NASA CR-134458

NASIS DATA BASE MANAGEMENT SYSTEM - IBM 360 TSS IMPLEMENTATION

IV - PROGRAM DESIGN SPECIFICATIONS

NEOTERICS, INC.

prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

NASA Lewis Research Center
Contract NAS 3-14979

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-134458		

4. Title and Subtitle	5. Report Date
NASIS DATA BASE MANAGEMENT SYSTEM - IBM 360 TSS IMPLEMENTATION IV - PROGRAM DESIGN SPECIFICATIONS	September 1973
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No. None
	10. Work Unit No.
9. Performing Organization Name and Address Neoterics, Inc. 2800 Euclid Avenue Cleveland, Ohio 44115	11. Contract or Grant No. NAS 3-14979
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes
Final Report. Project Manager, Charles M. Goldstein, Computer Services Division, NASA Lewis Research Center, Cleveland, Ohio

16. Abstract

The NASIS development workbook contains all the required system documentation. The workbook includes the following eight volumes:

I - Installation Standards (CR-134455)
II - Overviews (CR-134456)
III - Data Set Specifications (CR-134457)
IV - Program Design Specifications (CR-134458)
V - Retrieval Command System Reference Manual (CR-134459)
VI - NASIS Message File (CR-134460)
VII - Operating Specifications (CR-134461)
VIII - Data Base Administrator User's Guide (CR-134462)

17. Key Words (Suggested by Author(s))	18. Distribution Statement
	Unclassified - unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	595	$10.75

* For sale by the National Technical Information Service, Springfield, Virginia 22151

TABLE OF CONTENTS

TOPIC A.1 - MT/T MONITOR

A. MODULE NAME

Multi-Terminal Tasking Monitor
Program-ID - RMTTSUP
Module-ID - MTTSUP

B. ANALYST

Robert L. Rutledge
Neoterics, Inc.

C. MODULE FUNCTION

The MT/T Monitor is the single program which effects communication among the NASIS application program, the TSS/360 operating system and the NASIS user community. This program is responsible for the allocation of the NASIS resources to the user community, the handling of all terminal input/output for the user community and the processing of those NASIS functions which require more or less direct interface with TSS/360. In addition, the Monitor supports processors for those NASIS user commands which pertain to the Monitor itself (such as communicating with other users, listing active users and so on).

The MT/T Monitor is written completely in 360/67 machine code via the TSS/360 Assembler. The TSS user macro library (SYSMAC) and the TSS system programmer macro library (ASMMAC) are used to obtain the TSS system facility macros (for terminal communication, data management and so on) and some macros in the Monitor itself are used for convenience in coding. (These particular macros are described below.)

Although the primary consideration in the coding of the Monitor is execution time, special effort has been made to make the coding itself as lucid and informative to the reader as possible. Profuse self-documentation and comments make it quite probable that specific questions left unanswered by this document may be answered by a quick look at the listing of the Monitor.

Finally, all the facilities of the Monitor have been incorporated into one program. This is done merely to get everything in the same place for convenience; it is felt that nothing would be really gained by splitting the Monitor into several programs.

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

a. Monitor Macros

The macros incorporated into the Monitor itself (they are the first things in the program) are used only for ease of coding and reading. They are in no way necessary in the sense that they could be replaced by the expanded coding with no detremental effect to the execution of the Monitor.

Each macro included in the Monitor is now described along with the parameters it expects and its precise function.

1. FMT

This macro is used to convert an internal hexadecimal data item into a corresponding EBCDIC data item (i.e., convert internal hex into printable characters). It is used mostly for formatting error codes and so on for the operator. The operands for this macro are the target field address and length and the source field address and length.

2. RTRN

The RTRN macro is used to cause control to be returned to a point described by a TSS/360 interrupt push-down area. This macro merely points a register at the push-down area (the only operand) and issued the svc which causes control to be transferred to the MTTRTRN routine to switch push-down areas. (For a further discussion of this technique, see the description of the MTTRTRN routine.)

3. MSG

This macro is the means the Monitor uses

to send messages to the (NASIS) operator. The operands for this macro are the message, message and (optionally) a return address. This address is branched to after the message is sent if it is specified to the macro. This macro sets-up the message pointer in register one, the message length in register zero and the return address (if any) in register fourteen and calls the MTTPRMPT routine. (In addition, if operator response is expected, this macro negates register zero.) See the MTTPRMPT routine description for further information.

4. TIME

This macro is a more useful version of the (TSS) EBCDTIME macro. It accepts the same operands as EBCDTIME but also accepts defaulted operands and register-notation operands. This macro sets-up parameter registers and calls the MTTTIME routine so as to conserve space in the PSECT. For more information, see the description of the MTTIME routine.

5. MOVE

This is the macro which every program written having to move data around has in it in some form or another. Ours puts the operands for the macro (target field pointer, source field length and source field pointer) in registers and calls the MTTMOVE routine, the description of which later in this document will give you more information.

6. TRAN

This macro is analogous to the MOVE macro except that it translates instead of moving. The operands for this macro are the field pointer, the field length and the address of the translation table. After setting-up the parameters this macro calls the MTTTRAN routine to actually translate the text.

7. ENTR

This important macro is used to take care of all the linkage conventions, setting-up base registers, terminating timers and so on each time the application calls one of the Monitor's service routines. Basically, it saves all the calling registers (including the floating-point registers), the return address and the program mask in the push-down area in the task control table (see the description of the TCTE table below), kills the user's time-slice timer by calling MTTUNTIM and sets-up all the base registers for the Monitor. There are no operands for this macro which is kind of the mirror image of the RETN macro discussed next.

8. RETN

This macro is used to cause return to the caller of a Monitor service routine. It expects the registers and so on to have been saved by a corresponding ENTR macro and effects return by merely flagging the sub-task as dispatchable and branching to the queue-scanning routine (MTTFNDQ1). (By and by the queue-scanner will find this sub-task waiting for dispatch and issue a RTRN macro pointing to the push-down area in the user's task control table.)

9. RECORD

This macro is used to record an event and/or data within the Monitor. It uses a VMHOOK-like mechanism to cause the System Internal Performance Evaluation (SIPE) processors to record a data area onto the system's recording tape. There are various and sundry operands to this macro which describe the animal being recorded. You are referred to the section on event recording (below) for details on this and the rest of the data recording mechanism within the Monitor.

b. Overview

The MT/T Monitor, with the exception of the code to set-up NASIS and to take it back down, consists of one main routine to scan the work-to-do queues which the program uses to keep track of what's going on and a bunch of subroutines to actually do the indicated work. Thus, the flow of control through the Monitor is queue-scanner to subroutine to handle a requested function to queue-scanner and so on. When the queue-scanning routine finds nothing whatsoever to be done, it enters the (TSS) "WAIT" condition.

As most of the Monitor consists of the subroutines to perform specific functions for the Monitor, a quick list of these functions is in order. First, there are the task controlling functions: dispatching and time-slice-ending a sub-task. There are several routines for performing input/output on a sub-task terminal. There are quite a few routines to handle NASIS "user" commands which are more easily processed by the Monitor than by any other part of NASIS. And there are the normal subroutines which can be called by anybody for the grunge tasks: moving text, translating text, sending messages and so on.

One final section of the Monitor doesn't quite fit the description above. This is the asynchronous (SYSIN) attention interrupt processing routine. This is the program which gains control when the (MT/T) operator hits the ATTN key on the operator's terminal. Thus, this is the routine which actually communicates with the operator. All it does is read (Monitor) commands from him and exeute them.

c. External Specifications

1. Module Name - RMTTSUP

2. PSECT Name - MTTSUPP

3. CSECT Name - MTTSUPC

4. Entry Point Names

a. NASIS (To initialize and set-up for execution the entire NASIS system.

b. MTTWRITE (To write text to a sub-task terminal.)
c. MTTREAD (To read text from a sub-task terminal.)
d. MTTWREAD (To first write text to and then read text from a sub-task terminal.)
e. MTTFLUSH (To empty the output text buffer by writing it to a sub-task terminal.)
f. MTTXTR (To obtain various information about a sub-task.)
g. MTTPASS (To prompt a sub-task user for his "security code".)
h. MTTGETIM (To obtain the elapsed time statistics for a sub-task.)
i. MTTMUST (To enter "must-complete" mode for a sub-task.)
j. MTTTSE (To force a time-slice-end condition on a sub-task.)
k. MTTLMT (To obtain the current NASIS limits for printing, searching and so on.)
l. MTTPGMIN (To process a program interruption.)
m. MTTSPEAK (To process an operator asynchronous attention interruption.)
n. MTTTSEND (To process a time-slice-end timer interruption.)
o. MTTRTRN (To process a "return" SVC interruption.)
p. MTTRESET (To process the operator attention resetting (TSS) command.)
q. MTTKA (To process the user "KA" command.)
r. MTTKB (To process the user "KB" command.)
s. MTTMSG (To process the user "MSG" command.)
t. MTTHELP (To process the user "HELP" command.)
u. MTTUSERS (To process the user "USERS" command.)
v. MTTNUSER (To process the user "NUSERS" command.)
w. MTTDATIM (To process the user "DATETIME" command.)
x. MTTWAIT (To process the user "WAIT" command.)
y. MTTUTAB (Pointer to the MT/T User

Information Table for the executing sub-task (if any).)

z. MTTTCT (Pointer to the start of the terminal table.)

aa. MTTTRQ (Pointer to the executing terminal table entry (if any).)

bb. MTTTCTE (Pointer to the executing task control table (if any).)

5. External References

a. SYSINDCB (Pointer to the data control block for SYSIN.)

b. TSATIN (Pointer to the NASIS attention interrupt handling routine.)

d. Datasets

The Monitor requires two datasets which are used in its initialization phase. One contains the list of NASISIDs which are allowed on the system and the other contains Monitor commands which are to be executed automatically before NASIS is opened up to the user community.

1. NASIS.USERIDS

This dataset is the one containing the list of allowable NASISIDs. It also contains information about which NASISID is allowed to look at which file, but the Monitor only uses the first four fields in each record: the NASISID, the password (if any), the time-slice value to be initially assigned to the NASISID and the authority code to be initially assigned to the NASISID. (This last field is not used at the present.)

2. NASIS.COMMANDS(0)

This TSS/360 "line" dataset contains any Monitor commands which the operators wishes to have automatically executed each time NASIS is brought up. Examples of such commands are "limit", "pgmstop", "news", and so on.

e. CONTROL TABLES

The following section discusses the tables

which the Monitor uses to drive itself. Only general descriptions are given here for the formats of the tables; we would rather you know what their functions are. Each of these tables is described by a DSECT in the listing of the Monitor, so you are referred there for the formats of these tables.

1. Terminal Table (TRQ) (DSECT "TRQDSECT")

   This is the most basic table used by the Monitor. It is a list and count of the users attached to NASIS at any particular time. TSS, through the "Q" macros (WRITEQ and READQ) return a terminal identifier entitled the "relative terminal number". This is a number from zero to the current number of attached users and this is the number which is used to index the terminal table to find the entry for a particular user. All the pointers to information tables for the user originate in the terminal table. The things contained in this table are: the pointer to the task control table, the pointer to any message control blocks for this sub-task, the symbolic device address of the terminal, the NASISID of the user attached to this terminal (if any, this field is filled in during logon), the flag indicating whether the sub-task is waiting for dispatch and the "WAIT" timer for the sub-task, if any.

2. Task Control Table (TCT)
   (DSECT "TCTDSECT")

   This is the table that contains all the information about and working storage for a sub-task. All the (normal) register saveareas, task-related temporaries, task indicators, Monitor saveareas, timing information, I/O buffer information and so on are in this table. In addition, all the user information is here: NASISID, password, task-id and so on. When the Monitor can locate this table for a particular user, it knows all there is to know about him.

3. Message Control Block (MCB)

(DSECT "MCBDSECT")

This control block is constructed each time a message is to be sent to a sub-task. All it contains are the message text and length and a pointer to the next message control block on the chain, if there is one.

4. Attention Table (ATN)
(DSECT "ATNDSECT")

This table is dynamically built each time a user hits the ATTENTION (or BREAK) key at his terminal. The table contains all the information about the interrupt (which is almost asynchronous) to pass on to the NASIS attention processing routine. In addition, it contains a savearea for that routine to temporarily save the contents of the interrupted DSA. In addition, if the attention interruption processor wishes to modify the address to which control is to be returned after the interrupt is processed, it so notifies the Monitor through this table.

5. FINDQ Return List (DSECT "CHAFNQ")

This is a control block constructed and maintained by TSS. It is the information returned to the Monitor after it executed a READQ or WRITEQ and contains the terminal number mentioned earlier plus information about any text read in and some indicators for line hang and attention.

6. User Table Entry (DSECT "USRDSECT")

This thing is really only a descriptor for the format of a record in the NASISIDs list dataset, "NASIS.USERIDS".

7. Parameter List (DSECT "PRLDSECT")

This is the control table passed to the Monitor whenever the application calls it to process a terminal I/O request. It describes the parameter list expected from and returned to the caller. (This

parameter list is identical to the one used by TSS/360 GATE except that there is no ADCON pointing to control information in the first word.)

8. Operator Attention Savearea (DSECT "SAVDSECT")

This table is built by the Monitor each time the operator hits ATTENTION and causes entry to the operator attention interrupt processor (MTTSPEAK). This table contains all the registers and VPSW as of the interrupt and is kept (in a chain) because the operator is allowed to hit ATTENTION again while in this routine and we wish to be recursive. Thus, this table contains only the registers and VPSW as of the interrupt and a pointer to the next table on the chain, if any.

9. Limit Table (DSECT "LIMDSECT")

This table is where the Monitor keeps all its information about NASIS limits. (The things which may be limited are total number of users, number of users of a particular class, number of records in a set which may be searched on or printed and so forth.) This table consists of a header containing the limiting numbers for those things which are always limited and entries for each class of NASISID which has been manually limited by the operator.

10. Recording Area (DSECT "RECDSECT")

This table is the area in which the Monitor posts the data which it records (via the TSS "SIPE" mechanism, see the section on recording below). This table is also used to pass the data along to the recording mechanism in the TSS Supervisor. It consists of the actual data to be recorded plus some control and save areas which the recording mechanism expects.

11. Data Control Block (DCB) (DSECT "CHADCB")

This table is used by the Monitor to interface with the TSS access methods used in the handling of the various datasets the Monitor uses (see the section on datasets above). One DCB is set-up by the Monitor and serves for all the datasets it uses. For more information on the Data Control Block, you are referred to the "System Control Blocks" Program Logic Manual (GY28-2011).

12. Interrupt Storage Area (ISA) (DSECT CBAISA)

This control table is really page zero in virtual memory. The DSECT supplied by TSS is used to direct the Monitor to the correct memory locations corresponding to the items it wishes to reference. (For further information on the Interrupt Storage Area, you are referred to the "System Control Blocks" Program Logic Manual (GY28-2011).

f. Detailed Description

The following section contains a detailed description of the MT/T Monitor. It is assumed that the reader has a firm grasp of 360/67 machine language and the principles of the ISS/360 Operating System.

1. MTISTART routine

MTTSTART (with the entry point NASIS) is the routine which the MT/T Interface Module (CZCTC) calls when the MT/T operator enters the MTT command at the operator terminal. On entry, this routine establishes linkage and sets-up all the program base registers (R13 for the PSECT, R8-R12 for the CSECT). It next initializes some of the more important program variables--the flags, user counter, table pointers etc. It then initializes the (Monitor's) recording mechanism and obtains a two-page pool to build the limit table, news text area and terminal table in. The terminal table is aligned on a page boundary so as to minimize paging later on. Coincident with building these

tables, their pointers are posted in the PSECT. Next, the (operator) attention is specified and enabled, as is the program interrupt routine and the SVC 63 interrupt routine (for the RETURN SVC). In all cases, the SIR function is used to actually specify the interrupt routine. At this point, the NASISID-containing dataset (NASIS.USERIDS) is read and the four fields mentioned earlier from each record are posted in an in-core table (this is done so that the Monitor doesn't have to waste time reading the dataset each time a user logs on). After this dataset is processed, the command dataset (NASIS.COMMANDS(0)) is read in and the commands in it are executed. This is the last function for this routine, and after it is completed, it sends a message to the operator (MSG) to the effect that it is finished and commences execution of NASIS by exitting to the MTTFINDQ routine at MTTFNDQ1.

2. MTIEND Routine

This routine is the inverse of the MTTSTART routine and is entered by being the target of the STIMER issued in the shutdown routine (MTTSHUT). (It may also be entered by a direct branch to MTTEND1 in the case of an immediate shutdown request.) Upon entry in either case it first scans through the terminal table looking for active users and calling the MTTQUIT routine to log them off after sending them messages telling them that the system is shutting down. After it has gotten rid of all the users in this fashion, it returns the storage for the userid table and deletes all the interrupt routines via the TSS DIR function. After this is done, the routine sets a timer such that it sits idle for fifteen seconds. This is done so that all the user terminals have time to finish typing their FREEQ messages before the Monitor returns control to the TSS system which will terminate any I/O going on each terminal is it gets a chance. After the wait is over, the routine frees the pool of memory used to

build the three tables mentioned earlier and returns to the system through standard return linkage.

3. MTTFINDQ Routine

This is the routine which scans through the various queues looking for work for the Monitor to do. It has two entries: MTTFINDQ for callers who wish control returned to them at the termination of the I/O operation for the sub-task and MTTFNDQ1 for callers who are merely finished with a task. The only difference is that at MTTFINDQ, the registers are saved in the TCT for the sub-task in question and at MTTFNDQ1 they are not. The first thing either routine does is to zero out the pointers for the current TCT and TRQ so as not to confuse any of the interrupt routines. It then sets-up the pointers and indices to the terminal table in registers (for speed, it may have to scan the terminal table several times). The first scan through the terminal table is to look for sub-tasks with outstanding messages. If any are found, they are send via calls to the MTTWRITE routine at MTTWRIT1. (All internal calls on MTTWRITE are made to MTTWRIT1.) Before a message is sent, however, the MCB for the sub-task is locked out so that we don't try to send two messages to the task at the same time (this would confuse the Q routines which don't quite know what to do with a busy condition from a terminal). Also, no messages are sent to terminals with the "I/O active" flag on in the TCT for the same reason. After the terminal table has been scanned for messages, it is scanned for sub-tasks waiting to be forced. For each that is found, a message is sent and the MTTQUIT routine is called to actually disconnect the terminal. Care is taken so that this routine doesn't try to force a task more than once. These two scans are not particularly functionally important, but they must be done first because if they are done after the I/O and execute scans, the terminals would always appear

to be busy and both these functions rely on the terminal being free.

The next scan is done with the aid of the TSS FINDQ function. This facility scans through the terminals attached to NASIS and returns a non-zero return code if there are any with I/O completion status. If the Monitor finds one such, it records the return code (REC) and then checks for a hung-up phone line (a flag in the FINDQ list) and performs a logical disconnect on the terminal (MTTQUIT) if it finds this to be the case. It also manually checks for an indication of attention in addition to whatever code it has and calls the MTTCLEAR routine to "flush out" all other outstanding attentions from the terminal (because RTAM has the nasty habit of telling us that there were 1-3 attentions when there was in fact only one). After these checks are made (if it was a line hang, control has gone somewhere else) the routine locates the TCT for the terminal with the completion status by indexing into the terminal tabls with the relative terminal number and locating the TCT pointer in the terminal table entry. It then posts the terminal in question as the currently running task, restores the registers which it must have saved in the TCT (because everybody calls MTTFINDQ to await the completion of all I/O operations) and returns to the original caller through register 14.

If no terminal completion stati were found, the final scan through the terminal table is made. This scan looks for sub-tasks waiting to be dispatched. These will have the work-to-be-done flag on in the TRQ entry. After one of these is found, quick checks for terminal busy (it's receiving a message) and system down (somebody did a shutdown) are made. If the terminal is busy, the task is merely skipped over; if the system is shutting down, the task is marked as not waiting for dispatch and the scan is continued. If the sub-task is found to be OK to be dispatched, the pointers are

set to reflect the task now executing and that task is returned to by a direct branch to the dispatching routine (MTTDISPR).

If absolutely no work is found to be done, the Monitor relinquishes execution by telling TSS it must wait for some event to complete related to the NASIS task. This is done by merely issuing a (TSS) WAIT macro. Upon return from the WAIT control is transferred to the beginning of the queue-scanner to determine what event it was which completed.

4. MTTDISPR routine

This is the routine which transferrs control to a sub-task using the saved registers and vpsw in the task control table. All it does is start the user's time-slice timer by calling the MTTTIMER routine and then execute an RTRN pointing to the push-down area in the TCT.

5. MTTTIMER routine

This routine handles the initiation of the STIMER which starts a sub-task's time-slice execution time timer. Upon entry, it saves a few working registers in the TCT for the sub-task, picks-up the value to be used for the time-slice this time (TCTNTS), adds one to it (figuring that the coding in the Monitor plus the SVC processing tine for the RETURN will add up to a millisecond) and issues the STIMER using timer number seven. If the STIMER fails, the task is forced since we can't run without time-slicing. Return from this routine is to the caller through register 14.

6. MTTUNTIM routine

The MTTUNTIM routine is called when a Monitor service routine has been entered and there is still a time-slice timer running. It turns off that timer as the Monitor does not want to be time-sliced while it is handling a service request

for the applcation. Upon entry, some registers are saved in the TCT for the task and a TTIMER (CANCEL option) is issued to turn off the timer and return the amount of unused time in it. This unused time subtracted from the amount of time the last time slice started with is added to the user's total CPU time used (after the same one millisecond added in MTTTIMER is subtracted off). Also, a check is made here for a recent call on MTTMUST (flag TCTMUST). If this has happened, the next-time-slice value is reset to the user's time slice value (as it was originally set to an hour by MTTMUST). An error from the TTIMER does not kill the task, but the operator is notified of the anamoly. Return is, again, through register 14 to the caller.

7. MTTTSEND routine

MTTTSEND is the entry point specified as the exit routine to be called when the time-slice timer runs out. Thus, it is the routine which is called to recognize the fact that a time slice is finished and cleans-up and updated the timing statistics and places the time-slice-ended user back in the dispatch queue. After it is entered, it loads all the base registers including the bases for the terminal table entry and TCT for the executing task. It then checks to see if the timer interrupt occured in the Monitor and if it did it ignores the interrupt by merely there was indeed a user executing at the time of the interrupt by seeing of the TCT pointer is zero. If it is, a branch is taken to the code to ignore the interrupt (below).

At this point, MTTTSEND updates the user's timing statistics by adding the value used to initiate the last time-slice (TCTNTS) into the accumulated CPU time for the task (TCTCPUTM). TCTNTS is also reset to the user's time-slice value (TCTTIMER).

Now the routine checks to see if the

interrupt occured while the Monitor was executing by checking the interrupt VPSW address against the lower and upper bounds of the Monitor. If the interrupt occured within the Monitor, it is ignored by undoing the linkage and returning to the Task Monitor by branching through register 14. If the interrupt was legitimate, all the registers and VPSW as of the interrupt are saved by moving the push-down area provided by the Task Monitor into the user's TCT. The routine then places the task in the dispatch queue by turning on that flag in his terminal table entry (TCTWKSW). It now overlays the registers and VPSW in the push-down area with the registers for the Monitor and address of MTTFNDQ1 and causes control to return to the queue-scanner by returning to the Task Monitor. (This means of transferring control is used throughout the Monitor because the Task Monitor will get confused if we don't return to it after each interrupt so it can dequeue them from its interrupt chains.)

8. MTTTASKI routine

The MTTTASKI routine is the routine which is branched to directly when the queue scanner finds an "initial connection" interrupt as a return from a FINDQ operation. It means that a user has just typed BEGIN NASIS at a terminal. After some initialization, this routine checks to see if the application is shutting down (DOWNSW on) and if this is the case, it pretends the terminal hung up and calls MTTQUIT to disconnect it. At this point, the number of active users (MTTUSER#) is incremented by one.

MTTTASKI now checks to see if the terminal is one of the CRTs recognized by the application--a 2260 or CCI CC-30. If it is, the size of the area for the TCT is increased as these devices require a larger i/o buffer for their screens. (CC-30s are located by examining the list of CC-30 Symbolic

Device Addresses hard-coded into the Monitor; 2260s are located by checking the device-type field in the FINDQ return list.) Now that it knows how much space it needs, this routine obtains space for the task control table via GETMAIN. As in all cases within this routine, failure to obtain space for one of the tables results in the terminal being disconnected via the MTTQUIT routine. After the table is gotten, its pointer is posted in the terminal table entry for this sub-task and as many fields are filled in in the TCT as the Monitor can, for example all the buffer pointers and counters, terminal type, and so on. The sign-on time for the user is obtained via the REITIM macro (this is the number of micreseconds since March 1, 1900) and posted in the TCT.

As this point, the limit table is investigated to see if this user is allowed on, the criterion being the number of (total) users allowed on and the number that are already on. If this user overflows the limit, he is so notified and kicked off by calling MTTQUIT. If he is allowed on (so far) his NASISID and password are prompted for and read by calls on the MTTREAD and MTTWRITE routines (at MTTWRIT1 and MTTREAD1, respectively). After the user's NASISID is obtained, the limit on the members of his class (which is the first two characters of the NASISID) are checked and he is kicked off as before if the limit for his class has been reached. Also, only three tries to enter a correct NASISID or password are allowed. After all the information about the user is gathered in, his terminal table entry and TCT are filled in the rest of the way. At this point, a message is sent to the operator telling him (her) that somebody is logging on.

Now that the user is properly connected, he is informed of several things. If a shutdown has been requested, the user is informed of the time at which the

application will terminate. He is also sent any news that has been typed in (this is taken from the NEWSLTH and NEWSTEXT fields). Finally, a call to MTTFLUSH at MTTFISH1 is made to empty all this information plus his Monitor logon message to his terminal. Now the user is given virgin registers and a VPSW pointing to the application entry point (these are posted in his TCT) and he is placed in the dispatch queue. This routine exits to the queue scanner at MTTFNDQ1.

9. MTTQUIT routine

This is the logoff processor for the application. It also is used to force users off the system for one of a number of reasons. There are two entries to this routine: MTTQUIT for callers who don't wish to be returned to and MTTQUIT1 for those who do. This routine first gets itself a safe place to save registers and then decrements the count of active users by one (to match the early processing of this counter by the MTTTASKI routine) and flags the user as being ditched by posting the TCTQUIT flag in his TCT (if he has a TCT, remember he could have done something like hung up his terminal during the logon process). After cleaning-up the user, killing off any queue switches, turning off his time-slice timer, and so on, a message indicating the logoff is sent to the operator.

Now all MCBs and attention tables left un-released are gotten rid of by FREEMAINing them. (This is done because it is quite likely that attention tables in particular were not released during the user's session because some attention processing is of the GOTO form instead of the RETURN form.) Now the total elapsed CPU and connect times are calculated and formatted into a logoff message for the user. The user's TCT is now freed (after the terminal number is saved from it) and his terminal table entry is zeroed out. The logoff message is transmitted to the user's terminal as

an option on the FREEQ macro which is used to disconnect his terminal. This is the last operation this routine has to perform and when it is finished, it either returns to the caller through register 14 or to MTTFNDQ1 (queue scanner) depending on which entry was originally called.

10. MTTWRITE routine

This is the routine which is used to write text to a sub-task terminal. It has two entry points: MTTWRITE for calls from the application program (it assumes standard TSS/360 type I linkage and performs initialization with the ENTR macro) and MTTWRIT1 for calls from within the Monitor itself (it assumes register 14 to contain the return address and all the other base registers to be already set-up). For external callers, the parameters to the routine (text pointer and length) are taken from the parameter list pointed to by register 1 and for internal callers they are moved from registers 0 and 1. This is all the initialization that MTTWRITE does.

Checks are now made for one of the accepted CRTs and if one is found, control is transferred to one of the two routines to process these terminals. They are described below. At this point, all the registers used throughout the routine are set-up. They include registers containig constants, input string and output buffer pointers and counters and zeroed-out registers for character moving. The first thing checked for is a ':' at the end of the input string signifying that the caller wishes to inhibit the carriage-return that is normally added to the end of each line of output. A flag is turned on to remember about this later if the character is found. In addition, the colon is removed from the text. All trailing blanks are also stripped from the end of the text to be output (for neatness). A final check is made to see if the task's I/O buffer is completely

filled and if it is, it is emptied out by calling MTTFLSH1 to send the buffer to the user's terminal.

If the last operation completed at the terminal was a read, idles are added to the front of the text to give the carriage time to complete returning to the left margin. Also in this case, a linefeed is added to the front of text going to teletypes (as then do not automatically add a linefeed to their "new line" sequence). If the last operation to the terminal was a write and the terminal is a teletype and the last-written line was not carriage-hanged, a linefeed is added to the front of the text for the same reason. At this point, the caller's text is processed.

All processing of the caller's output text is done character by character. A character is fetched, examined, possibly given special treatment and then placed in the I/O buffer in the sub-task's TCT. Special treatment is given to the following text characters:

a. Idle characters are ignored completely (because they are used for special applications in the Monitor.

b. Linefeed characters are followed by idle characters for some terminals.

c. Backspace characters cause the distance across the carriage (for calculating the number of idles required after a carriage-return) to be decremented by two.

d. Carriage-return characters are followed by enough idles to allow the carriage to return to the left margin. Also, the distance across the paper is zeroed.

e. Tab characters are followed by an estimated number of idles (we don't know how far the tab is going to

move the carriage). The distance across the paper is also given an estimated increment.

If at any time during the moving of text the I/O buffer is filled, it is flushed to the terminal by a call to the MTTFLUSH routine at MTTFLSH1.

After all the text has been processed, a carriage return is added to the end of the line if the caller did not request the carriage to be hanged. Idles are added after this carriage return to allow the carriage to return to the left margin. If the TCT I/O buffer has been exactly filled by this write, it is flushed out by calling MTTFLSH1. After all this is done, all the buffer pointers and counters and distances are updated in the TCT and flags indicating whether the line was hanged or not and that the last operation was a write are posted in the TCT. The method of return to the caller is determined by the entry point called.

The two special routines for writing to CRTs merely move the text to the TCT I/O buffer, call the MTTWRQ routine to write it out, call MTTFINDQ to wait for the write to complete and branch to the common exitting code for MTTWRITE.

11. MTTCHARS subroutine

This subroutine is used to insert characters into the TCT I/O buffer. It is used extensively by the MTTWRITE routine to move idles, linefeeds and so on. It determines whether there is room for the characters it has been requested to post in the buffer and if there is it merely moves them in. If there is not room, it flushes the buffer to the user's terminal by calling MTTFLSH1 and then moves the characters into the buffer. Return is to the caller through register 14.

12. MTTREAD routine

This is the routine which reads text

from a sub-task terminal. It too has two entry points, one for the application to call and one for the Monitor to call. MTTREAD, the external entry, does its linkage/base register initialization via the ENTR macro. MTTREAD1, the Monitor's entry, merely moves the parameters to the appropriate registers. The parameters in the case of a call to MTTREAD are in a standard TSS parameter list. This routine first sees whether there is any output text in the TCT I/O buffer. If there is, it is written to the terminal and then text is read from the terminal by a call to the MTTWRQAR routine. If there is no text in the buffer, the terminal is merely read by a call to the MTTRDQ routine. If there are no error returns from the "Q" routine, the MTTFINDQ routine is called to await completion of the I/O operation.

After the text has been read from the terminal and presented to the MTTREAD routine by pointers in the FINDQ return list, it is translated from line code to EBCDIC with the TRAN macro. If the user is in "folded" (KB) mode, the text is translated once more to "fold" his lower case input to upper case. If there is a carriage return on the end of the input string, it is stripped off. If the line if input ends with either a "#" or a "/", the user has cancelled the line and control returns to the common entry to read another line from the terminal. Otherwise, the line is scanned for backspace characters which are processed by moving the text to the right of the backspace character two spaces left (over the backspace and the character preceeding it). Under no circumstances if the line backspaced over the "left margin". After the backspaces are processed, a check is made to determine whether or not the line ends with a "-", indicating continuation. If it does, the continuation flag is turned on in the return codes being built and the section for processing trailing blanks is skipped over. If there is no continuation indicator, all trailing

blanks at the end of the line are deleted.

Now the length of the text input is checked against the length requested by the caller. If the user typed in more text than the caller allowed for, the truncation indicator is turned on in the return code being built and only the amount of text requested by the caller is moved. Otherwise all the text the user typed is moved to the caller's target area. Now the terminal buffers and controls are released back to the system via the CLEARQ facility and the returns from the CLEARQ are checked for attention. If an attention is detected during the CLEARQ, the attention indicator is posted in the return codes being built.

At this point, all the flags indicating what has happened are posted in the TCT, the length of the user's input text is posted in the caller's parameter list and the caller is returned to either by a branch through register 14 for internal callers or a RETN macro for external callers.

13. MTTWREAD routine

This routine is mostly a convenience item for the application. All it does is internally call MTTWRITE and then MTTREAD (at MTTWRIT1 and MTTREAD1) to effect a "GTWAR"-type function. There is no internal entry to this routine. After the routine has performed initialization with the ENTR macro it moves the parameters from the TSS parameter list to the registers the internal entries to MTTWRITE and MTTREAD expect and calls each of the routines. Checks for errors from MTTWRITE are made before the call to MTTREAD is made and if any are found, the call to MTTREAD is skipped and the error indicators are returned to the caller. Return to the caller from this routine is made via the RETN macro.

14. MTTFLUSH routine

This routine is the routine which empties the I/O buffer (in the TCT) to the user's terminal and resets all the buffer pointers in the TCT to reflect the now-empty buffer. It has an entry for external callers (MTTFLUSH) which initializes with the ENTR macro, and an entry for internal callers (MTTFLSH1) which merely saves some working registers.

After initializaion if finished, this routine checks to see if there is anything at all in the buffer. If there isn't, it merely branches to the code to return to the caller. If there is, the routine checks to see if the terminal type is a 1050. If it is, an end-of-block character is added to the end of the text in the buffer. (This routine relies on there being an extra character in the buffer at the end of same in case the buffer is filled.) In the case of an external caller, the MTTWRQ routine is called to empty the buffer; in the case of an internal caller, the "Q" routine is one of the parameters to this routine and that is the routine which is called. After return from the "Q" routine, the MTTFINDQ routine is called to await the completion of the I/O operation. After this is finished, the caller is returned to either by branching through register 14 or by a RETN macro.

15. MTTRDQ, MTTWRQ and MTTWRQAR routines

These three routines are the routines which perform the actual I/O requests on user terminals. After performing the initialization unique to each operation and issuing the I/O request, they all user common coding to analyse the results and return (label QCOMMON).

MTTRDQ saves some working registers and then issues a READQ request to the appropriate relative terminal number with the operand TRNSL=N (because we do our own translating). It then joins the common finalization code.

MTTWRQ saves registers and translates the text to be written to line code by using the TRAN macro. (In the case of a 2260, no translation is done as of yet.) It then issues a WRITEQ request pointing to the appropriate relative terminal number, output text and output text length with the additional operand TRNSOUT=N (since we just translated the text ourselves). It then joins the common finalization code.

MTTWRQAR behaves almost the same as MTTWRQ except that the operands RESP=Y and TRNSIN=N are added to the WRITEQ request. It then falls through to the common finalization code.

After the "Q" operation has been requested, QCOMMON executes into a branch table indexing on the return code from the operation. This table either adjusts the return code to the Monitor's internal code set, branches to an error routine which tells the operator about the unexpected "Q" return code and then pretends it was an I/O error, or leaves a zero (good) return code alone. It then restores all the working registers (except the one with the return code) and returns to the caller through register 14.

16. MTTATTN routine

MTTATTN is the routine which is branched to directly by the queue-scanner when that routine finds a "naked" attention return code from one of the terminals after a FINDQ operation. This means that we have received almost an asychrounous interrupt from the sub-task and it requires some special processing. (It is almost an asychronous interrupt because we are assured that it can only be found at the end of a time-slice, so we really have available the "interrupt" registers and vpsw.) Also, the application has set-up routines which are to be called when attention interrupts (this kind, not the kind which merely add a bit to the I/O request return code) occur and the

routine itself requires special processing.

On entry, MTTATTN locates the TCT for the terminal which got the interrupt by indexing into the terminal table on the relative terminal number in the FINDQ list. If there is no TCT, the terminal is hung-up by the issuance of a FREEQ operation with no operands. After the TCT is found, it is marked not busy and the terminal table entry and TCT pointer are posted as being the currently executing ones. At this point, the MTTCLEAR routine is called to "flush out" any more attention indicators from the terminal.

Now, the routine has to check a bunch of special cases. If any of the following events are happening, the attention will be ignored:

a. The application is shutting itself down (MTTEND has been entered),

b. The user is not completely logged on (TCTONSW is zero),

c. The user is being logged off (TCTQUIT has been entered for him),

d. The user is waiting to be forced (TCTFORC is on).

Also, if this routine can determine that there is a set of saved registers for the MTTFINDQ routine, it "cheats" and merely signals to the caller of MTTFINDQ that it got an attention return code from whatever I/O operation was going on. This is done because it is simpler than calling the application attention routine and accomplished the same thing.

If control gets this far, the attention is deemed valid and the application attention processing routine has to be called. MTTATTN first obtains an area in which it can post all the information about the interrupt to be sent along to

the attention routine. If there is no space for this attention table, the operator is notified and the attention is ignored. Otherwise, this new attention table is added to the end of the chain of attention tables for this sub-task (the chain starts at label TCTATTN in the TCT). After the table is linked in, all the interrupt information (push-down area) is moved from the TCT to the attention table. Now all the registers necessary for the application's attention routine are set-up in the TCT (in the interrupt register/vpsw slots). After this is finished, the dispatch flag is turned on in the terminal table entry and the queue-scanner is exitted to so the task will be dispatched in turn later. (When it is dispatched, it will be dispatched to the application attention routine.)

Upon return from the attention processing return (it is not necessarily going to come back to us, remember), the registers from the attention table are moved back to the TCT, the attention table is un-chained from the attention table chain, the task's dispatch flag is turned on again and the control is returned to the queue-scanner at MTTFNDQ1. (This will cause the sub-task to either take up where it was interrupted or to take up where the application attention routine wants it to go, having modified the interrupt registers and VPSW in the attention table.)

17. MTTCLEAR routine

MTTCLEAR is a fudge which is necessary because the TSS supervisor terminal routine (CEATC) tends to tell us lies about the number of times a user hits his attention key. This routine merely issues FINDQs to the terminal in question until it quits returning stati. It indexes into a branch table with each return code from the FINDQ until either the status becomes zero or an error occurs, in which case it calls the routine in the queue-scanner to hang up

the terminal. After a zero status is found, the caller is returned to by branching through register 14.

18. MTTGETIM routine

This routine is called by the application when it wishes to obtain the elapsed CPU and connect times used by a sub-task thus far. After initializing the linkage by an ENTR macro it issues a REDTIM macro to obtain the current system time-stamp and subtracts the sign-on time from the result. (This is done in 64-bit fixed-point arithmetic.) The other result sent back by this routine is merely the TCTCPUTM field which already contains the sub-task's elapsed CPU time. Return is made via the RETN macro.

19. MTTXTR routine

This routine is called by the application to obtain the following information about a particular sub-task:

a. The NASISID of the user,

b. The password (if any) for the user,

c. The taskid of the sub-task,

d. The indicator for whether the application is running in MT/T mode or in standalone mode,

e. The indicator for whether the user is conversational or non-conversational.

f. The indicator for whether or not usage statistics is to be disabled.

After this routine performs its initialization with the ENTR macro it starts looking at the parameter list sent to it by the caller. It only fills in the positional output fields as long as there are available target spots in the parameter list. The first output is

the NASISID which is put into a string dope vector (SDV) pointed to by the first parameter. The password in put into the SDV pointed to by the second parameter. The taskid is a 32-bit fixed number which is pointed to by the third parameter. (Note: This output used to be a character string also.) The next two parameters, the flags, are always set "on" (one) and are put into bit-string dope vectors as pointed to by the last two parameters. The last flag (usage statistics disabled) is first turned off then the internal flag for this condition is checked and if it is on, the parameter flag is turned on. After it is all finished, this routine returns to the caller by using the RETN macro.

20. MTTMUST routine

MTTMUST is the routine which the application can call when it wishes to perform a function without being time-sliced. All it does is initialize with ENTR and then overlay the next-time-slice value in the TCT (TCTNTS) with a value of one hour and turn on the indicator which says MTTMUST has been called so that the MTTUNTIM can make the appropriate corrections the next time it is called. The caller is returned to with the RETN macro.

21. MTTPASS routine

This routine is called to prompt the user for his "security code". This routine is used instead of in-line code because it already has the mechanism in it to prompt for a parameter with "black-out". (It uses this mechanism to prompt the user for his password at logon time.) After this routine is entered, it performs linkage and base-register initialization with the ENTR macro. Calls to MTTWRIT1 and MTTREAD1 are used to actually perform the prompting I/O at the user's terminal and this routine does not check the entered string. It merely sends whatever it gets back to the caller.

Return is made with a RETN macro.

22. MTTTSE routine

This routine is used by the application program to force a time-slice-end on a sub-task (for example, to get out of "must-complete" mode). All it does is ENTR, REC and then RETN which has the effect of taking the user out of execution mode, setting him up to be dispatched again later and calling the queue-scanner to see if there is anything to be done before re-dispatching this user.

23. MTTLMT routine

This routine is called by the application to obtain the current limits on the NASIS resources (prints, searches, sorts and records). After initializing with the ENTR macro, this routine merely moves those four words from the limit table header (label LIMPRNTS in LIMDSECT) to the 16-byte field pointed to by the caller's parameter list. Return to the caller is made via the RETN macro.

24. MTTSPEAK routine

MTTSPEAK is the entry point specified in the interrupt control block which handles attention interrupts from the SYSIN terminal (operator's terminal). When it is called, it conforms to the linkage laws and sets-up all the base registers for the Monitor. In addition, if there was an executing sub-task, its time-slice timer is turned off. This routine then obtains an area to save all the interrupt registers and VPSW in and chains this area to a chain of similar saveareas. This is done because the operator is allowed to hit attention while a previous attention is being serviced. After the table is chained in and the interrupt information moved into it, the Task Monitor is returned to briefly so it can dequeue the interrupt from its own chains. At this point, the operator is permitted to hit attention

again--before this time, the attention will be ignored by TSS.

After all its initialization is done, MTTSPEAK prompts the operator to enter a Monitor command by typing a question mark at his terminal and unlocking the keyboard. If the response from the operator is just a carriage return, he is leaving "debug" mode and a branch is made to the code to return to the point of interruption. If text was typed in, MTTSCAN is called to parse it into a command name and operands. Errors from that routine cause a message to be sent to the operator and a branch taken to the exitting code. Otherwise, the OPERCOM (operator command) table is searched for a match with the command entered by the operator. If a match is found, the appropriate command processing routine pointer is picked out of the table and the routine is called. If no match was found, a check is made to see if the Monitor is in "debug" mode. If it is, the line entered by the operator is fed to an OBEY macro in case it is a TSS command. If the Monitor is not in "debug" mode, an error message is sent to the operator and the exitting code is branched to.

After a command has been accepted one way or another and control returns from either the command processor or the OBEY macro, another check is made to see if the Monitor is in "debug" mode. If it is, the operator is prompted again (he will be prompted until he enters a null line)--if it isn't, the exitting code is fallen through to.

At exitting time, the attention routine is deleted and re-specified (CLATT, DIR, SIR, USATT) so that TSS doesn't lose it. After this is done, all task interrupts are disables by turning on a flag in the ISA and the interrupt information is moved from the current savearea to the Monitor's PSECT. The savearea is now un-chained and returned to the system (FREEMAIN), any sub-task time-slice timer is restarted, task interrupts are

re-enables and the routine returns to the point of interruption by executing a RTRN macro pointing to the interrupt push-down area which it moved into the Monitor's PSECT.

25. MTTPGMIN routine

This is the entry point specified in the interrupt control block for program interruptions. On entry, it saves the interrupt registers and VPSW in the Monitor's PSECT and causes the Task Monitor to dequeue the interrupt from its lists by returning to it and telling it to return to the Monitor instead of the point of interruption. If a sub-task was executing, his time-slice timer is turned off. In addition, if a user was running, his dispatch flag is zeroed, just in case somebody decides to dispatch him later.

Now this routine composes an operator message containing the program interrupt VPSW and all the interrupted general registers. After this message is composed and formatted, it is sent to the operator via the MSG macro. After the message is sent, the PGMSW switch is interrogated to determine whether to pause so that the operator can "look around". If that switch is on, a CLIC macro is issued, causing control to return to TSS command language until the operator enters a "GO" command.

Upon return from the CLIC or if no CLIC was issued, the routine checks to see if a user was responsible for the interrupt. If no user was running, exit is made directly to MTTFNDQ1. If a user was responsible, he is sent a message via MTTWAIT1 telling what happened to him and then he is forced off with a call to MTTQUIT.

26. MTTSCAN subroutine

This subroutine is called by sections of the Monitor that wish to parse a string containing a Monitor command. It merely separates the line into elements

delimited by commas or (strings of) blanks and posts pointers and counters for each so-delimited operand into a table called PARAMS. In addition, it recognizes the quote-mark convention used by TSS itself to allow the use of commas, blanks or quote marks inside of operands. If it finds an unmatched quote mark somewhere in the string, it sets the condition code to non-zero before returning, otherwise the condition code is set to zero. Return in either case is through register 14.

27. MTTKA routine

This routine is the routine which the application calls to process a user "KA" command. After initializing the entry (ENTR) it merely turns on the TCTKA flag in the user's TCT and returns via the RETN macro.

28. MTTKB routine

This routine is the processor for the user "KB" command and turns the TCTKA flag off instead of on. The same linkage is used as for MTTKA.

29. MTTMSG routine

This is the routine which is used to send a message from one user to another (where one of the users in question is allowed to be the MT/T operator). Entry at MTTMSG is for the application to use when it finds a user "MSG" command and entry at MTTMSG1 is used by the Monitor to process an operator "MSG" command.

At MTTMSG, initialization is accomplished by the ENTR macro. The userid of the message recipient is then verified by a call to the MTTGTUSR subroutine. A check is then made to see if there is any text to be sent. If there is, the MTTHANG subroutine is called to place the message on the receiving user's MCB queue or to send the message immediately to the operator (whose userid is OPERATOR). Return to the application is made by the RETN

macro.

At MTTMSG1, the parameters (from PARAMS) are picked-up and the userid is checked by a call to MTTGTUSR. The text length is also checked and if there is some text to be sent, it is attached to the appropriate user's MCB chain by a call to the MTTHANG subroutine. Return from this routine is via register 14.

30. MTTHANG subroutine

This is the subroutine which actually causes a message to get sent to either a user or the MT/T operator. In the case of a user, it obtains space for an MCB, puts the message into it and hangs the MCB to the end of the chain of MCBs pointed to by a word in the terminal table entry (TRQMSGPT) and turns on the flag in the terminal table entry indicating that there is at least one message in that queue.

In the case of a message intended for the MT/T operator (userid OPERATOR), the message text is moved to a (Monitor) message area (after a time-stamp field) and the whole thing is sent immediately to the operator via the MSG macro.

Return from either section of this subroutine is through register 14.

31. MTTHELP routine

MTTHELP is the entry called by the application to process a user "HELP" command. This command is a "MSG" with the receiving userid assumed to be the operator. After performing initialization with the ENTR macro, this routine locates the text of the message to be sent and sends it to the operator by calling the MTTHANG subroutine with a userid of OPERATOR. Return to the application is effected with the RETN macro.

32. MTTBCST routine

This is the routine which handles the

(operator) "BCST" command. It is used to send a message to all attached users. Upon entry and after the text of the message to be sent is verified, the terminal table is searched for active users. For each, the MTTHANG subroutine is called to hang an MCB containing the message text onto the MCB chain for the user. After they are all finished, return to the caller is made through (saved) register 14.

33. MTIUSERS routine

This is the routine which processes either the user or operator "USERS" command. Entry to MTTUSERS is for the application to call for the user command, the operator command causes entry MTTUSER1 to be called.

After MTTUSERS is called, normal initialization is accomplished by the ENTR macro. For users, a header message is composes stating the time and that following will be a list of NASISIDs. This is sent to the user by a call to MTTWRIT1. Then a column pointer and character counter are set-up and control transferrs to the common accumulating routine.

At entry to MTTUSER1, no header message is sent (it is assumed the operator knows what the line of output consists of). Only the column pointer and character pointer are initialized. They are different for the operator since his message has to start with the time-stamp field. Control falls through to the common accumulation code.

Now MTTUSERS adds a border ("***") to the left part of the line if a user message is being composed. It then searches the terminal table for active users, moving the NASISID of each one it finds into the print line with trailing blanks stripped off and separating each NASISID by a blank. Again, if a user message is being processed, the border is put on the right-hand end of the message. In the case of a user,

MTTWRIT1 is used to send the list; in the case of the operator, the MSG macro is used. Return is either through (saved) register 14 (for the operator) or via the RETN mechanism (for a user call).

34. MTTFORCE routine

This routine is the processor for the (operator) "FORCE" command. Upon entry, it merely looks for the NASISID which it was passed in the terminal table (by calling the MTTGTUSR subroutine) and turns on the force flag in the user's TCT (TCTFORC) if the NASISID can be located. (The user will actually be forced the next time the queue-scanner checks through the terminal table for users to be forced.) Return to the caller is through (saved) register 14.

35. MTTNUSER routine

MTTNUSER contains the two routines to handle the user and operator "NUSERS" commands. Entry to MTTNUSER is for the application to call when it finds the user command. Initialization is performed with the ENTR macro. Then the number of users is picked-up from MTTUSER#, made decimal and formatted into a message. The message is sent to the user via a call to MTTWRIT1. Return is made with the RETN macro.

MTTNUSR1 is the entry to the MTTNUSER routine for processing an operator "NUSERS" command. It takes the number of users from MTTUSER# and formats it into the appropriate operator message which is sent via the MSG macro. Return from this section is through (saved) register 14.

36. MTTSHUT routine

This routine processes the MT/T operator SHUTDOWN command. After it is called, it examines the parameter it was given. If this operand is defaulted, the value five is used as a default (number of minutes). After locating or defaulting

the operand, this routine converts the EBCDIC for it into binary and converts the number into milliseconds for use by the STIMER. A REAL timer (number six) is set for the resultant interval if the operand had a positive value. If the value of the operand was zero, the MTTEND1 routine is branched to directly to effect an immediate shutdown. After the timer for a non-zero operand has been successfully set, the caller is returned to through register 14.

37. MTTKILL routine

This routine is used to sloppily kill off an application user. If the user specified by the operand to the command can be located (MTTGTUSR) and if he is in control (execution), his resuming VPSW from the attention routine is overlaid to transfer control to memory location as soon as the attention routine transfers control back to the point of interruption. If the userid is not found or is not in control, an error message is sent to the operator with the MSG facility. Return from this routine is to the caller through register 14.

38. MTTLIMIT routine

MTTLIMIT is the processor for the operator LIMIT command which can be used to limit certain application facilities as well as various numbers of users allowed onto the system at one time. After it is called, this routine attempts to find out what manners of operands it was given. If the first of two operands is not two characters long, it is considered to be a keyword (for one of the facilities) and the table of keywords in the Monitor's CSECT is searched to find a match. (If none are found, an error message is sent to the operator.) If one is found, it is pointed to and control is passed to the number scanning code.

If the first of two parameters is two characters, it is a class name (where class is the first two characters of the

userid). The portion of the limit table beyond the header area is searched until the entered class name is located or the end is reached. If the end of the table is reached, the entered class name is added to the bottom and pointed to for the number scanner. If the entered class name is found, it is merely pointed to and control falls to the number scanning code.

After the appropriate class name is located, the second parameter is scanned if it is present. (If it is defaulted, the number 32767 (infinity, in this case) is used for the default.) The number is scanned left to right digit by digit. Invalid decimal digits cause an error message to be sent to the operator and the enterprise to be abandoned. After the number is scanned, it is posted in the slot in the header table (for keywords) or after the class name (for class names). Return from this routine is to the caller through register 14.

39. MTTDEBUG routine

This routine is the processor for the operator DEBUG command and consists of two instructions. The first one turns the debug switch (DEBUGSW) on and the second one returns to the caller through register 14.

40. MTTDATIM routine

MTTDATIM is the processor for the user DATETIME command. It performs its initialization and entry code with the ENTR macro, fills in the message to be sent to the user with the current date and time with the TIME macro and then calls MTTWRIT1 to send this message to the user. It then returns to its caller with the RETN macro.

41. MTTWAIT routine

After it is written sometime in the future, this routine will be the processor for the user WAIT command.

42. MTTPGMS routine

This routine processes the operator PGMSTOP command. This command alters the status of the program-interrupt-stopping switch. On entry, this routine turns off that switch and checks the operand to the command for a length of two (for "on"). If the length is not two, the routine returns to its caller through register 14. If it is two, the character string ON is checked for. If it is ON, the switch is turned on; otherwise the caller is merely returned to.

43. MTTNEWS routine

This routine processes the operator NEWS command. After it is called, it checks the entered operand for the character string OFF. If it finds it as the first three bytes of the operand, it causes the news buffer to be emptied by merely setting its length to zero. If the operand is real news text, it is added to the bottom of the news buffer if there is enough room in it. If there isn't, an error message is sent to the operator and the command is ignored. After the text is moved to the buffer, the length of that buffer is updated and the caller returned to through register 14.

44. MTTREC routine

This is the routine which allows the operator to manipulate the recording level by processing the RECORD command for him. (See the section on event recording for a description of what the recording level actually means.) The numeric operand to this command is processed much the same way that the number for the LIMIT command is processed except that an error message is also sent to the operator if the number is greater than 255. If the number is valid, it is posted as a one-byte value in the field MTTRECSW in the PSECT. Return to the caller is through register 14.

45. MTISTATS subroutine

This routine is called to process the operator STATS command. All it does is verify that the operand to the command is either the string "on" or "off" and set a flag in the PSECT. Return is to the caller through register 14.

46. MTTGTUSR subroutine

This is the subroutine which is called whenever somebody wants to see if a particular userid is currently connected to the application. After verifying the length of the operand (between 1 and 8), the operand is moved to a temporary and right-padded with blanks. Now the terminal table is searched top to bottom for the userid passed to MTTGTUSR (field TRQUSRID in the terminal table). After (if) the entry containing the userid is found, the logged-on field in the TCT for the terminal table entry is interrogated. If the user is not yet logged on, the userid is considered invalid. Valid userids are indicated by returning the terminal table entry pointer and a condition code of zero. Invalid userids are flagged by a condition code of two if the error return address (R6) is zero. (In both the preceeding cases, the caller is returned to through register 14.) If there is an error return address (R6 non-zero), the MTTBUSR routine is branched to. (It will make use of the error return address after sending the operator an error message.)

47. MTTBUSR subroutine

This subroutine merely sends the operator an error message about an invalid userid and then causes control to be transferred to an error routine by branching through register six.

48. MTTBMSG routine

This routine does the same thing as MTTBUSR except that the error message is for a missing message.

49. MTTERMPT routine

This routine is used to send a message to the operator. It is called by the MSG macro after that macro located the message pointer and length. This routine first puts a time-stamp field intc the message and causes it to be filled in by use of the TIME macro. It then posts the length of the message in core and uses GATWR or GTWAR to send it to the operator depending upon whether or not response fron the operator is expected. (If response was expected, the receiving field is blanked out before text is read.) Return from this routine is to the caller through register 14.

50. MTTTIME routine

This is the routine called by the TIME macro to cause an EBCDTIME to be performed on some text. This routine merely moves the descriptors of the field and does a register-form EBCDTIME on the field. Return is to the caller through register 14.

51. MTTMOVE subroutine

This subroutine merely moves text from here to there and is called by the MOVE macro. It is coded to be as quick as possible under the unaligned text case on a 360/67. It makes no special efforts to detect aligned fields so it can use register moves on them. After the text specified is moved, the caller is returned to through register 14.

52. MTTTRAN subroutine

This routine is called by the TRAN macro and does for translating text the same thing that MTTMOVE does for moving text. Return is again to the caller through register 14.

53. MTTRTRN routine

This routine is actually the processor specified for "SVC 63" instructions

which are issued to cause transfer of control from one push-down area to another. Since this routine is provided with a push-down area by the system, it can merely overlay it with the one to transfer control to. After doing this, it just returns through the linkage to the task monitor to cause the desired push-down area to receive control.

54. MTTRESET routine

This routine is called from the TSS command language by the operator when the attention processing routine is somehow disconnected (by TSS). After is has observed type I linkage conventions to get itself initialized, it merely re-specifies the attention routine in the same manner as the MTTSTART defined it to start with. If it is unable to do so, MSG is used to send indication to the operator. In either case, return is to the caller (TSS) through the exit linkage.

g. Recording Mechanism Description

The event recording mechanism is incorporated into the Monitor to allow events and some data to be recorded onto the (TSS) system SIPE (System Internal Performance Evaluation) tape. There are some fields which may be recorded in addition to the data area which may be specified. At present, the things which are recorded are input/output to user terminals, dispatches, time-slice ends, WAITs and returns from WAITs. Any event may be recorded in the Monitor by the insertion of a RECORD macro at the event to be recorded. (RECORD is described below.) Each event recorded must have a unique "key" so the events can be retrieved from the tape with identification. Furthermore, each recorded event (RECORD) must have a recording level attached to it. This level is used to determine whether or not the event is actually going to be recorded. The level corresponds to the numeric parameter to the RECORD operator command. If the event level at the event being recorded is higher than the level set at MTTRECSW by the RECORD command, the event is not recorded. Thus,

all the recording levels are higher than zero so that recording can be disabled by entering zero as the parameter to the RECORD command.

1. RECORD macro description

This macro is the only one which is used by record events. Its format is:

```
TSVAL=,NTSVAL=,CPUTIM=,REG=,
DATA=,DATALTH=
```

where KEY is the key number to be assigned to the event (they actually start at 4000 but for this macro we start at zero and it remembers what the base is) and LEVEL is the level to be used in determining whether or not the event is to be recorded as discussed above. TERM# is the relative terminal number (TCTTERM# or FNQPDV) for the user in question. #USERS is the number of users attached to the application (MTTUSER#). TSVAL is the user's time-slice value, usually from TCTTIMER, NTSVAL is the field containing the number of milliseconds he was dispatched with last time or is going to be dispatched with next time, taken from TCINTS and CPUTIM is the amount of CPU time the user has used so far, taken from TCTCPUTM. REG is a register which may be recorded if you wish; it is useful for dumping return codes and suchlike. DATA is the pointer to a data area which you need recorded (input/output text, for example) and DATALTH is the length of that field. KEY and LEVEL must always be specified and if DATA is specified, DATALTH must also be specified. Any of these field may be specified in register notation and any register may be used. Thus, if you have the elapsed CPU time sitting in a register, you could code CPUTIM=(R7) instead of CPUTIM=TCTCPUTM. In addition, the DATALTH field may be either an address of a length field, a register containing the length or an absolute (or undefined) expression containing the length.

2. Expanded code

The RECORD macro expands code to collect the fields specified into either registers (USERID, TERM#, #USERS, TSVAL, NTSVAL and CPUTIM) or into an area in the recording area itself (DATA, DATALTH, KEY). Since SIPE records the registers as well as the recording area, they are used to hold as much of the recorded data as possible. Future fields will have to be put into the recording area, however, as there are no more registers available for data-holding. Code is also expanded to test the level specified against NTTRECSW and to check the return code from SIPE itself to make sure it intercepts the SVC issued to activate it. The SVC itself is in the beginning of the recording area (which is always pointed to by register 1 during RECORDs) and is the target of an EXECUTE instruction (so that SIPE is insured that the area is in physical core).

F. CODING SPECIFICATIONS

1. Source Language

TSS/360 Assembler Language

2. Suggestions and Techniques

Not Applicable

TOPIC A.2 - INITIAL ENTRY ROUTINE

A. MODULE NAME

Initial Entry Routine
Program-ID - RDBMTT
Module-ID - DBMTT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The function of this module is to perform allocations of the external data items used by the system. It also issues the initial prompt, which is used to determine which NASIS sub-system the user wishes to invoke, and then calls the proper module for that sub-system.

D. DATA REQUIREMENTS

1. I/O Block Diagrams

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

The program makes use of the following tables:

a. USERTAB

b. VERBTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Initialize

This routine initializes the interrupt (ATTN and END) processing routines and the PL/I error handler. It allocates and initializes the user data table. The program also allocates and initializes the verb table (including user specified commands) which it uses in the prompt routine.

b. Define

This routine performs all of the file control block allocations and initializations for the proper operation of the rest of the NASIS system.

c. Prompt

This routine sets a temporary END condition handler which results in a new prompt on an END condition. It prompts the user for a command and searches the verb table for a matching entry. If no match is found a diagnostic message is written to the user and the prompt is re-issued.

The verb table entry is analyzed and if an immediate command has been entered, the program branches to the routine which processes that command. Otherwise, the program optionally establishes a new strategy and then calls the entry point of the processer for the command entered. When control is returned to DBMTT, the user is

prompted for the disposition of the current strategy and it is either renewed or erased.

When the command entered has been completely processed, control is passed back to the prompting routine. The entry of an END command causes the program to be terminated.

F. CODING SPECIFICATIONS

1. Source Language

TSS/360 PL/I

2. Suggestions and Techniques

Not Applicable

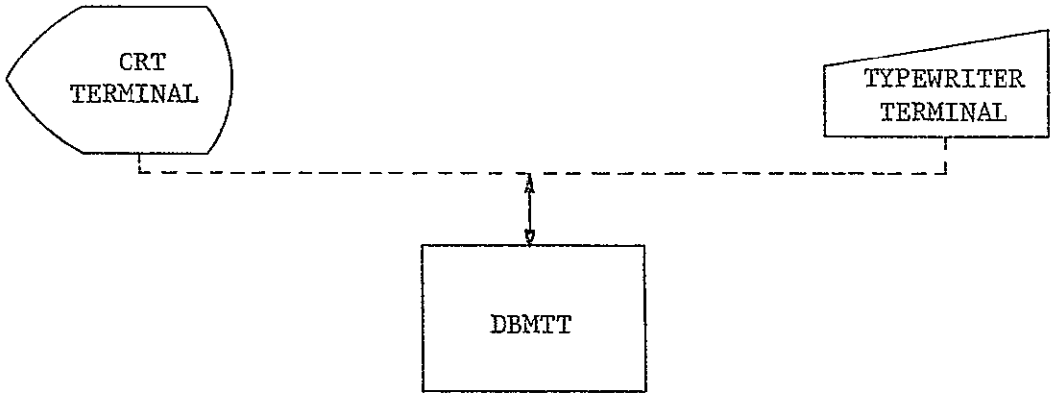


Figure 1. I/O Block diagram

IV A, 2a 1

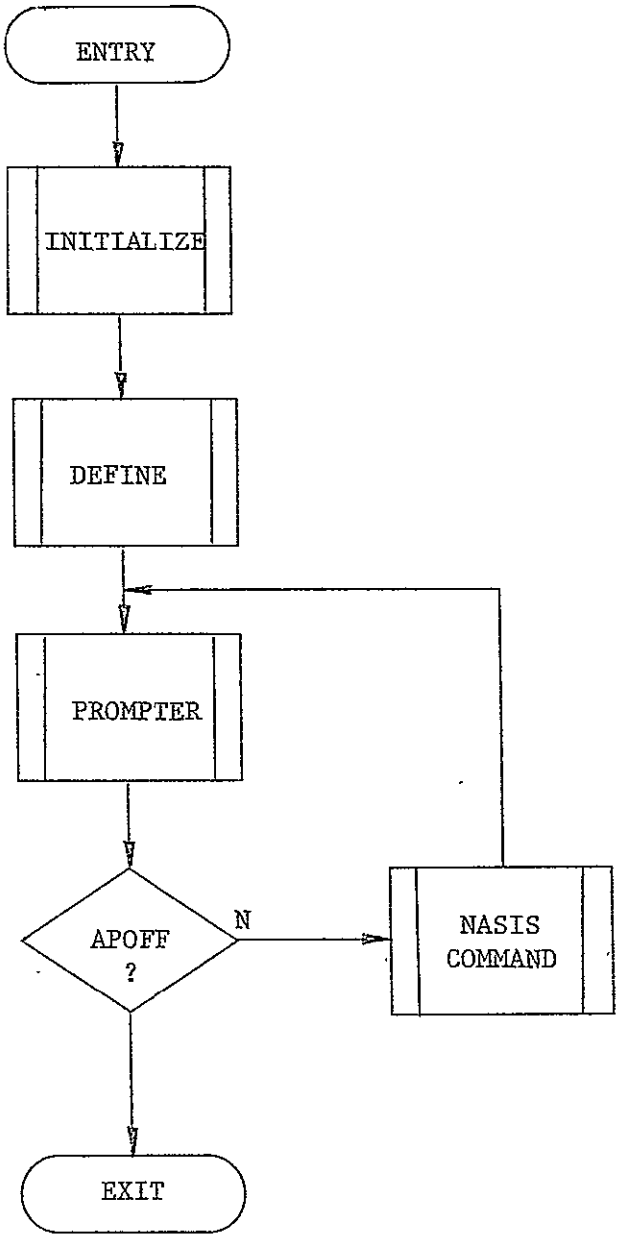


Figure 2. Top level flowchart

IV A2 2

TOPIC A.3 - SINGLE TERMINAL TASK MONITOR

A. MODULE NAME

Terminal Support - Single Terminal Task Monitor
Program-ID - NASISX
Module-ID - MTTSUPX
Entry Points - NASIS1, MTTWRITE, MTTREAD, MTTWREAD, MTTGETIM, MTTXTR, MTTMUST, MTTPASS, MTTATTN, MTTPGMIN, MTTSVC1, MTTSVC2, FIXATTN, MTTKA, MTTKB

B. ANALYST

Frank Reed
Robert L. Rutledge
Neoterics, Inc.

C. MODULE FUNCTIONS

1. Organization Chart

See Figure 1

2. Overview

The function of this module is to provide the MTT monitor services to the non-MTT user of the NASIS system, allowing NASIS to be invoked in ''standalone'' mode.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 2

2. Input Data Sets

a. Parameter Cards

Not applicable

b. Punched Card Input Files

Not applicable

c. Input Files

LISRIDS

d. On-line Terminal Entries

The user enters his NASIS-ID and his security-code when prompted.

3. Output Data Sets

Not applicable

4. Reference Tables

a. External Tables

1. MTTUTAB
2. MTTLOAD

b. Internal Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

a. Entry Points

1. NASIS1 - See Figure 3
2. MTTWRITE - See Figure 4
3. MTTREAD - See Figure 4
4. MTTWREAD - See Figure 4
5. MTTGETIM - See Figure 5
6. MTTXTR - See Figure 6
7. MTTMUST - See Figure 7
8. MTTPASS - See Figure 8
9. MTTATTN - See Figure 9
10. MTTSVC1 - See Figure 10
11. MTTPGMIN - See Figure 11
12. MTTSVC2 - See Figure 12
13. FIXATTN - See Figure 13
14. MTTKA - See Figure 14
15. MTTKB - See Figure 14

2. Narrative

a. Entry Point NASIS1

This section of code begins a NASIS application in standalone mode. The preliminary functions it performs include specifying interrupt routines, initializing timers and checking the user's userid and security code. After these have been successfully completed, the location of RDBMTT is retrieved from the table MTTLOAD and control is passed to it. On return, the interrupt routines are deleted and control is returned to the caller (terminal user).

b. Entry Points MTTREAD, MTTWRITE and MTTWREAD

These routines are the ones which do actual I/O to the user terminal. This is accomplished by an ENTER to the GATE junction using a control block initialized by the calling program.

A TWAIT is executed for write-only outputs to insure a coherent interchange should the user press the attention key before the output is complete. On return from all I/O functions, the GATE return code is placed in register 15 and control is passed to the caller.

c. Entry Point MTTGETIM

MTTGETIM returns elapsed connect and CPU time to its caller. All times are kept via the REDTIM AND XTRTM macros.

d. Entry Point MTTXTR

This routine returns information about the terminal user to its caller. Specifically, the userid, security code, taskid, MT/T and conversational mode flags are returned.

e. Entry Point MTTMUST

This entry point is not required in standalone mode, hence, it is a no-op.

f. Entry Point MTTPASS

MTTPASS prompts the terminal user for his password (security code) and returns the

response to its caller.

g. Entry Points MTTKA and MTTKB.

These routines are used to get the user into either KA or KB mode. The OBEY macro is utilized to accomplish this feat.

h. Entry Point FIXATTN

This entry point is called by a terminal user to re-activate the attention routine whenever it goes away (which occurs frequently during debugging). The attention routines are deleted and re-specified to TSS and control is returned to the user.

i. Entry Point MTTATTN

This routine gains control of attention interrupts from the system and passes then along to the attention routine specified in the MTTLOAD table. Two savereas are reserved in the NASISX PSSCT to hold the interrupted registers and VPSW.

On entry the interrupted registers and VPSW are moved to one of the saveareas and the interrupt is dequeued. Then, the processor is called with register 1 pointing to the savearea. On return from the processor, MTTSVC1 is invoked to return control to the point of interrupt.

j. MTTSVC1

This routine moves the registers from a NASISX savearea to the system savearea pointed to by register 0 (zero). Control is returned to the system, which loads the moved registers and VPSW and passes control to the point where execution was interrupted by the user.

h. MTTPGMIN

This routine is called by the system whenever a program interrupt occurs within the application. The user is notified and the interrupted registers are displayed at the terminal. Next, the user is prompted via the CLIC macro. If his response is ''GO'', control is returned to the point of interrupt

by invoking entry point MTTSVC

i. MTTSVC2

This routine moves the registers active at the time of a program interrupt from a NASISX savearea to the system savearea pointed to by register 0 (zero). Control is returned to the system, which in turn returns control to the point of interrupt.

F. CODING SPECIFICATIONS

1. Source Language

TSS/360 Assembler Language.

2. Suggestions and Techniques

Not Applicable

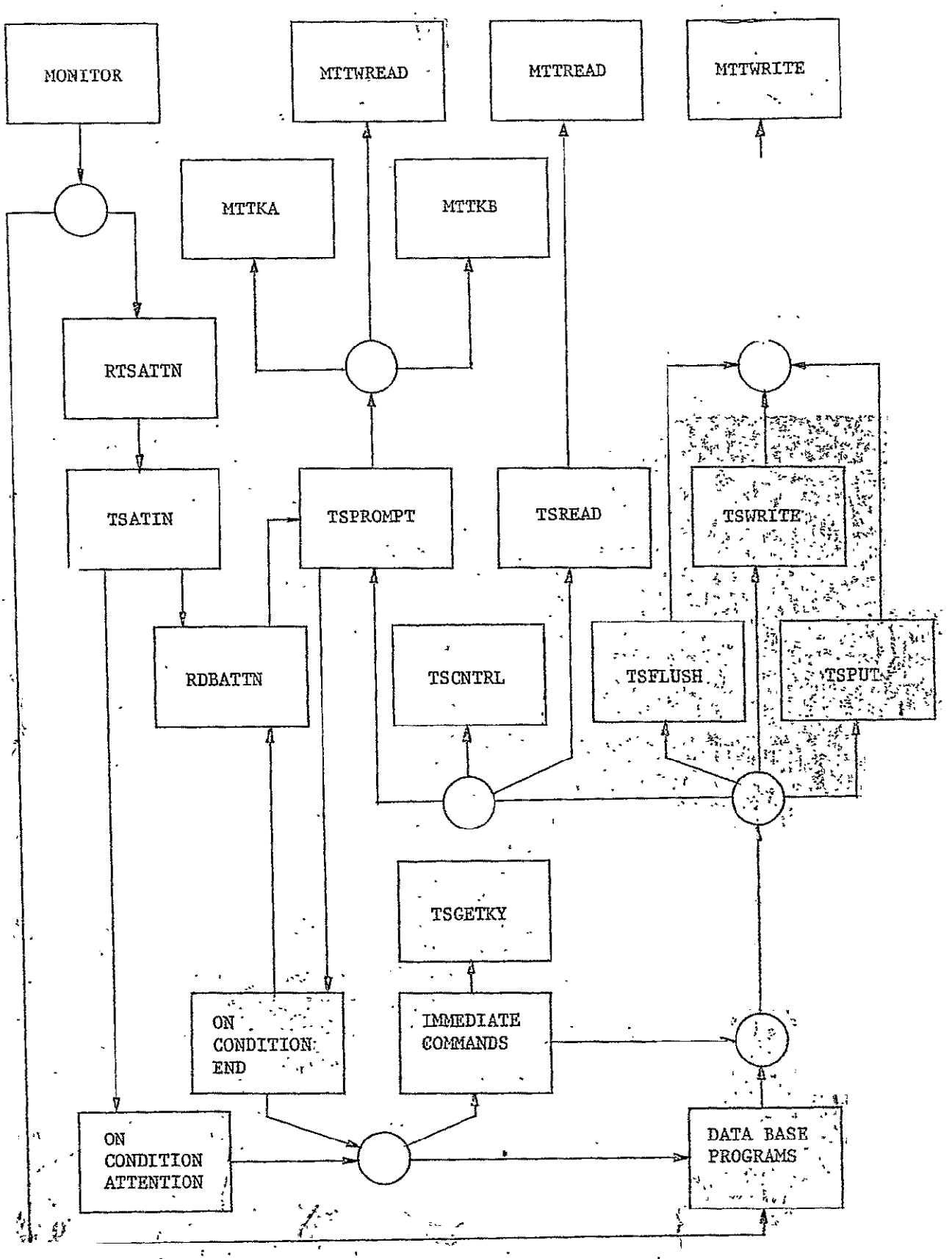


Figure 1. Terminal Support Organization Chart

IV A.3.1

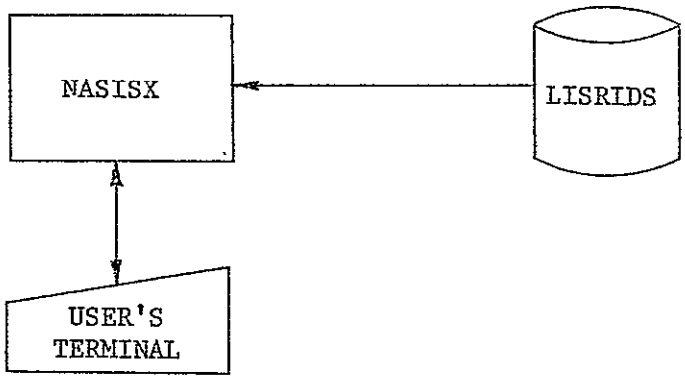


Figure 2. I/O Block Diagram

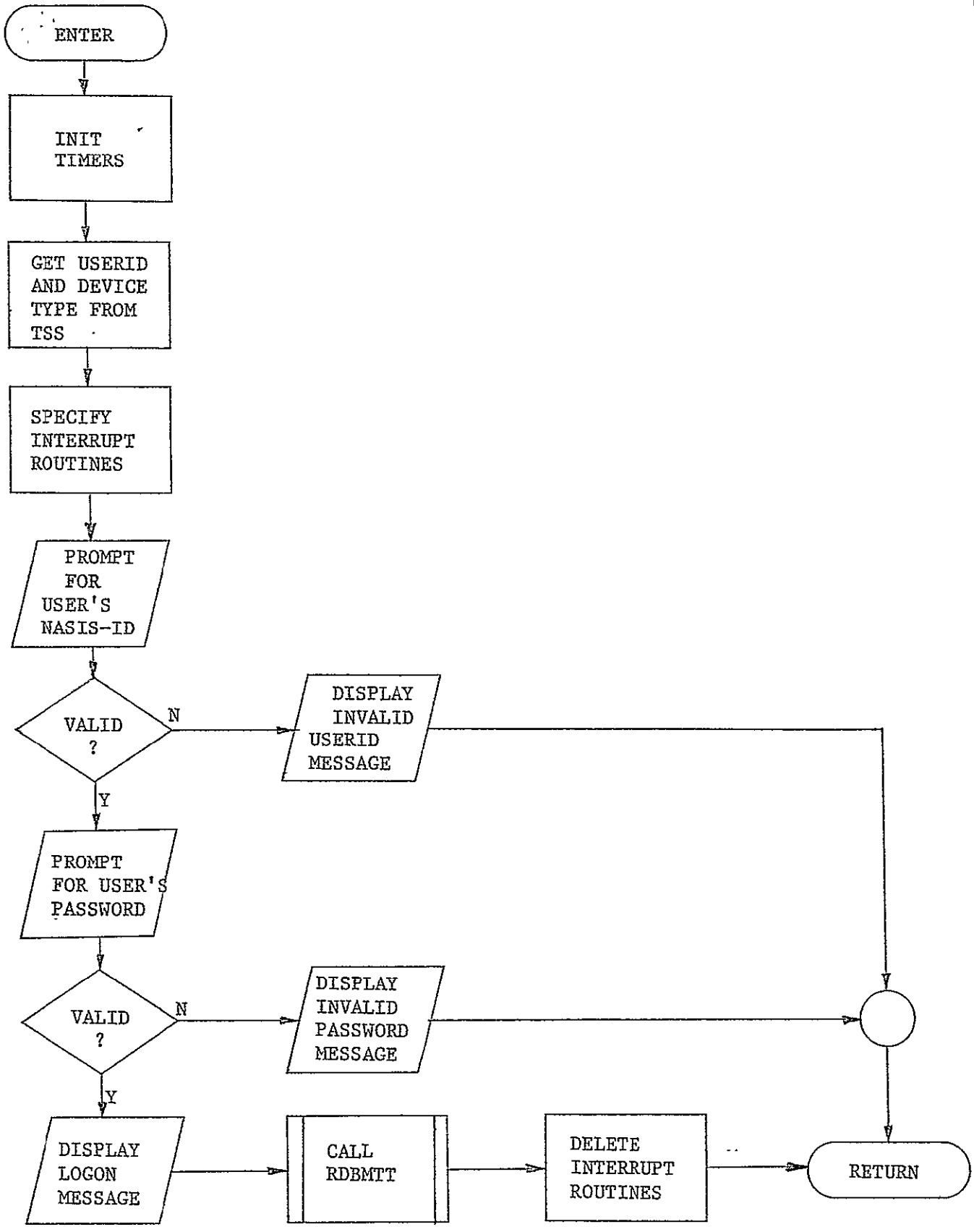


Figure 3. Entry Point NASIS1

IV A3-3

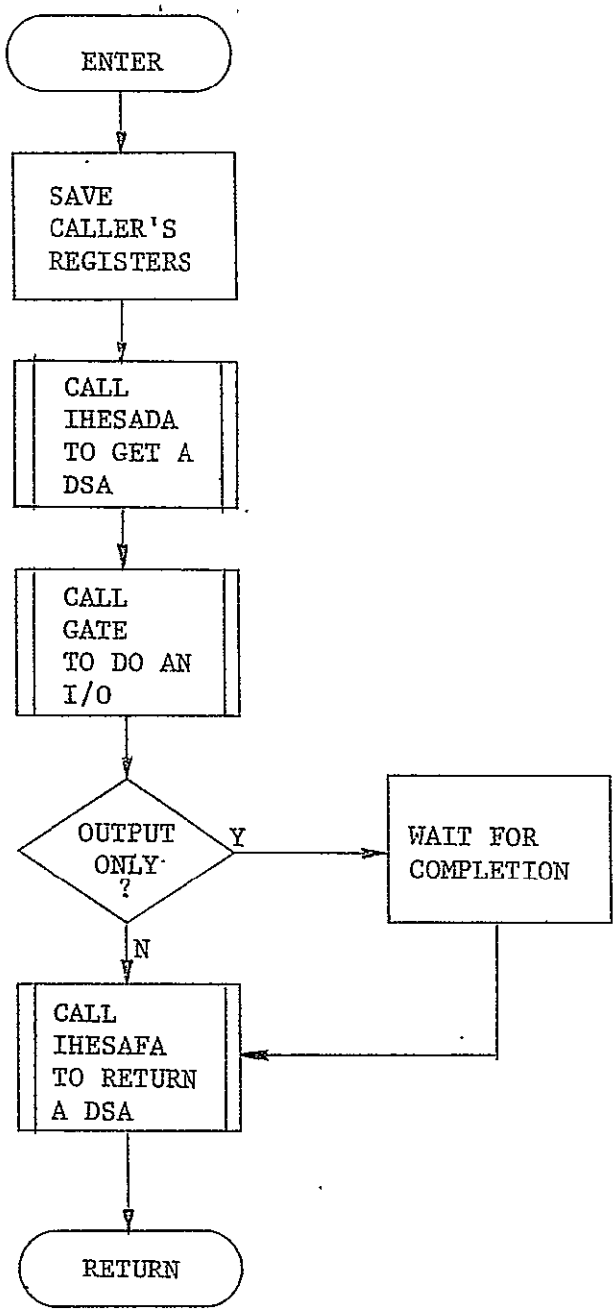


Figure 4. Entry Points MTTREAD, MTTWRITE, MTTWREAD

IV A3-4

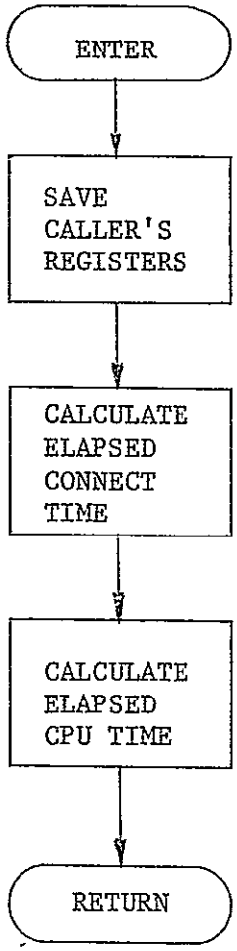


Figure 5. Entry Point MTTGETIM

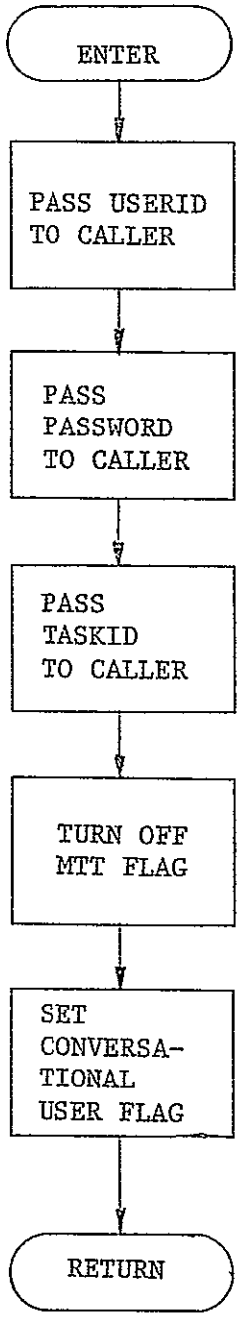


Figure 6. Entry Point MTTXTR

IV. 13-6

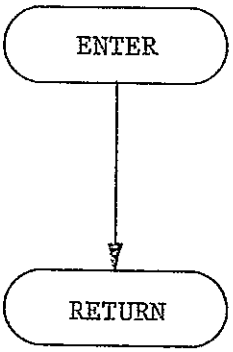


Figure 7. Entry Point MTTMUST

IV A3-1

Figure 8. Entry Point MTTPASS

IV A章-8

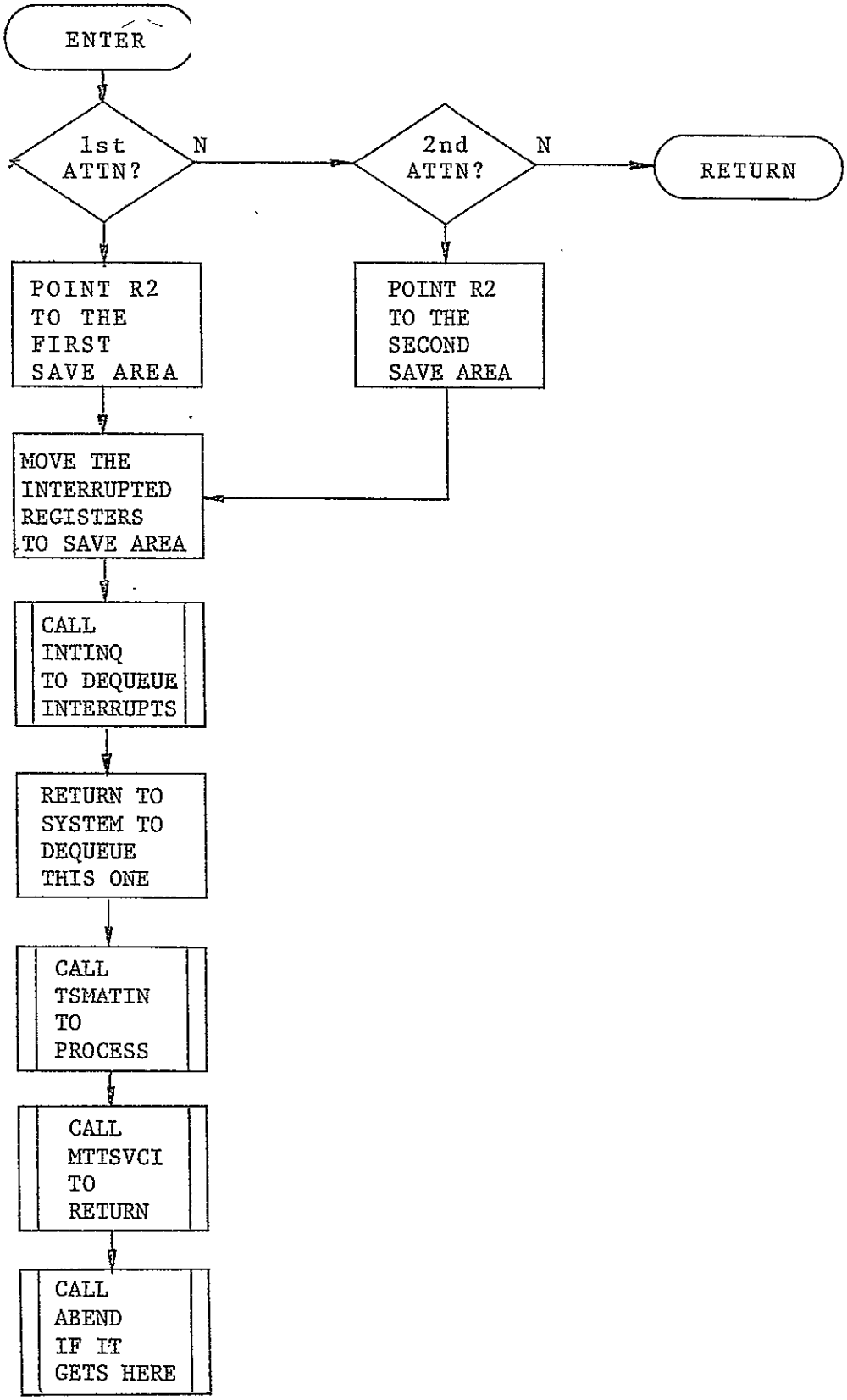


Figure 9. Entry Point MTTATTN

IV 453-9

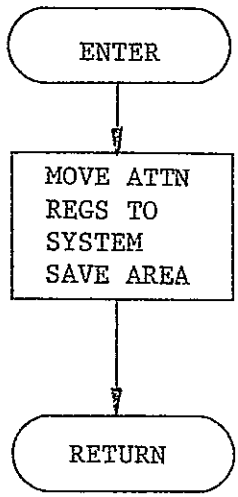


Figure 10. Entry Point MTTSVCI

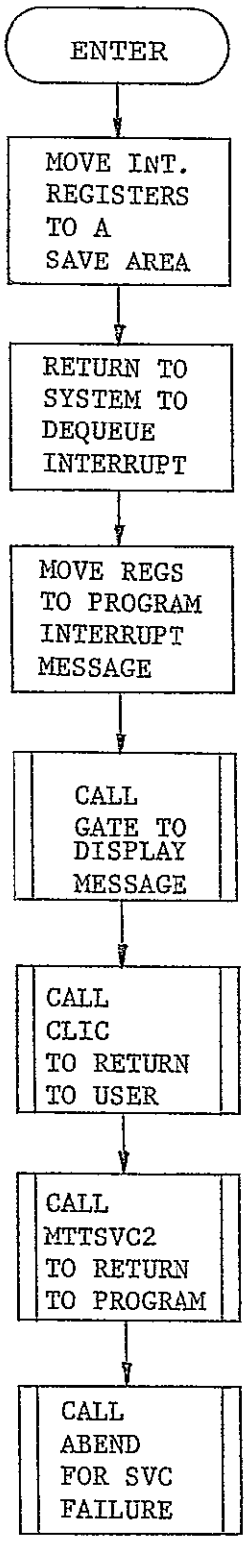


Figure 11. / MTTBGMIN

IV A3-11

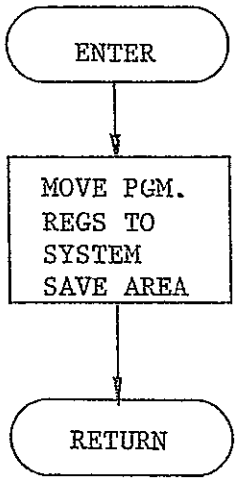


Figure 12. Entry Point MTTSVCZ

IV A3-12.

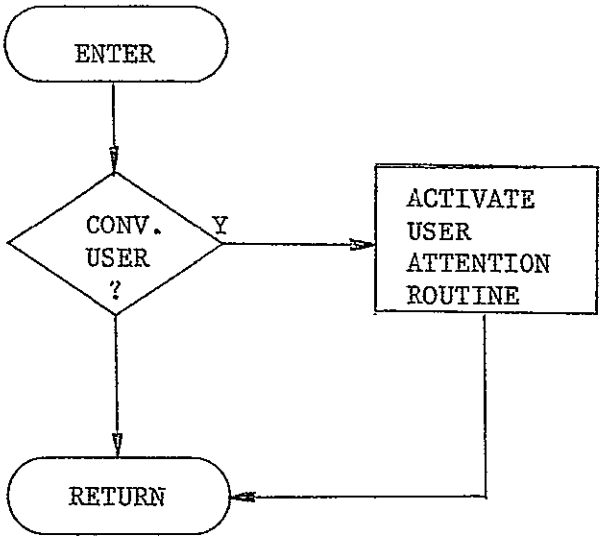


Figure 13. Entry Point FIXATTN

IV A3-13

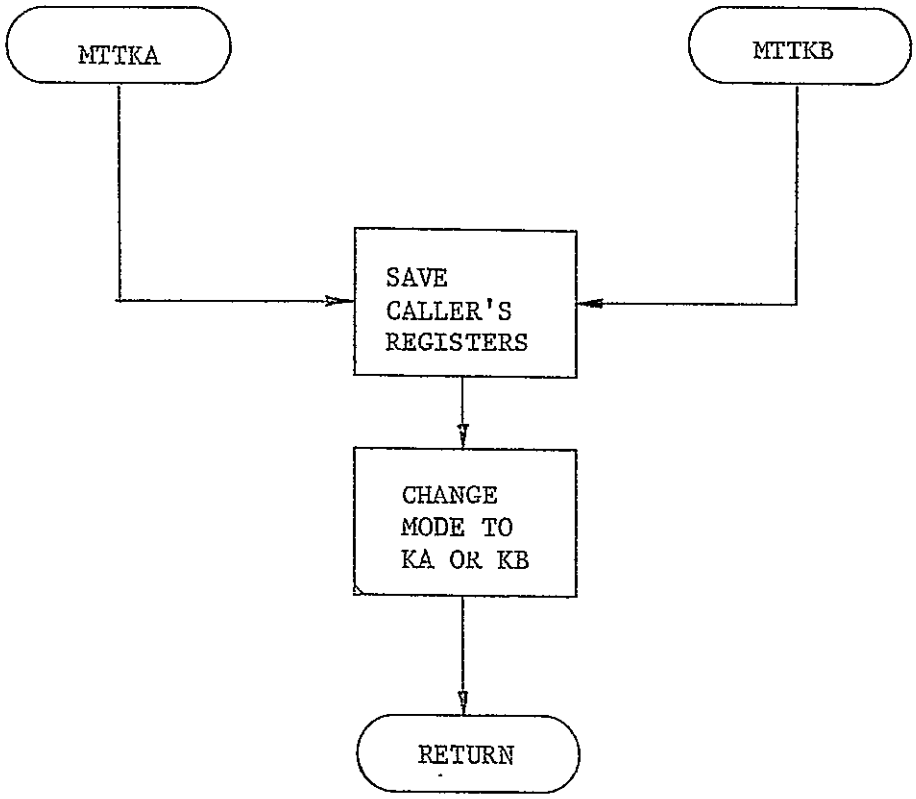


Figure 14. Entry Point MTTKA and MTTKB

IV A3-14

TOPIC B.1 - EXECUTIVE PRE-PROCESSOR

A. MODULE NAME

Data Base Executive - Preprocessor
Program-ID - DB
Module-ID - DB

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

DB analyzes Data Base PL/I language extension (DBPL/I) statements and generates, in their place, in a source program, PL/I statements for communication with the Data Base Executive (RDBPAC OR RDBLIST). Diagnostic comments are generated for errors that can be detected by DB during preprocessing.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Job control parameters for operation under TSS are those required for PL/I precompilation and immediate compilation. Refer to the appropriate IBM PL/I Programmer's Guide (Form C28-2049 for TSS). The PL/I compiler parameters-MACRO, SOURCE2, and COMP (among others) are specified to indicate that precompiling, precompiler input listing and compiling are desired.

b. Punched Card Input Files

1. DB Text

The DB Text deck is text for insertion into the source program as a result of a % INCLUDE DB; statement in the source program. This text is composed of the source statements of the DB preprocessor function procedure, itself, and any PL/I

statements to be unconditionally inserted at the % INCLUDE DB; point in the source program. DB Text is coded as specified in this report, formatted according to PL/I source language standards and catalogued once in a data set for compile-time use by all programs using DB.

2. Source Deck

The SOURCE Deck is any PL/I source program using DB for its DBPL/I statements. It is prepared according to the DBPL/I User's Manual (DWB Section V, Topic B.2) to access a self-describing data base and formatted according to PL/I source language standards.

c. Input Files

DB Text is catalogued as a member, named DB, of a partitioned direct access data set for retrieval by the IBM PL/I precompiler. The data set is accessed via ddname LISRMAC.

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The object module consists of the relocatable machine instructions and constants generated by the PL/I compiler for the source program. It is stored as a member of a program library (Partitioned data set) for subsequent loading by the TSS system loader.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-outs

1. Precompiler Listings

Two precompiler listings are produced: a source listing before precompilation, and any precompiler diagnostics (these diagnostics are any errors in the use of

preprocessor PL/I, not DBPL/I). The appropriate IBM PL/I Programmer's Guide explains the listing formats.

2. Compiler Listings

The compiler listings include an intermediate source listing (between precompiling and compiling) and any compiler diagnostics. Any errors in the use of DBPL/I generate diagnostic PL/I comments in the intermediate source listing. Serious DBPL/I errors may result in compiler diagnostics, particularly for undeclared qualified names when DB has suppressed automatic generation of a declare statement. The appropriate IBM/I Programmer's Guide explains the listing formats.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

MFCB - Mainline file control block.
See Section III, Topic B.4, of the DWB.
DBPL/I - Diagnostic comments.
See Section III, Topic B.1, of the DWB.
DBPL/I - DBPAC Interface.
See Section III, Topic B.2, of the DWB.
DBPL/I - DBLIST Interface.
See Section III, Topic B.10.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Top Level

The mainline PL/I source program is required, according to the DBPL/I User's Manual (DWB Section V, Topic B.2), to have a% INCLUDE DB1 statement once in the program before all DB preprocessor function references. This statement directs the PL/I precompiler to take text from member DB of the partitioned data set accessed via ddname LISRMAC and

incorporate it into the source program. (See the I/O Block Diagram in Figure 1.).

The DB text includes the following statements:

```
ON FINISH GO TO FINISH;
```

for "automatic" data base file closing. DBPL/I requires that the PL/I FINISH ON-condition be reserved for this purpose.

The DB text declares and activates the DB preprocessor name by:

```
%   DECLARE   DB   ENTRY(CHARACTER)
RETURNS(CHARACTER);
```

The DB text following the end of the DB preprocessor function procedure invokes DB once as follows:

```
DB(INITIALIZE)
```

This statement is a special function reference to be recognized by DB as the first reference (directing DB to initialize itself).

The remainder of this narrative specifies the DB preprocessor function procedure which is depicted in the Top Level Flowchart in Figure 2.

DB receives one argument from a preprocessor function reference: a varying length character string consisting, in general, of labels, comments, valid DBPL/I statements and, possibly, invalid text. DB's objective is to analyze the argument and generate a varying length character string, called the "generated text", consisting of valid PL/I labels, comments and PL/I statements for communication with the Data Base Executive.

If the special argument, 'INITIALIZE', is received, (i.e., the first reference to DB), the Initialize DB routine is performed and a comment, such as:

```
/*  DB001  INITIALIZATION  COMPLETE. */
```

is returned for insertion into the source

program and DB is terminated. Otherwise, the Argument Initialization routine is performed.

Following the Argument Initialization routine DB is logically between DBPL/I statements in its processing of the argument. The Find Subargument routine is performed there. If it finds the right parenthesis at the end of the argument, the generated text is returned for insertion into the source program, and DB is terminated. If Find Subargument finds an inter-statement comment, a statement label, or a null statement (simply a semicolon), then the subargument is concatenated to the right end of the generated text (i.e., "passed through" to the intermediate source text), and preprocessor control is transferred back to the inter-statement point. Otherwise, the Process Statement routine is performed, and preprocessor control is transferred back to the inter-statement point.

b. Diagnostic Comment Generation

Wherever this narrative specifies the generation of a diagnostic comment, the following specifications apply. A diagnostic comment is concatenated to the right end of the generated text for insertion into the intermediate source program. If the diagnostic is for an error, the precompiler count of diagnostics is incremented. If more than four errors are detected in one DB reference further processing of that reference is stopped to prevent the possibility of unpaired quotes, parentheses or comment delimiters looping the preprocessor. A diagnostic has the following general format:

```
/*  DBnnn diagnostic-message. */
```

The "DB" preceding the message number indicates that the comment was generated by the DB preprocessor. The three-digit message number guides the user to a more detailed explanation of the message which is documented in the DWB Section III, Topic B.1.

c. Initialize DB

Precompiler variables for file attributes, file usages and diagnostic counts are appropriately initialized. These variables are subsequently set or incremented as DBPL/I statements are processed and are examined when the finish statement is processed. A precompiler indicator is set to indicate that the FINISH statement has not yet been processed.

d. Argument Initialization

The argument is examined to find the left parenthesis at its beginning. If any other non-blank character is found, a diagnostic comment is generated and DB is terminated. A precompiler variable pointing to the "current argument character" is initialized to point to the character following the beginning left parenthesis. The generated text is initialized as one blank character.

e. Find Subargument

A subargument, as used in this specification, is a substring of the argument that is one of the following classes of syntactic units:

1. The right parenthesis at the end of the argument.

2. A label, including its colon.

3. An inter-statement PL/I comment.

4. A Null statement consisting only of a semicolon.

5. A DBPL/I statement terminated by a semicolon.

6. A syntax error; i.e., none of the above.

A class (5) subargument may contain paired parenthesis (possibly nested) or string constants enclosed in string quotes. A class (6) subargument will be terminated by a semicolon if one is found but will never include the right parenthesis at the end of the argument.

The Find Subargument routine is used at the

inter-statement point in the Top Level Flowchart. The argument is examined beginning at the current argument character and ignoring leading blanks to find the next subargument. A precompiler variable pointing to the beginning character of the subargument, and another indicating its length in characters, is set. The current argument pointer is advanced to point to the character following the subargument.

f. Process Statement

This routine analyzes a single DBPL/I statement body (i.e., apart from any statement labels), generates suitable PL/I statements for communication with the data base executive and returns preprocessor control to the inter-statement point. The PL/I statements and comments that are generated are concatenated to the right end of the generated text string for subsequent insertion into the intermediate source program.

A diagnostic comment containing the subargument and any intra-statement comments is generated for documentation and for reference in case of other diagnostics. If the FINISH statement has already been processed or if the subargument has a syntax error, an appropriate diagnostic comment is generated, and control is returned to the inter-statement point.

If the Find Keyword routine does not find a keyword that identifies a DBPL/I statement, then a diagnostic comment is generated and control is returned to the inter-statement point. If the keyword identifies a SET, FINISH, FREE or ON statement, control is transferred to the relevant specific statement routine. The Find File clause routine is performed if the second clause is not a FILE clause then a diagnostic comment is generated, and control is returned to the inter-statement point. The Find File routine is performed, and control is transferred to the relevant specific statement routine.

1. Find Keyword Routine

A clause, as used in this specification,

is a substring of the subargument that is one of the following classes of syntactic units:

-the semicolon at the end of the subargument,

-a comma separating DBPL/I substatements;

e.g., in a multiple OPEN,

-a keyword with an associated parenthesized argument,

-a keyword without a parenthesized argument.

A keyword-with-argument clause may contain paired parenthesis (possibly nested), or string constants enclosed in string quotes.

The Find Keyword routine is used to find the keyword that will identify a statement to branch to the specific statement routines.

2. Find File Routine

The Find File subroutine extracts the file name from a given FILE clause. If the file-name is not a valid PL/I external name, a diagnostic message is generated, and the statement abandoned by control being transferred to the inter-statement point. Otherwise, the precompiler's file table is searched to determine if the file-name has been used previously in the program. If it has not, a new entry is appended to the file table. In either case, a precompiler variable is set to indicate the current file, and control is returned to the point from which Find File was invcked.

3. Specific Statement Routines

Each specific statement routine examines the statement from left to right until the semicolon is found. (The CLOSE and OPEN statement routines recognize a

comma as separating substatements and loop accordingly). The keywords are verified for correct spelling and order. The FREE LIST routine for specific lists recognizes a comma separating list-pointers and loops accordingly. Routines that process a statement having a FIELD clause recognize a comma separating field-name expressions, find the corresponding element in the FROM or INTO clause and loop accordingly. If any error is detected, a diagnostic comment is generated, and the statement abandoned by control being transferred to the inter-statement point.

For those statements having a FILE clause, the precompiler's file table is posted to record the file usage (for analysis in the FINISH routine).

Following successful analysis, each specific statement routine generates PL/I statements for communication with the DBPAC or DBLIST executive and then loops back either to process another FIELD or FREE LIST element, to process another OPEN or CLOSE substatement, or to the inter-statement point. Special processing for the ON-- and FINISH statements is specified after the general specifications for all other specific statement routines.

For those statements having an entry in the DBPL/I - DBPAC Interface table (Section III, Topic B.2, of the DWB), an assignment statement is generated in the following format:

```
filename.OPERATION = 'operation'B;
```

For example, when processing the following argument:

```
LOCATE  FILE(SAMPF)  KEYFROM(REC#) ;
```

The following assignment is generated:

```
SAMPF.OPERATION = '11010000'B;
```

For statements having a FIELD clause, the operation assignment need only be

generated once for the statement, even if it contains multiple field names.

For an OPEN statement having a TITLE clause the following assignment is generated:

```
filename.ONFILE = title-expression;
```

If it has no TITLE clause the following is generated:

```
filename.ONFILE = 'filename';
```

For an OPEN statement having an "access" option and/or a "function" option, a bit-string value is assigned to filename.ATTRIBUTES according to the definition of a Mainline File Control Block (described in Section III, Topic B.4 of the DWB); otherwise, the following assignment is generated for an OPEN:

```
filename.FUNCTION = '10'B;
```

For each field-name in a FIELD clause, an assignment statement is generated as follows:

```
filename.ONFIELD = fieldname;
```

Where the field-name may be an expression, for example, when processing the following argument:

```
GET    FILE(EXAMP)      FIELD('DATEPUB')
INTO(DP) ;
```

The following assignment is generated:

```
EXAMP.ONFIELD = 'DATEPUB';
```

For those statements having an entry in the DBPL/I - DBPAC Interface table, a CALL statement is generated in one of the following formats, depending on whether the "Arg1" and "Arg2" columns of the table have entries:

```
CALL entrypoint  (arg1);

CALL entrypoint  (arg1, arg2);
```

```
CALL entrypoint  (arg1, arg2, arg3);
```

For example, when processing this statement:

```
LOCATE FILE(DAMPF)  KEYFROM(REC#);
```

This CALL is generated:

```
CALL DBPACFV  (SAMPF, REC#);
```

For those statements having an entry in the DBPL/I-DBLIST Interface Table (Section III, Topic B.10), a CALL statement is generated according to the table.

The ON statement routine examines the second clause. If an ERRORFILE clause is found, the Find File subroutine is performed. The statements shown below at the right are generated for the ON statement shown at the left.

```
ON ERRORFILE(f)  GO TO label;

     f.ERROR.ROUTINE = label;

     f.SYSTEM = '0'B;

ON ERRORFILE(f) SYSTEM;

     f.SYSTEM = '1'B;

ON LISTERROR GO TO label;

     LISTERR.ERROR.ROUTINE = label;

     LISTERR.SYSTEM = '0'B;

ON LISTERROR SYSTEM;

     LISTERR.SYSTEM  = '1'B;
```

The FINISH statement routine sets a precompiler indicator to indicate that a FINISH statement has been processed. Also, the following statement is generated:

```
FINISH:  ON FINISH SYSTEM;
```

Then each entry in the precompiler's

file table is analyzed. If the file was used inconsistently in the program, a diagnostic comment is generated and the next file analyzed. Otherwise, a Mainline File Control Block (MFCB) declaration is generated, using the file-name as the major structure name and as the initial value of the title. Any file attributes implied by the usage of the file are generated into the initial value of the filename.ACCESS and filename.FUNCTION fields. Statements are generated to "automatically" CLOSE the file, just the same as for a CLOSE statement.

After all files have been analyzed, the following statement is generated:

```
RETURN;
```

In all programs, a declaration of the entry points to the Data Base Executive (DBFAC) is generated.

In all cases, a summary diagnostic comment is generated giving the number of DB diagnostic error comments in the program.

F. CODING SPECIFICATIONS

1. Source Language

The DB preprocessor function procedure is coded using the preprocessor PL/I statements permitted in preprocessor PL/I procedures.

Statements to be INCLUDEd or generated into the intermediate source program are coded using PL/I.

2. Suggestions and Techniques

The DB preprocessor function procedure is coded in a modular manner so that the syntax analysis of the argument is separate from the generation of statements. This modularity will allow much of the DB coding to be usable for any other extensions to PL/I that may be designed, such as a Terminal Support PL/I language extension.

The coding of the specific statement routines are

"table-driven" where possible to facilitate any future changes in the generated text for a particular statement.

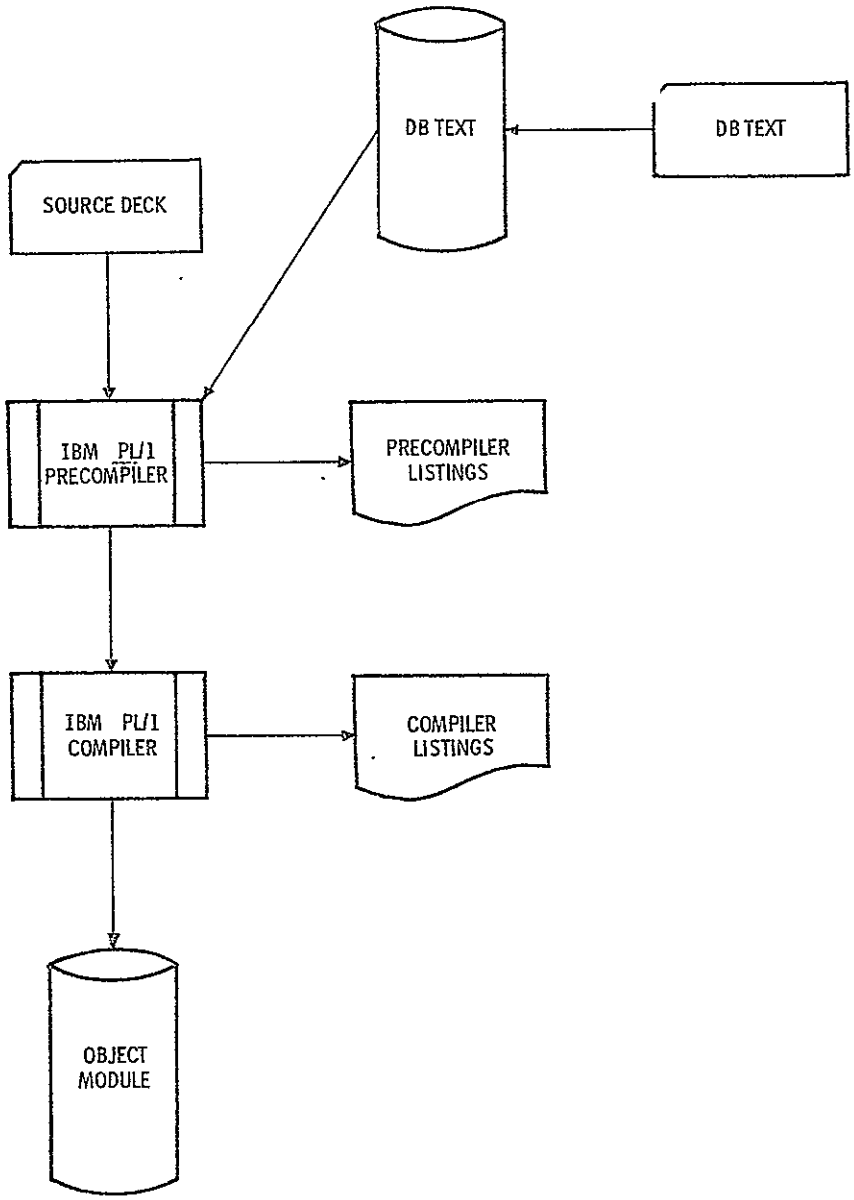


Figure 1. - I/O Block diagram.

IV B.1.)

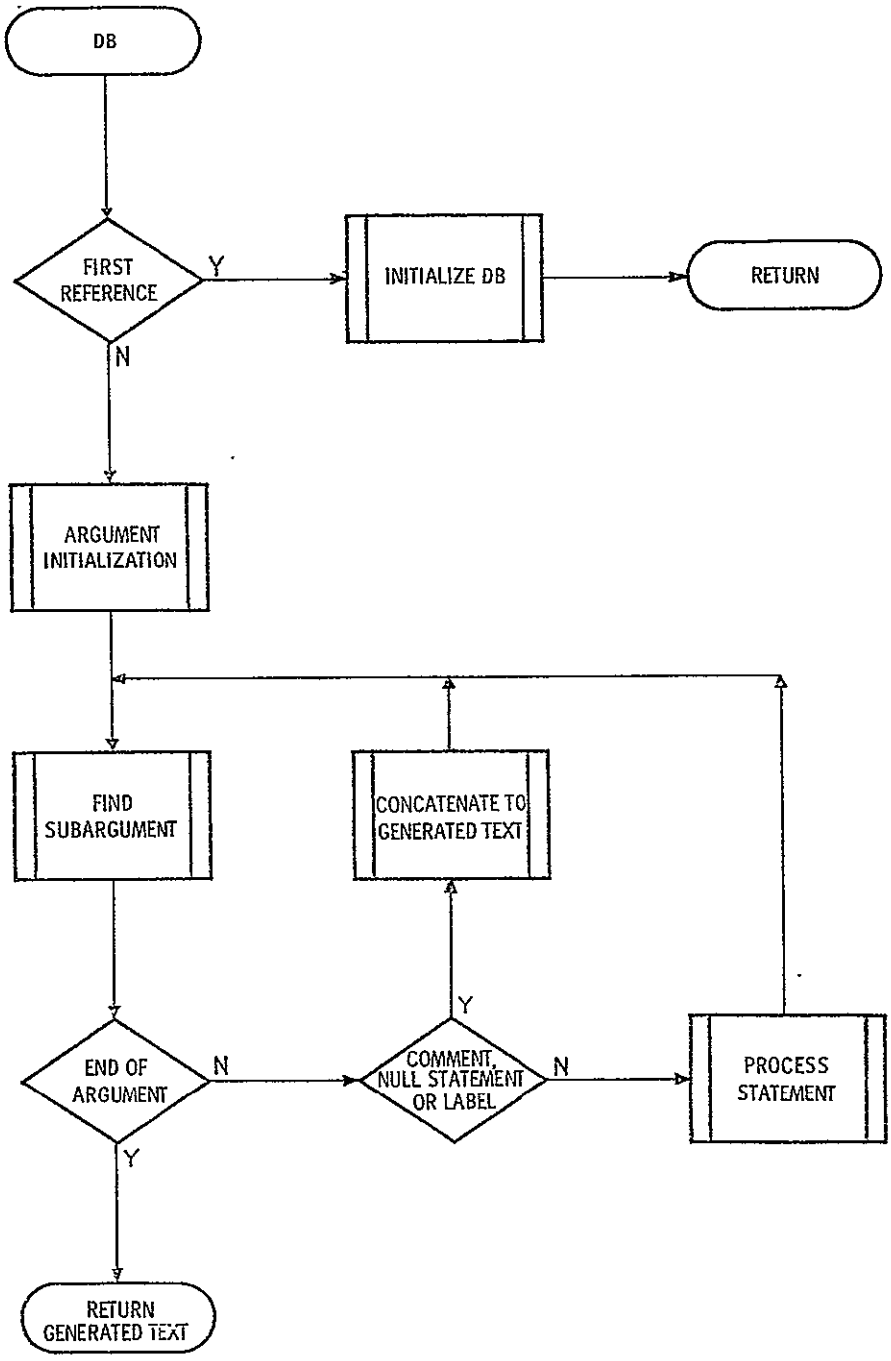


Figure 2. - Top level flowchart.

IV. B3.1.2

TOPIC B.2 - DATA BASE EXECUTIVE EXECUTION PROCESSOR

A. MODULE NAME

Data Base Executive Execution Processor
Program-ID - RDBPAC
Module-ID - DBPAC
Procedure Entry Point (control section name): #FIELD
Other Entry Points - #XREF,DBPACFR,DBPACFP
DBPACPF,DBPACFV,DBPFLDT

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

RDBPAC executes all data base input/output for mainline programs.

Mainline PL/I programs are written with DBPL/I statements for data base input/output. (See the DBPL/I User's Guide, Section 8, Topic B.2). These statements are processed during compilation and CALL statements are generated (according to the DBPL/I-DBPAC Interface Specification, Section 3, Topic B.2). The first parameter passed in a CALL to RDBPAC is a Mainline File Control Block (see Section 3, Topic B.4).

RDBPAC executes the request indicated by the operation code in the MFCB. For physical input/output operations it CALLs appropriate entries in the RDBTSSIO module. Whenever RDBPAC detects either a logical error or a physical input/output error it posts an error code in the MFCB. (See DBPAC Error Codes, Section 3, Topic B.3).

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

The RDBPAC module does not do any terminal input/output or print any reports.

2. Input Data Sets

When a mainline program is accessing the descriptor data set of a data base, "descriptor descriptor" tables coded in RDBPAC are used

instead of an input descriptor data set.

3. Output Data Sets

The descriptor data set is updated as part of OPEN and CLOSE processing (setting and reseting the MNTNABLE and MNTNING switches).

4. Reference Tables

a. DBPL/I - DBPAC Interface (see Section 3, Topic B.2)

b. DBPAC Error Codes (see Section 3, Topic B.3)

c. Mainline File Control Block (see Section 3, Topic B.4)

d. List Structure (see Section 3, Topic B.5)

e. Dataplex Descriptor File (see Section 3, Topic B.7)

f. Inverted Index Format (see Section 3, Topic D.5)

g. FLDTAB Table (see Section 3, Topic F.10)

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Receive Control

The entries at the beginning of the module are described here; entry DBPLDT is described in paragraph "f" below. All entries receive a Mainline File Control Block (MFCB) as their first parameter. RDBPAC treats the MFCB as a simple parameter; that is, RDBPAC does not know that the MFCB is a CONTROLLED structure allocated by the mainline: RDBPAC never ALLOCATEs or FREEs an MFCB.

For the #FIELD and #XREF function entries, an appropriate operation code is posted in the MFCB and the second parameter, which is a file name, is copied into the MFCB. This is necessary because the function references in

the mainline have not been expanded by the DB preprocessor.

The DBPACFR entry handles a user record in the form of a character string as its second parameter.

The DBPACFP and DBPACPF entries both handle a user list pointer as their second parameter. DBPACPF additionally accepts a user subscript as its third parameter. A switch indicating the absence or presence of a user subscript is set.

The DBPACFV entry handles a user field value in the form of a varying length character string as its second parameter. The DBPACFV entry is also used for all statement calls that only pass an MFCB without a second parameter.

b. Common Code

Handling for PL/I errors that may occur in RDBPAC is initialized so that they will cause a jump to paragraph "m" below before returning to the mainline.

If the MFCB is closed and a redundant CLOSE operation is attempted then control branches directly to the common return paragraph "m". If an OPEN operation or an operation that can imply opening (most record level operations) is encountered then control branches to the open routine - paragraph "d". If the operation can not imply opening then an error is raised: a specific error code is posted in the MFCB and control jumps to the common return - paragraph "m". This is an example of the general method RDBPAC uses when it detects an error.

If the MFCB is open the operation code is checked for validity. Close and open (which is re-open in this case) operations branch to the close routine-paragraph "c". Record operations branch to paragraph "e". Get operations branch to paragraph "h". Put and Reput operations branch to paragraph "i". An invalid operation code raises an error and jumps to the common return - paragraph "m".

c. Close Routine

For each data set in the data base the unlock subroutine is called and the ASMCLOS is called.

For a simple close operation control branches to the common return. For a close erase operation, ASMERSE is called either for the descriptor data set or for the "data" data sets and control branches to the common return.

d. Open Routine

The TSS userid is obtained by calling ASMID. For a re-open operation on the same data base with the same security password, the following descriptor read-in and File Control Block (FCB) initialization steps are bypassed. For an open operation on a descriptor data set, a pointer to the hard-coded descriptor descriptor table in main storage is posted in the MFCB and the following descriptor read-in step is bypassed.

To read in the descriptor records, ASMDCB, ASMFNDS, and ASMOPEN are called. Then for each region (describing one date set) ASMGETK is called to read the file descriptor record and ASMGET is called repeatedly to read the field descriptor records. A descriptor for the RECLEN field is bypassed except on the first data set. When the descriptor for the key field is found, it is stored at the top of the DESC table, other descriptors are stored sequentially which is alphabetical by field name. Superfield descriptors are reread (by calling ASMGETK) so that their component fields may be checked (if a component has failed security checking then the superfield also fails). Finally ASMCLOS is called to close the descriptor data set.

Next for each data set a File Control Block (FCB) is allocated and a skeleton Data Control Block is copied into it and ASMFNDS is called. For OUTPUT or UPDATE mode a null record is composed in the FCB by finding the primary field descriptors. After the FCBs are all initialized, then file and field subscripts (INVFLCUR,ASSOCCUR,SUBFLCUR, and RELFLDSS) are determined.

If a non-descriptor file is being opened for output or update then the MNTNABLE or MNTNING switch in the anchor file descriptor record is updated. For output or update, any subfile data set in the data base is opened and the highest id-key in use is found by calling ASMOPEN and ASMGETK. If the operation was an explicit open then control branches to the common return. Otherwise it was an implicit open and control proceeds to the record routine.

e. Record Routine

The record-level routine is used for WRITE, LOCATE, READ and UNLOCK operations. The WRITE operation is handled separately by calling ASMPUTK and branching to the common return.

For LOCATE and READ operations, the element GET cursors are reset for the particular data set or for the anchor and associate data sets. The LOCATE SUBFILE operation is handled separately at this point: control branches into the Put routine to find the anchor or associate control field for the subfile. A subrecord id-key is determined from the highest id-key in the control file or, if it is null, from the highest id key previously used in the subfile. A current subrecord is built by copying the null record built by the open routine, posting the new id-key and posting the parent key field by copying the anchor key field. Control branches into the Put routine again to put the new element into the control field. To better ensure data base integrity, the anchor or associate record containing the control field is immediately written or rewritten and reread by calling ASMPUTK and ASMGETK. If the control field is on an associate and the anchor record was newly located then it is written and reread too.

An anchor LOCATE operation is handled separately at this point: control branches to the Validate key routine (described with and also used by the READ KEY operation) and then an attempt is made to read the new key using ASMGETK. If the new key is found, the record is made current (just as if a READ KEY operation had been requested) and an error is

raised. Normally, the new key will not be found and a current anchor record is built by coping the null record built by the Open routine (or, for a descriptor data set, a hard-coded null file or field descriptor record is copied) and the new key valve is posted in it.

Spanned index reads are handled separately at this point: their fundamental objective is to make the last record of a spanned region current. For read INDEX BACKWARDS either ASMSTLP (if the old suffix was 0) or ASMSTLK (to read the old region suffix0) are called and the ASMSTLP is called to position at the last record of the previous region. For read INDEX forwards ASMGET is called to read the first record of the next region and then ASMSTLK is called with suffix FF to position at the last record of the new region. For read INDEX KEY the validate key routine (described later) is used and then ASMSTLK is called with suffix FF to position at the last record of the new region. Then for all types of read INDEX, ASMGET is called to read the last record of the new region.

Normal (un-spanned) reads are processed as follows. For read BACKWARDS, ASMSETL is called with the 'P' option to position to the previous record. For read forwards, it is unnecessary to do any file positioning. For read PER SUBFILE, the parent key value is taken from the current subrecord for use without validation. For read by KEY, the Validate key routine is used. The Validate key routine (also used for LOCATE KEYFORM) calls the generic conversion routine, if specified in the key field descriptor, and then calls the validation routine, if specified, using "CALL CALL" service for both purposes. For read LIST, the appropriate key value is taken from the list (next forward, next backward, or by subscript) for use without validation. Then for all non-locking direct reads (PER SUBFILE, by KEY, or from LIST), ASMSTLK is called to position to the desired record. Now the file is positioned for all reads (except direct locking) and ASMGET is called to actually read the desired record. Then if the record is to be locked and for direct locking reads, ASMGETK is called to reread or read the record and lock

it for exclusive use. Next, for INPUT modes, any record level security checking is done; if it fails and it was a sequential read (forwards or backwards), control loops back to do another sequential read until a record that passes security or end-of-file is encountered. If record security fails for a direct read, a key-not-found error is raised. For reading a descriptor data set only, the region is compared to determine if the read stayed within the region that was opened and the key is checked to determine if a file or a field descriptor was read so that the pointer to the appropriate hard-coded descriptor descriptor table can be posted in the MFCB to govern subsequent field level operations. If an anchor record was read, then all subfiles are checked: any having a current subrecord with a different parent key to the new anchor key are marked "not current". If a subfile record was read, then the anchor and all other subfiles are checked: any having a current (sub) record with a different (parent) key to the new subrecord's parent key are marked "not current".

- DBPFLDT Entry

  The DBPFLDT (Post FLDTAB) entry is provided to build a Field name Table by reference to RDBPAC's main storage descriptor tables built by the Open routine. This entry is not supported by a DBPL/I statement a mainline program must:

  i. execute a DBPL/I OPEN statement or a record level statement implying opening.

  ii. "CALL DBPFLDT(mfcb); where mfcb is the file name of the data base that was opened.

  iii. have a "% INCLUDE LISRMAC(FLDTAB);" statement to copy in the declaration for FLDTAB. Use of this entry is optional; RDBPAC makes no use of FLDTAB.

- FLDTAB Routine

  FLDTAB is allocated or freed and reallocated with its size adjusted to hold the number of

field names in the data base. RECLEN and the key field name are posted. The anchor descriptors are searched to find anchor field names to post for Format 2. The anchor descriptors are searched again to find associate field names to post for Format 3. Each subfile's descriptors are searched in turn to post subfile field names for Format 4. If any superfields were noticed in the anchor or associate searches, they are found again and their components analyzed to determine whether to post the superfield in Format 2 (all components from anchor) or Format 3 (one or more associate components but no subfile components) or Format 4 (one or more subfile components).

h. Get Routine

The Get routine is used for all GET operations and for the #FIELD and #XREF functions.

When a field name has been passed or posted in the MFCB, it is found in the DESC table to determine the data set for the GET3 otherwise, the first data set (the one specified by the OPEN TITLE cluse) is implyed.

If that data set does not have a current record, then for the #XREF function a zero is returned. If it is the anchor data set and any subfile has a current record, its parent key will be used to read (using ASMGETK). The anchor record whose record security will be checked: if it fails, a null value will be returned and control branches to the common return or, for #FIELD, a zero is returned.

The GET RECORD operations is handled by coping the record from the FCB to the user's string and branching to the common return.

For the GET LIST SET statement (DBI) and GET INDEX LIST SET statement (DB2) the cross-reference field descriptor is found and control branches down to the Get Field routine.

For the GET KEY SET, GET SUBFILE KEY SET and GET INDEX KEY statements the appropriate key descriptor is found and control branches down

to the Get Field routine.

For the #XREF function the cross-reference field descriptor is found and control branches down to the Get Field routine.

For GET FIELD, #FIELD and GET SUBFILE LIST SET if the descriptor found previously was a dummy, then the corresponding real descriptor must be found in an associate descriptor table. For GET FIELD and #FIELD of a superfield, a loop is initialized to take each component field, starting with the first, find its read descriptor and record (using ASMOPEN and ASMGETK for an associate record if necessary) and perform the Get Field routine repeatedly until the superfield has been composed or its count determined.

Get Field handles a bit field, a fixed length byte field, a simple variable field, a fixed length element of a multi-element field or a variable length element of a multi-element field.

GET KEY SET operations are handled separately after the fixed length key has been extracted. If necessary, a list segment is allocated and chained and initialized.

For the SUBFILE option the subfile id-key field name is found in the subfile descriptor table. For an index option, if the index is spanned and the last suffix is greater than zero, the first record in the region is read using ASMGETK and control branches back to the Get Field routine. A list segment is allocated, with its size governed by the field's length, and chained and initialized and posted with the whole multi-element field value. For a spanned index, if the suffix is less than the last in the region, then the next index record is read using ASMGET and control branches back to the Get Field routine; this repeats until the whole region has been copied into list segments and the data set is positioned at the last record of the region again.

The #FIELD function is handled for the null and real value cases of all five types of direct fields and for the case of an empty associate data set or an absent associate

record. Superfields are handled by effectively evaluating #FIELD for each component to determine the net count. The #XREF function for a spanned index calculates the number of cross-references on records preceding the last in the region by assuming full maximum length records and adds the number of cross-references on the last record. The #FIELD and #XREF functions are thus complete and return their function value directly (without branching to the common return).

The GET INTO operations are handled for the null and real value cases of all five types of direct fields and for the case of an empty associate data set or an absent associate record. Superfields are handled by looping back to get each component field and concatenating them together.

i. Put Routine

The field name passed in the MFCB is found in the DESC table to determine the data set implicated. If it is an associate data set, it is opened, if necessary, by calling ASOPEN and read, if necessary, by calling ASMGETK and if the record is absent a current associate record is built by copying the null record built by the open routine and the anchor key value is copied into it. If an anchor key or a subfile id-key is being REPUT to null, then control branches to the Delete routine described in paragraph "j" below.

For a fixed length field or element, the new value is justified right or left depending on the NUMALIGN switch in the field descriptor. For a variable length or multi-element field, the field length and record length (RECLEN field) are adjusted as necessary. If the field is indexed and had a non-null value, then the Delete XREF subroutine (described in paragraph "l" below) is called. If the new value is non-null and the field is indexed, then the XREF subroutine (described in paragraph "k" below) is called.

j. Delete Routine

Nulling a subfile id-key indicates that a subrecord is to be deleted. The subfile

control field descriptor is found and if it is on an associate, the associate data set is opened, if necessary, by calling ASOPEN and read, if necessary, by calling ASMGETK. The control field element is found and excised and the field length and RECLEN are decremented. Then the subfile descriptors are searched: for all indexed fields, the Delete XREF subroutine (described in paragraph "l" below) is called for each element value. Finally the subrecord is deleted by calling ASMDELR.

Nulling an anchor key indicates that an anchor record and its associated and subordinate records are to be deleted. The anchor descriptors are searched for subfile control fields and for indexed anchor or associated fields. If a control or indexed field is found on an associate data set, it is opened, if necessary, by calling ASMOPEN and read, if necessary, by calling ASMGETK. For each control field, the subfile is opened, if necessary, by calling ASMOPEN and each element is used to read a subrecord using ASMGETK. The subfile descriptors are searched for every subrecord: for all indexed fields, the Delete XREF subroutine is called. Each subrecord is deleted by calling ASMDELR. During the anchor descriptor search, when an indexed anchor or associate field is found, the Delete XREF subroutine is called.

k. XREF Subroutine

The XREF subroutine is called from the Put routine when a non-null value is PUT or REPUT to an indexed field. The inverted index data set is opened, if necessary, by calling ASMOPEN. Then an index read is attempted using ASMGETK (with a suffix of zero if it is spanned). If the record is not found, then the null record built by the Open routine is copied, the cross-reference and the indexed value are copied in, it is written by calling ASMPUTK and control returns to the calling program.

If an index record is found, then its highest (rightmost) cross-reference value is compared with the new cross-reference. If the new reference is lower, then the insertion point is found by a binary search and the new

reference inserted; otherwise the new reference is appended. If the index is not spanned or if the region only needs one record, the cross-reference field length and RECLEN are incremented, the index record is rewritten using ASMPUTK and control returns to the calling program.

In a spanned index region when the zero suffix record is full, if its last reference is less than or equal to the new reference then it is released by calling ASMREL; otherwise the insertion point is found by a binary search, the new reference is inserted, the last reference overflows to become the new reference to be propagated forward, and the record is rewritten using ASMPUTK. The suffix is incremented and control loops back to attempt a read of the next record of the region. This continues as long as full records are found. finally a short record is found to append to or a fresh record is created and the process is completed like a non-spanned case and control returns to the calling program.

1. Delete XREF Subroutine

The Delete XREF subroutine is called from the Put routine when an indexed field that had a non-null value is being REPUT. It is also called exhaustively by the Delete routine for indexed fields. The inverted index data set is opened, if necessary, by calling ASMOPEN. If the index is spanned, the last suffix of the region is determined by calling ASMSTLK with a suffix of "FF" and ASMGET. Whether or not the index is spanned, ASMGETK is called to read the index record (with the highest suffix if it is spanned). If the index is not spanned or if the region only has one record, then the cross reference is found by a binary search and excised, the cross-reference field length and RECLEN are decremented, the index record is rewritten using ASMPUTK and control returns to the calling program. In the exceptional case of the index record only having the one cross-reference, it is deleted using ASMDELR and control returns to the calling program.

In a spanned index region having more than one record, the lowest (leftmost)

cross-reference value is examined before the binary search. If it is greater than the cross-reference to be deleted, then the whole cross-reference falls off to be rolled backward in the region. The record is then rewritten (with the field length and RECLEN decremented if necessary) using ASMPUTK or deleted using ASMDELR. Then the previous record is read using ASMGETK with the next lower suffix and the lowest cross-reference examined. This process repeats rolling one cross-reference backward in the region until the record is found with a lowest cross-reference less than or equal to the one to be deleted. The cross-reference is found by a binary search and excised, the rolled cross-reference from the record just processed is posted at the right end, the record is rewritten using ASMPUTK and control returns to the calling program. If the cross reference is not found on the record, it belongs on then the record is released using ASMREL and in the simple case control returns to the calling program. However, if rolling back had been started in a spanned region, one cross-reference is still in limbo, so control branches into the XREF subroutine which will roll one cross-reference forward from that point to reconstruct the region before returning to the calling program; this should be an extremely infrequent occurrence.

m. Return

The common Return is used by all routines. The only exception is that when the #FIELD or #XREF functions complete successfully they return directly.

When an error has been detected, an error code is posted in the MFCB. The address of the MFCB is posted in DBEFCBP to assist any mainline having multiple MFCBs. If the mainline has a current DBPL/I ON ERRORFILE GO TO ... action, then RETRNPT is called to post MFCB.ONRETURN and RDBPAC is left by branching to the mainline label in MFCB.ERROR.ROUTINE. Otherwise RDBPAC is left by signalling the PL/I ERROR condition which, unless the mainline catches it, will terminate the mainline program.

Normally, RDBPAC is left by a simple RETURN statement and control returns to the mainline that called.

F. CODING SPECIFICATIONS

1. Source Language

RDBPAC is written in PL/I. The DB preprocessor and DBPL/I are not used in RDBPAC. Various Assembler language subroutines are used as mentioned in the Processing Requirements Narrative.

2. Suggestions and Techniques

When a desired field descriptor has been found by subscript in the tables, its address is held in a pointer variable and based structure references are used to avoid frequent re-evaluation of the subscript. Similar techniques are used whenever possible.

Binary search techniques are used to maintain the cross-reference lists in inverted index records in ascending sequence.

The facilities available in the RDBTSSIO module are used to the best possible advantage with the TSS operating system VISAM access method.

The RDBPAC module is designed and implemented to be reentrant under multi-programming; automatic, controlled and based storage are used appropriately. One known exception is that the main storage descriptor descriptor tables are static for efficiency; if two or more users attempt to access the same descriptor data set region concurrently they may encounter interference on the multi-element field cursors (only RSECTYCD, NAMEFLD and SECURITY fields are affected).

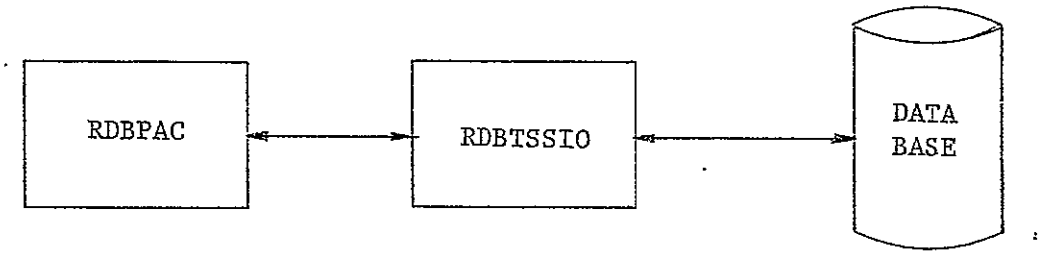


Figure 1. I/O Block diagram

正、倍2.1

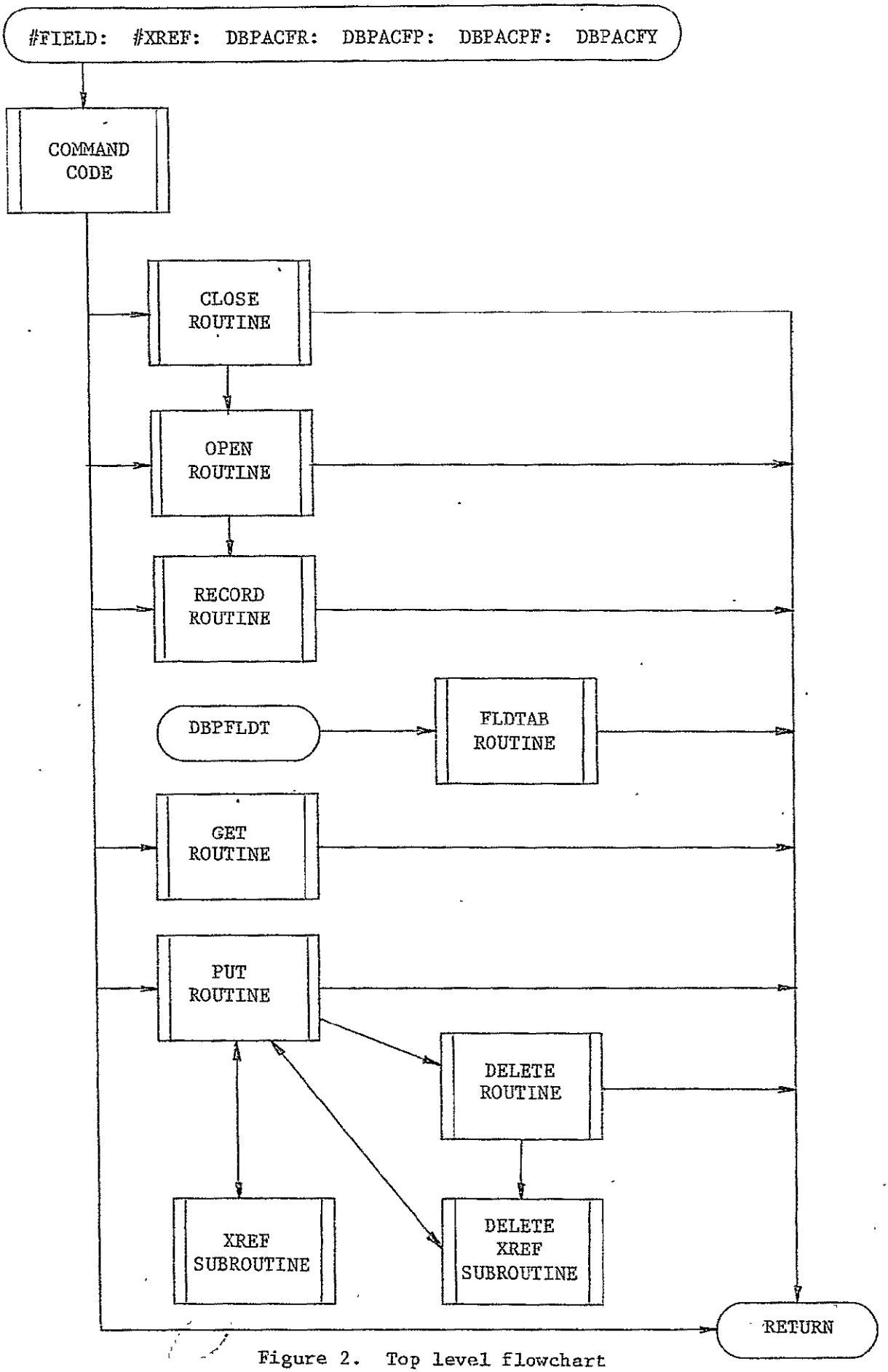


Figure 2. Top level flowchart

IV. B2.2

TOPIC B.3 - EXECUTIVE ASSEMBLER PROGRAMS

A. MODULE NAME

Executive Assembler Program
Program-ID - RDBTSSIO
Module-ID - RDBTSSIO

B. ANALYST

Phillip D. Pritchard
Neoterics, Inc.

C. MODULE FUNCTION

This program works in conjunction with the Data Base Executive Program (RDBPAC) and provides the assembler language macros required to handle the input, output and updating of VISAM files, as well as the handling of error conditions.

These VISAM files are the files of a data base and the Data Base Executive will call the Executive Assembler Program when it needs an I/O operation performed.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

All of the files which make up a data base could conceivably be input, including descriptor files. The only real restriction is that the files be VISAM.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Same as input files.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

This program is designed to handle the input and output functions for the Data Base Executive (RDBPAC). It deals strictly with VISAM files.

The program is divided into many routines, and each of these routines has a unique function (Illustrated in Table 1). The Data Base Executive (RDBPAC) calls these routines individually to perform the various functions which are required. Associated with each of these calls is the passing of the required parameters.

The abilities of these assembler routines are comprehensive enough to handle any situation which might arise in the Data Base Executive. This includes the abilities to: open files for input, output, or update; read the file sequentially, read the file by key, exclusively or non-exclusively; position the file to the beginning, the end, the previous record or the next record; and close the file. For example, if the Data Base Executive were required to open a data base in the update mode and process records,

the sequence of calls would be as follows:

```
CALL ASMDCB   (parameters)
          establish the files DCB (data
          control block).

CALL ASMFNDS  (parameters)
          link the DCB with the JFCB (job
          file control block).

CALL ASMOPEN  (parameters)
          open the file.

CALL ASMGETK  (parameters)
          read a record by key.

CALL ASMPUTK  (parameters)
          rewrite the record.

CALL ASMCLOS  (parameters)
          close the file.
```

The Executive Assembler Routines (RDBTSSIO) is called from the Data Base Executive (RDBPAC). If no errors are detected by the assembler routines, the error switch (one of the parameters) is set equal to zero upon return to RDBPAC and the return is to the specified 'Good' return address (one of the parameters). If an error is detected by the assembler routines, the error switch is set with the proper error code and the return is to the next sequential instruction in RDBPAC. The error codes will have the following values when an error occurs in ASMOPEN, ASMPUTK, ASMGETK, ASMGET, ASMPUT, ASMSETL, ASMESTL, ASMREL, ASMCLOS, ASMDELR or ASMSTLK:

a. 04 - keys equal (sequence error)
b. 08 - key not found
c. 12 - key out of sequence
d. 15 - keys do not coincide
e. 20 - keys coincide
f. 24 - invalid retrieval address
g. 28 - invalid record length
h. 31 - position past end of data set
i. 36 - position before start of data set
j. 40 - exceed maximum number of overflow pages
k. 44 - exceed maximum size of stored data set

The assembler routines will add 100 to all of the above error codes prior to returning to the Data Base Executive (RDBPAC). The end of data exit sets the error switch to 99. The error switch is

a fixed binary half-word.

The first parameter is always the DCB address (DCB means Data Control Block). The second parameter is the record area, except for:

a. The open (ASMOPEN) - in this instance, it is a one byte function code -
   I = input
   O = output
   U = update

b. The close (ASMCLOS)
   ESETL (ASMESTL)
   STLK (ASMSTLK)
   REL (ASMREL) - in these instances, it is a one byte dummy character (no meaning.)

c. The DEIREC (ASMDELR) - in this instance, it is the key.

d. The SETL (ASMSETL) - in this instance, it is a one byte function code -
   B = beginning
   E = end
   N = next
   P = previous

The third parameter indicates the routine to which return is made if there are no errors.

NOTE: The error switch parameter for the following routines must be preset.

a. ASMGETK - 01 if KY(Read by key)
   00 if KX(Read by key exclusive)

b. ASMPUTK - 01 if KT(Write)
   00 if KS(Rewrite)

The routines and their functions are as follows:

a. ASMFNDS: This routine obtains the location of the JFCB corresponding to a given data set name. If the data set name specified is not in the task definition table (DDEF'ed) but is in the catalog, the JFCB is created.

   If the data set name specified is in the task definition table (DDEFed), the JFCB is already in existance. If the data set name is neither DDEFed nor cataloged and the key

length is passed as a parameter in the error switch, the file is DDEFed and the JFCB is created.

The DDNAME used is posted from the JFCB to the DCB, and the owner-ID is posted from the JFCB to the user's area.

The parameters required for successful execution of the FINDDS are as follows:

1. The DS name (35 characters)
2. The DCB address
3. The owner's ID
4. The error switch (key length)

b. ASMCAT: This routine does a standard catalog of the string passed as the first parameter. The string has the standard CAT macro format. The parameters are as follows:

1. The catalog parameter
2. The return code

c. ASMPR: This routine tries to do a macro print on the string that is passed as the first parameter. It has the format of a standard PRINT macro parameter. The parameters are as follows:

1. The PRINT parameter
2. The return code

d. ASMERSE: This routine erases the direct-access storage for a data set. In addition, it will remove the entry for a catalogued data set from the catalog. The DSNAME passed is padded with blanks to 35 characters. If a stored data set is opened by many users concurrently, a particular user cannot erase that data set until every other sharer actively using that data set issues a close.

Once a user is the only currently active task using the data set, he may erase it regardless of whether he has closed it or not.

The parameters required are the DSNAME and the error switch.

NOTE: For both ASMFNDS and ASMERSE, the

error switch upon return to the Data Base Executive is equal to zero only if no error occurred.

e. ASMOPEN: This routine connects the data set to the system by completing the DCB (containing the attributes), indicates the manner in which the data set is to be processed and positions the data set for processing. The address of the SYNAD routine (SYNADRTN) and the address of the EODAD routine (EODADRTN) are posted to the DCB. The address of the save area is also posted to the DCB.

The parameters are as follows:

1. The DCB address
2. The function code
3. The 'Gocd' return address
4. The error switch

f. ASMPUTK: This routine moves a selected record from a user specified area to an output buffer. The system then includes the record in the output data set by key. This operates in one of two modes: Rewrite (KS) or Write (KT). Write releases any page level interlocks set for the data set. The parameters are as follows:

1. The DCB address
2. The record area (address)
3. The 'Good' return address
4. The error switch (preset:
   0 means Rewrite (KS),
   1 means Write (KT)).
5. The key (address)

g. ASMGETK: This routine obtains a selected logical records from an input data set and moves it to a user specified area. There are two modes, read with interlock (KX) and read with no interlock (KY), both by key. The parameters are as follows:

1. The DCB address
2. The record area (address)
3. The 'Good' return address
4. The error switch (Preset: 01 means read with interlock (KX), 00 means read with no interlock (KY)).
5. The key (address)

h. ASMGET: This routine obtains the next sequential record and moves it from an input buffer to a user specified area. The parameters are as follows:

1. The DCB address
2. The record area
3. The 'Good' return address
4. The error switch

i. ASMPUT: This routine has the same parameters as the ASMGET routine. However, instead of reading a record, it writes a record.

j. ASMSTLK, ASMSETL: These routines position a data set. The parameters for both routines are as follows:

1. The DCB address
2. The code : K (by key)
   B (beginning)
   E (end)
   N (next)
   P (previous)
   NOTE: the 'P' actually does two SETL 'P's in order to allow for reading the file backward.
3. The 'Good' return address
4. The error switch
5. The key (address, for ASMSTLK only)

k. ASMSTLP: This routine has the same parameters as the ASMSTLK and ASMSETL routines. It does one SETL 'P'.

l. ASMESTL: This routine releases a page level interlock imposed by another macro.

The parameters are as follows:

1. The DCB address
2. A dummy character
3. The 'Good' return address
4. The error switch

m. ASMDELR: This routine deletes a record from a VISAM file. The parameters are as follows:

1. The DCB address
2. The key
3. The 'Good' return address
4. The error switch

n. ASMREL: This routine makes the record available to other users.

The parameters are the same as in the ASMESTL routine.

o. ASMCLOS: This routine closes the file (VISAM).

p. ASMDD: This routine does a general purpose DDEF after trying to release the 'UNIQUE' DDNAME it creates. It then passes the DDNAME to the calling program. The parameters are as follows:

1. The DDEF parameter string
2. The returned DDNAME
3. The return code

The first parameter is the DDEF information, identical to the parameter string for the DDEF macro, less the leading 'DDNAME'.

The second parameter like the first is a varying character string. It is the variable to which the DDNAME is returned. The third parameter is the return code for the DDEF.

q. SYNADRTN, EODADRTN: When an end of file or some error is detected during any of the routines in this program, these routines set the proper error code in the error switch and return control to the Data Base Executive for appropriate action.

r. GETRECRD, OTHGET: When a read by key, non-exclusive, is executed and an error is detected, they will function as follows:

When the ASMGETK is called to read non-exclusively and the record is not found (ERROR X'08') then a SETLK to the not-found key is performed. This action positions the data set to: The last record if the key is beyond the end of the data set, to the next lowest key if the key is in the central portion of the data set, and to the first record if the key is prior to the beginning of the data set. The SETLK returns to the SYNAD routine. At this point, the record indicated is read. An error code of X'108' is returned to the calling module (RDBPAC), but in fact, the record indicated is

current.

s. ASMDCB: This routine takes the DCB created in this program and moves it to the user's specified area. The only parameter is the user specified area (address).

t. ASMID: This routine determines the user - ID (TSS - ID) of the task and places it in the user specified area. The only parameter passed is the address of the user specified area.

u. ASMRELS: This routine is used to release the JFCB created by the DDEFing of a particular file. The only parameter passed is the DDNAME associated with the DDEF. Any errors that occur are ignored.

v. CALL: This routine allows a PL/I program to call an external routine by specifying its name at execution time. Any parameters other than the called routine name, are passed on to the called routine for interpretation. The name specified must conform to the name construction standards of TSS/360.

w. RETRNPT: This routine is used by the Data Base Executive error routine. It posts the double word in the MFCB so that the user (of DBPAC) can return to the next sequential instruction in his program after the occurrence of an error. The first word is the invocation count; the second word is the address.

x. ASMMODE: This routine is used to determine if the maintenance task is running in a batch mode. It returns a 'C' if running conversationally; or it returns a 'B' if not.

y. DBUCHEK: This routine is used to validate the construction of an external name. The rules used are:

1. the name must begin with an alphabetic character (including #, $, @),
2. the name must be eight characters or less,
3. the second and subsequent characters of the name must be alphanumeric (including #, $, @, ?).

The parameters passed are the name and the name length (in the event that the user wishes to restrict it to less than eight). If the name is invalid, the length paramter will be set to one, as an error indicator, otherwise it will be set to zero.

z. ASMXTR, ASMPASS, ASMMUST: These entry points simply transfer control to the MTT monitor to maintain linkage conventions.

F. CODING SPECIFICATIONS

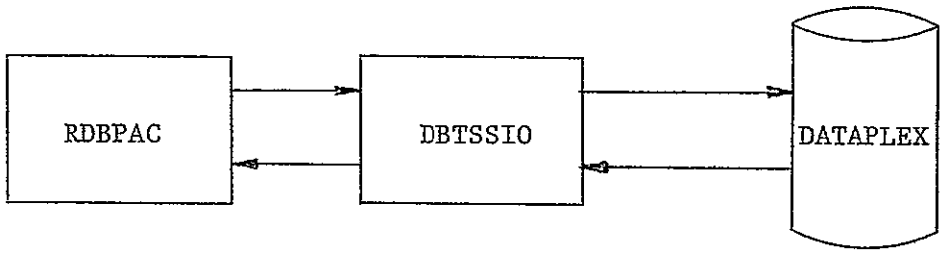
1. Source Language

Unlike most other modules for the NASIS system, the Executive assembler program (RDBTSSIO) is written entirely in Assembly language.

2. Suggestions and Technigues

a. Special attention is paid to the linkage conventions of the current PL/I compiler.

b. The Data Base Executive, by design, is the primary user of this program. However, the program is written so that programs other than the Data Base Executive can use it.


RDBPAC
DBTSSIO
DATAPLEX

IV 33.1

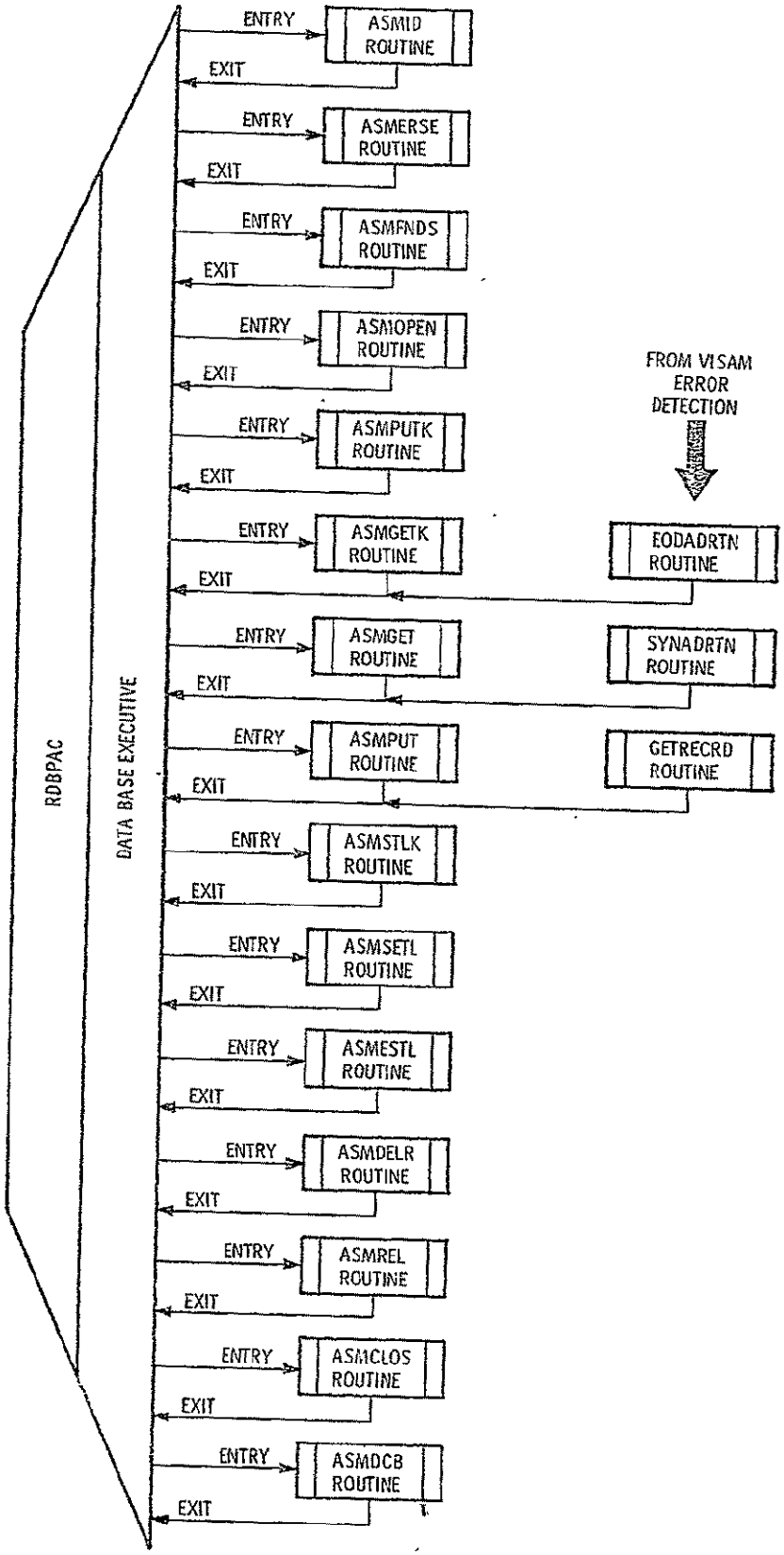


Figure 1 - Top level flow chart

IV B 3.2

PARAMETER NUMBER	REGISTER NUMBER	PARAMETER
1	2	DCB ADDRESS
2	3	RECORD AREA FUNCTION DUMMY KEY
3	4	'GOOD' RETURN
4	5	ERROR SWITCH
5	6	KEY

TABLE I. - PARAMETERS.

IV B3.3

TOPIC B.4 - DATA BASE EXECUTIVE CONVERSION AND REFORMATTING ROUTINES

A. MODULE NAME

Standard Conversion and Reformatting routines for the Descriptor Editor and the Data Base Executive.

Program-ID - RDBEXITS
Module-ID - DBEXITS
Entry Points - See Table 1.

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

This module provides 31 standard general field conversion and reformatting routines. They are called by the Data Base Executive field processing routines (PUT, GET, and REPUT) if they are specified in the field descriptor record. The routines are written according to the DBPAC Exit Routines User's Guide (Section 8, Topic B.1) and may be used for user's database fields, if desired.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

The conversion routines (DBCVT__) are for use during PUT or REPUT field processing. They all accept a varying length character string argument and all allow the value to have leading and trailing blanks. They check the argument value according to the Notes in Table 1. If the argument value is invalid, they return with the BAD parameter left set. Otherwise they copy the value or convert it to the internal form and length shown in Table 1, reset the BAD parameter switch and return.

The reformatting routines (DBFMT__) are for use during GET field processing. They all accept a varying length character string argument (from the dataplex). If the argument length is not as shown under "Internal bytes" in Table 1, then the routine is being misused and the value "BAD. HEX=" is generated followed by the hexadecimal expansion of up to eight bytes of the argument. Normally the internal form of the value is reformatted to the external form and control is returned. These routines all produce exact length output (i.e. without leading or trailing blanks).

F. CODING SPECIFICATIONS

1. Source Language

PL/I with no DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

TABLE I

CONVERSION ROUTINE	NOTES	PURPOSE	INTERNAL BYTES	REFORMATTING ROUTINE
DBCVTS	1	Scientific (long float)	8	DBFMTS
DBCVTSS	1	Short Scientific (short float)	4	DBFMTSS
DBCVTLN	1	Long Numeric (fullword binary)	4	DBFMTLN
DBCVRSD	1	Scaled Decimal	5	DBFMTSD
DBCVTSN	1	Short Numeric (halfword binary)	2	DBFMTSN
DBCVTBN	1	Byte Numeric (quarterword binary)	1	DBFMTBN
DBCVTRL	1,2	RECLEN	4	DBFMTRL
DBCVTID	1	subfile ID key	3	DBFMTID
DBCVTFT	3	header descriptor FILETYPE	1	DBFMTFT
DBCVTFV	4	field descriptor Fixed or Varying	¼	DBFMTFV
DBCVTOO	5	Off or On	¼	DBFMTOO
DBCVTRS	6	header descriptor RSECTYCD	9	DBFMTRS
DBCVTNF	7	super-field descriptor NAMEFLD	9	DBFMTNF
DBCVTEN	8	External Name	1-8	(DBFMTSB)
DBCVTSB		Strip Blanks	variable	DBFMTSB
DBCVTHX	9	HeXadecimal	variable	DBFMTHX

1. Arethmetic conversion and size checking.
2. Value between 4 and 4000.
3. Valid values are: anchor, 1: associate, 2; subfile, 3; index, 4.
4. Valid values are: fixed, f, 0; varying, v, 1.
5. Valid values are: off, no, n, false, f, 0; on, yes, y, true, t, 1.
6. Has the form: nasis-id <=> hexmask.
7. Has the form: <external|e|internal|i> component-field-name.
8. First non-blank alphabetic; other characters alphanumeric.
9. Valid non-blank characters are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

IV. B.4.1

TOPIC B.5 - DATA BASE EXECUTIVE LIST PROCESSOR

A. MODULE NAME

Data Base Executive List Functions and Statements
Program-ID - RDBLIST
Module-ID - #LIST
Entry points - DBPAC,DBPACP,DBGLKN,DBGLIK,DBGLKI,
DBGLKO,DBSLLL,DBGLKS,DBPLIK,DUPLIST,
ULIST,LIST

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION:

RDBLIST performs services on lists (of keys in main storage) which do not require access to a data base. (Services requiring access to a data base are done by module,RDBPAC) The list services are:

1. Getting the number of keys in a list,

2. Getting a key from a list in various ways,

3. Building a new list like or from an old list,

4. Boolean combination of two lists, and

5. Freeing a list or all lists.

There are two means by which mainline PL/I programs use the DBLIST services:

1. By function reference. The #LIST, DUPLIST, ULIST and LIST functions are invoked by reference in a PL/I expression.

2. By use of DBPL/I. The other services are all used by having DBPL/I statements in a PL/I program. (See the DBPL/I Language Extension User's Guide, Section 8, Topic B.2.) These are processed at compilation time by the DB preprocessor function which transforms DBPL/I statements into normal PL/I CALL statements. (See the DBPL/I - DBLIST Interface, Section 3, Topic B. 10.) At execution time the various entries of DBLIST are called for the various services.

D. DATA REQUIREMENTS

See the DBPL/I Language Extension User's Guide, Section 8, Topic B.2.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

The routines all receive their parameter values as specified in the DBPL/I - DBLIST Interface because the DB preprocessor function generates a DECLARE of the entry points and their parameter attributes.

The routines all recognize if a list pointer parameter has the NULL value and process accordingly.

The routines all handle lists having continuation segments by stepping from segment to segment as necessary. In this regard DBLIST has three internal subroutines:

a. GET which gets the next sequential key from a (segmented) list.

b. PUT which appends a new key to a (segmented) list possibly allocating and CHAINing a new segment.

c. CHAIN which connects a new list segment to the previous segment.

When the routines detect any logic error, they post an error code number in LISTERR.ERROR.ONCODE. Then if the user has done a DBPL/I ON LISTERROR statement, a return is made to the user's error routine. Otherwise, the PL/I ERROR condition is raised.

#LIST is a function entry that accumulates and then returns the current count of keys in a (segmented) list.

LIST is a function entry that compares the keys in two given (segmented) lists and builds a (segmented) list to return consisting of the union (the OR operation) or the intersection (the AND

operation) or the difference (the MINUS operation) of the two given lists. For the OR operation, if one list is well then the other list pointer is returned immediately. For the MINUS operation, if either list is null then the first list pointer is returned immediately without building a copy.

DBPAC is an entry that systematically deallocates all list segments for a subtask thus freeing all lists.

DBPACP is an entry that deallocates the segments of one list and unchains them from the chain for the subtask. Deallocation is not done if the LIST.PERMANENT flag has been turned on.

DBGLKN is an entry that gets the key specified by a subscript from a (segmented) list. If the subscript is zero or too high, the list is reset and a null string returned. If the subscript is any negative value, then the previous key is returned (except if the list was reset then the last key is returned or if the first key was current then the list is reset and a null string returned).

DBGLKI and DBGLIK are entries that get the next key from a (segmented) list (except if the list was reset then the first key is returned or if the last key was current then the list is reset and a null string returned). DBGLKI will call a conversion routine for the key value (if specified); DBGLIK always returns the unconverted interval key value.

DBGLKC is an entry that resets a (segmented) list so that the first (or last) key will be available.

DBSLLL is an entry that allocates the first segment for a new list and initializes it to be like an existing list (except that it has no keys yet).

DBGLKS is an entry that copies the current internal key from a (segmented) list to the end of another compatible (segmented) list.

DBPLIK is an entry that puts an internal key value at the end of a (segmented) list.

DUPLIST is a function entry that returns a copy of a (segmented) list. The copy is "condensed", that

is, it has full maximum size segments.

ULIST is a function entry that steps through the keys of a (segmented) list checking for duplicated internal key values. If none are found, i.e. the keys are unique, the list pointer is returned without building a copy. If duplicated keys were found, a copy of the list having only a single instance of any keys that were duplicated is returned. The copy is "condensed", i.e. it has full maximum size segments.

F. CODING SPECIFICATIONS

1. Source Language

PL/I.

The LIST and LISTERR declarations are included from the SOURCER.LISRMAC dataset.

No Assembler routines are used.

2. Suggestions and Techniques

The internal procedures GET, PUT and CHAIN simplify, standardize and expedite key by key processing of segmented lists.

TOPIC B.6 - DATA BASE EXECUTIVE PARENT - CHILDREN PROCESSOR

A. MODULE NAME

Data Base Executive Parent and Children List Functions.
Program-ID - RCCLIST
Module-ID - CCLIST
Entry points - UPLIST,CPLIST

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

RCCLIST builds a list of children (or parent) keys in main storage from a list of parent (or children) keys.

Mainline PI/I programs use the RCCLIST services by function reference in a PL/I expression.

D. DATA REQUIREMENTS

See the DBPL/I Language Extension User's Guide, Section 8. Topic B.2.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

The routines all receive a MFCB (Mainline File Control Block) as their first parameter. They pass it though when they call RDBPAC. RCCLIST's second parameter, a subfile control field name, is posted in MFCB.ONFIELD for RDBPAC. The routines all receive a list pointer parameter. If it has the NULL value, they return a NULL list pointer immediately.

Then READ FILE LIST KEY (0); is done to reset the READ cursor of the input list and the list's key field name is compared with the anchor key field name in the core descriptor tables. For CCLIST they should be equal (the input list should be an anchor key list); if unequal, the input list pointer is returned immediately. For CPLIST and

UPLIST they should differ (the input list should be a subfile key list); if equal, the input list pointer is returned immediately.

The #LIST function is invoked on the input list to obtain the count to govern further processing.

For the CCLIST function READ FILE LIST NOLOCK is done iteratively to process all (parent) records in the input set. From each one a GET FILE SUBFILE LIST SET is done using the second parameter for the subfile name. This returns a list pointer to a temporary set consisting of the subfile control field. If it is null, control loops back to the next READ and GET. If it is the first non-null control field encountered, it is made the basis for the output list. If it is a subsequent non-null control field it must be merged with the previous output. If its first key is higher than the last key of the previous output (usually) the temporary set segment is appended or the last segment. Otherwise (rarely) the OR LIST function of the RDBLIST module must be invoked to perform the merge and then the temporary sets must be freed. Control loops back until all the READ and GET's have been processed. The output list pointer is returned.

For the UPLIST and CPLIST functions a switch is set indicating whether duplicate-keys are to be dropped after the parent list has been built by code common to both entry points. One or more list segments of up to 32767 bytes of keys are allocated and initialized as necessary to hold as many parent keys as there are subrecord keys in the input list. READ FILE LIST NOLOCK is done iternatively to process all subrecords in the input set. From each one the internal parent key value is extracted and posted to the output list.

If the output list has only one key, its pointer is returned for either UPLIST or CPLIST. Otherwise the output list must be sorted into ascending collating sequence. For the CPLIST function the output list pointer is returned at this point.

For the UPLIST function a final pass over the output list is made to detect any duplicate keys. Each time a duplicate key is found it is deleted by shifting the remainder of the segment to the left and decrementing the segment count.

(This leaves a non-compressed list.) The output list pointer is returned.

F. CODING SPECIFICATIONS

1. Source Language

PL/I

The LIST and LISTERR declarations are included from the SOURCE.LISRMAC dataset. Declarations for MFCB, DESC, DESC_FLD and FCB structures have been taken from the source for RDBPAC.

No assembler routines are used.

2. Suggestions and Techniques

The name conflict between the LIST structure and the LIST entry to RDBLIST (which may be invoked from RCCLIST) can be circumvented by using PL/I preprocessor facilities to rename the LIST structure LISS during the compilation.

TOPIC C.1 - UTILITIES JOIN (RDBJOIN)

A. MODULE NAME

Joining new NASIS users
Program-ID - RDBJCIN
Module-entries - LBJOIN, JOIND, PAGERR

B. ANALYST

Edward J. Scheboth, Jr.
Neoterics, Inc.

C. MODULE FUNCTION

This program gives the NASIS DBA the ability to create and maintain the data set NASIS.USERIDS. This data set contains the NASISIDS under which users of the NASIS system are given access to MT/T, the Retrieval system and the various data bases. The data set NASIS.USERIDS is organized under VISAM, and has as a key composed of eight byte NASISID of each joined user, with a variable record format containing his password, timeslice, user authority, and list of permitted files.

This prcgram has as a secondary function the task of displaying for rdbinit the files available for retrieval to a specific user.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data sets

a. Parameter cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The NASISIDS data set. (For complete detailed specifications of this file see Section III of the Development Workbook).

d. On-line Terminal Entries

Valid JOIN commands.

3. Output Data Sets

a. Output Files

See 2.c

b. On-line Terminal Displays

See 2.d

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The primary entry point of this program (DBJOIN) is responsible for maintenance and display of the NASIS.USERIDS file.

The main routine has a prompt validation loop which calls the subordinate functions such as Join, Quit... etc. making the program more modular and much easier to modify.

Program termination is thru the common END convention set up in TS/2. All parameters to the commands shall be obtained using the new TS/2 facilities.

The secondary entry point of this routine (JOIND) displays the available files for DBINIT. This is really a sub function of the main routine's Display function and paging entry and should be coded as such to facilitate coding.

F. CODING SPECIFICATIONS

1. Source Language

   As much as possible of the RDBJOIN module is coded in the IBM PL/1 programming language. The input and output coding for accessing the file NASIS.USERIDS is handled by a direct call to the DBPAC assembler routines. All terminal access is handled by TS/2.

2. Suggestions and Techniques

   Refer to Section III of the Development Workbook for all data set specifications.

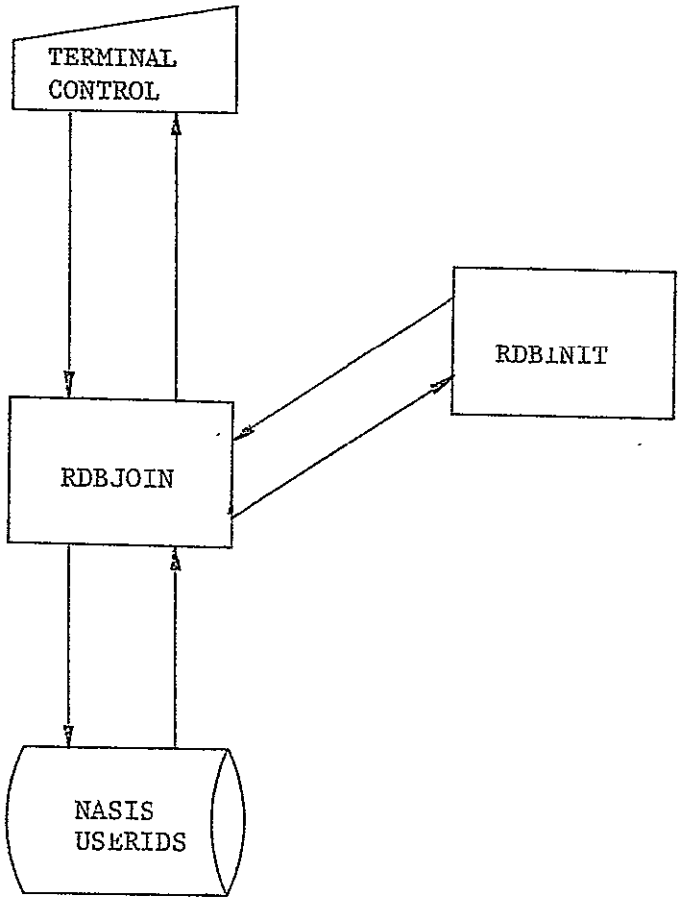


Figure 1. I/O BLOCK DIAGRAM

IV C. 1-1

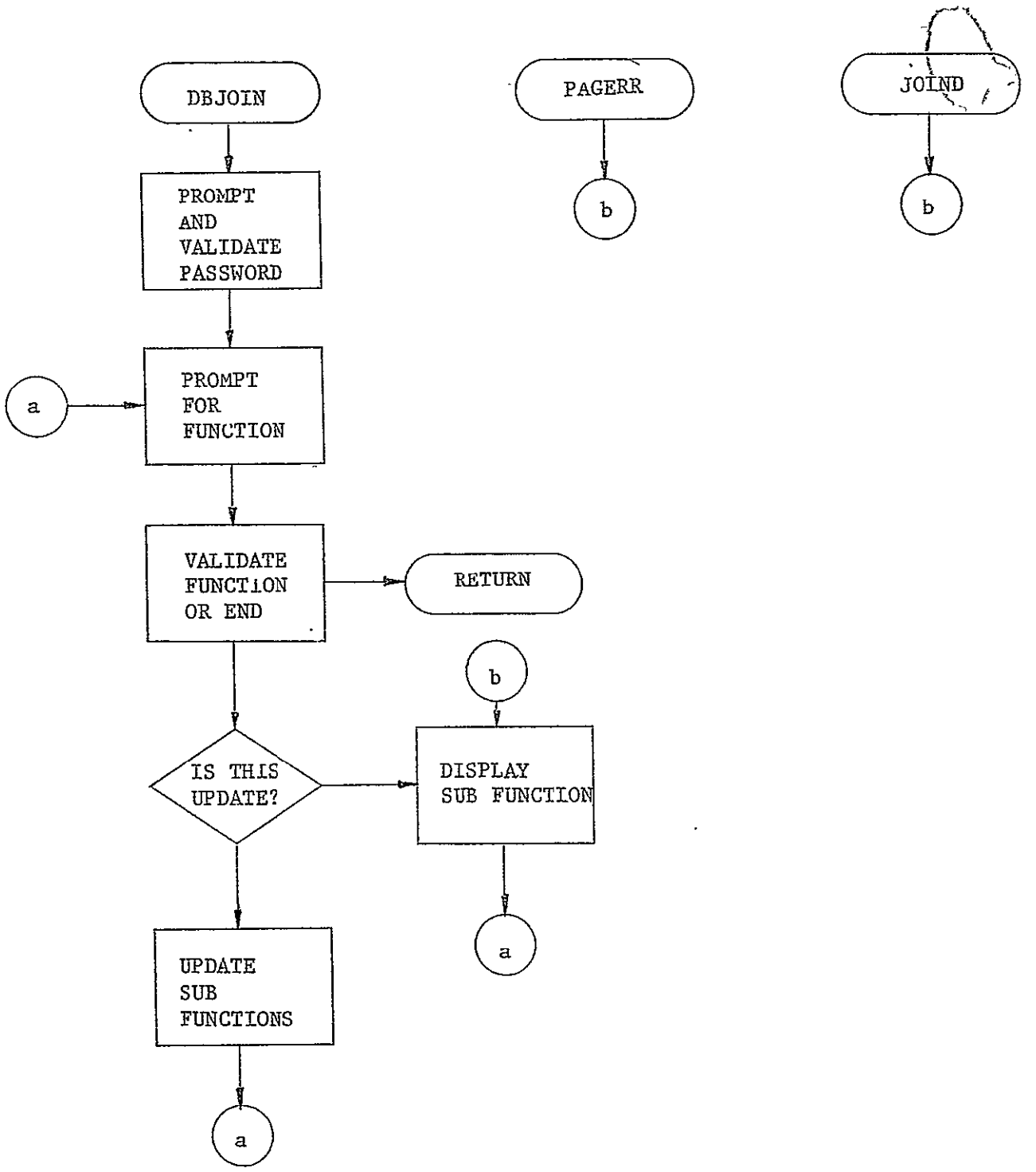


Figure 2. Top Level Flowchart

IV C 1-2

TOPIC C.2 - MESSAGE FILE EDITOR

A. MODULE NAME

Enter new EXPLAIN text into LISRMLF
Program-ID - RDBMLF
Module-ID - DBMLF

B. ANALYST

George F. Oswald
Neoterics, Inc.

C. MODULE FUNCTION

DBMLF will enable a systems programmer to enter NASIS EXPLAIN text into a library similar and replaceable with LISRLIB(LISRMLF). A person may copy LISRMLF into a VISAM file, DDEF it with the DDNAME of LISRMLF, perform editions to it with DBMLF, and test the new editions. Upon approval, the editions may replace the original LISRMLF member of LISRLIB.

The module will perform interactive editions consistent with TS2 conventions. The edit commands will provide the user the capability to ADD, DELETE, REPLACE, and DISPLAY a message; to PREFIX (set the filter prefix); and to END (terminate editions).

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Parameter Cards

Not Applicable

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The input file will be a VISAM copy of the current LISRLIB(LISRMLF).

d. On-Line Terminal Entries

All entries will be either command and parameters or command only; the form of the entries is either full words, codes, or no-entry. A default is made, where possible, when no-entry is made. Prompting for new command or parameters will occur whenever necessary.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Display

All on-line terminal displays for DBMLF follow the same format. All such displays are handled by TSPL/I commands.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. The purpose of DBMLF is to provide a system programmer the capability to ADD, DELETE, REPLACE, and DISPLAY, NASIS EXPLAIN text. The program is an interactive command-driven maintenance routine which creates a library similar and replaceable with LISRLIB (LISRMLF).

b. Mainline (Prompt and Validation Routine)

This routine is entered externally from program execution and internally upon command completion and error detection.

1. Program Execution

The user will be prompted to enter a command. The user has the option of entering either a command plus parameters or only a command. The routine will then validate the entire command string. However, if a parameter is in error, the user will be notified and prompted to re-enter the particular parameter. If a no-entry is detected for a parameter, the parameter will be defaulted, or if it is a mandatory parameter, the user will be prompted for the required parameter. As each parameter is accepted or defaulted, an entry will be made in the command argument data string. The command itself will be used to determine which command processor is to be invoked.

2. Command Completion

Upon completion of a command the user will be notified of its success full termination, and prompted for a new command or parameters. If the new entry contains a command keyword (ADD, DELETE, etc.), then execution will continue as described above (E.2.b.1). If the new entry is not a command keyword, then the program will assume the new entry is to be processed as prescribed by the previous command.

3. Error Detection

If an error has occurred during subsequent processing of the command argument data string, the user will be notified of the error and the user will be prompted for a new entry as described in 2 above.

c. Command Processor Routines

There are two unique forms of the command processors: Group 1 contains the commands that cause file access; they are ADD, REPLACE, DISPLAY, and DELETE. Group 2 contains program or data control; they are PREFIX which sets the filter prefix but causes no access to the file, and END which causes program termination.

1. File Access

The file access commands have two forms: those which write or rewrite data and those which read or delete data.

a. Write or Rewrite

These commands require a source key plus new text information.

1. ADD

The ADD command will specify with the type parameter the position into which the new text is to be placed; example, within ID, MSG parameter preempts line numbers 0 through 99 in increments of 5, EXP parameter preempts line numbers 100 through 199 in increments of 10, and the RESP parameter preempts line numbers 400 through 9999999. The lack of one of the above qualifying params designates the type and text of the command argument data string designates an explanation of a file oriented term or a global term. This form causes a default to the next available line number within that region.

Note that when positioning for adding a record to the file and a line number is not specified, a read will be attempted using the highest line number possible within the above-mentioned specification and thus provide the line number of the last record within type or term within the region.

In combination with the type param the line number may be specified; this combination denotes a line to be inserted within type or term, and

within region.

When the new line number has been established, the source key will be complete and a write of the new line will be initiated. Note: the writing of a new data line can cause a duplicate key error condition; this condition will be handled by a separate routine.

2. REP

When a request is made for a line to be replaced, all params are mandatory. The routine will first construct the source key and then initiate a read. Successful completion of the read determines that there is an existing line to be replaced. A key-not-found condition denotes that an invalid line number was supplied in the command argument data string.

Note: The existence of a key-not-found condition will cause an entrance into the error handling routine.

The data line obtained by the read will be replaced as denoted in the command argument data string, and a rewrite will be initiated. Note: There are no valid exceptions for the failure or the rewrite statement.

b. Read and Delete

These commands do not manipulate the data file per se, but provide for deleting or displaying of text within the delimeters provided by the command argument data string.

In both commands only the ID param is mandatory; therefore, it is

possible for a command to request the deletion or display of text from a single line number to an entire ID.

1. DELETE

   Delete will obtain the delimeters from the command argument data string and remove all references to the text as specified by the delimeters. Note: There are no valid exceptions for the failure of the delete statement.

2. DISPLAY

   Display will obtain the delimeters from the command argument data string and display all references to the text as specified by the delimeters. When expiration of the delimeters occur and/or no data is available for displaying, a message will return notifying the user of the condition. Note: The key-not-found condition will be used to determine the expiration of the delimeters.

c. Control Commands

There are two control commands provided.

1. PREFIX

   The PREFIX command provides for the setting of the filter prefix for each line to be added or replaced.

   Note: The prefix code in the command argument data string will be ignored by all other command processor routines. Once the prefix command is issued, it will remain in

affect for all ADDs and RSPs commands until a subsequent prefix command alters the setting of the filter prefix.

2. END

An orderly close of all files will occur and the program will terminate.

F. CODING SPECIFICATIONS

1. Source Language

Not Applicable

2. Suggestions and Techniques

Not Applicable

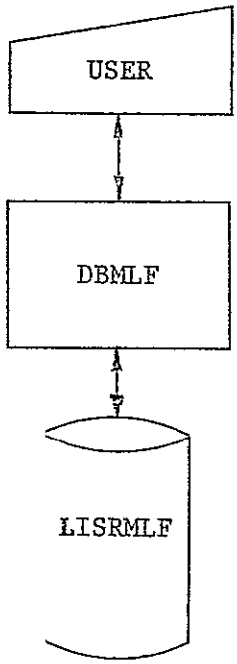


Figure 1. I/O Block Diagram

IV C.2.1

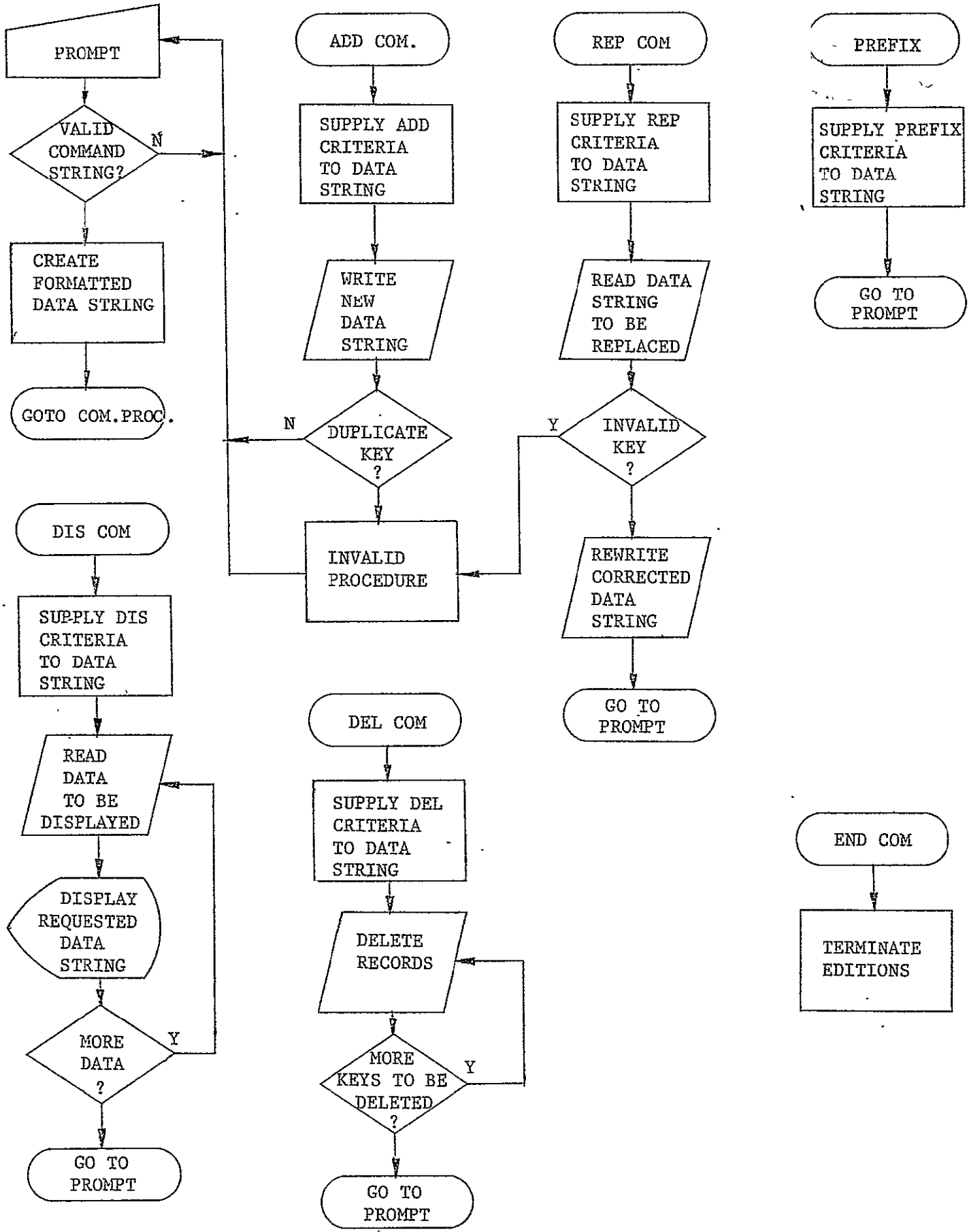


Figure 2. Top Level Flowchart

IV C 2.2

TOPIC C.3 - CONVERSION AND FORMATTING ROUTINE TEST DRIVER

A. MODULE NAME

Conversion and Formatting Routine Test Driver
Program-ID - RDBDRIVE
Module-ID - DBDRIVE

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

RDBDRIVE is a facility to allow the application programmers to test conversion, validation and reformatting routines conversationally. The user can specify the routine names and input data values to simulate the activities of RDBPAC.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

The user is prompted for all the input data required.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

All input data is displayed to the user for verification in two forms; in the form as entered, and after any necessary conversion. The output from each routine is displayed in hexadecimal, and the output from reformatting routines is also displayed in its character form.

C. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The user is prompted to enter the input mode, this is the mode which the character string he enters at his terminal will be converted into before inputing it to his selected routine(s). The possible input modes are:

a = alphanumeric,
f = full word,
h = half word,
p = packed decimal,
l = long floating point,
s = short floating point,
x = hexadecimal.

A null response to this prompt is the only way out of the module. Null responses to any other prompt will eventually filter back to this prompt.

The type of input mode selected determines the setting of a label subscript which is later used to branch to the routine necessary to convert the input data to the selected mode.

The user is next prompted for the routine names. They must be entered in one string, separated by commas, and must not exceed eight characters. A null response here returns to the prompt for input mode. Also, the routine names must be entered in the order; conversion, validation, reformatting, and any missing routines must be defaulted; eg:

```
DBCVTSN,,DBFMTSN
or
,,DBFMTHX
```

If a validation routine has been specified, the user is prompted for the validation arguments. These can be any character string, up to a maximum of 50 characters; or null.

The user is now prompted for input data. A null response here returns to the prompt for routine names. The input data is converted to the mode selected by the user.

The data is now passed to all the routines specified, in the order; conversion, validation, reformatting. The output from one routine is used as the input to the next routine.

The output from each routine is displayed in hexadecimal and the output from the reformatting routine is also displayed in character.

Successful completion returns the user to the prompt for data. Any error results in a diagnostic message and the return to the prompt for data.

F. CODING SPECIFICATIONS

1. Source Language

This module is coded in the IBM/360 PLI language. The TSPL/I Language Extension is used for all terminal I/O.

2. Suggestions and Techniques

Not Applicable

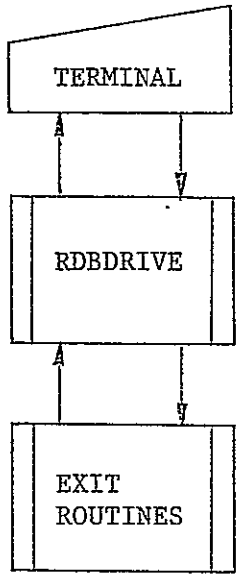


Figure 1. I/O Block Diagram

IV e.3-1.

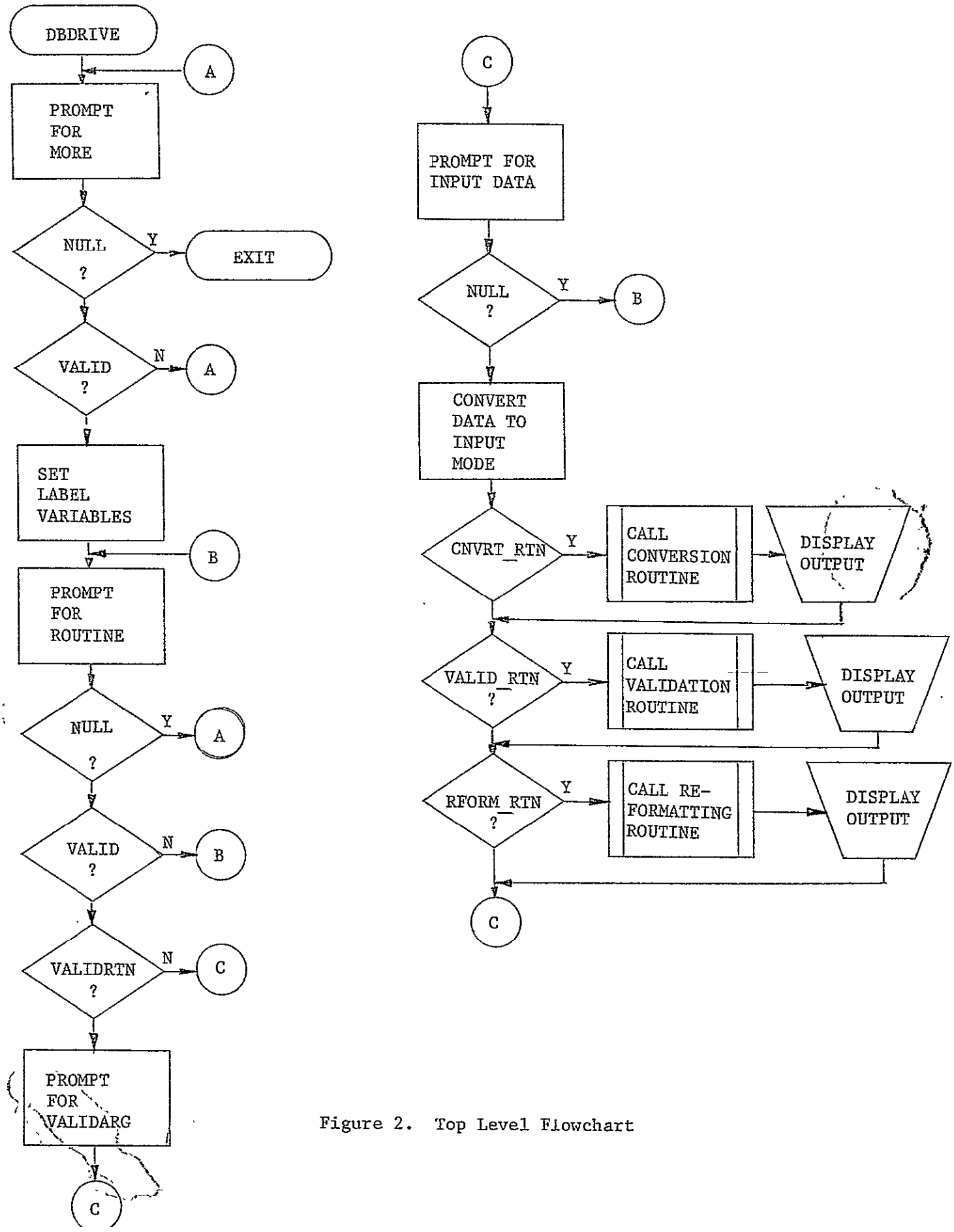


Figure 2. Top Level Flowchart

IV C3 -2

TOPIC C.4 - UTILITIES RUSERID

A. MODULE NAME

Get the TSS USERID
Program-ID - RUSERID
Module-ID - USERID

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This program is used to obtain the TSS userid and save its value in the TSS profile as a default for the symbol USERID. This symbol is then used during USERJOIN to create the transaction and statistics files.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1

2. Narrative

Upon entry, normal TSS linkage requirements are fulfilled. Next, the program uses the XTRCT macro to obtain the userid of the task being executed. The trailing asterisks are removed from the userid and the resultant value is assigned to the default symbol USERID and posted in the user's TSS profile by means of the OBEY macro. The program then returns to its caller.

F. CODING SPECIFICATIONS

1. Source Language

Because of its use of TSS system functions, this module is coded in the TSS/360 assembler language.

2. Suggestions and Techniques

Not Applicable

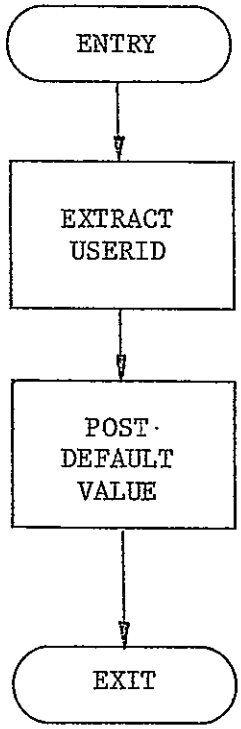


Figure 1. Top level flowchart

$\overrightarrow{IV}_1$ . . . C4.1

TOPIC D.1 - MAINTENANCE TRANSACTION MERGE

A. MODULE NAME

Maintenance - Transaction Merge
Program-ID - RDBMERGE
Module-ID - DBMERGE

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

This module is responsible for taking the contents of the varicus userids' transaction data bases and merging all of the transactions affecting a particular data base into the transaction data base of the data base owner. The resultant combined set of transactions is then processed by Maintenance itself.

The input transaction data basees are shared with read/write access to the data base owner to enable him to delete each transaction as he copies it over.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The two possible sources of input for this program are, (1) the VISAM dataset which contains the list of TSS userids which have been joined to the NASIS system, and (2) the transaction data bases of the NASIS system users.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The only output file used by the program is the transaction data base of the owner of the data base whose transactions are being merged.

b. On-line Terminal Displays

Standard prompting messages are directed to SYSOUT, as are all error messages written by the program.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

A special table is used by the program to control the input files to be processed. This table is built by the program at initial entry from the special input file containing the names of all TSS userids which have been joined to the system.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Initialize

The routine opens the dataset containing the list of TSS userids joined to the system. This dataset is read sequentially and each entry is placed in the name table for future reference. When the end-of-file is sensed for this dataset, it is closed and normal execution continued.

The final function of this routine is to open the transaction data base of this userid for

direct output.

c. Open Input

This routine uses the data base name and the next entry in the name table to construct the name of a transaction data base, which it then attempts to open for direct update. Any errors cause a diagnostic to be written and a bypass of that file. The routine then constructs the lowest value key possible for that userid and the specified data base and attempts to read that record.

d. Copy Record

This routine performs a sequential read of the input transaction data base. The record key of the transaction read is examined to ensure that it applies to the data base specified. If not, control is passed to the Close Input routine. Once the transaction has been validated it is written to the data base owner's transaction data base and then deleted from the input data base. Control is then passed to the beginning of this routine for the next record. Any DBPAC errors that are encountered, except end-of-file on the input data base, cause program termination, following an appropriate diagnostic message.

e. Close Input

This routine closes the input transaction data base. If the entries in the name list have not been exhausted, control is passed back to the Open Input routine to process the next user's transactions. Otherwise, control flows to the next section.

f. End-of-Job

This routine closes the owner's transaction data base. It then returns to the calling module.

F. CODING SPECIFICATIONS

1. Source Language

The Merge program employs the IBM PL/I programming language. The special extensions of that

language, called DBPL/I and TSPL/I, are utilitzed for all access to files in the data base and for all terminal communication, respectively.

2. Suggestions and Techniques

Not Applicable

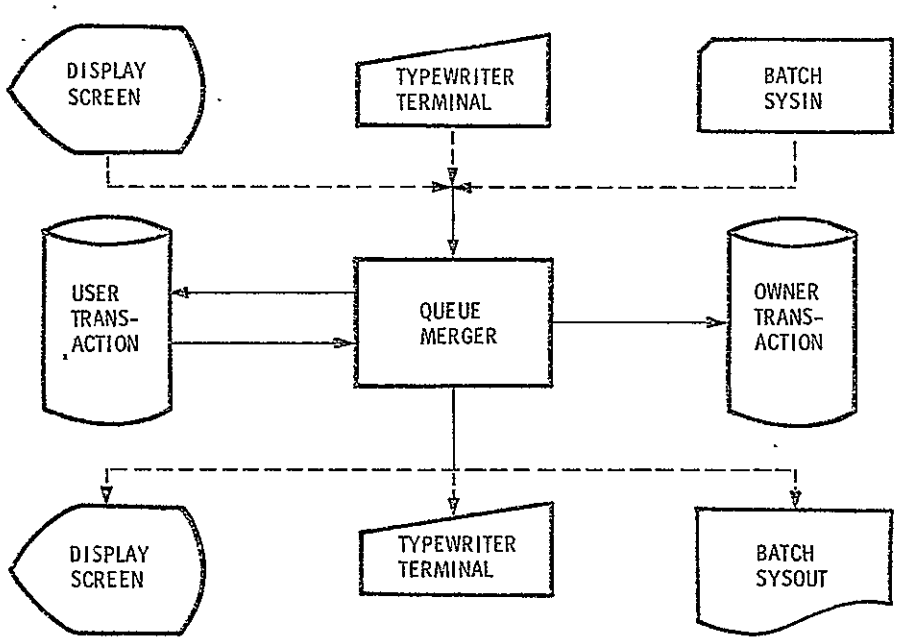


Figure 1. I/O Block diagram

IV. D1.1

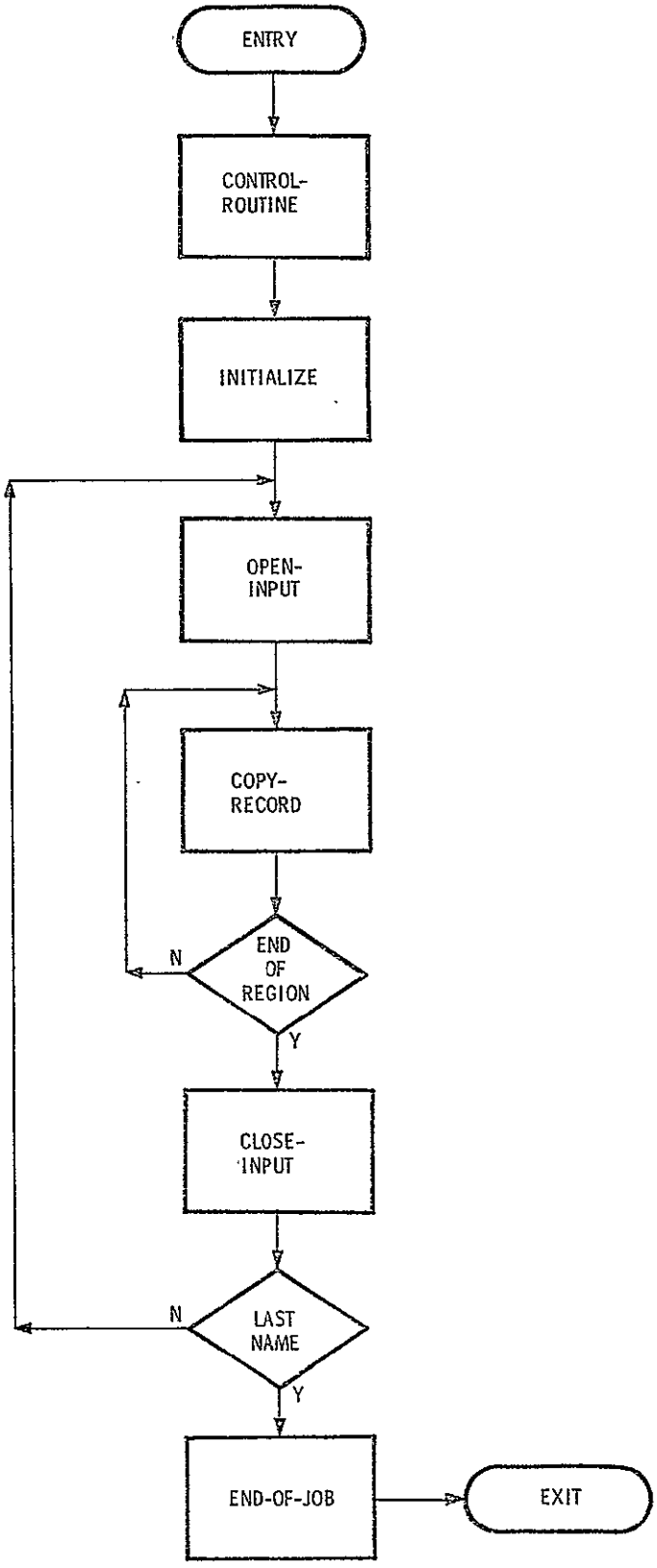


Figure 2. Top level flowchart

IV D1.2

TOPIC D.2 - MAINTENANCE MAINLINE

A. MODULE NAME

Maintenance Mainline
Program-ID - RDBMNTN
Module-ID - DBMNTN

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The Maintenance Mainline program is an independent module which carries out any actual changes necessary to correct, update, or expand the files comprising a data base. The specific changes, which can be additions, deletions, or replacements, are accepted by Maintenance in the form of transactions. The transactions are kept on a data base named 'TRNSCT' and are created and maintained by the CORRECT command.

The transactions can be applied to the data base on a record, field, or element basis. Those transactions which are successfully applied to the data base are deleted. Therefore, after the successful completion of a maintenance run, the only transactions remaining on the 'TRNSCT' data base are those which need correcting. The Maintenance Mainline acquires the necessary statistics while executing and causes the 'STATIC' data base to be updated (via a call to RDBUPDST). The Maintenance Mainline is run only in background or batch mode. The restart capability of the maintenance run is inherant because of the deleting transactions as they are applied and because the statistics are updated after the successful processing of each transaction record.

The Maintenance Mainline then has external interfaces with modules of the usage statistics. The RDBUPDST module is called after the successful processing of each transaction in order to update the maintenance statistics.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

While the Maintenance Mainline is normally invoked from a terminal and, therefore, has no punched card input, it is also possible to initiate the task in the non-conversational mode. The maintenance is always a batch task. The Maintenance User's Guide describes completely the procedure for invokation.

c. Input Files

The maintenance program requires all of the files which make up a data base as input to the module.

The files in a data base are the source of the old or current data for maintenance, the transaction data base (TRNSCT) is the source of the new or replacement data (i.e., the changes). The complete description of the transaction queue is found in the dataset specifications. The transaction data base (TRNSCT) contains information concerning the data base, file, record, field and element to be maintained, as well as the type of maintenance and the new data.

d. On-Line Terminal Entries

There is basically only one terminal entry to the maintenance routine and that is the command entered to initiate the program. The complete explanation of this procedure is available in the Maintenance User's Guide.

3. Output Data Sets

a. Output Files

All of the files of a data base may be used as output files for maintenance. As in the case where the files of a data base are used for input, the individual data files are output files only if specific transactions require them.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Since DBPL/I is used extensively in this module, the various combinations of DBPAC errors should be handled properly. These are in an array to determine program processing after error occurs.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. RDBMNTN(DBMNTN-entry point)

The Maintenance Mainline program is an maintenance module which carries out changes to the files comprising a data base. The program receives directives to modify a data base file or files from the maintenance transaction data base (TRNSCT).

b. Initialization

A table of the fields which are inverted is created from FLDTAB. The key fields descriptor is read and upon finding it, the key field's length is saved.

If any errors are incurred while reading the descriptor file, the proper message is emitted and the run terminated. If not, then the use of the descriptor file is at an end and it is closed.

It is now time to initialize the transaction file by opening it, positioning it and making the first record to be processed available.

An error on opening of the transaction data base could mean that there is no data on the 'TRNSCT' data base, or that the 'TRNSCT' dataplex is already opened for update or output. In either case, appropriate error messages are issued and the run is terminated.

To position the data base (after opening), we do a Read by Key NOLOCK. The key we create consists of the data base name concatenated with the owners-ID contatenated with all bits off. This should represent a low key value. This yields either a successful read or a DBPAC error of 108. We expect the error to occur. Then a sequential read is performed and we obtain the first transaction to be processed. Before continuing a get field is executed on the key and its contents are checked. If the key does not represent the proper data base name, owner-ID combination an error message is emitted and the run is terminated.

Otherwise, we are prepared for the final stage of initialization.

The regular transaction data base (TRNSCT) routine is set, the data base which is being updated has its error routine set and it is opened for direct update or sequential output.

The initialization process is complete.

c. Updating Statistics

The rules for gathering and keeping the statistics for maintenance are as follows:

1. Add transactions with a field name equal to the anchor files (TRANCNEW).

2. Add transactions with a field name of '*NEW SUB*' will post the new subfile record count for subfiles (TRSUBNEW).

3. Delete transactions with a fieldname equal to the anchor key name will post the delete record count for the anchor files (TRANCDEL).

4. Delete transactions with a subkey field

will post the delete subfile record count (TRSUBDEL).

5. Add transactions with a field name other than the key field name will post the update count for the anchor file (TRANCUPD).

6. Add transactions with a field name other than the key field name and a SUBKEY field will post the update count for the subfile file (TRSUBUPD).

7. Change transactions will post the update count for the anchor file (TRANCUPD).

8. Change transactions with a SUBKEY field will post the subfile count for the subfile file (TRSUBUPD).

9. If the filed is inverted then appropriate index file count will be posted.

These statistics are accumulated only if the transaction was successfully used to update the data base. The statistics data base is updated after each transaction has been processed. This means that if the system crashes or otherwise fails during execution of a given maintenance run, the statistics will be correct. Since each transaction (after it has successfully updated the data base) is deleted from the TRNSCT data base, then the restart procedure is automatic. The call to the module to update the statistics is as follows:

```
CALL DBUPDST(UPDTFLAG,DATAPLEX,
     TRANCNEW,TRANCDEL,TRANCUPD,
     TRSUBNEW,TRSUBDEL,TRSUBUPD,
     TRINVNEW,TRINVDEL,TRINVUPD);
```

If the statistics are updated successfully, the DBUPDST module returns a 'G' in UPDTFLAG. If the statistics are NOT updated successfully, the DBUPDST module returns a 'B' in the UPDTFLAG.

d. Delete the Transaction

If the transaction is successful, it is now deleted from the TRNSCT data base.

e. Read Transaction

The transaction file is a data base which consists of only an anchor file and no associated or inverted files. The transactions are read sequentially. The Data Base Executive performs all of the necessary I/O operations. After a transaction record is located by the Data Base Executive, GETS are executed on all of the desirable fields. These fields are disseminated to various work areas. Then, checking is performed based on the presence and/or absence of data. The validation of this data is based upon the following:

See Figure 3

If there is an error detected during this initial processing, then the transaction is in error.

f. E.O.D. (end-of-data)

The end-of-data is only detected on the transaction data base. When this condition is detected, all the files are closed, appropriate messages are issued and the processing continues at the reset the switches, section (g).

g. Reset the Switches

This section of code is executed antecedent to the occurence of an end-of-data condition. The files of the data base are manipulated to detect the existence or non-existence of data and the 'DATA' switches of the corresponding files are set accordingly.

h. DEL_RTN: Delete field routine.

This routine uses the #FIELD function to reput all the elements in the field to null. If it is the key field, then the entire record is deleted.

i. ADD_RTN: Add routine.

This is the add record and add element routine. If the field name is the key field then this name is stored to indicate to the

maintenance routine that a new record is to be added to the file. If the field is not the key field, then a test is made to see if the transaction key is already present. If not, then the key is compared to the stored key from the last add transaction with a key field. If they do not match, an error has occurred and is flagged; otherwise, a record is created with the stored key. The new element is then put to the record. Control is passed back to section (e) on completion of this transaction.

NOTE: If subfile key is present in transaction then subfile record is obtained. If SUBCTI field is present then new subfile record is located.

j. CHG_RTN: Change Routine.

If no start or end field, given element is replaced. Using the key passed in the transaction record, the appropriate record is read in from the data base. At that point, the value of the returned field element is compared to the 'old' data element in the transaction. If no match is made, a test is made as to whether the returned element is null, thereby signifying the end of the field. If that is the case, then an error has occurred and is indicated. If the null element was not detected, then the next element is obtained and the process repeated. If a match does occur, then the 'new' data element from the transaction record is reput to the record. If the 'new' data element is null, then the element is deleted. Continue processing with section (e). If a start and end field is present then a field context operation is performed.

The maintenance program can carry out changes to portions of large fields without the entire field on the transaction entry record. To begin, the record is read into a large enough area to hold the maximum record using the key provided in the transaction. The field in question will then be obtained and an interactive process is applied wherein the 'old' data value is compared sequentially across the field from the starting location to the ending location. Whenever a match is

found, the 'new' data value is used to replace the 'old' in the field and a count is kept of the number of replacements. When the end of the search range is reached, the count is tested. If no matches were made, then that error is recorded. The processing will continue with section (e).

F. CODING SPECIFICATIONS

1. Source Language

As much as possible of the Maintenance Module is coded in the IBM programming language PL/I. The input and output coding for access to files in a data base is handled through an extension to that language, known as DBPL/I. Where indicated in the narrative, it was necessary to use the assembler language for IBM System/360 in order to interface with the command processor and the DDEF instructions. All terminal communication is handled through the terminal support preprocessor, TSPL/I.

2. Suggestions and Techniques

a. Much of the verification of correct access to files in a data base is handled within the DBPAC routines.

b. During implementation, all appropriate messages are included to increase the understanding of the user.

c. While not noted in the narrative, it is necessary to test the return codes from every input and output operation. In those cases where errors occur, messages are written out and the task terminated unless a correction can be applied, in which case the processing can then continue.

d. Whenever it becomes necessary to terminate the maintenance routine at any point, it is desirable to make every attempt to restore the data base to a normal condition. In most cases, this action involves resetting control switches found in the header records of the descriptor file. This action makes possible subsequent processing on the data base which might correct the original problem and also allows continued retrieval from all usable portions of the data base.

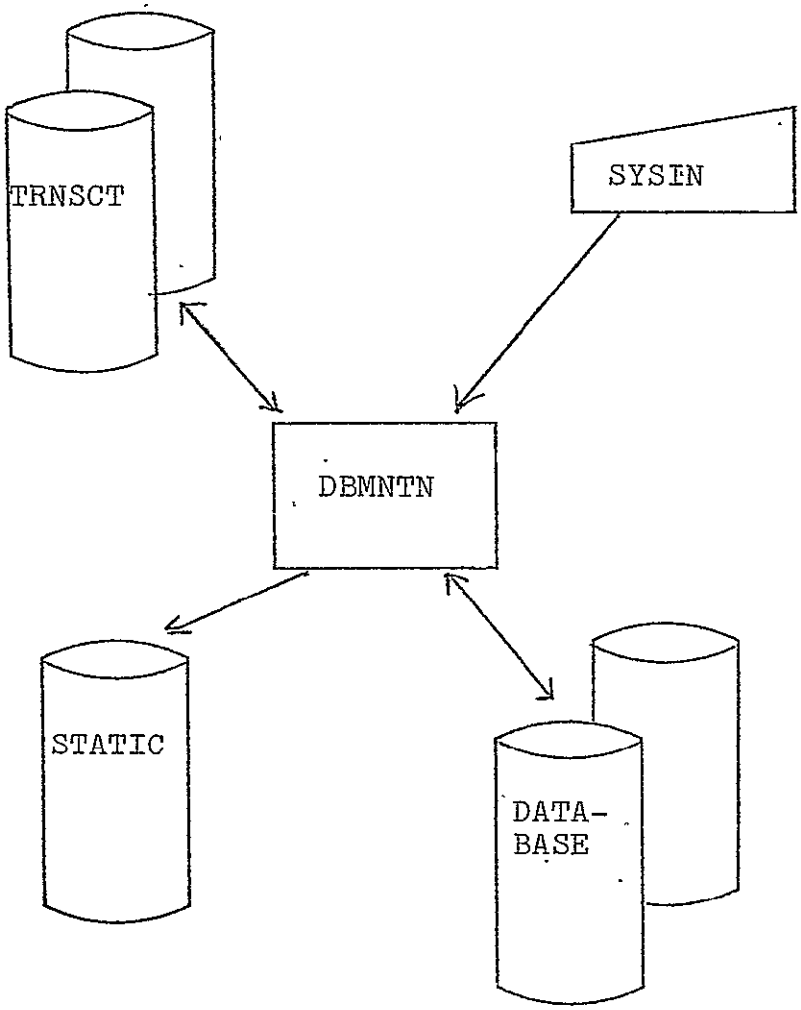


FIGURE 1 I/O BLOCK DIAGRAM

IV 52.1

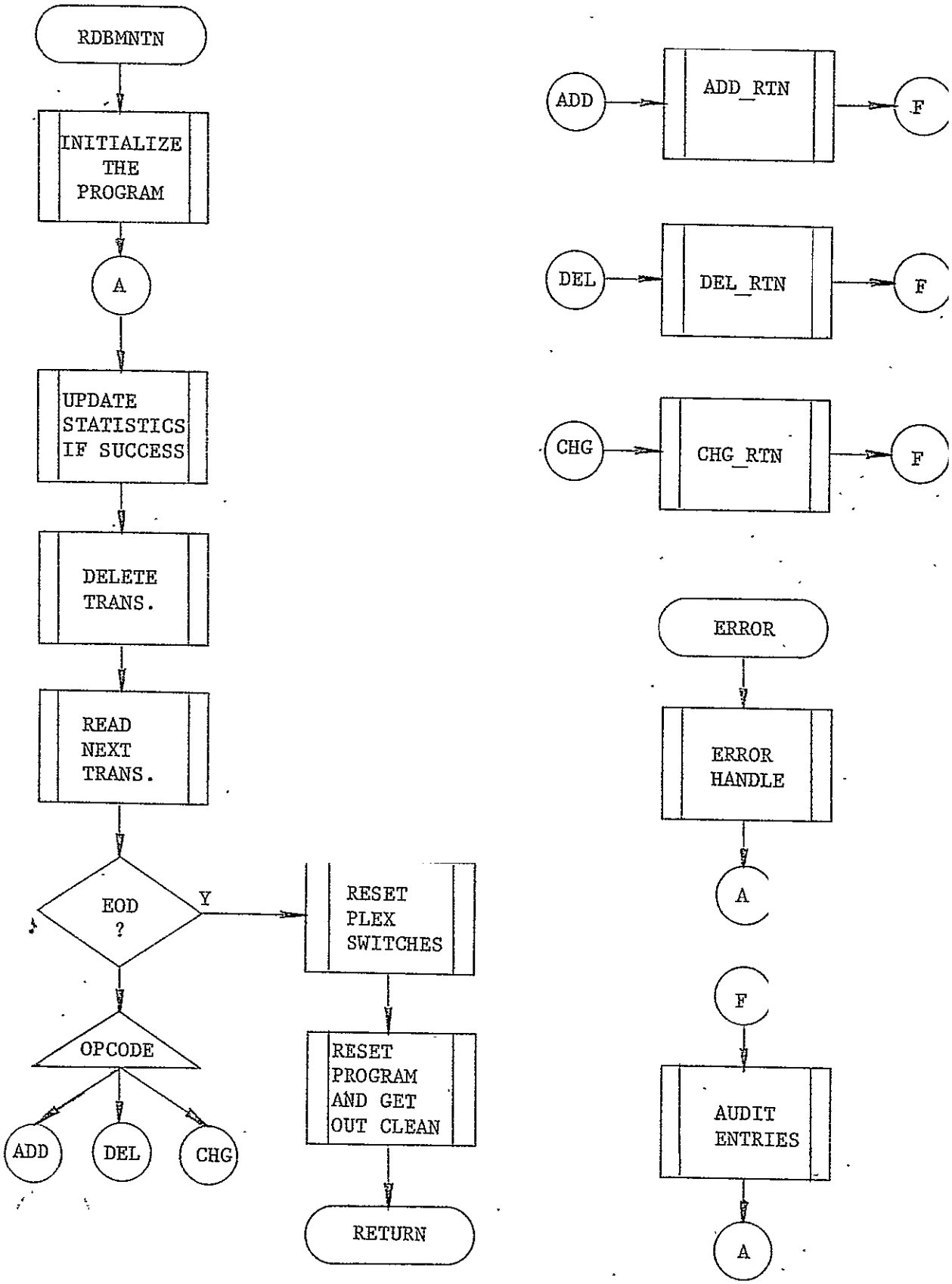


Figure 2. Top level flowchart

IV. 52.2

√ : REQUIRED PARAMETER

X : NOT REQUIRED

	ADD	DEL	CHG
KEY	√	√	√
NASISID	√	√	√
OPCODE	√	/	√
FIELD	X	√	√
START AND END	X	X	X
OLDDATA	X	√X	√
NEWDATA	√X	X	√
SUBKEY	√X	√X	√X
CTLFLD	√X	√X	X

Figure 3. Parameter table

IV D.2.3

TOPIC D.3 - MAINTENANCE CORRECT COMMAND

A. MODULE NAME

Maintenance, CORRECT Command
Retrieval, CORRECT Command
Program-ID - RDBCORR
Module-ID - DBCORR
Entry Point (DBCORR) MAINTENANCE
(DBCORR1) RETRIEVAL

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The CORRECT command is a routine, called by the RETRIEVAL system, whose purpose is to allow the retrieval system user to create certain maintenance transactions during retrieval. When a user observes an error during a display, he is able to have any or all of the fields of a given record displayed and then be able to specify any deletions, additions, or changes to those fields. The transactions created are not executed, but are placed in a transaction data base which is examined by the data base owner before the actual maintenance takes place. The calling sequence is: CORRECT field,key.

The CORRECT command, when invoked through the maintenance sub-system, performs on-line creation and updating of files.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

The parameters available to the CORRECT command are "key" and "fieldname". The Terminal Support system applies default values to the parameters, if they are available, when no original values are entered.

Additional terminal entries are requested of the user. These responses indicate what alterations, if any, to the field are desired. These entries take the form of sub-commands available to the user while running under control of CORRECT. The sub-commands are:

```
ADD data
CANCEL
CORRECT field,<key>
DELETE element
DISPLAY
END
FIELDS
INSERT field,...
REPLACE element1,<element>,olddata<,newdata>
VERIFY
```

3. Output Data Sets

a. Output Files

The only output file from the CORRECT command is the transaction data base. This file is a VISAM data set containing maintenance transactions from all sources for all data bases. The fields of the transaction data base and their format are completely described in the Dataset Specifications.

b. On-line Terminal Displays

The CORRECT command outputs a formatted display of the specified field on the terminal. Each field to be processed begins on a new screen image with appropriate header information. Each element of multi-element fields begins on a new line. No attempt is made to end lines of the display on word boundaries. In addition to the display of the field in question, a prompting message

requesting the action to be taken is issued in the input area of the screen.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROGRAM REQUIREMENTS

1. Flowchart

See Figure 2

2. Narrative

a. CORRECT

The CORRECT command is called by the maintenance sub-system at entry point DBCORR. The Retrieval subsystem calls CORRECT at the entry point DBCORR1. Any default parameters which are applicable are supplied by terminal support.

b. Real Entry

This routine initializes the routines for handling the exceptional error and interrupt conditions. Attention interrupts cause the user to be prompted for a decision. If he defaults, execution continues from the point of interruption. Terminal support prompting errors cause program termination, unless the error is for input transaction, in which case, a warning message is issued to the user and execution continued. Any other errors cause program termination, following an appropriate diagnostic message.

c. Main Line

The routine allocates the screen buffers, if not already done, and obtains the julian date for time stamping the transactions. The current retrieval data base is then opened

for input, unless that data base is the user's transaction data base, in which case, it is opened for update.

d. Get Record

If necessary, this routine reads a new record from the input data base and gets the value of the key field. Again, if necessary, the routine reads in the values for all elements of the field, maintaining a count of the number of elements processed. Finally, the routine copies the input data to a temporary storage area for the user to process against.

e. Format Screen

This routine, unless running from a typewriter with the verify option equal to no, formats and displays the status of the data most recently referenced by him. It first constructs a heading, composed of the record key, data base name, field name and element count. It then proceeds to fill the remainder of the screen with data beginning at the element indicated by the calling routine. If the element length is less than the length of the data portion of the line, the element is written on a single line preceeded by the element number. If the element is too large, the first line is processed as above, but the remaining data is split across succeeding lines.

f. Re-prompt

This routine prompts the user for his next request. It extracts the command keyword, and if valid, calls the appropriate calling routine. If any type of error is encountered, the routine re-prompts for the correct information.

g. Add Routine

This routine takes the input data and uses it to create a new element for the field being processed. If FIELD is key field, then new record is created. If there is no data entered, or if the maximum allowable number of elements has been reached, a diagnostic is written and processing bypassed. After

processing, that data control is passed to Format Screen to display the updated data.

h. Replace Routine

This routine expects four parameters to be entered, a starting point, ending point, old data value and a new data value. The starting and ending points are expressed in terms of element numbers. If the element numbers are invalid, or if no old data value is entered, a diagnostic is written and processing bypassed. The data is then searched, character by character, from the starting point to the ending point. If any occurrence of the old data value is found, it is replaced by the new data value. If no occurrence of the old data value was found, a diagnostic is written. Otherwise, control is passed to Format Screen to display the updated data.

i. Cancel Routine

This routine re-initializes the data in the field to its initial status when read from the data base. Control is then passed to Format Screen.

j. Page Routine

This routine expects one parameter, which it uses to adjust the current element pointers to adjust the segment of the data which is displayed on the screen. The parameter may be a default for forward paging, a 'B' for backward paging, a number for a specific element number. If the data is invalid or the request cannot be honored, a diagnostic is written and processing bypassed. Otherwise, control is transferred to Format Screen to display the data.

k. Verify Routine

This routine sets the switch that determines whether the user, operating from a typewriter, receives a verification display of the data following each command. The data entered should be 'YES' for verification or 'NO' for none. If the data is invalid, a diagnostic is written and processing bypassed. Otherwise, control passes to

Format Screen.

l. Delete Routine

If the field is to be deleted, it is done and control passed to Format Screen. If the record is to be deleted, it is done and control passed to End Routine. If elements are to be deleted, the routine will accept a list of elements or element ranges as input. For each, it analyses the element number to determine its validity. An invalid element will cause a diagnostic to be written and further processing bypassed.

m. Insert Routine

This routine allows the user to specify the fields of subfile records to be inserted into the data base. If no field is specified, a diagnostic is written and processing is bypassed. If the previous field's data has been changed, Output Routine is called to create the necessary transactions. Control is then passed to the Correct Routine for further processing.

n. Correct Routine

This routine allows the user to specify the key of a new record to be processed, the name of the next field to be processed, or both. If the previous field's data has been changed, Output Routine is called to create the necessary transactions. The routine first checks for a signed numeric value in the key operand, and if found, reads the file sequentially forward or backward to the desired record. If sequential processing is not indicated, the routine extracts the new key and the new field name, if present, and transfers control to Get Record for further processing.

o. Fields Routine - CALL DBFLDS

This routine displays a list of the field names for the data base for the user. It calls DBFLDS to extract the field names. It also checks each field until it has identified the key field, whose name it maintains separately. It then moves the field names into the output area, fitting as

many as possible on each line, and displays them to the user. If more names exist than may be displayed on the screen at once, the routine prompts the user for a decision as to whether he wants to see the remaining names or to continue correcting.

p. Output Routine

This routine analyses the data maintained for the field being processed, and for each element whose data has been changed, creates transactions to represent the change. The routine calls Write Tranplx to actually write the transactions. The routine handles three cases, an added element, deleted element and a changed element. Upon completion, the routine returns to its caller.

q. Write Tranplx

This routine performs the actual creation of transactions, based upon the data supplied to it. If the user is correcting the transaction data base itself, the routine updates the data directly, otherwise it generates transactions. If any DBPAC errors are encountered, this routine calls End Routine; otherwise, it returns to its caller.

r. End Routine

This routine processes any transactions remaining to be written. It closes all of the files, resets switches, restores the NASIS status to what it was when the program was invoked and returns to the calling program.

F. CODING SPECIFICATIONS

1. Source Language

The correct command program employs the IBM PL/I programming language. The special extensions of that language, called DBPL/I and TSPL/I, are utilized for all access to files in the data base and for all terminal communication, respectively.

2. Suggestions and Techniques

Not Applicable

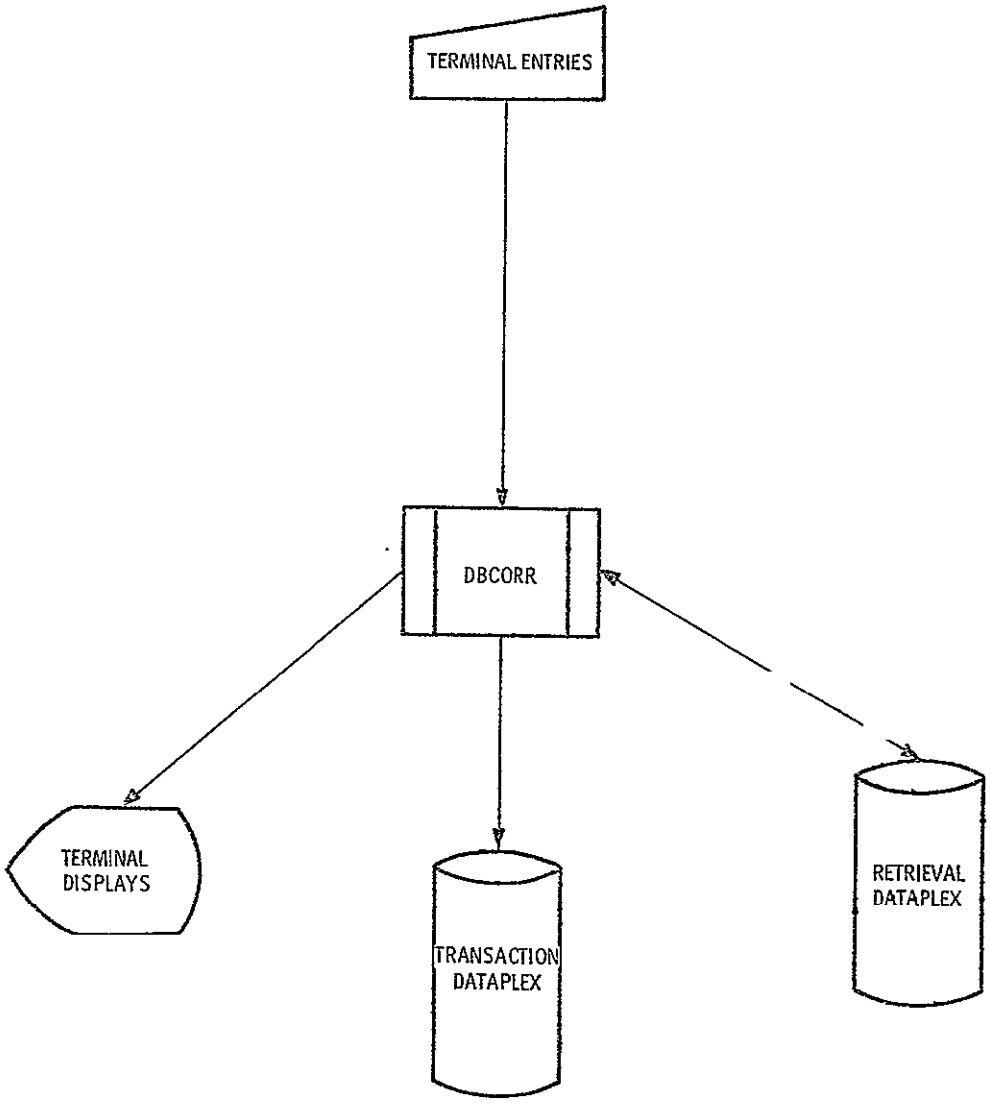


Figure 1. - I/O Block diagram

IV D3.1

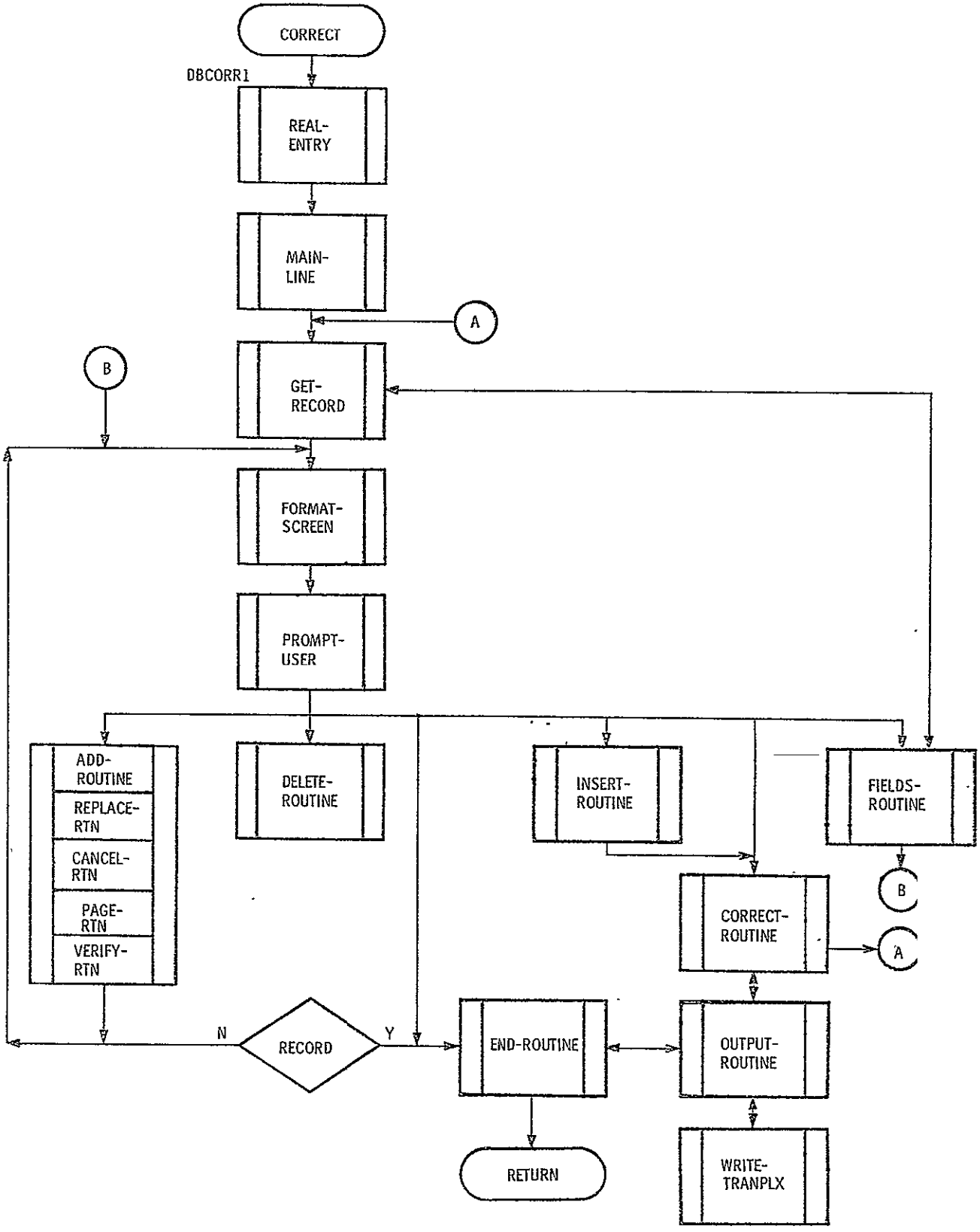


Figure 2. - Top level flowchart.

IV 53.2

TOPIC D.4 - MAINLINE MAINTENANCE INVOCATION

A. MODULE NAME

Mainline - Maintenance Invocation
Program-ID - RDBCLMN
Module-ID - DBCLMN

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The maintenance is always run in a batch mode. A data set must be created to execute. This program accomplishes the creation of the required data set.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The data base descriptor file (defined elsewhere).

3. Output Data Sets

a. Output Files

The output file is a line data set whose name is CLDBMAIN./PLEX$$/. Where PLEX$$ is the data base name. It is a VISAM data set RKP=4, LRECL=132, KEYLEN=7, RECFM=V. It contains the necessary TSS commands to run maintenance in batch.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

This module accepts the data base name as a parameter. The execution of this module proceeds as follows:

a. Write output line data set consisting of:

1. ERASE CLDBMAIN.'FILENAME'
2. LOGON OWNER-ID(TSS-ID)
3. CALL RDBMNTN,'FILENAME'
4. LOGOFF

b. Close files and quit.

F. CODING SPECIFICATIONS

1. Source Language

TSS PL/I.

2. Suggestions and Techniques

Not Applicable

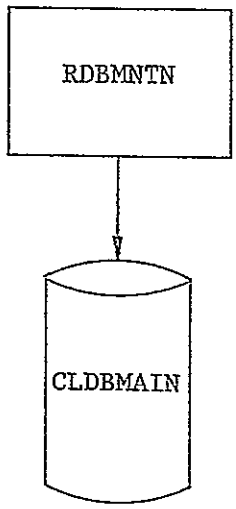


Figure 1. I/O Block Diagram

IV. DSN-1

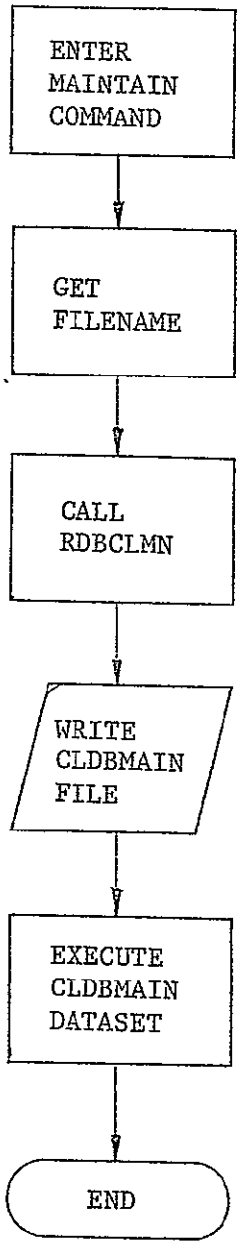


Figure 2. Top Level Flowchart

IV D4-2

TOPIC D.5 - MAINTENANCE DIRECTOR

A. MODULE NAME

Program-ID - RDBMAIN
Module-ID - DBMAIN

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module serves as the initializer and command director for the maintenance sub-system. It prompts the user for the file name and invokes the maintenance function requested.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The module opens the file specified by the user for update, for those modules which require it.

d. On-line terminal Entries.

The program initially prompts the user for the name of the file to be maintained. Subsequently, the program prompts the user for his maintenance commands.

3. Output Data Sets

a. Output Files

Nct Applicable

b. On-line Terminal Displays.

The program displays various diagnostic messages to the user.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Nct Applicable

4. Reference Tables

The following tables are referenced by the program,

USERTAB
VERETAE
FLDTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry, the program establishes control of attention interrupts and initializes terminal support. It then prompts for the file name. If a null value is returned, the program is terminated. Otherwise, the name is validated and the strategy is posted. If the name is invalid the user is reprompted for a valid file name.

The program then allocates and initializes its verb table, including any user defined commands. Next, the program prompts the user for a maintenance command. If the command is not valid a diagnostic message is issued and the user is reprompted for the maintenance command. Five successive invalid commands cause the program to be terminated.

If the command entered was CORRECT or UPDATE, the file specified is opened for update and FLDTAB is initialized. Otherwise, the file name is posted in the MFCB named PLEX. The entry point

specified for the command is then called to effect the processing requested. Upon return, the program prompts for the next maintenance command.

When an END command is entered, the program closes the file, releases VERBTAB and returns.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and techniques

Not Applicable

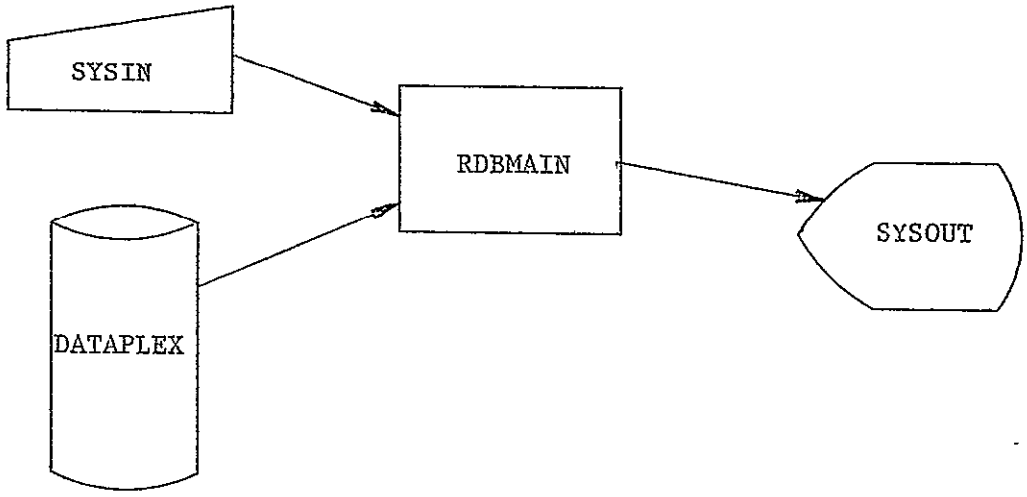


Figure 1. I/O Block Diagram

IV 55-1

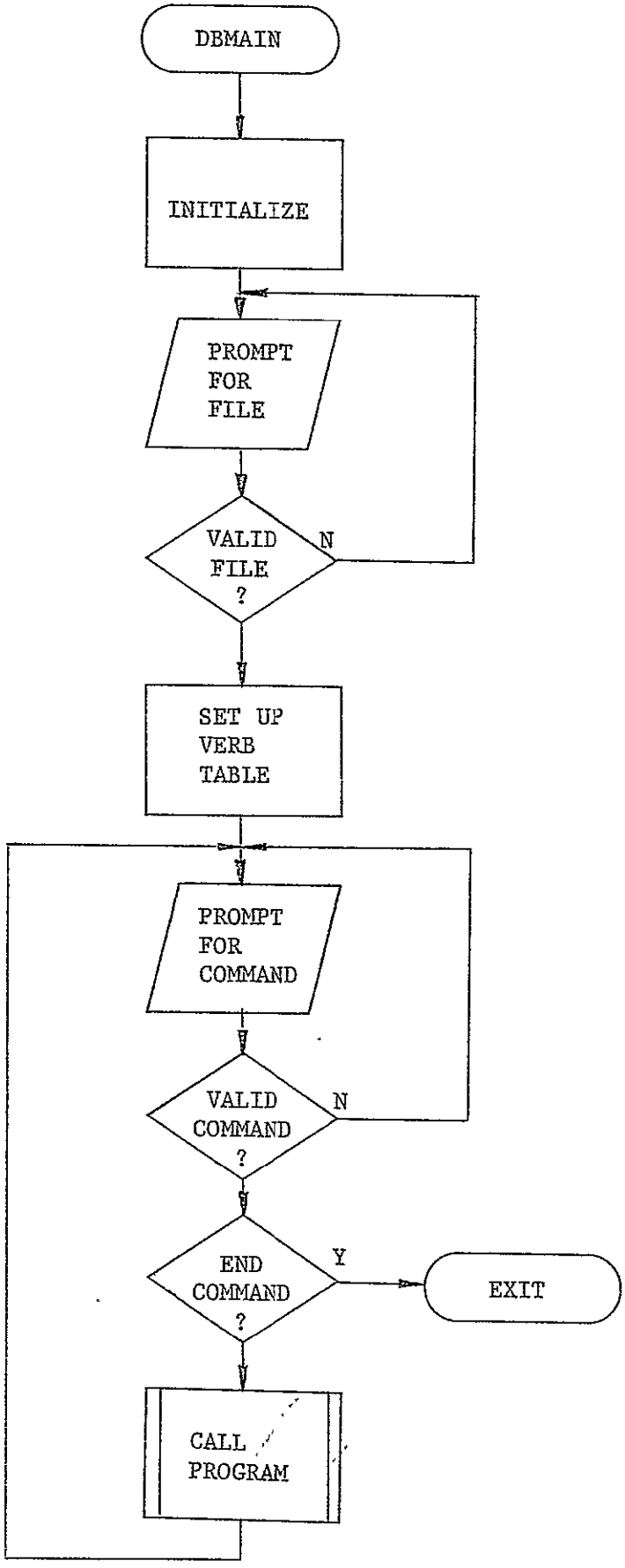


Figure 2. Top Level Flowchart - DBMAIN

IV DS-2

TOPIC D.6 - MAINTENANCE LOAD/CREATE

A. MODULE NAME

Load/Create
Program-ID - RDBLOAD
Module-ID - DBLOAD

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

This module provides a generalized file loading capability for NASIS.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

1. The primary input file is the data set containing the records to be loaded to the data base. This file may be indexed, with the keys having the same format and value as that of the final data base, or sequential.

2. The only other input file is the descriptor data set for the data base being loaded.

d. On-line Terminal Entries

The parameters required by the program must be entered with the command, or default values will be assumed.

3. Output Data Sets

   a. Output Files

      1. The primary output file is the data base which is being loaded.

      2. The other output file is the error file, on which is written exact duplicates of any input records that cannot be successfully loaded.

   b. On-line Terminal Displays

      The only output to the terminal is the informational and diagnostic messages that are displayed as the program executes.

   c. Formatted Print-outs

      Not Applicable

   d. Punched Card Output Files

      Not Applicable

4. Reference Tables

   The module contains a table of error switches which control the action to be taken for each possible DBPAC error; abend, skip record, or skip field.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   a. Upon entry the program establishes interrupt handling routines which will terminate if any PL/I errors occur, terminate if any PROMPT errors occur, or display statistical data and prompt the user if an attention interrupt occurs.

   b. The program next prompts for the input parameters applying defaults for any parameters that are not entered.

   c. The next step is to open the descriptors for

the file specified. The file header switches are reset in case the system crashed. The index file headers are read, and if the field is not set for inversion, the loadable switch is turned off.

d. The next function performed is the definition and opening of the input file, the error file and the data base itself. At this point, the program checks the user's mode parameter, and if it is restart passes control to section (g) before continuing.

e. Finally, the program is ready to process data. It reads an input record, passes the record to the user written exit routine for separation into its component fields. Upon return from the exit routine, the program tests the status bits, and if set properly, begins writing the input data to the data base, field by field. If any errors are detected, and appropriate diagnostic is written to the user and the action indicated by that error's code in the ERROR_CODE table is taken. The options are to abend the program, to skip the remainder of the record, or to skip the field. When the field has been completely processed, the routine continues with the next input record, until the data is exhausted.

f. When all of the data has been processed or when a terminal error has been detected, statistical counts are written on the user's terminal along with a termination message. All of the files are closed, and the status bits of the descriptor header records of each of the component files of the data base are posted to indicate whether date exists on the file or not. The program then terminates.

g. If the user specified a restart, the program retrieves the last record written to the data base. It then accesses the next record to be written from the input data set. Checkpoint backup is automatically done after every 1000 records are processed. When the operation is complete, processing continues with section (e).

F. CODING SPECIFICATIONS

1. Source Language

This module should be written in the TSS/360 PL/I Language.

2. Suggestions and Techniques

   a. Because of the function of this module, extreme care should be taken to code it as efficiently yet as indestructibly as possible.

   b. Any place in the program where there is any remote possibility of an error, there should be a meaningful diagnostic.

   c. The ERROR_CODES table was designed to be used in conjunction with a label array. The digits in the table are to be converted to index values and an indexed branch taken based onthe label array.

   d. The user-written exit routine is responsible for assigning field names, field off-sets, and field lengths.

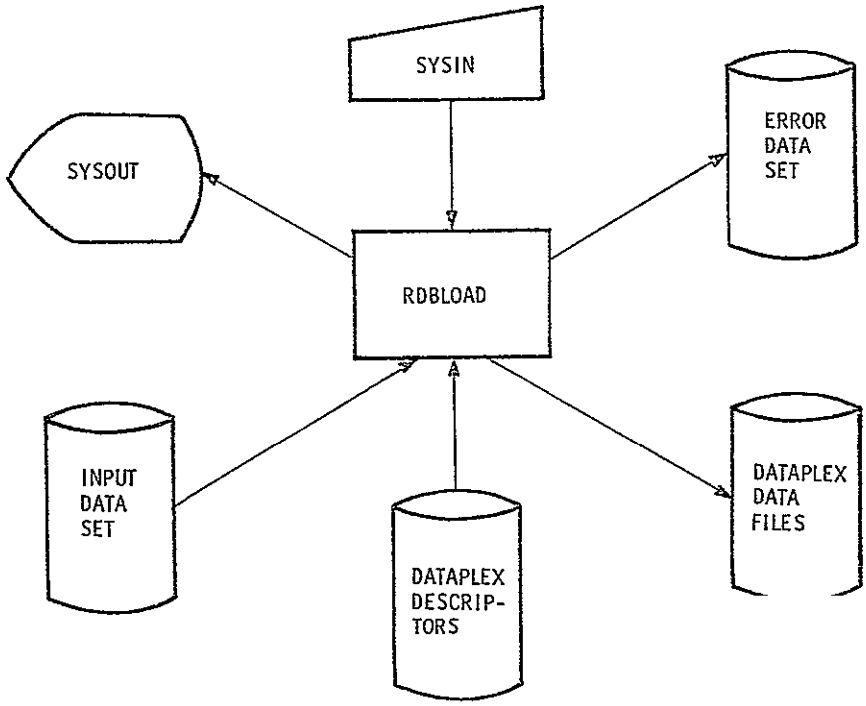


Figure 1. I/O Block diagram

IV. Dle. 1

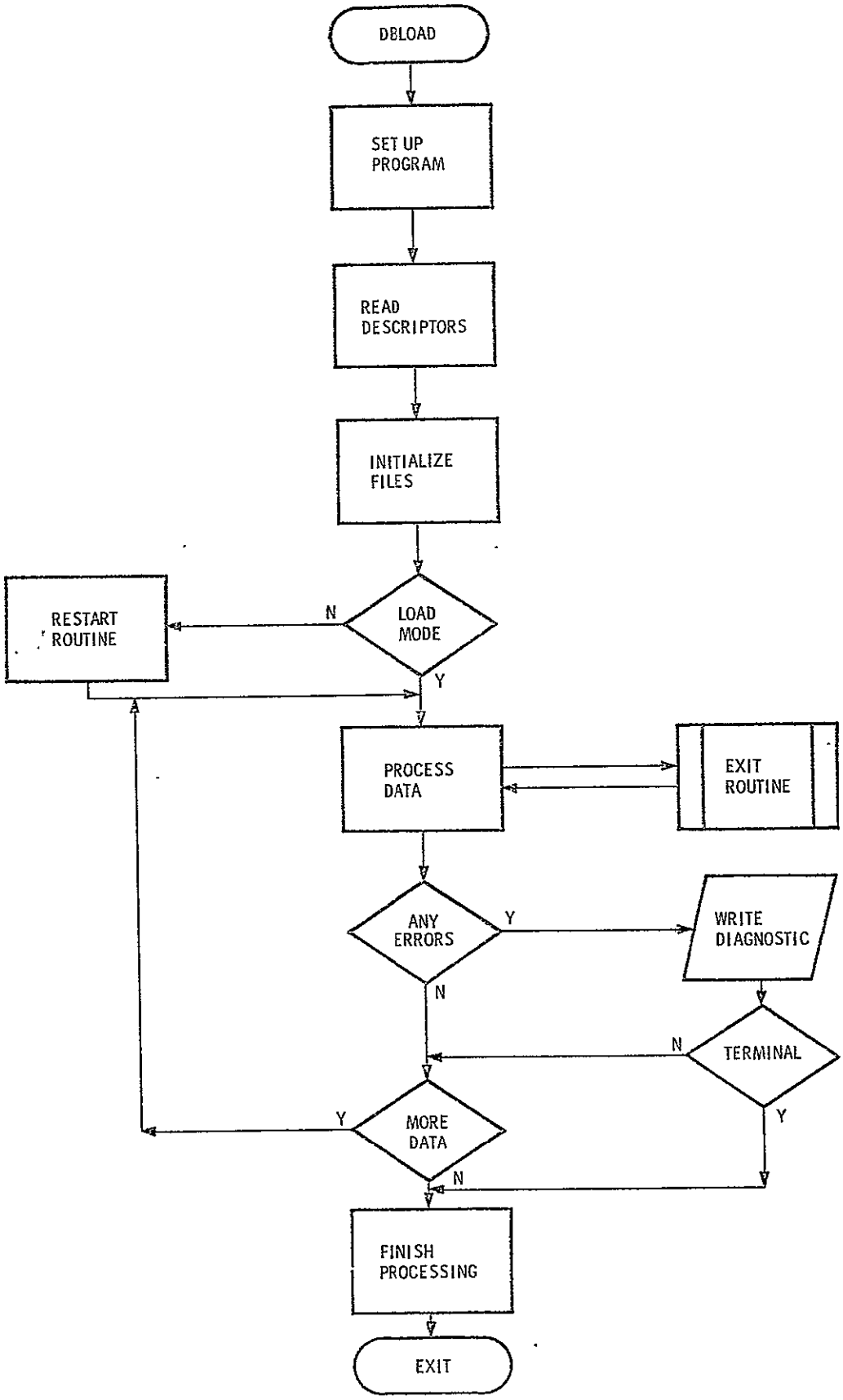


Figure 2. Top level flowchart

IV 1 ЛЬ6.2

TOPIC D.7 FILE INVERTER

A. MODULE NAME

Maintenance - File Inverter
Program-ID - RDBSIVRT
Module-ID - DBSIVRT

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

The purpose of the program is to take data from certain fields of a data base and to post this data to an inverted index file.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

1. Data base: The primary input to the Inversion module is the file being inverted.

2. Data base Descriptors: The file descriptors are needed to provide information.

3. Restart file: If the program is invoked in restart mode, a restart file with the restart key is needed.

d. On-Line Terminal Entries

All of the terminal entries to the Inversion

program, except one, are in the form of responses to prompting messages from the program itself. The exception is the entry of the initial command with its parameter to invoke the procedure. The purpose of terminal entries are to establish field names, to establish the mode of operation, to establish number of records to process, and to establish range of keys to process.

3. Output Data Sets

   a. Output Files

      1. Sortin File: This file is created by the first step and is a VSAM file with the value of the field being inverted concatenated with the file key. This file becomes the input for step two, the sort.

      2. Sortout File: This file is the sorted output from step two, the TSS sort utility. The file becomes the input to the third step.

      3. Plex File: This file is the output of step three in the form of an index file with the internal field value as the key. This file becomes the input to step four, the translation step.

      4. Range File: This file is the output of step three if field is indexed with internal format, and is the output of step four if field is indexed with external format. Range of keys to invert must have been specified for this file to be produced. This file becomes the input to the index merge program.

      5. Database Index File: This is the final index file. It is the output of step three if field is indexed with internal format, and is the output of step four, if field is indexed with external format.

   b. On-Line Terminal Displays

      All on-line terminal displays for the Inversion program follow the same format. The TSPL/I facility of the system is utilized

to request entires at a terminal and display progress information.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Parameter Prompting routine

Prompt the user for program parameters. Initialize program swtiches and tables. Read appropriate field descriptor for each field name, saving field length. If applicable, fill external indexing table, inverted suffix table, and spanned table. If field is indexed external, read index file header descriptor and get external field length. Read appropriate region descriptor and save the file key length and key name. Check the cperating mode and go to appropriate section.

b. Field stripping routine (step one)

Loop through fieldname table and DDEF the sortin files. DDEF the restart file. If restart run, read in restart key and read this file record. If range run, read first file key to start at. Read sequentially the input file, save the internal file key. Loop through the index file suffixes. Loop through the field name table. Loop through the #FIELD for this field. Write out a sortin file record. If end of file reached, go to next section. If end range key reached go to next section. If number of records to process is reached, write out restart file

C3

and terminate.

c. Sort step (step two)

DDEF the sortin file, sortout file and invoke the TSS sort utility.

d. Write VISAM file (step three)

Find the key length of the VISAM output file. This will be the maximum length of the fields being inverted on the same index file. If field is indexed external, use external length. If index file is spanned, increase by one. If external indexing, DDEF the output file as 'PLEX.', if range keys and no external indexing, DDEF output file as 'RANGE.' If not a range run, and no external indexing DDEF output file as final index DSNAME. Read input file. If keys are the same, concatenate file key onto list of keys. If list has reached maximum list length, up the span character and initialize list to null. If keys are different, write out index record. If end of input file reached, write out last index record and check to see if external indexing. If external indexing, proceed to next section. Display record counts for user and post the index file descriptor data bit.

e. Translate Keys routine

DDEF the input file and the output VISAM file. Read input file sequentially. Search Key for first blank character after first non-blank character. Use this parsed string to pass to field formatting routine. Replace internal value in key with external value. If end of file reached, post data bit in index header descriptor record. If more field names in table, go to sort step. Terminate the program.

F. CODING SPECIFICATIONS

1. Source Language

The Inversion program employs the IBM PL/I programming language. The special extensions of that language, called DBPL/I and TSPL/I, are utilized for all access to files in the data base and for all terminal communication,

respectively.

2. Suggestions and Techniques

The module can very easily be broken up into four separate modules, step one - strips the fields from file; step two - the TSS sort step; step three - the internal index file creating step; step four - the external or translation step.

This could be a major enhancement for core consideration and storage requirements.

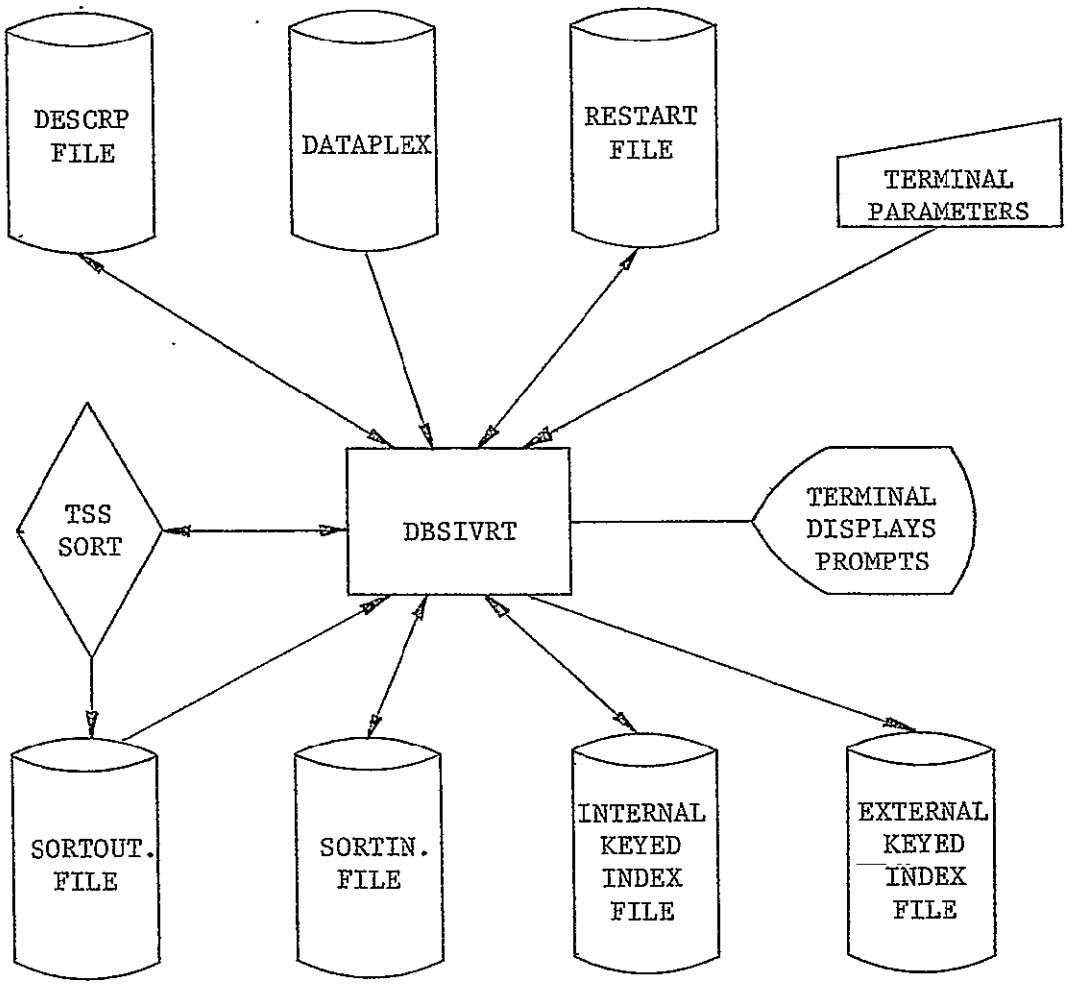


FIGURE 1. I/O BLOCK DIAGRAM

IV

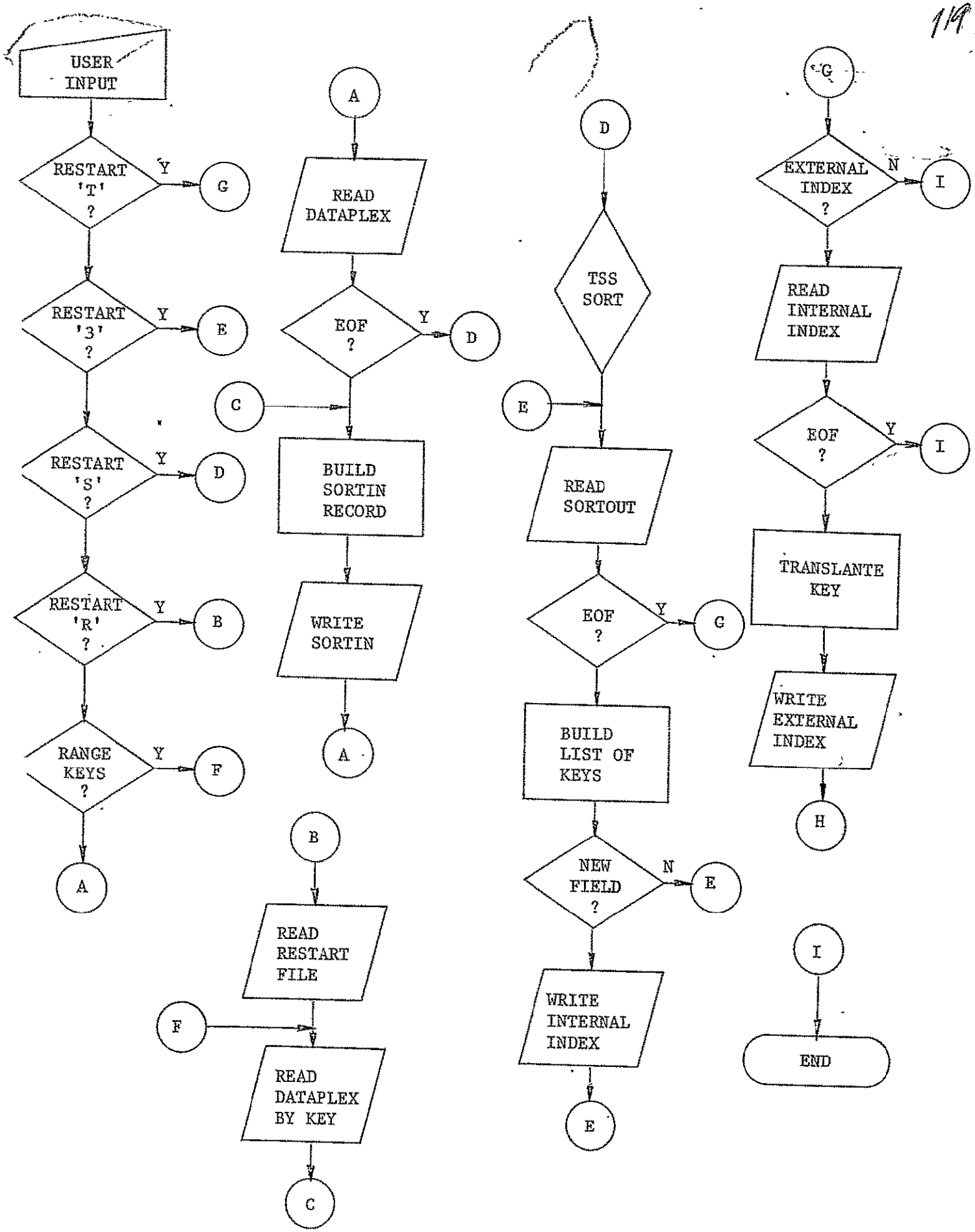


Figure 2. Top level flowchart

IV D7.2

TOPIC D.8 - DBLOAD FILES BACKUP

A. MODULE NAME

Maintenance - DBLOAD Files Backup
Program-ID - RDBLLBK
Module-ID - DBLDBK

B. ANALYST

Richard D. Graven
Neoterics, Inc.

C. MODULE FUNCTION

This module is a utility called by DBLOAD to backup all files of a data base being loaded at a checkpoint of every (1,000) records.

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Pnched Card Input Files

Not Applicable

c. Input Files

The input files are all files of the data base being loaded (such as anchor. associated, index, subfile).

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The output files are backup generation data group (2 levels) of all files of the data base being loaded. The DSNAME will be OWNERID.'BACKUP'.PLEXII file suffix.

Example:

SAFETY.BACKUP.COMAT$Y

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-out

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

Not Applicable

2. Narrative

This module is coded as a sub-procedure to DBLOAD. It deals with three parameters:

a. The ownerid of the file.

b. The data base name.

c. The character string of file suffixes.

The ownerid is a 1-8 character string of the file owner such as: SAFETY

The data base name is a six (6) character string of the file name padded with '$' such as: COMAT$.

The string of file suffixes to be backed up looks like: ' 1,A,B,Y,Z'.

The string of file suffixes is searched and concatenated to the data base name to construct the backup file DSNAME. Index file A for the COMAT file owned by the TSSid SAFETY would be backed up with a DSNAME of SAFETY.BACKUP.COMAT$A

The PL/I routine SYSOBP is used to CATALOG a level 2 generation data group, DDEF the file, CDS the file, and RELEASE the DDNAME.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with no DBPL/I statements.

2. Suggestions and Techniques

Depending on storage available under a given userid, it might be advantageous to copy the files to a data cell or private disc pack.

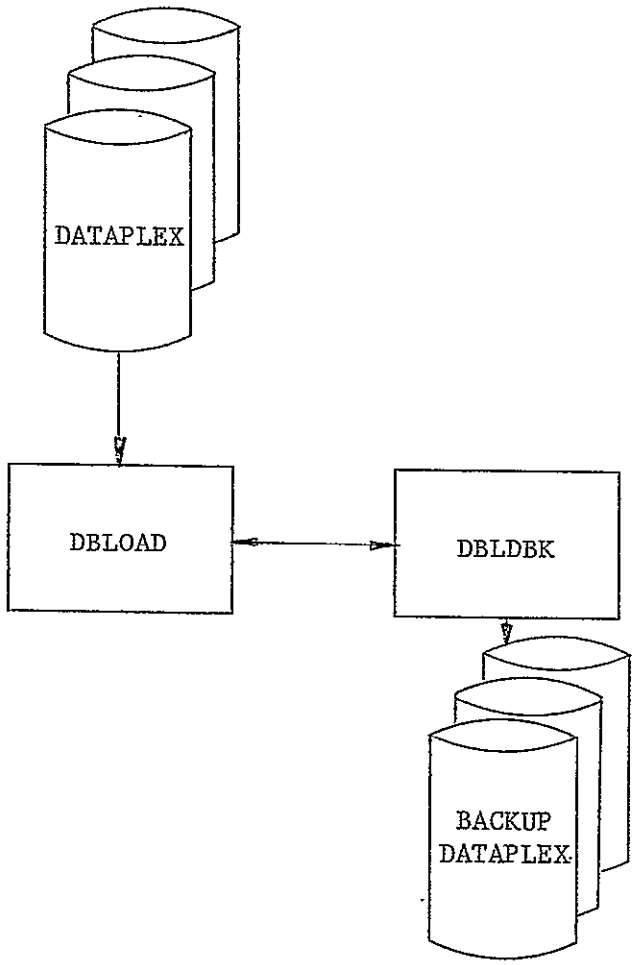


Figure 1. I/O Block diagram

TP 58.1

TOPIC D.9 - DESCRIPTOR EDITOR - ADD - CHANGE COMMANDS

A. MODULE NAME

Program-ID - RDBEDAC
Module-Name - DBEDAC
Entry Points
DBEDAC1 - ADD Command
DBEDAC2 - CHANGE Command

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

Those commands allow the user to create and modify field descriptors.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDAC:

1. FIELD
2. FLD
3. FLD_STRING

4. HDR
5. HDR_STRING
6. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into RDBEDAC a flag is set to indicate if the ADD or CHANGE command was entered. The routine DBEDGF is called to obtain a valid fieldname. In ADD mode it must be a valid non-existent and non-reserved fieldname. In CHANGE mode it must be an existent non reserved fieldname with the exception of the fields FREEFORM and COMMENTS and it must not be a superfield nor a subfile control field. If in ADD mode a FLD structure is allocated, initialized and posted with the fieldname.

The user is prompted for the field type and the input is validated. If it is invalid, the user is given a diagnostic and prompted for a new value. If in update mode, the user is not allowed to change the field type if it affects the field length and the field appears in the fixed part of the record. The user is also not allowed to make the key field a bit field.

If there is more data in the parameter list 'TYPE' the user is prompted for an alignment value. If it is invalid, the user is given a diagnostic and prompted for a new value. If no value is entered a built in default value is assumed and posted in FLD.

The field form is prompted for and validated. If invalid, the user is given a diagnostic and prompted for a new value. The anchor file key field and bit field can only be of fixed form. In Update mode, the user can not change fixed field to a varying or elemental field. the necessary values are posted in FLD.

The user is prompted for a field length and the value is validated against prestored maximum

values for single and multielement fields. If it is invalid, the user is given a diagnostic and prompted for a new value. The correct values are posted in FLD.

If the field is non-elemental, go prompt the user for a conversion routine, otherwise the user is prompted for an element length value if necessary. For several field types the only element length value is posted in FLD. If the element value is invalid, the user is given a diagnostic and prompted for a new value.

The user is prompted for the number of elements and the input validated. If the value is invalid, the user is given a diagnostic and prompted for a new value. A correct value is posted in FLD. The parameter unique element is prompted for and processed in the same manner.

The routine IEEDGR is called to obtain and process the conversion, formatting, validation routine names and validation argument.

At this point of adding the key field or in update mode, the rest of the parameters are ignored. in update mode the changed information is posted to the descriptor dataset by calling DBEDFL, and then go save the command string. The use is prompted if the field is to be indexed. If the answer is no then go prompt the user for associated file information, otherwise the user is prompted as to which index file the field is to appear. If no defining fieldname is entered, a new index file is created for this field. Otherwise the field is placed on the same index as that of the defining field. In the CHANGE command, if the field was already indexed on a different file, it must be deleted from that index file before it is placed on the new index file.

The user is prompted as to whether the index key is to be in either internal or external form. If no value is entered, internal is assumed. If the value is external then the user is prompted for and must enter the maximum length of the external value.

The user is prompted if the index is to be spanned. If no value is entered, it is assumed not to be spanned. At this point, the index is ready to be setup. If it is a new index a header descriptor is allocated and setup for the index,

else the new information is posted to the existing header.

The routine DBEDGA is called to determine if the field is to be placed on an associate file.

The user is prompted if the field is to be on a subfile. If not go prompt the user for a subfile value as obtained, the subfile header is updated accordingly.

The user is prompted for the defining base field name if the field is to be a subfield. If no value is entered, the field will not be a subfield. If it is to be a subfield, the user is prompted for an offset value. If none is entered, a value of 0 is assumed. In case the defining base field is either RECLEN or the anchor key field the user is prompted for the particular file on which this subfield is to be placed. The user can specify the actual file name if known or indicate the type of file on which the subfield is to be placed. If ASSOCIATE or SUBFILE is specified the user is prompted for a field defining which associate file or which subfile. the subfield information is posted in FLD.

At this point all of the parameters have been entered, processed and the information posted. It is now determined which file list the field is to be placed and if not in the proper place already, threaded onto the end of that file list.

If adding the anchor key field then the fields FILEKEY, FREEFORM, and COMMENTS are setup on the appropriate files and an index file is setup for FREEFORM.

The command string is saved in the current strategy and control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPI/I statements.

2. Suggestions and Techniques

Not Applicable

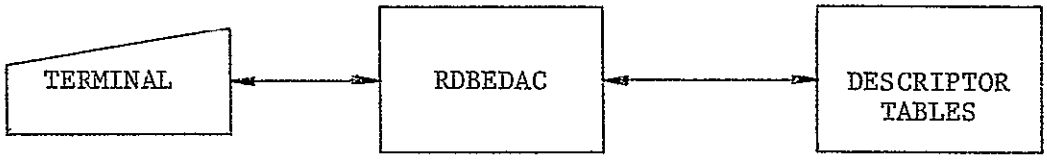


Figure 1. I/O Block diagram

IV

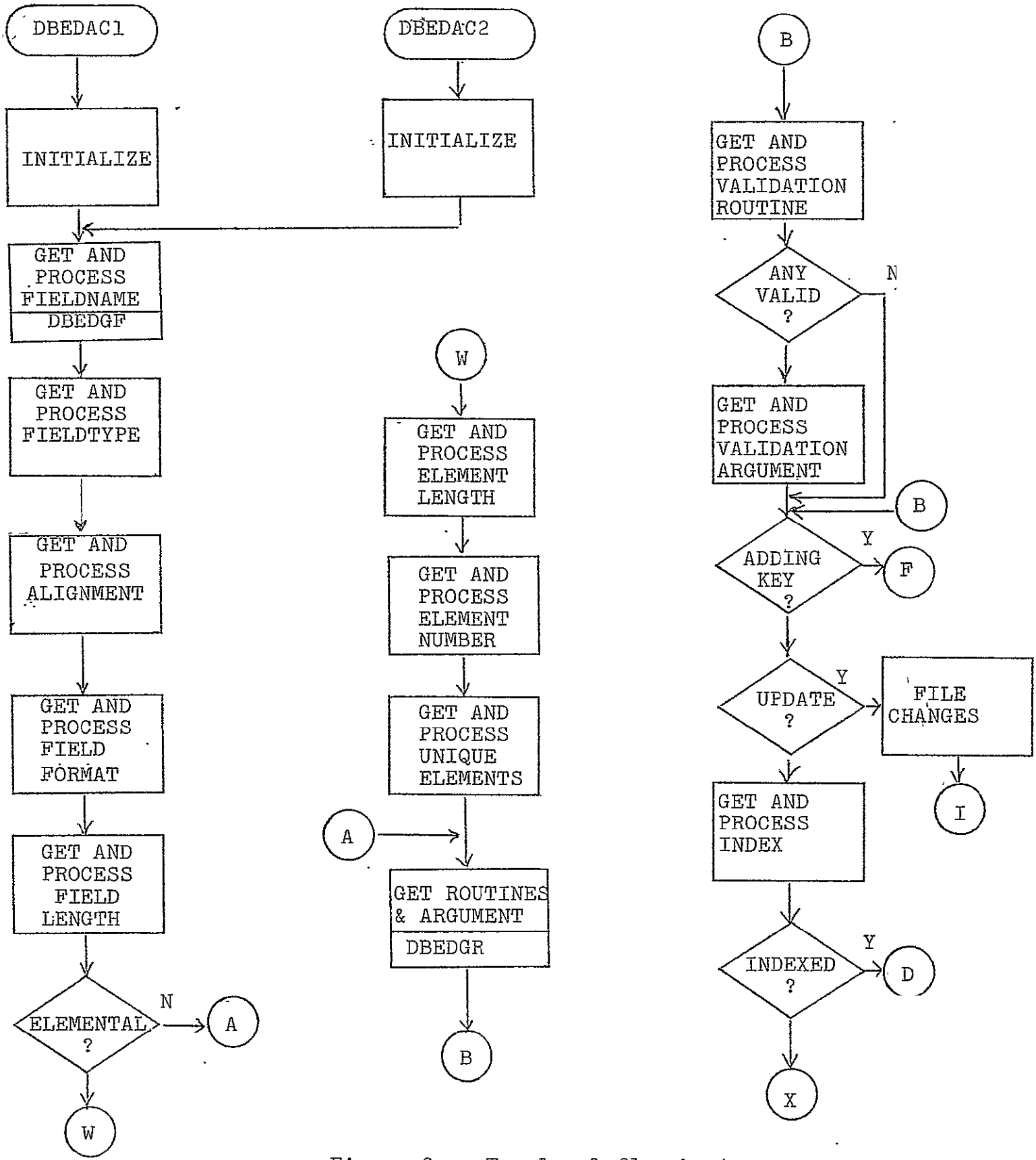


Figure 2a. Top level flowchart

IV D.92.a

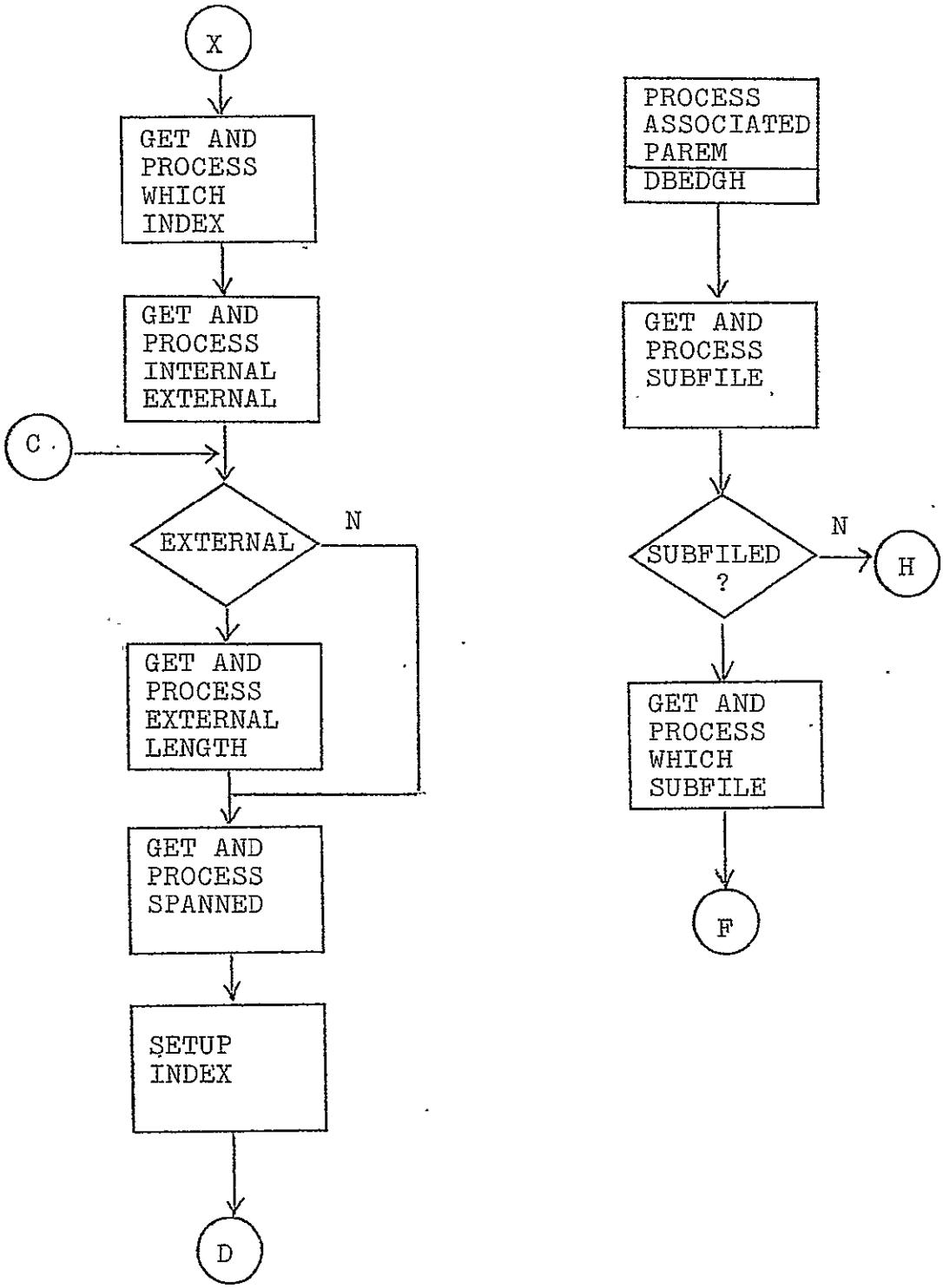


Figure 2b. Top level flowchart

.59.2b

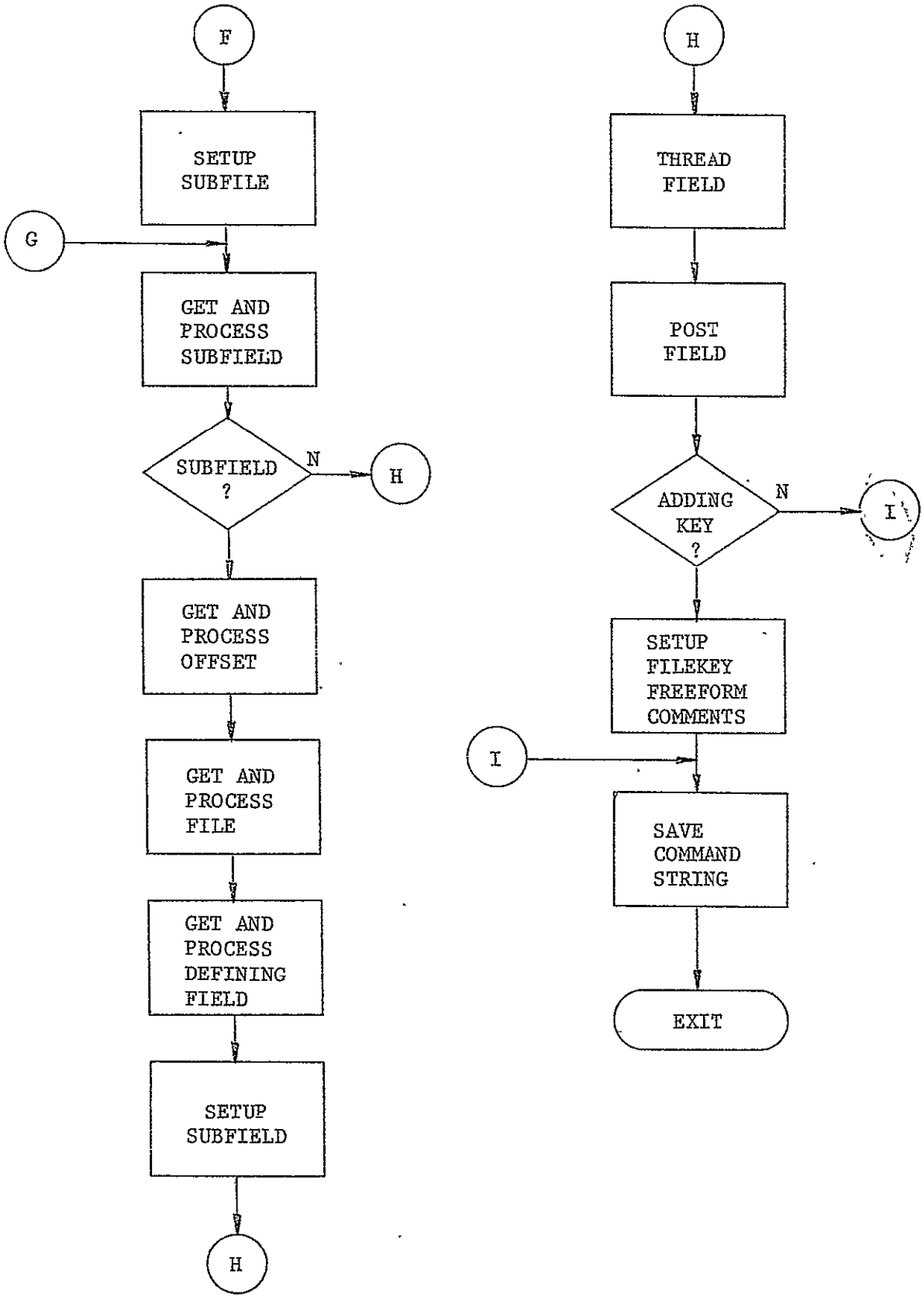


Figure 2c. Top level flowchart

IV D9.2c

TOPIC D.10 - DESCRIPTOR EDITOR - ADDLIKE - RENAME COMMANDS

A. MODULE NAME

Program-ID - RDBEDAR
Module-ID - DBEDAR
Entry Points:
DBEDAR1 - ADDLIKE Command
DBEDAR2 - RENAME Command

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The ADDLIKE command creates a new field descriptor exactly like an existing descriptor with a different name. The RENAME command changers the name of an existing descriptor.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDAR:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. SECURITY
7. SUPER
8. SUPER_STR
8. SUPER_STR
9. VALID
10. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The entry points are ADDLIKE command - DBEDAR1 and RENAME command - DBEDAR2. Upon entry into either command a flag is set indicating which command was called. After which the two commands share common code for parameter processing.

Routine DBEDGF is called to obtain a valid new field name. To be valid the new field name must be of alphanumeric of at most eight characters long, must not already exist and must not be a reserved field name.

Routine DBEDGF is called to obtain a valid existing fieldname. This field must exist and must not appear in the reserved fieldname list.

If in the RENAME command, then the name of the specified existing field is changed to the specified new field name and the field name change is posted in the FIELD structure. At this point the command string is stored in the current strategy and then control is returned to the

calling routine.

If in the ADILIKE command the existing field is duplicated, the new fieldname posted in the copy. The new fieldname is posted in the FIELD structure. The ADDLIKE command string is saved in the current strategy, after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

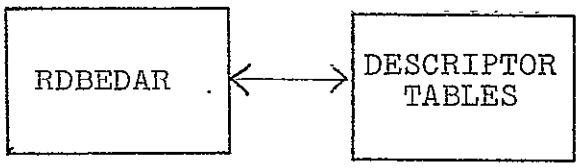


Figure 1. I/O Block Diagram

D10.1

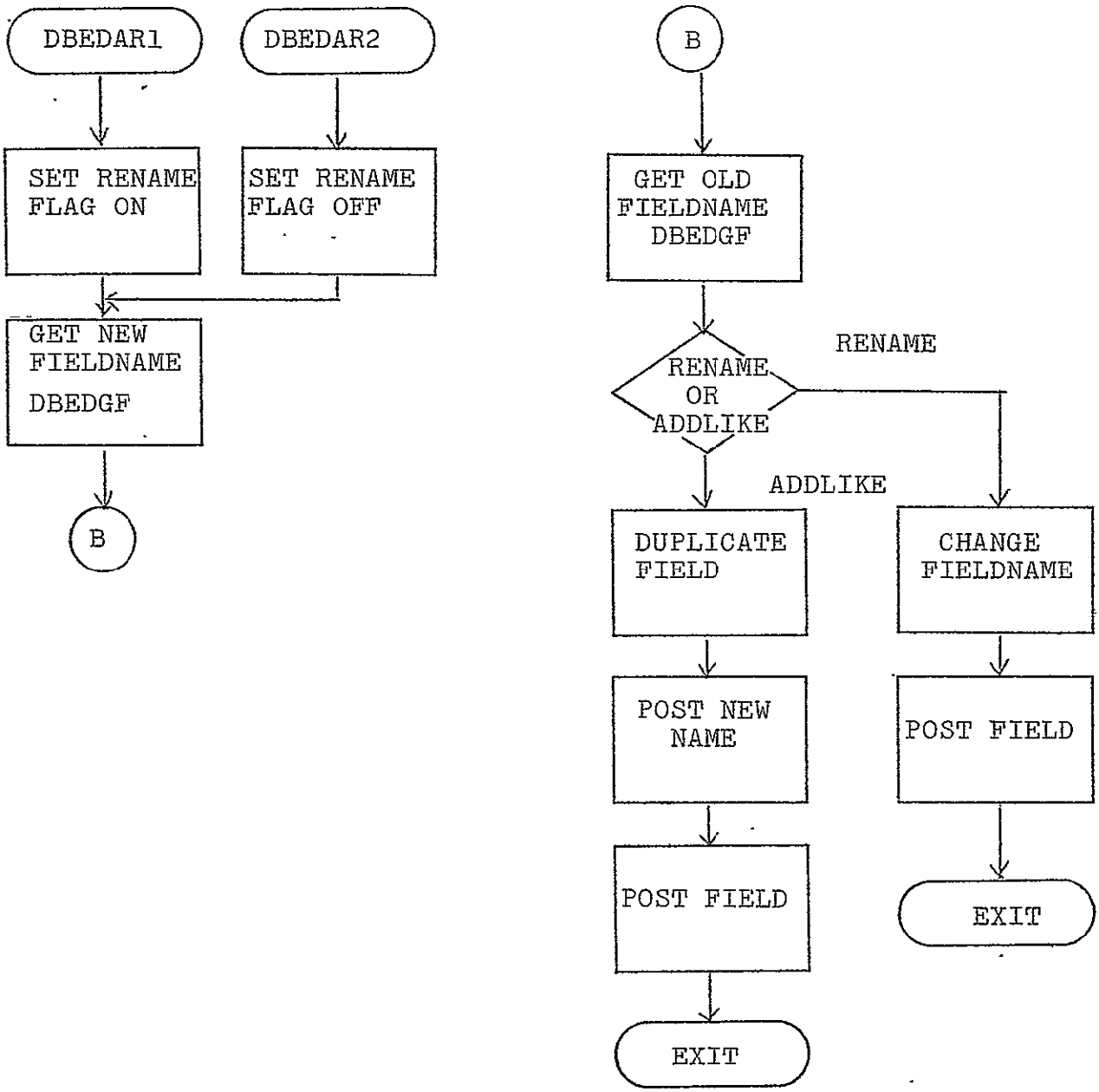
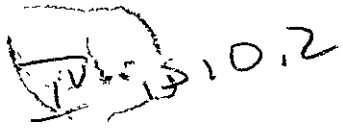


Figure 2. Top Level Flowchart



TOPIC D.11 - DESCRIPTOR EDITOR - CHKPOINT COMMAND

A. MODULE NAME

Program-ID - RDBEDCP
Module-ID - DBEDCP

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTIONS

This command is used to save the descriptor tables as they exist in memory in a VSAM data set as that they may be retrieved at a future time by use of the RESTORE command and then continue to create the descriptors from that point.

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

The cutput file is a TSS VSAM data set named DESCRP.CHKPCINT. Refer to the data set specifications for a description of this data set.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDCP:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC_STR
7. SECURITY_STR
8. SUPER_STR
9. VAIID
10. X

The description of these tables is specifications in the dataset of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into CHKPOINT, any previously existing checkpoint dataset is erased by use of ASMERSE routine. The data set record length is dynamically determined by calculating the length of that part of the X structure that must be saved, the current length of the FIELD structure and using the larger of the two values.

The data set CHKPOINT.DATASET is created and initialized by use of the routine ASMDCB to create the DCB for the data set, routine ASMDD to data def the dataset routine ASMENDS to initialize the JFCB, and routine ASMOPEN to open the checkpoint data set.

The variable part of the X structure is put into the data set by use of the ASMPUT routine, and likewise the FIELD structure.

Each of the fields are saved through use of ASMPUT routine by creating the following character string: the FLD_STRING concatenated to SUPER_STR if it is a superfield, concatenated to SECURITY_STR if there is field security on this

field, concatenated to VALID.ARGUMENT if the field has a validation argument.

Each of the headers are saved through use of ASMPUT routine by creating the following character string: the HDR_STRING concatenated to RECSEC_STR if the file has record security.

The checkpoint dataset is closed by use of the routine ASMCLOS, after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

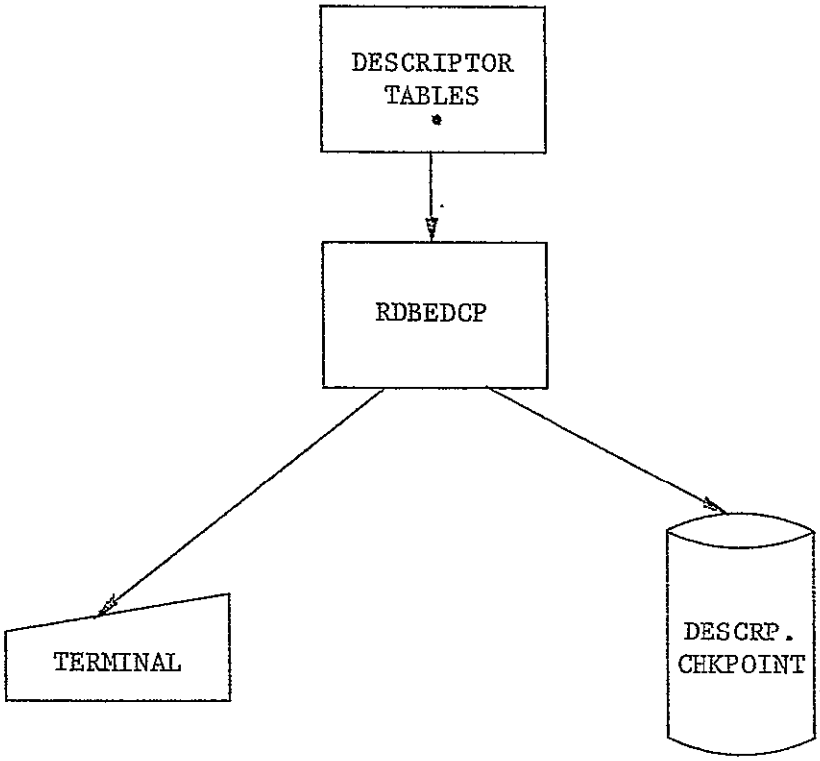


Figure 1. I/O Block diagram

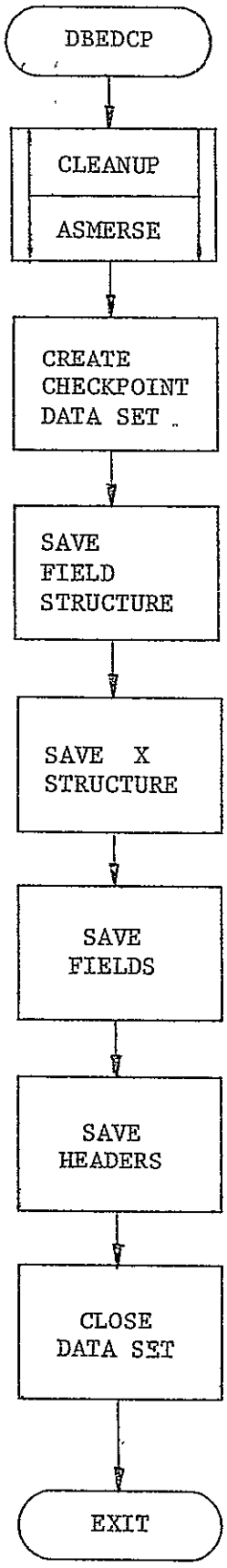


Figure 2. Top level flowchart

IV $\Delta > 11.2$

TOPIC D.12 - DESCRIPTOR EDITOR - CREAT SUB COMMAND

A. MODULE NAME

Program-ID - RDBEICS
Module-ID - DBEDCS

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This routine is used to define and setup the necessary field to create a subfile.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables ----

The following external tables are referenced by RDBEDCS:

1. FIELD
2. FID
3. FLD_STRING
4. HDR
5. HDR_STRING
6. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into CREATSUB, module DBEDGF is called to obtain a valid subfile control fieldname. To be valid this field name must not be longer than six characters long and be a valid alphanumeric character string. The following field names are then created. The subfile key field name, the subfile parent key field name, and the subfile record security field name. Of the above mentioned four fieldnames, none must exist and none must be a reserved field name for the entered subfile control field name to be valid.

The user is prompted for the maximum number of subfile records per anchor file record that may be loaded into the subfile. The number must be less than or equal to 1325. If the number is valid, processing continues, else the user is given a diagnostic and prompted for a new number. This number then becomes the number of elements on the subfile control field.

Routine DBEDGA is called to determine if the subfile control field is to appear on an associate file.

The subfile control field, the subfile key field, and the subfile parent key field are now created and posted with the proper values. The subfile control field is placed in the varying field list of either the anchor file or the appropriate associated file. The subfile key field and the parent field are placed in the fixed list of the

appropriate subfile.

The afore mentioned field names are placed in the reserved field name list. The command string is saved in the current strategy, after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

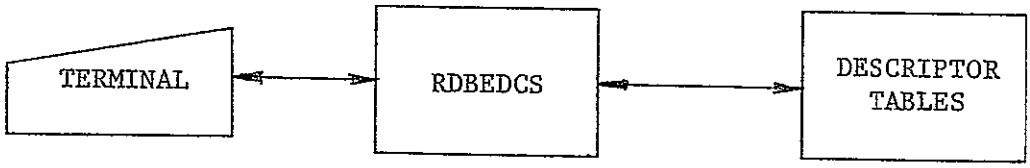
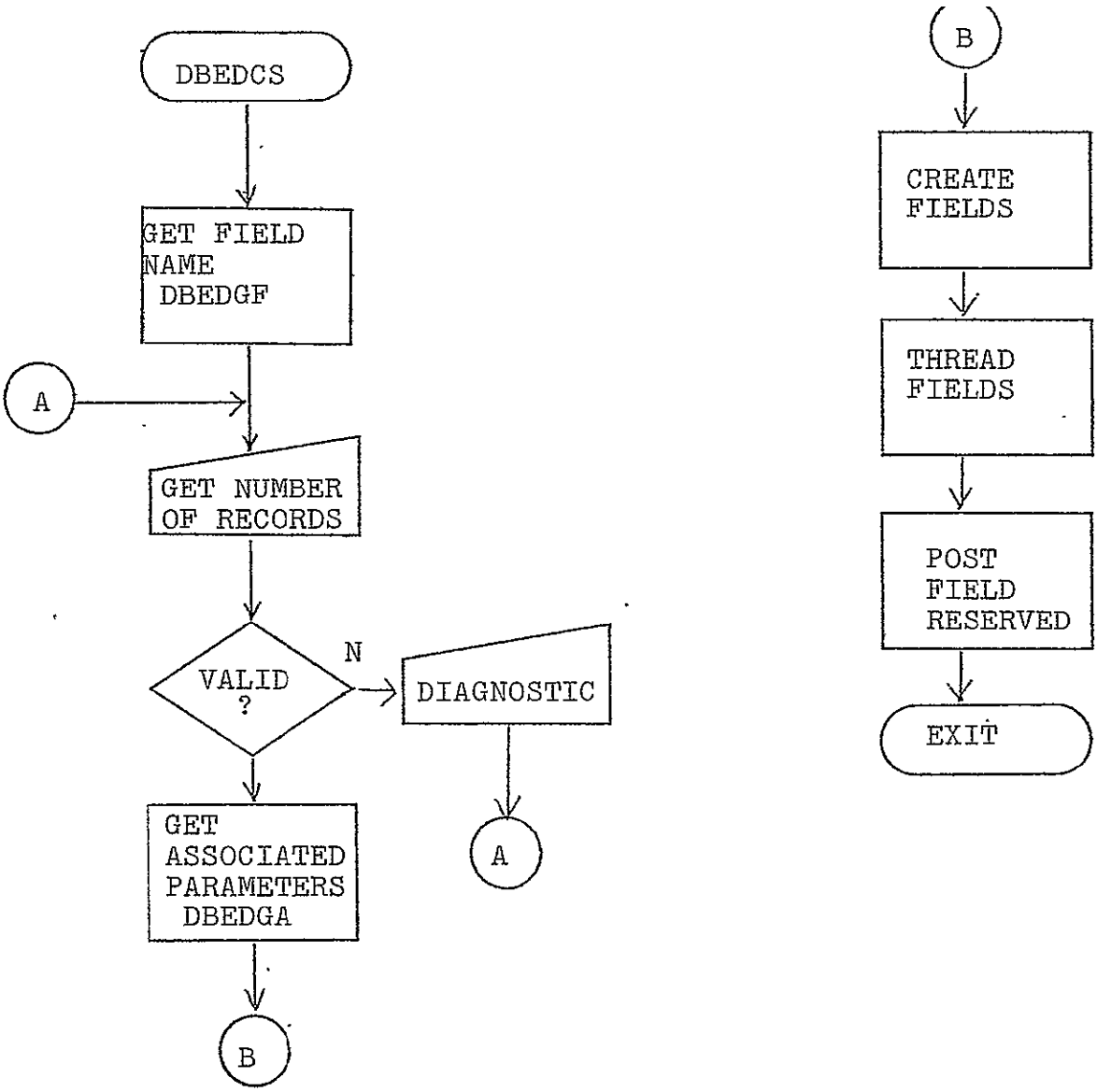


Figure 1. I/O Block diagram

IV


DBEDCS
GET FIELD NAME DBEDGF
A
GET NUMBER OF RECORDS
VALID ?
N
DIAGNOSTIC
A
GET ASSOCIATED PARAMETERS DBEDGA
B
B
CREATE FIELDS
THREAD FIELDS
POST FIELD RESERVED
EXIT

10 Oct 2022,

TOPIC D.13 - DESCRIPTOR EDITOR - END COMMANDS

A. MODULE NAME

Program-ID - RDBEDDE
Module-ID - DBEDDE

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is the entry point into the Descriptor Editor. It prompts for and processes Descriptor Editor commands and calls the appropriate command routine. The END command is used to terminate Descriptor Editor processing and return control to the maintainence director.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDDE:

1. FIELD
2. X
3. VERBTAB

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Module DBEDIN is called to set up mode of operation and all of the tables necessary to the running of the descriptor editor.

The user is prompted for his next Descriptor Editor command. If the command is invalid as determined by a search of the verb table, the user is given a diagnostic and prompted for a command string.

If the command is not END, then the appropriate command is called by use of the CALL routine when control is returned, the user is prompted for his next command.

If the command is END then if the user has not filed his corrections, additions, or changes, he is prompted informing him such and asked if he really wants to terminate the Descriptor Editor. If the anser is no then the user is prompted for his next Descriptor Editor command, else the Descriptor Editor run is terminated.

At termination each field storage and each header storage area is released. The FIELD and X structures are then released. Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DBPL/I and TSPL/I statements.

2. Suggestions and Technigues

Not Applicable

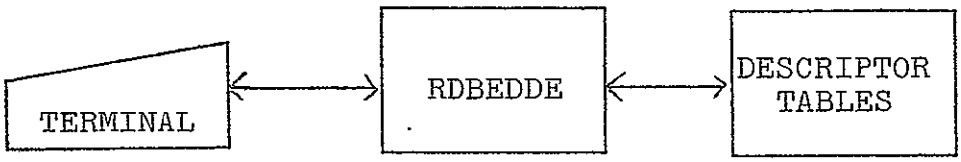


Figure 1. I/O Block diagram

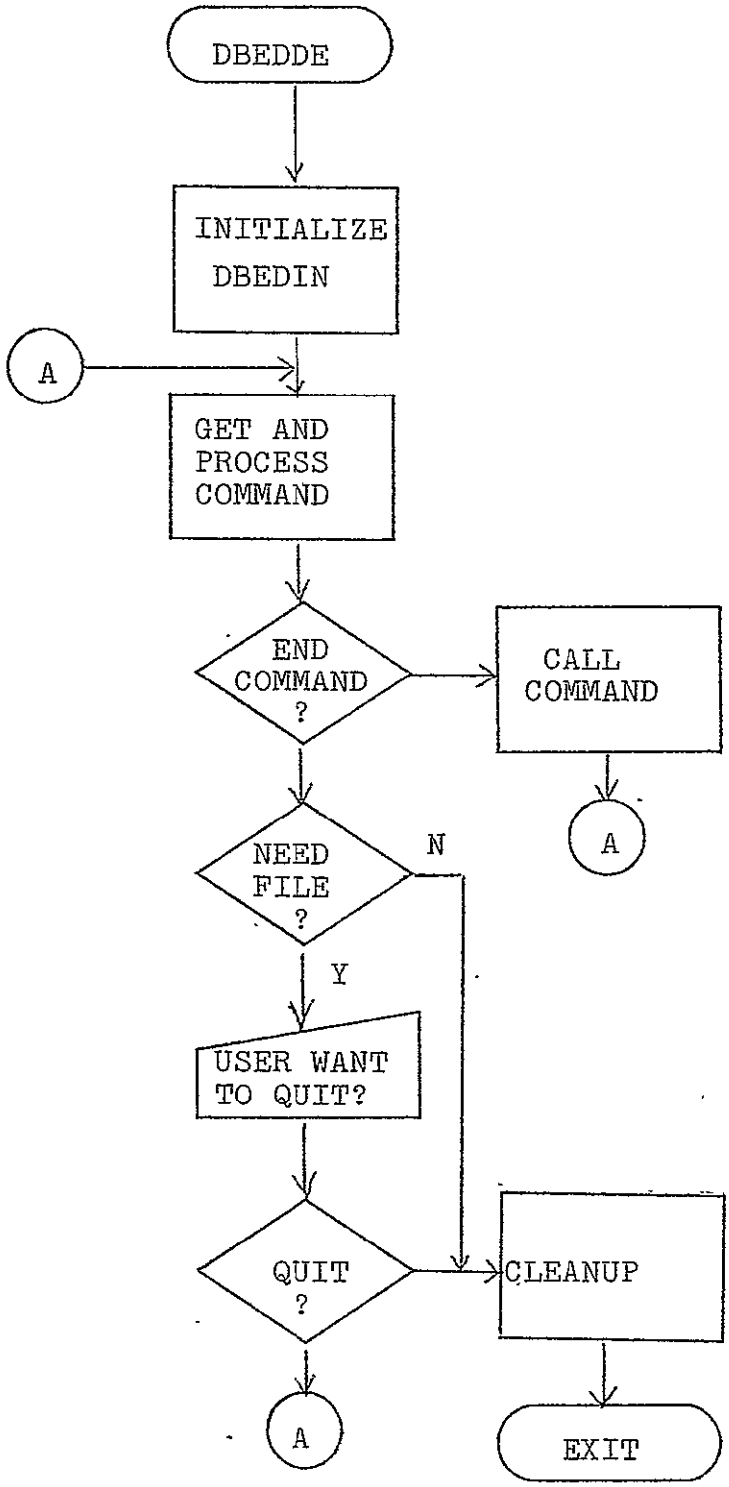


Figure 2. Top level flowchart

$\overline{ID} > 13.21$

TOPIC D.14 - DESCRIPTOR EDITOR - DISPLAY INTERNAL COMMAND

A. MODULE NAME

Program-ID - RDBEDDI
Module-ID - DBEDDI

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is a debugging tool used to display the various external descriptor tables (DESCTAB), by their internal name, field descriptors by their field name and header descriptors by their file ids.

DISPLAYI DISTYPE=<I,F,H>,DISNAME=structure-name

where:

DISTYPE is the type of variable to be displayed I for internal, F for field descriptor and H for file or header descriptor.

DISNAME is the name of the variable to be displayed. For internal mode the following structures may be displayed.

ERRORFILE
FIELD
FLD
HDR
RECSEC
SECURITY
SUPER
VALID
FLD_COMMENTS
FLD_FREEFORM
FLD_RS
FLD_SUBCNTRL
FLD_SUBID
FLD_SUBPK
HDR_ASSOC
HDR_INDEX
INIT_FLD
INIT_HDR
INIT_RECSEC
INIT_SECURITY
INIT_SUPER

IO_FLD
IO_HDR
IO_RECSEC
IO_SECURITY
FLEX
RESERVEC
XS
X

For field mode the name of the field to be displayed is supplied. For header mode the file suffix id is entered.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

1. FIELD
2. FLD
3. HDR

4. RECSEC
5. SECURITY
6. SUPER
7. VALID
8. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into RDBEDDI the user is prompted for the display type. If the display type value is not 'I', 'F', or 'H' the user is given a diagnostic and prompted for a new value.

Depending on the display type the user is prompted for either an internal structure name, a field name, or a header id. If the internal structure name is not contained in the list of names in the module function section, or the field does not exist or the file does not exist, the user is given a diagnostic and prompted for a new display name value.

When displaying an internal name, a label variable is used, one label for each structure that may be displayed. At each of these pieces of code, a generalized display subroutine is called to display the desired type of structure passing the address of the particular structure to be displayed. This is done for all structures except for the structures PLEX, ERRORFILE, and XS. A word about these display procedures later. The information from the structures PLEX and ERRORFILE is setup and displayed. The display of the X structure is a service of calls to the different display procedures, one for each minor structure of X to be displayed.

When displaying a field descriptor, a call to the procedure DIS_FLD is called to display the proper FLD structure. If the field is a superfield, has a validation argument, or has field security, calls are made to the routines DIS_SUPER, DIS_VALID, and DIS_SECURITY to display the proper structures. This is done to display all of the

information associated with the field.

When display a header descriptor, a call is made to DIS_HDR to display the proper HDR structure and if the file has record security, a call is made to DIS_RECSEC to display the appropriate record security information.

After the information has been displayed, control is returned to the calling routine.

For displaying the actual desired data several internal procedures are set up, one for each type of structure. They are DIS_FIELD, DIS_FLD, DIS_HDR, DIS_RECSEC, DIS_RESERVED, DIS_SECURITY, DIS_SUPER, DIS_VALID, and DIS_XS. These procedures build the output information in a work area in predefined formats. The information is output to the terminal thru use of the TS PROMPT facility. The output consists of a title line followed by the data usually displayed beneath the title line.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I and DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

Figure 1. I/O Block diagram

IV-D14.1

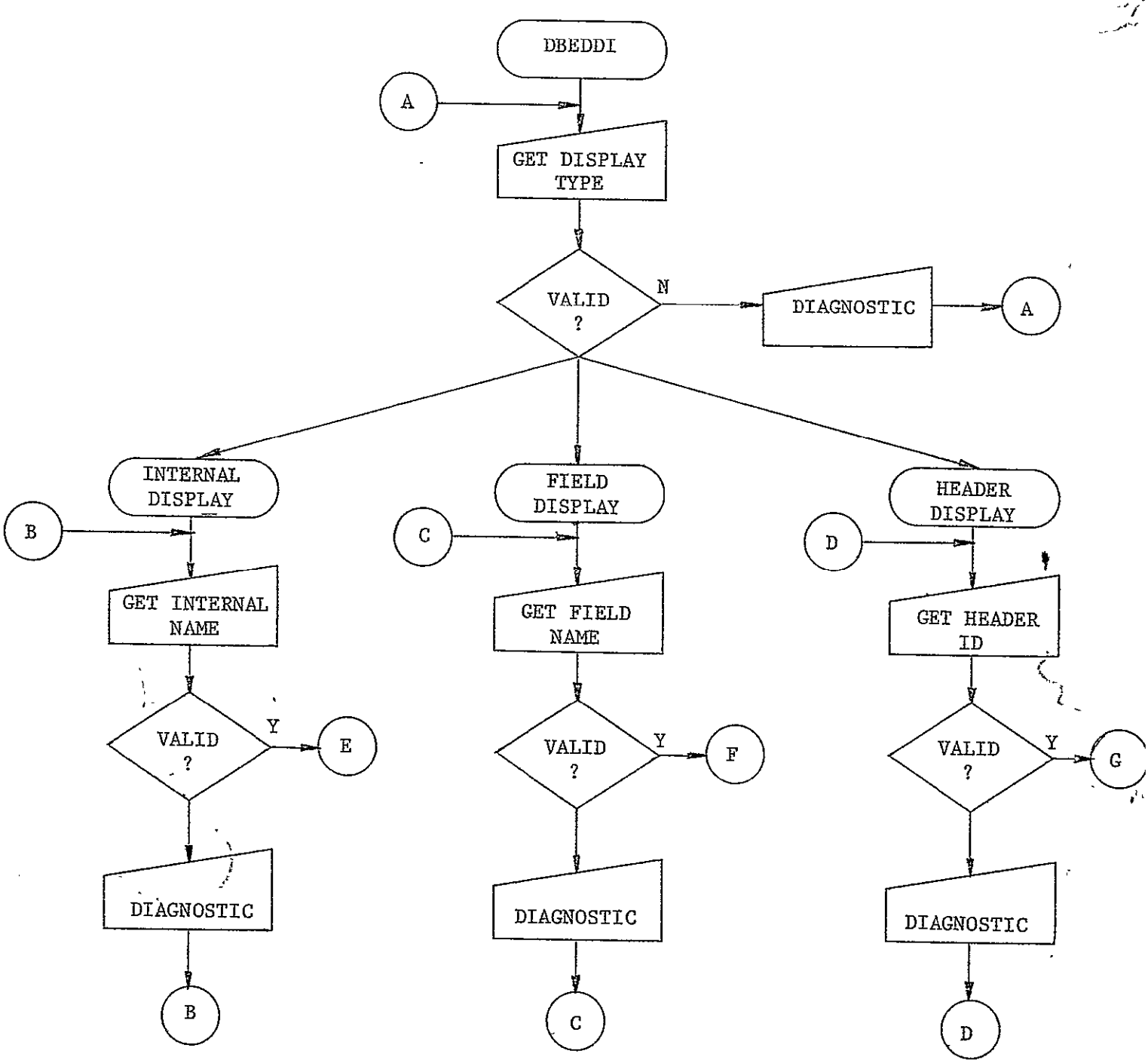


Figure 2a. Top level flowchart

IV D14.2a

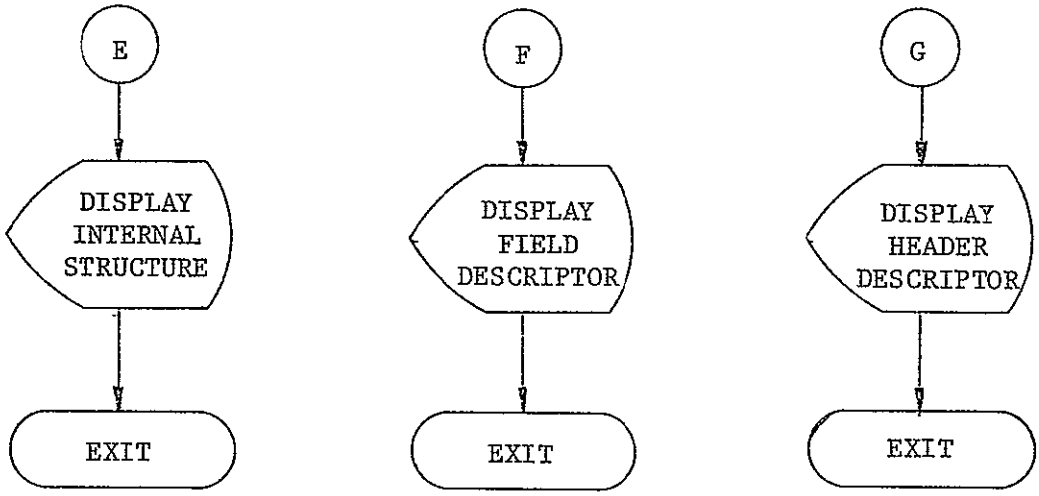


Figure 2b. Top level flowchart

IV B14.2b

TOPIC D.15 - DESCRIPTOR EDITOR - DELETE FIELD COMMAND

A. MODULE NAME

Program-ID - RDBEDDL
Module-ID - DBEDDL

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is used to delete a previously defined field descriptor.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDDI.

1. FIELD
2. FLD
3. HDR
4. SUPER
5. X

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Routine DBEDGF is called to obtain a valid fieldname. To be valid, the field must exist. If the field name appears in the reserved list, then it must be a subfile control field and there must be no other fields on this subfile to be valid.

A further check is made to determine if the field to be deleted is a component of any superfields or is the defining base field for any subfields. If so, the user is told of all superfields and all subfields that make use of this field. The user is then prompted for a new field name value. If here then the field can be deleted.

At this point, the internal delete entry point is defined. If the field name to be deleted does not exist, control is returned to the calling routine. Otherwise a pointer is set to the field to be deleted. At this point delete forms common code.

If the field appears on an associate or subfile or is indexed, then the appropriate file descriptor counts are updated. If the associated file or index file is depleted of fields, the file headers are deleted, and the file ids made available for reassignment.

At this point the field is deleted by the internal delete field routine. If the deleted field is a subfile control field, the subfile key field and the parent key field are also deleted.

The next field in the list to be deleted is now processed in the afore mentioned manner. After all of the fields have been processed, the command string is saved in the current strategy, if it was the delete command that was called. Control is then returned to the calling routine.

The internal procedure DELETE FIELD is used to release the work areas containing the field

information, and to post the deleted field list if this field exists on the disc storage version of the descriptor file.

F. CODING SPECIFICATIONS

1. Source Language

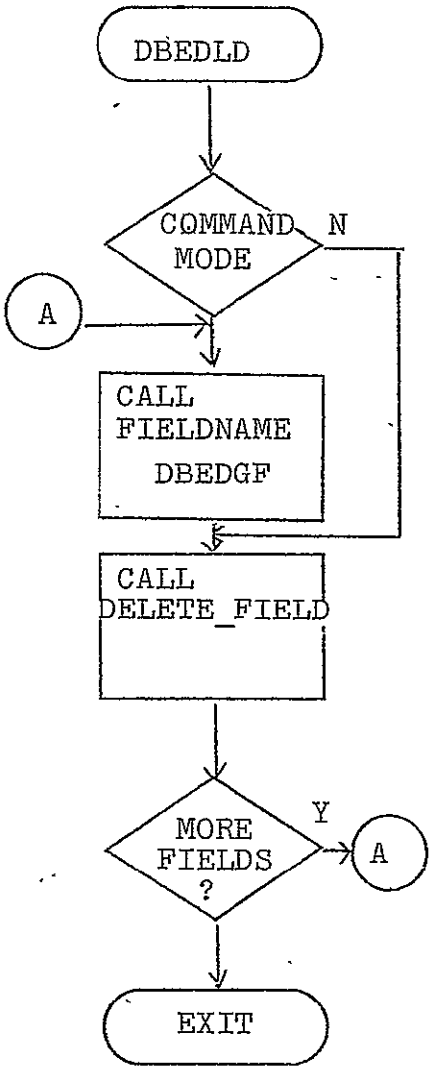
PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

Figure 1. I/O Block diagram

IV. DIS. 1



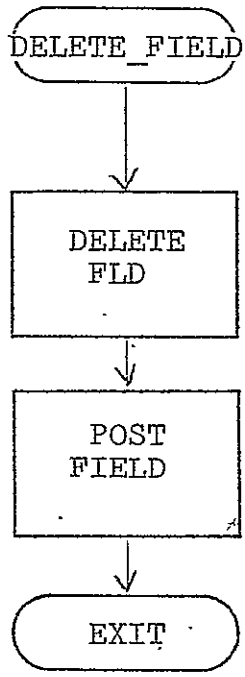


Figure 2. Top level flowchart

IV D 15.2

TOPIC D.16 - DESCRIPTOR EDITOR - DISPLAY FIELD COMMAND

A. MODULE NAME

Program-ID - RDBEDDP
Module-ID - DBEDDP
Entry Points
DBEDDP1 - DISPLAY Command
DBEDDP2 - Paging Entry

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This routine is used to display the information defining a field.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input File

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The various pieces of information are displayed on the screen one item per line preceded by a descriptive title. Refer to the dataset specifications for a description

cf this display format.

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDDP:

1. FIELD
2. FLD
3. HDR
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

At the command entry point the paging information structure is allocated and initialized and routine DBEDGF is called to obtain the fieldname to be displayed.

At the paging entry point, the paging information is set to point to the proper page to be displayed and then join common code with the command entry point.

At the start of the common code the number of the next item to be displayed is retrieved from the paging information and a branch is taken to the appropriate code to obtain the next piece of field information. If there is no information for this item number, the next item is pointed to and processed as above. After the line of information is built, it is placed in the screen buffer.

If there is more room in the buffer, the next item is pointed to and processed as above. Once the screen is full and there is more information to be output in the forward direction, a paging entry

point is setup and next page information is posted in the paging information structure. The buffer is then flashed to the screen after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

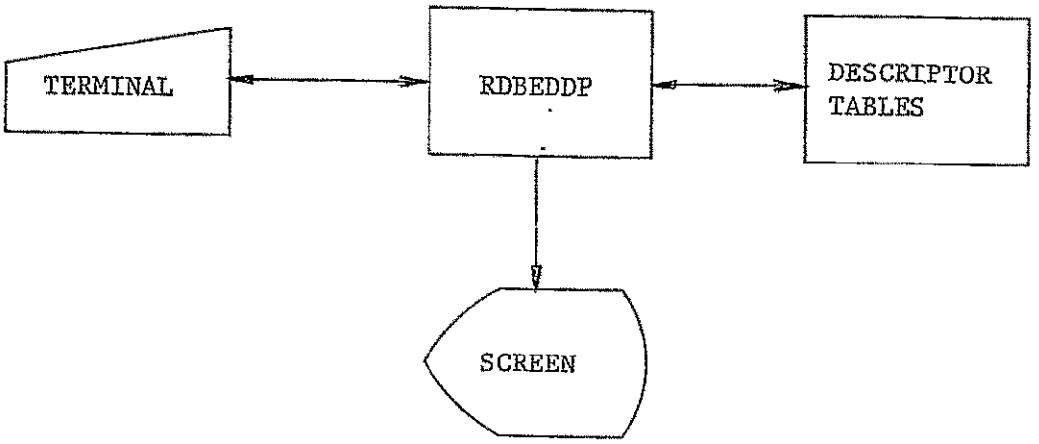


Figure 1. I/O Block Diagram

IV 1611

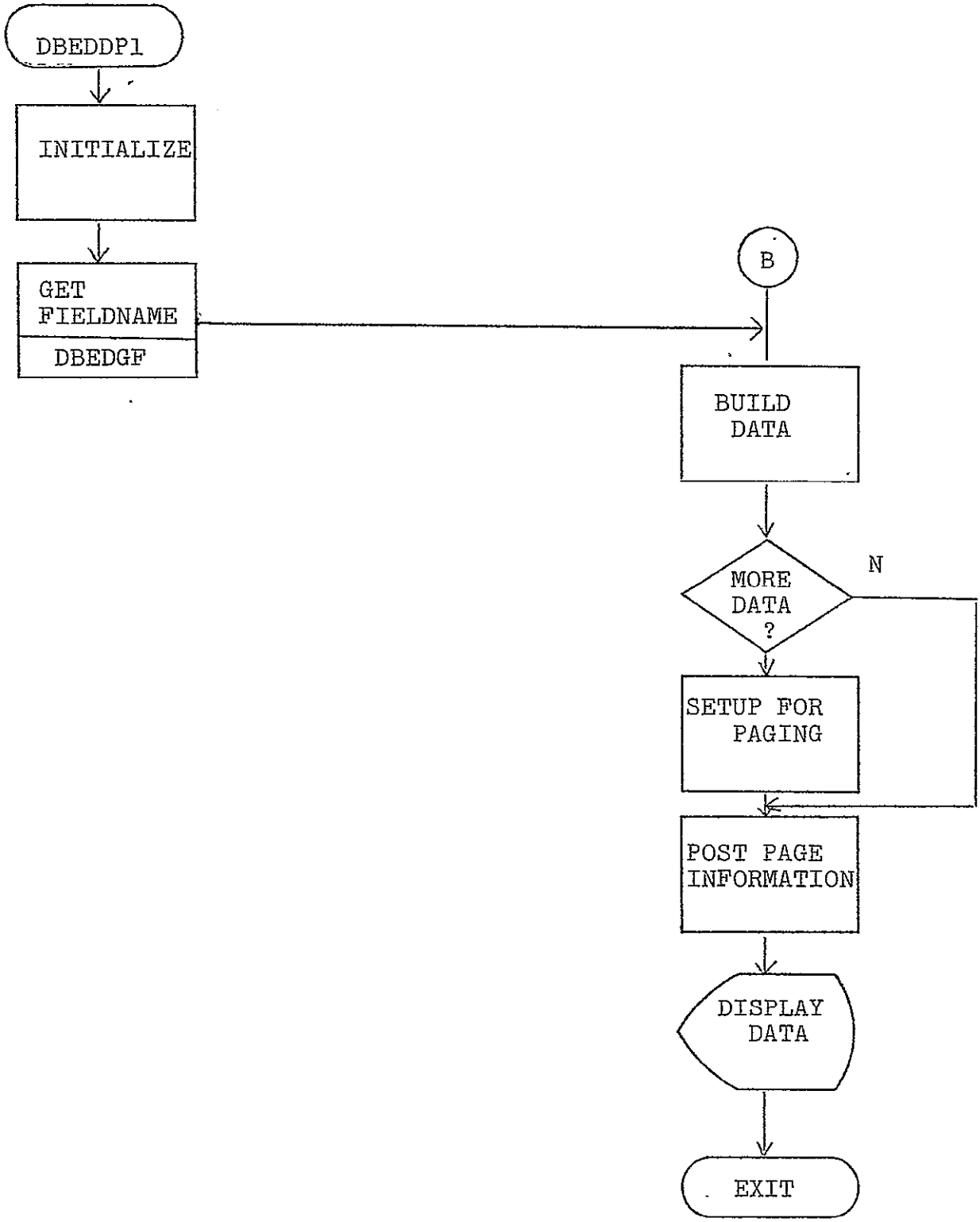
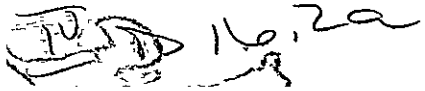


Figure 2a. Top Level Flowchart



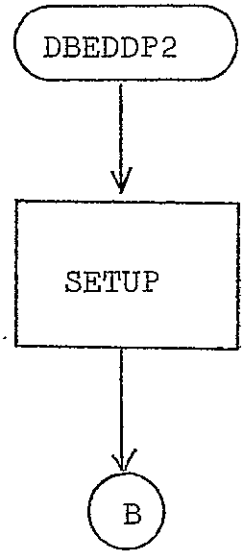


Figure 2b. Top Level Flowchart

TO: D16.2b

TOPIC D.17 - DESCRIPTOR EDITOR - COMMON ROUTINES MODULE

A. Program-ID - RDBEDCM
Module-ID - DBEDCM
Entry Points DBEDDA
DBEDDS
DBEDDX
DBEDEF
DBEDFA
DBEDGA
DBEDGF
DBEDGR
DBEDPG1
DBEDPG2

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module consists of several routines commonly used by the Descriptor Editor. They are:

1. DBEDDA is used to delete a field from an associate file.

2. DBEDDS is used to delete a field from a subfile.

3. DBEDDX is used to delete a field from an inverted index file.

4. DBEDEF is used to expand the field structure when it is full.

5. DBEDFA is used to release the work areas containing all of the field descriptor and file descriptor information.

6. DBEDGA is used to get and process the ASSOCED parameter group.

7. DBEDGF is used to get and process a valid fieldname.

8. DBEDGR is used to get and process the conversion, formatting, and validation routine names and validation argument.

9. DBEDPG1 is used for a common paging entry point.

10. DBEDPG2 is used to flush the buffer to the screen

and setup to allow paging.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The displays output to the screen by DBEDPG2 are those setup by the DISPLAY, REVIEW, and FIELDS commands. Their descriptor can be found in the Dataset Specifications section of the DWB.

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDCM:

1. FIELD

2. FLD

3. HDR

4. RECSEC

5. SECURITY

6. SUPER

7. VALID

8. X

A description of these tables can be found in the Dataset Specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into DBEDDS the appropriate subfile header table is addressed. The count of records on this subfile is decremented.

If the deleted subfile field exists on disc, then X.DELETE is posted so that the appropriate descriptor will be deleted. The field descriptor table is then updated after which control is returned to the calling routine.

Upon entry into DBEDDA the appropriate associate header table is addressed. The count of records on this associated file is decremented. If no fields are left on the associate file, the associate file header is released and the associate file-id made available for reassignment.

If the deleted associated field exists on disc, then X.DELETE is posted so that the appropriate descriptor will be deleted. The field descriptor table is then updated after which control is returned to the calling routine.

Upon entry into DBEDDX the appropriate index header table is addressed. The count of records on the index file is decremented. If no fields are left on the index file, the index file header is released and the index file-id made available for reassignment.

If the deleted indexed field exists on disc, then X.DELETE is posted so that the appropriate descriptor will be deleted. The field descriptor

table is then updated after which control is returned to the calling routine.

Upon entry into DBEDEF, the number of permissable items in FIELD is raised by 100 and a new larger FIELD structure is allocated. The information from the old field structure is moved to the new structure and then the old structure is released. Control is then returned to the calling routine.

Upon entry into DBEDFA, a control loop is set up to step through all of the existing field descriptor table. If the field descriptor has a validation argument, then the VALID pointer is setup and the VALID area released. If the field descriptor has field security, the SECURITY pointer is setup and the SECURITY area released. If the field descriptor defines a super field, the SUPER pointer is setup and the SUPER area released. The FLD area is released. After all the field descriptor areas have been released, the FIELD structure is released.

A control loop is setup to step through each header descriptor table. If the file has record security, the RECSEC pointer is setup and the RECSEC area released. The HDR area is then released.

After all file descriptors have been released if the RECLEN field has field security, the SECURITY pointer is setup and the SECURITY area is released. Control is then returned to the calling routine.

Upon entry into DBEDGA, the user is prompted whether or not the field is to be placed on an associate file. If the response is not an acceptable boolean value, he is given a diagnostic or prompted for an acceptable boolean response. If the response is no and in UPDATE mode, if the field is already on an associate file, the field is deleted from the associate file by calling DBEDDA. Control is then returned to the calling routine.

If the user wants the field to be associated, he is prompted on which associated file the field is to be placed. If no field name is entered, a new associate file header is created and posted. If a fieldname was entered, it must exist and be associated. If the entered name does not meet

this criteria, the user is given a diagnostic and reprompted for which associated file. If the entered field is associated, then this associated file header record is addressed, and posted.

The field descriptor structure is then updated after which control is returned to the calling routine.

Upon entry into DBEDGF the user is prompted for a fieldname. The field name is then verified.

If the name is invalid, the user is given a diagncstic and reprompted for a correct field name. The field name validation criteria is set up by the calling routine. It consists of the message id to prompt with, whether it is to be a new field name or if it must already exist and whether or not it can be a reserved fieldname.

If it is a new fieldname and the caller so requests, a new FLD structure will be allocated and the fieldname posted therein.

If the entered name is unacceptable for any of the prestated reasons, the error flag is turned on. Contrcl is then returned to the calling routine.

Upon entry into DBEDGR, a control loop is setup to prompt for each of the 3 routine names: conversion, fcrmatting, and validation routines names. If a name is entered, it is verified. For conversion and formatting routines if no name is entered and if the field type is not alphanumeric, a default routine name is supplied. The routine names are posted in the FLD structure.

If a validation routine was specified, a validation argument is prompted for. To be valid it must contain an even number of hexadecimal characters. If an argument is entered, the value is saved in a VALID structure, the pointer of which is posted in the FLD structure. Control is then returned to the calling routine.

Upon entry into DBEDPG1 the user is prompted for a paging direction and it is validated. If it is in error the user is given a diagnostic and reprompted fcr the direction. If in the desired direction, there is no more information to output, the user is given a diagnostic and control is returned to the calling routine. The appropriate routine is called to setup the next page of

information to be displayed, after which control is returned to the calling routine.

Upon entry into DBEDPG2 if there is more information to output the MORE DATA flag is turned on. If it is possible to page in either direction, a paging entry point is setup. The information is flushed from the buffer to the screen after which control is returned to the calling routine.

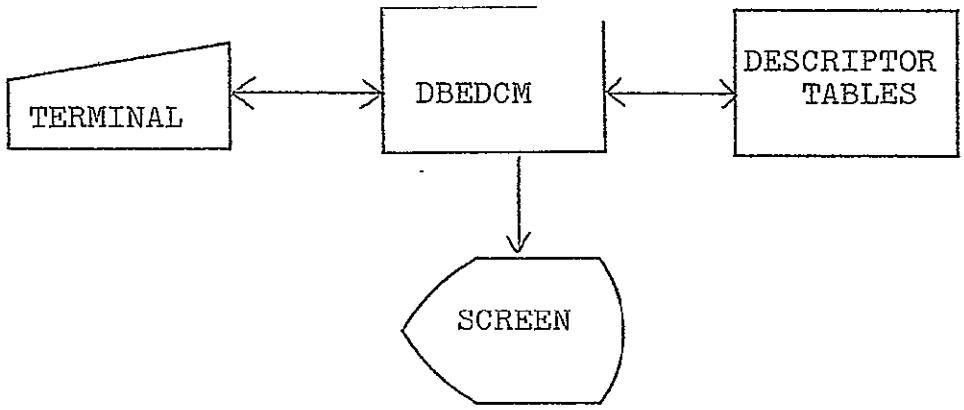
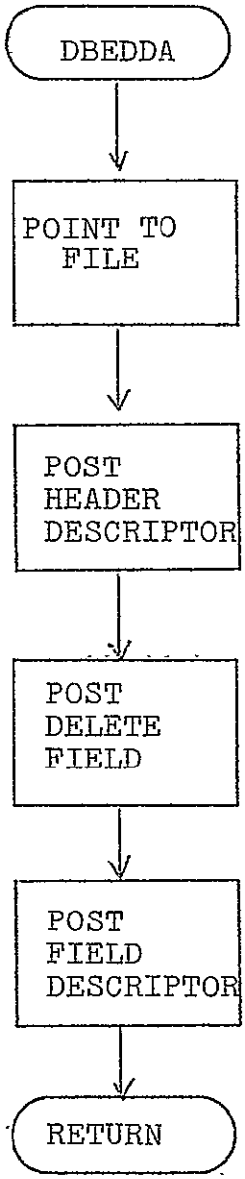


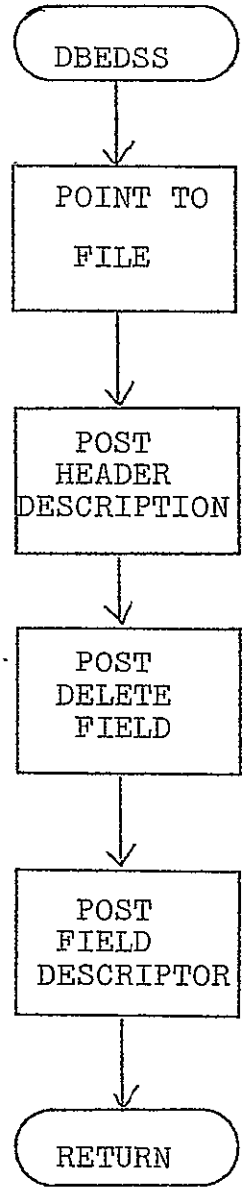
Figure 1 - I/O Block Diagram

TD: D.17.1


DBEDDA
POINT TO FILE
POST HEADER DESCRIPTOR
POST DELETE FIELD
POST FIELD DESCRIPTOR
RETURN

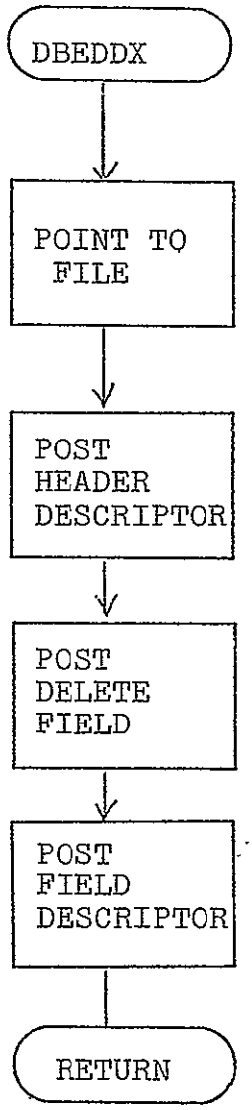
17.2

CM5


DBEDSS
POINT TO FILE
POST HEADER DESCRIPTION
POST DELETE FIELD
POST FIELD DESCRIPTOR
RETURN


17.3

CM6


DBEDDX
POINT TO FILE
POST HEADER DESCRIPTOR
POST DELETE FIELD
POST FIELD DESCRIPTOR
RETURN

[illegible] 17.4

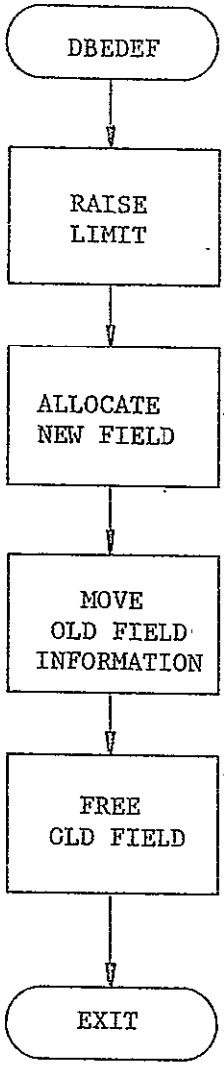
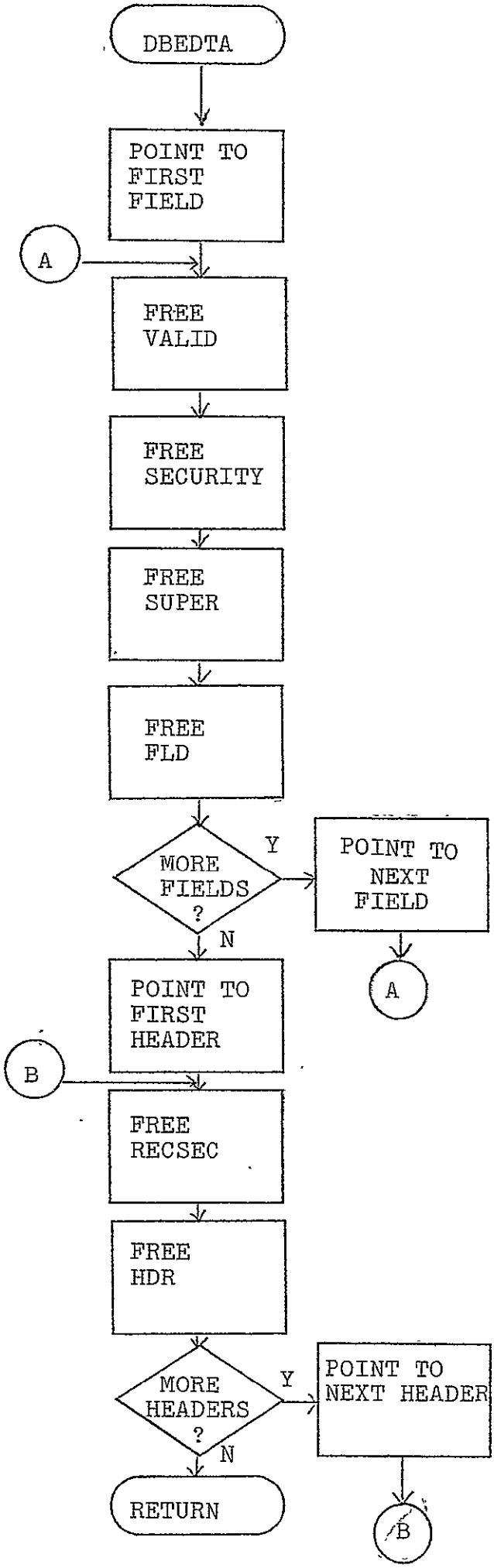


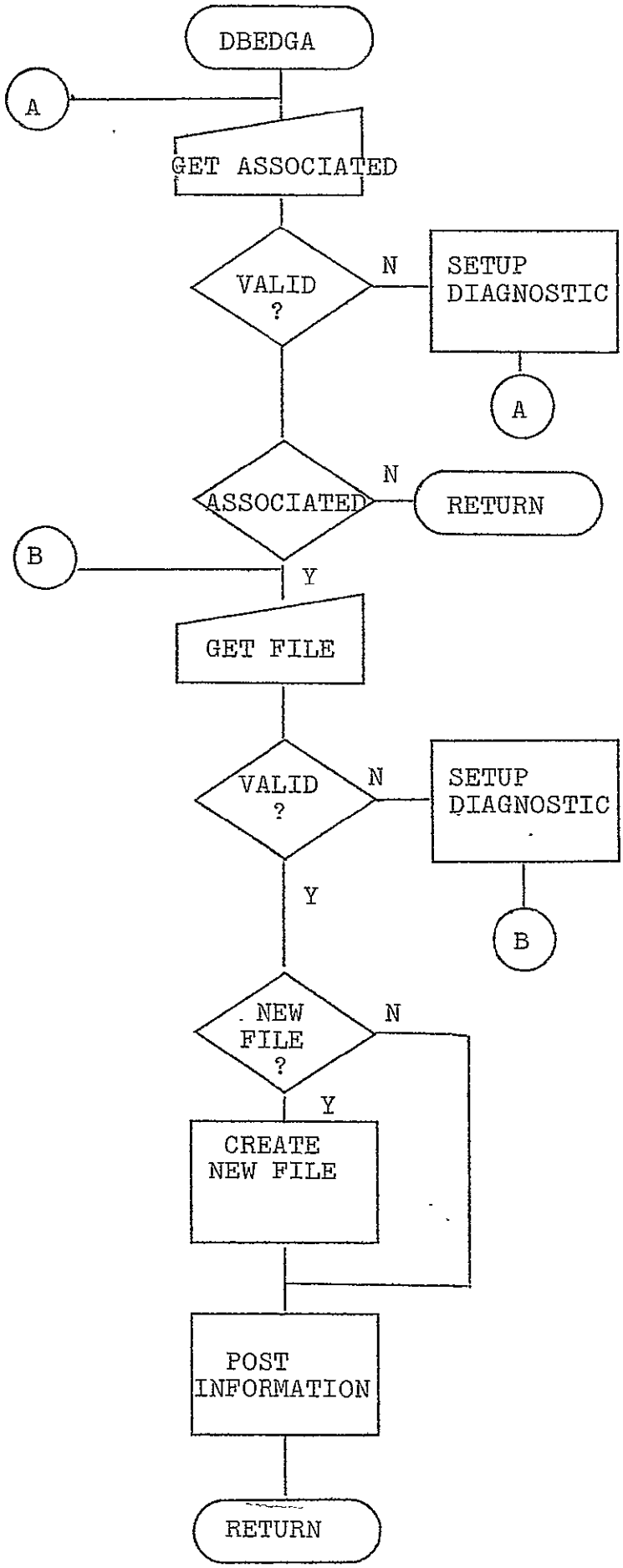
Figure 2. Top Level Flowchart

IV. 15.17.5

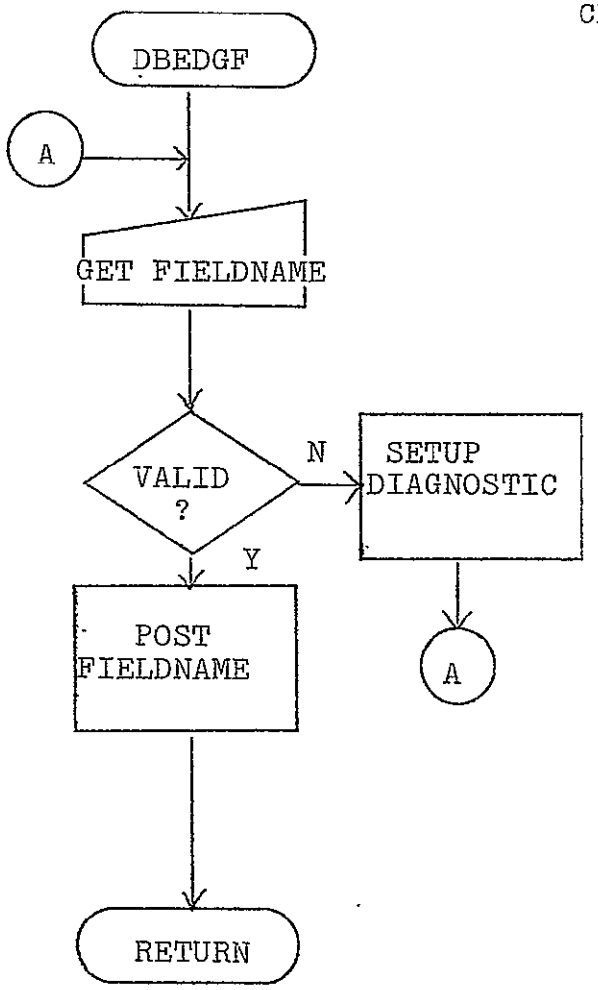

DBEDTA
POINT TO FIRST FIELD
A
FREE VALID
FREE SECURITY
FREE SUPER
FREE FLD
MORE FIELDS ?
Y
POINT TO NEXT FIELD
A
N
POINT TO FIRST HEADER
B
FREE RECSEC
FREE HDR
MORE HEADERS ?
Y
POINT TO NEXT HEADER
B
N
RETURN

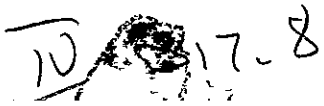
10 ס 17.6

CM9

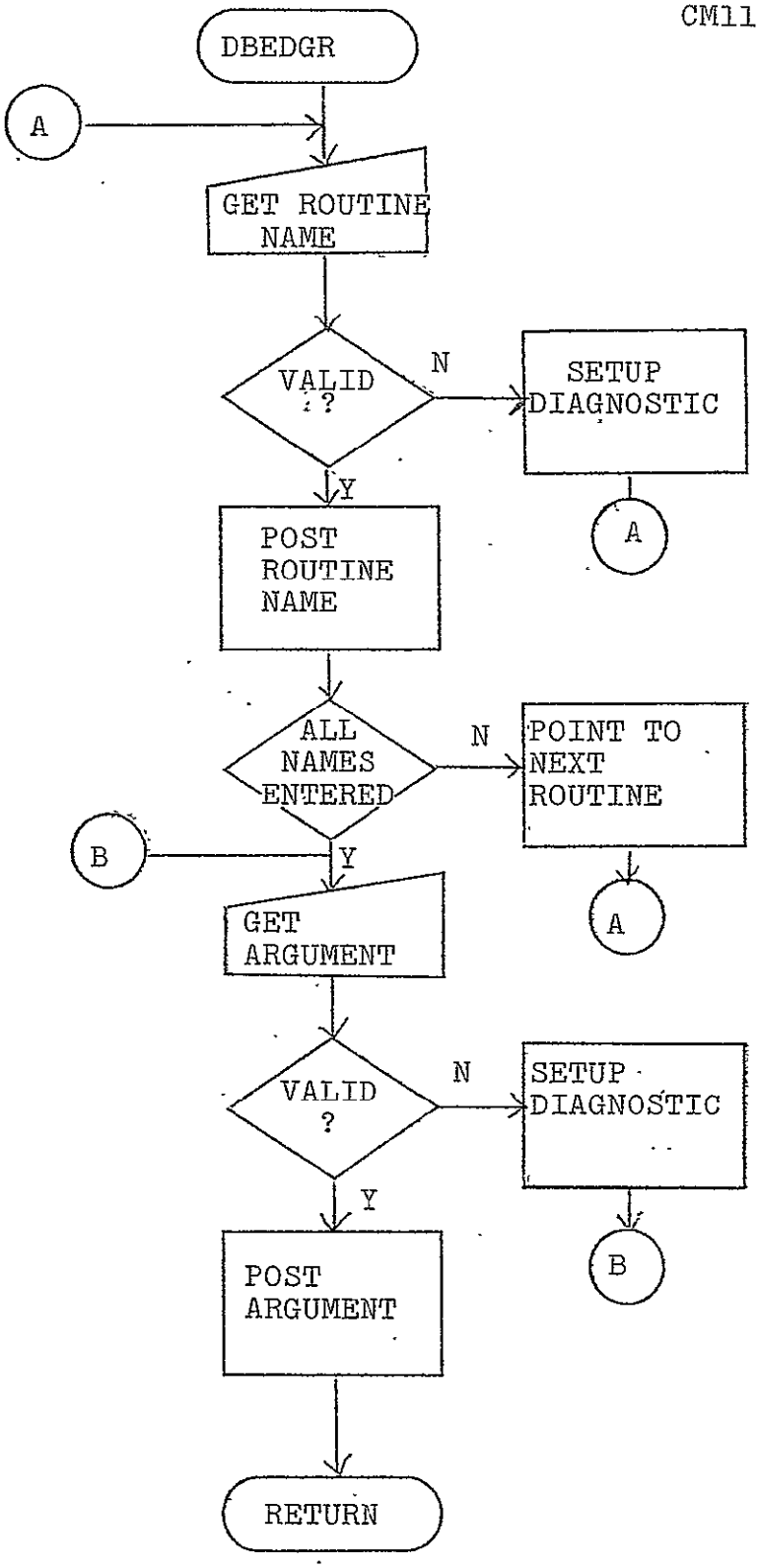

DBEDGA
A
GET ASSOCIATED
VALID ?
N
SETUP DIAGNOSTIC
A
ASSOCIATED
N
RETURN
B
Y
GET FILE
VALID ?
N
SETUP DIAGNOSTIC
B
Y
NEW FILE ?
N
Y
CREATE NEW FILE
POST INFORMATION
RETURN

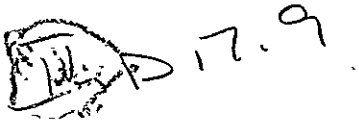
IV 5 12.7


DBEDGF
A
GET FIELDNAME
VALID ?
N
SETUP DIAGNOSTIC
Y
POST FIELDNAME
A
RETURN

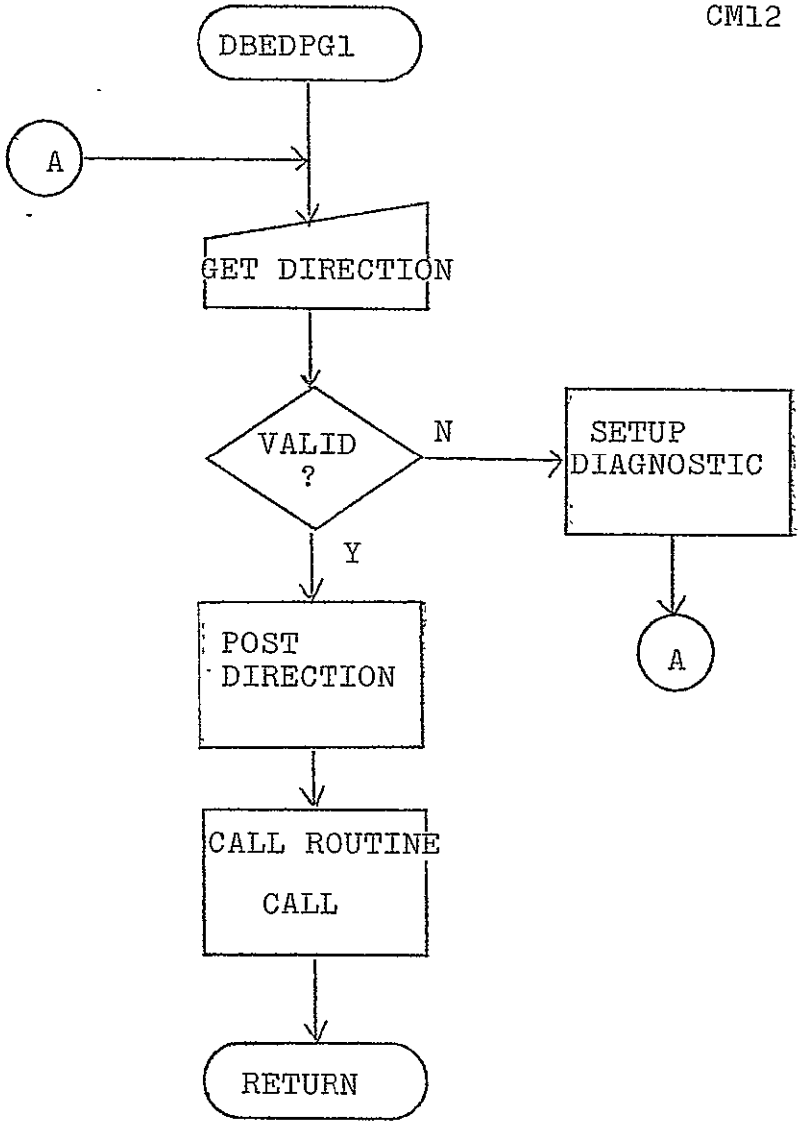


CM11


DBEDGR
A
GET ROUTINE NAME
VALID ?
N
SETUP DIAGNOSTIC
A
Y
POST ROUTINE NAME
ALL NAMES ENTERED
N
POINT TO NEXT ROUTINE
A
B
Y
GET ARGUMENT
VALID ?
N
SETUP DIAGNOSTIC
B
Y
POST ARGUMENT
RETURN

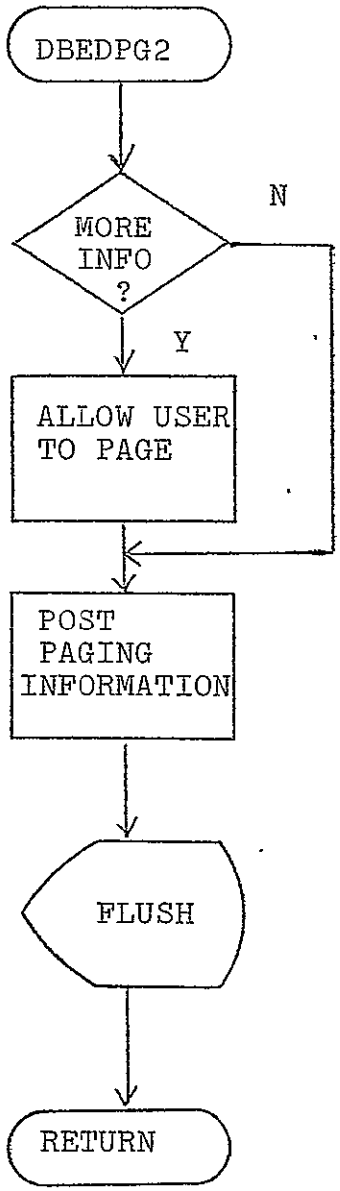


CM12


DBEDPG1
A
GET DIRECTION
VALID ?
N
SETUP DIAGNOSTIC
Y
A
POST DIRECTION
CALL ROUTINE CALL
RETURN

IV > 17.10

CM13


DBEDPG2
MORE INFO ?
N
Y
ALLOW USER TO PAGE
POST PAGING INFORMATION
FLUSH
RETURN

TOPIC D.18 - DESCRIPTCR EDITOR - FIELDS COMMAND

A. MODULE NAME

Program-ID - RDBEDFD
Module-ID - DBEDFD
Entry Points
DBEDFD1 - FIEIDS Command
DBEDFD2 - Paging Entry

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

In CREATE mode the FIELDS command outputs the names of the fields thus far created. In UPDATE mode the descriptor descriptor names are displayed.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Cutput Files

Not Applicable

b. On-Line Terminal Displays

The fieldnames are placed on the screen, the number of names per line determined by dividing the screen width by 20.

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDFD:

1. FIELD
2. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

If in CREATE mode, a pointer is set to the FIELD structure; otherwise in UPDATE mode the pointer is set to an internal list containing the descriptor descriptor field names.

At the paging entry the proper page number is set up in the paging information structure.

At this point, the code becomes common for both the command and paging entry points. The number of the next field name to be displayed is obtained from the paging information structure. Two control loops are set up, one to build every line to fill the screen and the other to fill each line of the screen.

If there is more information to be displayed, the paging entry point is set up. The paging information structure is posted, the buffer is flushed to the screen and control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

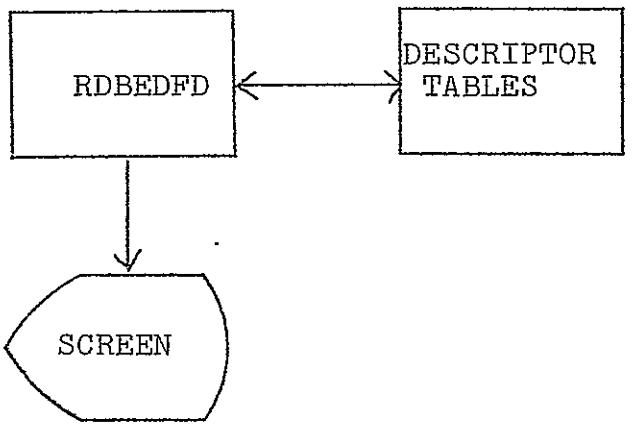


Figure 1. I/O Block Diagram

- $\overline{IV} > 18.1$

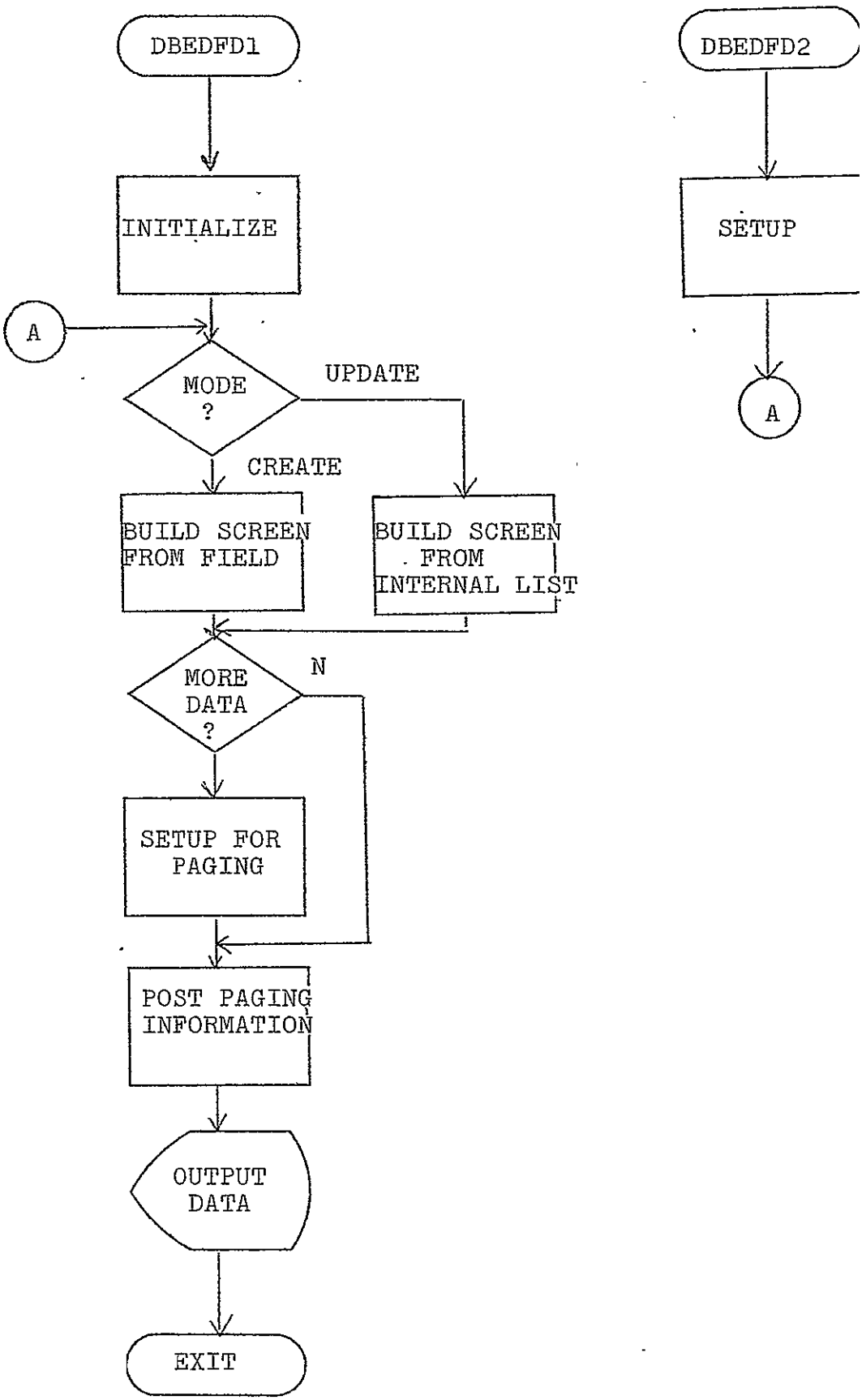


Figure 2. Top Level Flowchart

$\overline{IV}$ $\nu > 18.2$

?

TOPIC D.19 - DESCRIPTOR EDITOR - FILE COMMAND

A. MODULE NAME

Program-ID - RDBEDFI
Module-ID - DBEDFI

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module is used to place those additions and/or changes from the descriptors in core to the descriptor file.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

C. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

The descriptor file is a region TSS VISAM dataset containing all the information necessary to completely define the data base.

b. On-Line Terminal Displays

Not Applicable

C. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDFI:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC
7. SECURITY
8. SUPER
9. VALID
10. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entering FILE command, if just one descriptor record is to be updated, the appropriate file identified is setup, the file opened and the descriptor record is updated after which control is returned to the calling routine. Otherwise all the descriptor information is to be filed to the descriptor file. The user is prompted for the parameter DESCOK and the value saved for posting each header record. If the input value is in error the user is given a diagnostic and prompted for a new value.

If the anchor key field needs to be deleted, it is deleted from the anchor and all associate files. The delete file list is then processed deleting the header, RECLEN key field and when applicable the parent key field of all files listed.

For outputting descriptor information, the files are processed in the following order: anchor, all associate files, then all subfiles. If the file does not exist on disc the RECLEN field is written out. If it is a new region and the file is either the anchor file or an associate file the

anchor key field must be written out for a new subfile the subfile key field and parent key field are written out. If these or any other fields already exist on the file then only the changes if any, to these fields are written out. Record security if any is then written out.

As these fields are output the field position value for each field in maintained and updated. This value is then placed in the FLDPOSIT position for each field.

The packed bit fields for the file are then processed in the order in which they appear in the list. They are packed four to a byte and the field position and field length indicating which byte and where in the byte respectfully the field can be found. After all packed bits fields are processed, the fixed fields for the file are processed shipping over the key field, parent key field and record security field where applicable. Then all varying fields are processed in order.

If it is an anchor or associate file all descriptors if any are set up and processed, Then the header record is setup and processed and the file closed. The next file is processed in this manner until the anchor file, all associate files and all subfiles are processed.

The index files are processed next. If it is a new file the RECLEN field is written out. Each field to be indexed on this file is located, setup and written out. The anchor or key field on the appropiate subfile key field is setup and written out. If the index file already exists then only those changes applicable are written out to the dataset. Each index file is processed in this manner until all index files have been processed.

After all the fields have been processed the various external structures are marked indicating that the descriptor data is on the dataset. The command string is saved in the current strategy and control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DBPL/I and TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

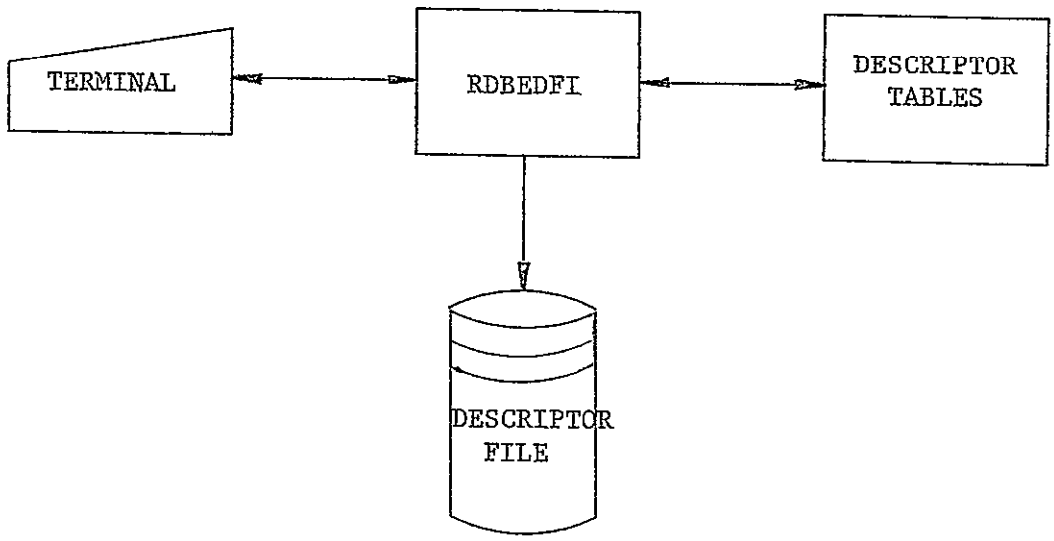


Figure 1. I/O Block Diagram

19.1

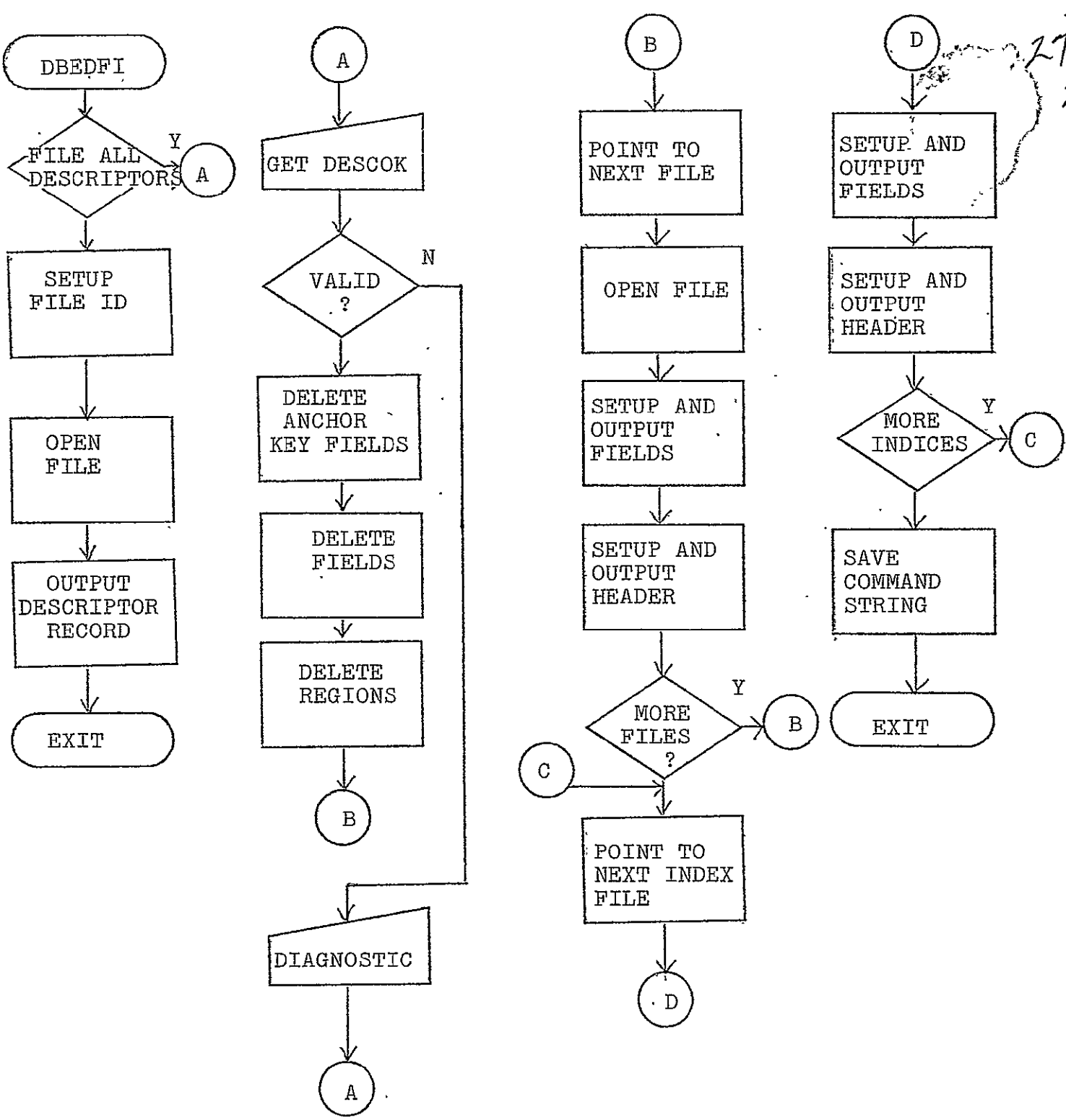
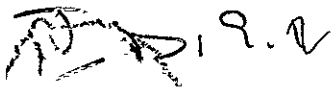


Figure 2. Top Level Flowchart



TOPIC D.20 - DESCRIPTOR EDITOR - FIELD SECURITY COMMAND

A. MODULE NAME

Program-ID - RDBEDFS
Module-ID - DBEDFS

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This command is used to define and setup field security for a field or a group of fields.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDFS:

1. FIELD
2. FLD
3. SECURITY
4. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Routine DBEDGF is called to obtain an existing field name.

If CREATE mode the user may enter up to 18 field names at one time. If several field names are entered, they are processed as above.

The user is then prompted for a security code and an add-delete indicator. If no indicator is present, add is assumed. If the indicator is invalid or the security code is not an alphanumeric character string, the user is given a diagnostic and prompted for a new security code value.

The used may enter up to 18 security codes in a parenthesesed list.

If the field already has field security a pointer is set to it, otherwise a SECURITY work area is allocated.

If the security code is to be added to the list, and it is not already there, it is added at the end of the list. Otherwise it is ignored. If the security code is to be deleted, a search is made through the existing codes deleting all occurrences of the code if any.

Once all of the entered security codes have been processed a check is made if any security codes are lift on this field. If none, then the SECURITY work area is released, else the pointer to SECURITY is posted in the FLD structure.

If in UPDATE mode, routine DBEDFI is called to post the security codes to the descriptor file.

The command string is saved in the current strategy and control is then returned to the calling module.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

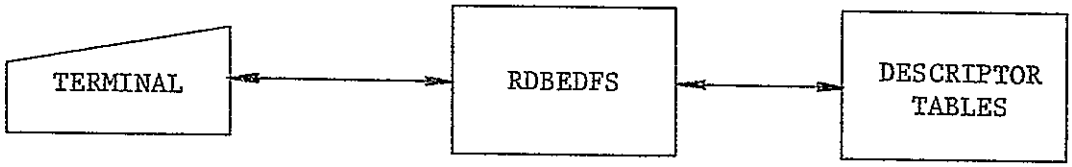
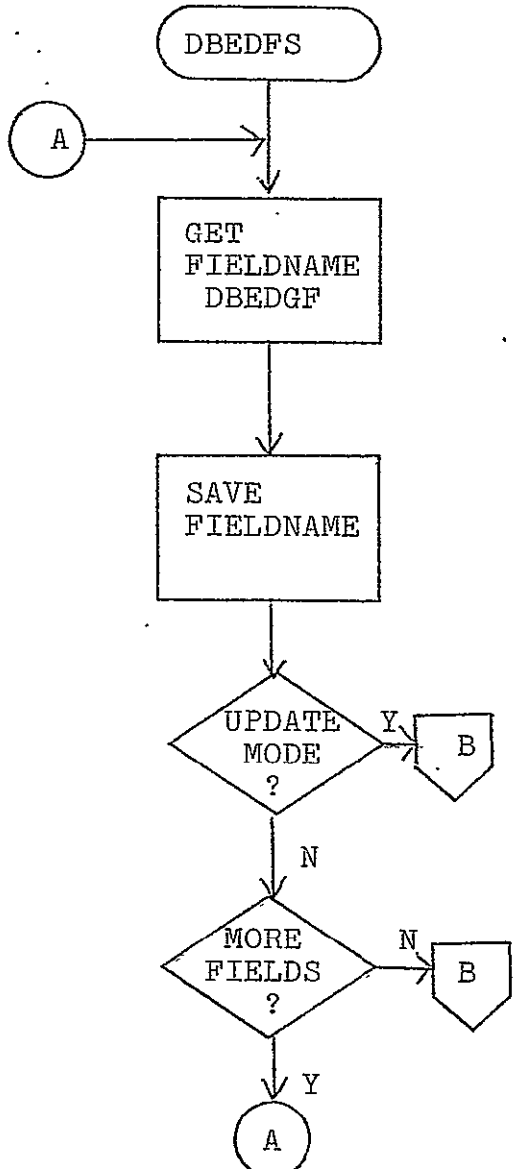
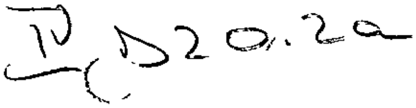


Figure 1. I/O Block Diagram


DBEDFS
A
GET
FIELDNAME
DBEDGF
SAVE
FIELDNAME
UPDATE
MODE
?
Y
B
N
MORE
FIELDS
?
N
B
Y
A



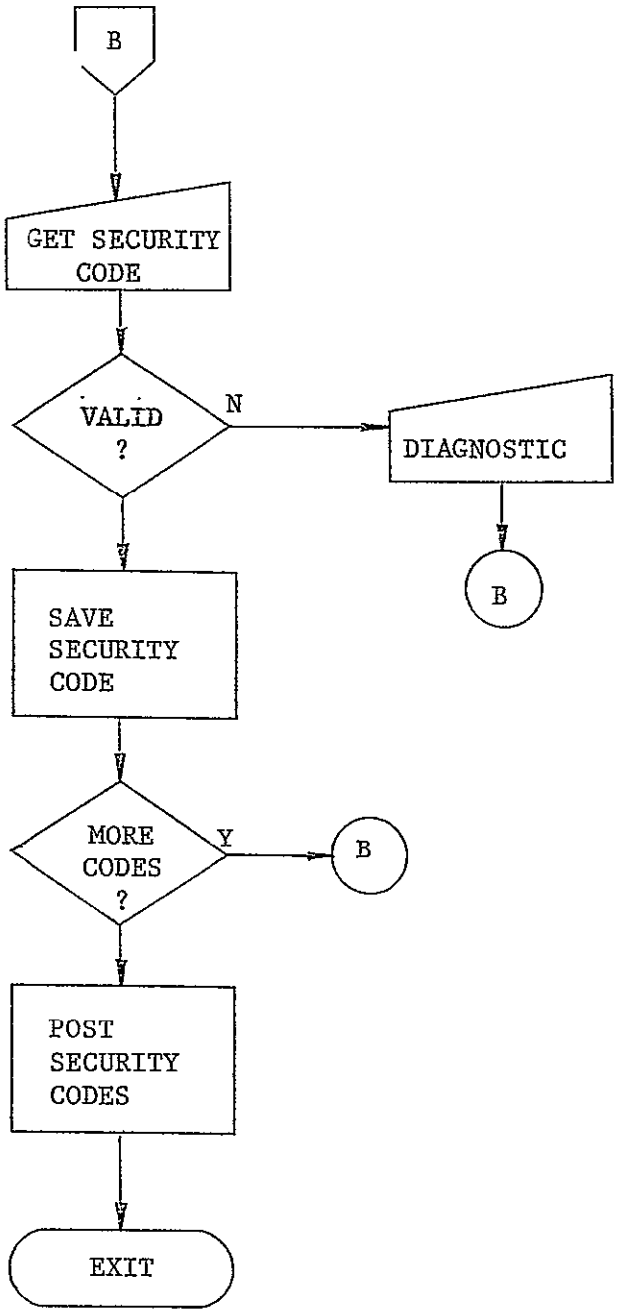


Figure 2b. Top Level Flowchart

TOPIC D.21 - DESCRIPTOR EDITOR - LOAD DESCRIPTORS MODULE

A. MODULE NAME

Program-ID - RDBEDLD
Module-ID - DBEDLD

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

In create mode the load module loads and sets up all field and header descriptor information. In update mode the load module loads the desired descriptor record, including file descriptors and dummy descriptor records.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The descriptor file is a region TSS VISAM dataset containing all the information necessary to completely define the data base.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDLD:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC
7. SECURITY
8. SECURITY_STR
9. SUPER
10. VALID
11. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into DBEDLD if all descriptors are to be loaded, the anchor file is first pointed to, otherwise the appropriate file identifier is set up. If call from REVIEW command branch to retrieve the appropriate header on field descriptor fields as the file has been opened and the appropriate descriptor read into core.

In update mode any fields which have been loaded and still exist in work areas are released. This is a control so that no more than one field descriptor can be loaded at any one time. Note: this is not true for header descriptor.

The next descriptor region is opened starting with the anchor region and the descriptor header record read in. The header fields are obtained and all bit switches converted to an alphabetic character. A HDR structure is allocated and the header information saved therein. If the file has record security, the security codes obtained, placed in a

RECSEC structure and hooked up to the HDR structure.

If in update mode, the desired field descriptor record is read in, otherwise the next sequential field descriptor is read in. If not in review mode, it must be determined if the field is a dummy descriptor. If it is then a list of file ids is built eventually containing all of the descriptor regions on the file once all of the field descriptors on the anchor file have been processed. This list is built from non-blank entries in the ASSOCFIL, INVFILE and SUBFILE descriptor fields. If the field is a dummy, and in update mode, the correct file is pointed to and a branch goes to open the file and read the desired field descriptor. In create mode, this record is skipped and the next descriptor record in the region is read.

If this field descriptor is saved, all of the field descriptor bit field values are translated to an alphabetic character.

The field validation argument, if any, is obtained and saved. If the field is a superfield, the component values are obtained and saved. Likewise, if the field has security, the security codes are obtained and saved.

A FLD structure is allocated and the field information saved therein. The field name and pointer are posted in the next available slot in the FIELD structure, and if in create mode, the FLD structure is chained to the end of the proper file list.

When the anchor region is finished. A list of all existing descriptor regions is complete. The next descriptor region in that list is selected and loaded as described.

In review mode once the desired descriptor record from the desired descriptor region has been processed, as the correct non dummy field descriptor has been loaded in update mode, control is returned to the calling routine.

In create mode a search is made through all file lists to locate all subfields. For each subfield, the defining base field is located and the base field name and offset are posted in the subfield FLD structure.

The fields within the file lists are ordered by their field positions within each file list with all subfields and superfields appearing at the end of the ordered lists. Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

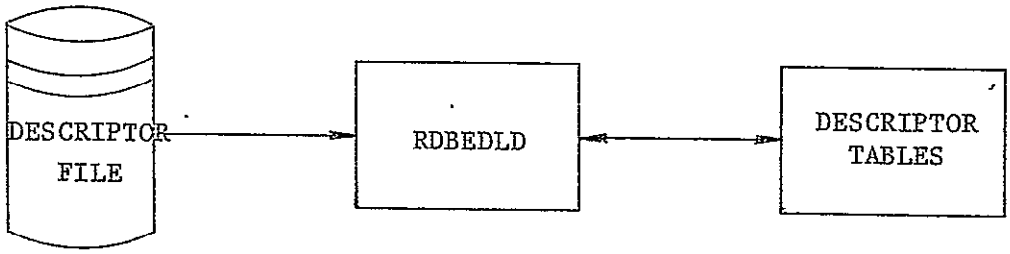


Figure 1. I/O Block Diagram

١٧ آذار ٢٠١٠

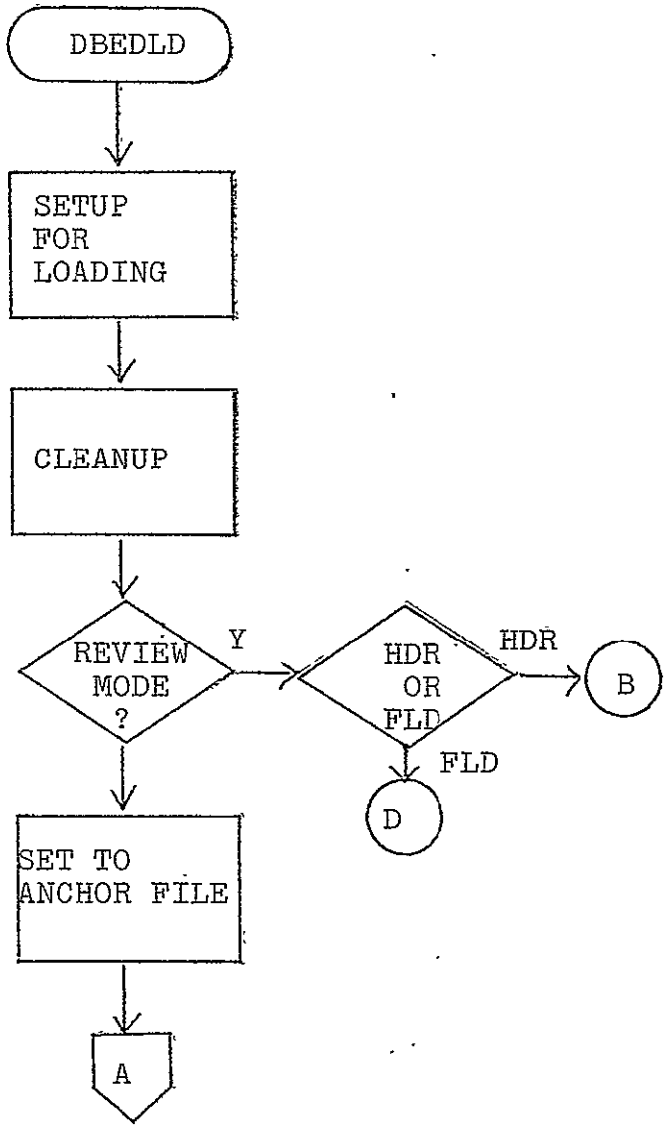


Figure 2a. Top Level Flowchart

IV 52.21.2a

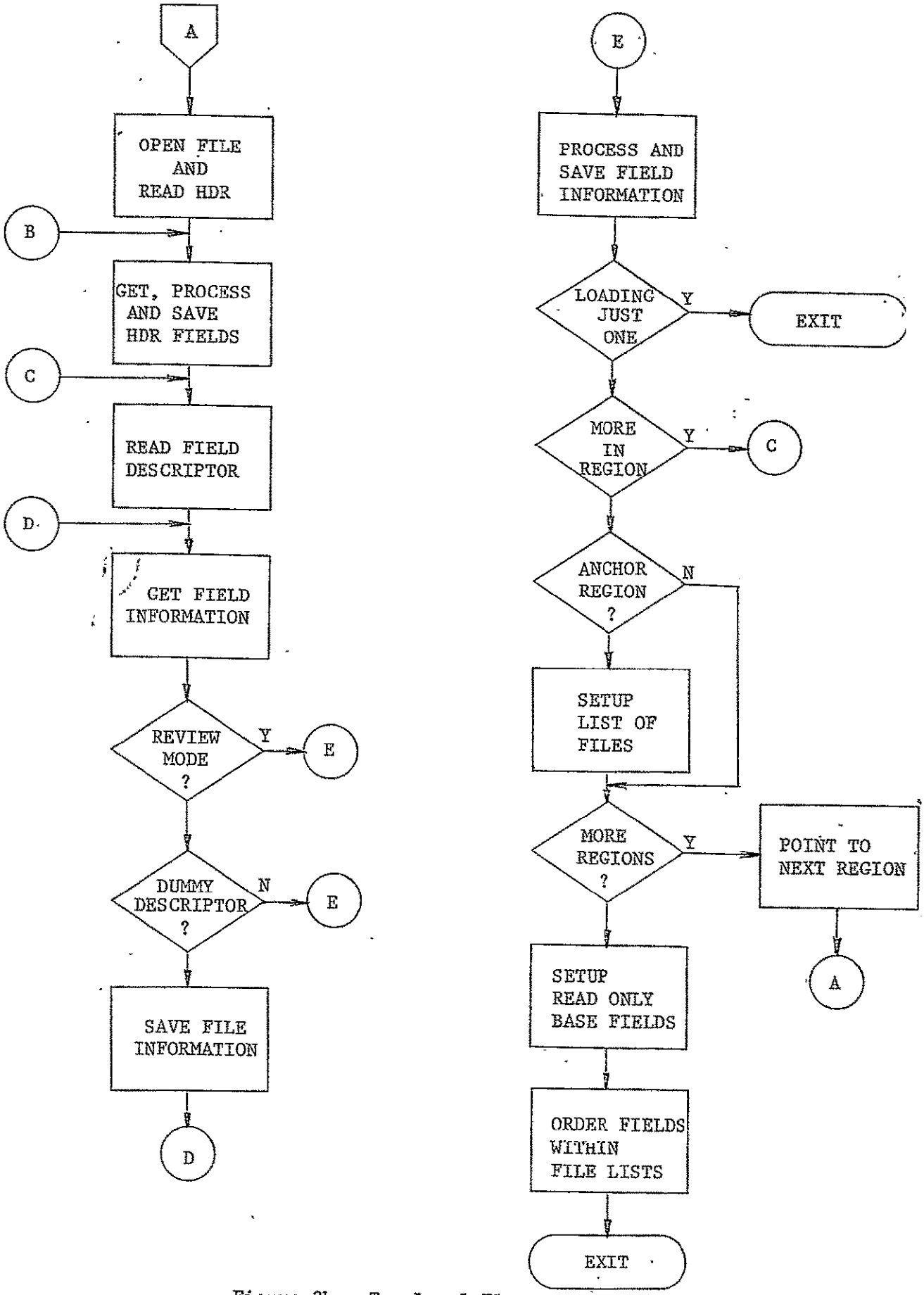


Figure 2b. Top Level Flowchart

正反 21.26

TOPIC D.22 - DESCRIPTOR EDITOR - MOVE COMMAND

A. MODULE NAME

Program-ID - RDBEDMO
Module-ID - DBEDMO

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The MOVE command permits the user to reorder fields within any field list.

D. DATA REQUIREMENTS

1. I/O BLOCK DIAGRAM

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDMO:

1. FLD
2. HDR
3. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The user is prompted for the new position field name. If the entered field name does not exist, the user is given a diagnostic and prompted for a new fieldname. The new position field name cannot be, the anchor key field if the anchor file has record security, the subfile parent key field if the subfile has record security or the subfile key field, or the RECLEN field. If any of these conditions are met, the user is given a diagnostic and reprompted for a new position fieldname. A superfield has no field position. If a subfield is specified, the defining base field is located and used as the new position fieldname. All other fields are unacceptable.

The user is prompted for the field to be moved. To be valid, the field must exist and must not be a reserved fieldname, must appear in the same field list as the new position field name and must not be a superfield or a subfield. If the field is invalid, the user is given a diagnostic and reprompted for the field to be moved.

The field to be moved is decoupled from the list by resetting the appropriate forward and backward pointers. It is then threaded into its new position by setting the appropriate forward and backward pointers.

The command string is saved in the current strategy and then control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

   PL/I with TSPL/I statements.

2. Suggestions and Techniques

   Not Applicable

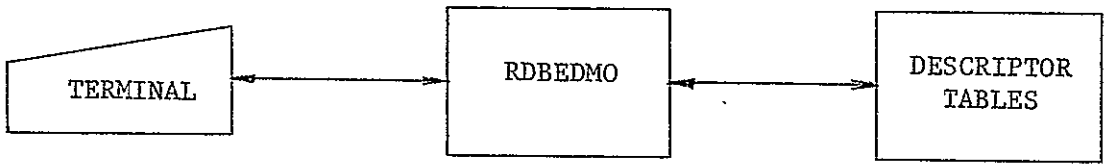


Figure 1. I/O Block diagram

IV ▷ 22.1

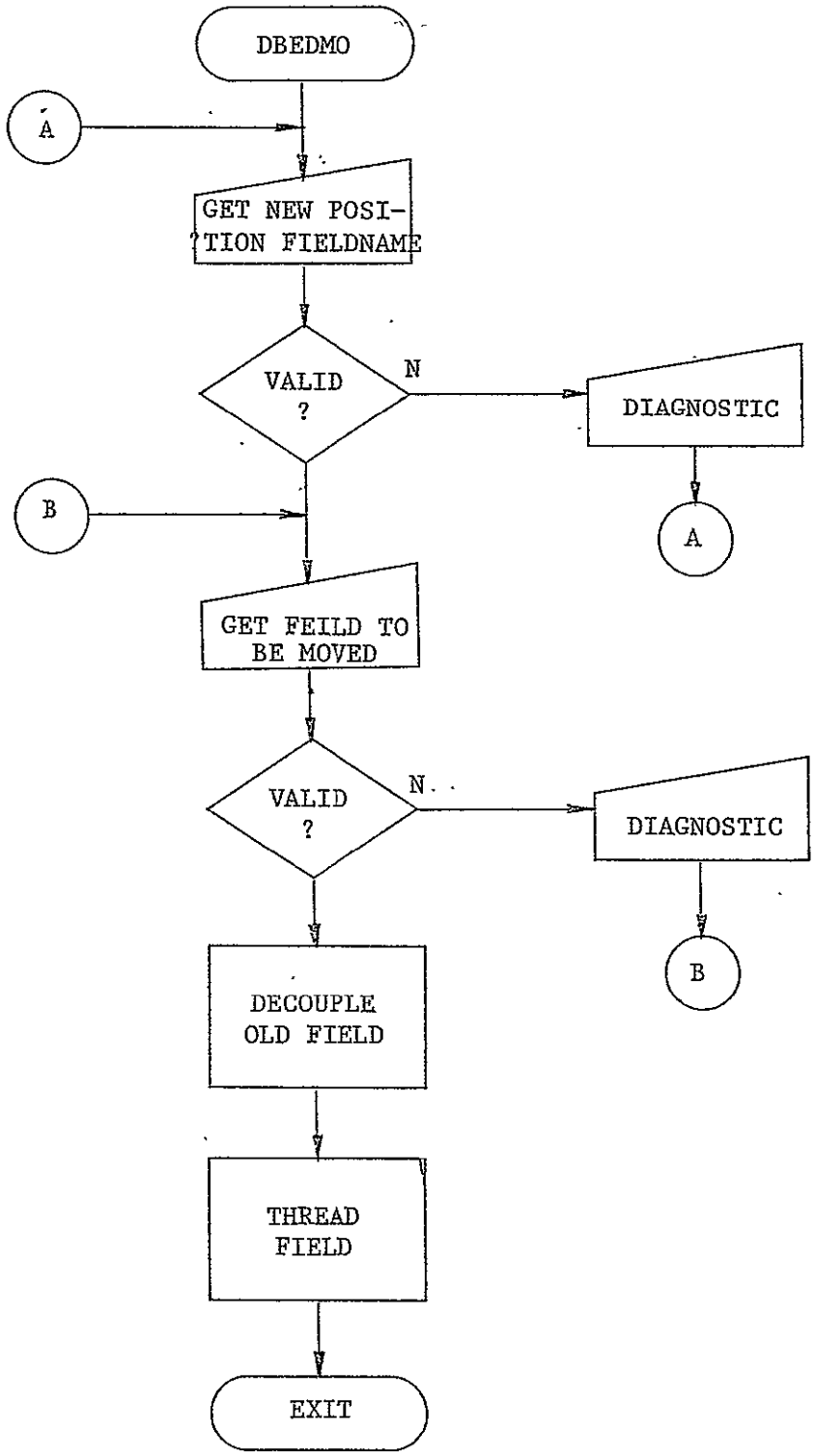


Figure 2. Top level flowchart

IV. 22.2

TOPIC D.23 - DESCRIPTOR EDITOR - PATCH COMMAND

A. MODULE NAME

Program-ID - RDBEDPA
Module-ID - DBEDPA

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The Patch command permits the user to patch the value in any descriptor record in any description region in the descriptor file. The record to be patched must be identified by use of the REVIEW command.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Cutput Files

Not Applicable

b. On-Line Terminal Displays

Nct Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDPA:

1. FLD
2. HDR
3. RECSEC
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The REVIEW command indicates in X that a REVIEW has been done and it is alright to patch, REVIEW also indicates whether the field to be patched is a field or header descriptor. If a REVIEW has not been done the user is given a diagnostic and control is returned to the calling routine.

The user is prompted for his patch in the form "keyword=test". The keyword is checked to see if valid. If not, the patch is ignored, the user given a diagnostic and reprompted for the patch. If the name is valid, a branch is taken to the piece of coded which actually posts the appropriate field.

In each of the sections of code, one for each descriptor field name, a resonableness check is made on the patch text, to assure that the data will be accepted by the validation routines when posting the information to the descriptor file.

Refer to the Descriptor Editor Users Guide for the acceptable range and form of the patch texts.

The user may enter a parenthesesed list of patches.

After all the patches have been posted in the descriptor table work areas, they must then be

posted to the descriptor data set. The routine DBEDFI3 is called to post the appropriate field descriptor, or the routine DBEDFI is called to post the appropriate header descriptor. The routine called depends on whether the user is patching a field descriptor or a header descriptor. Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

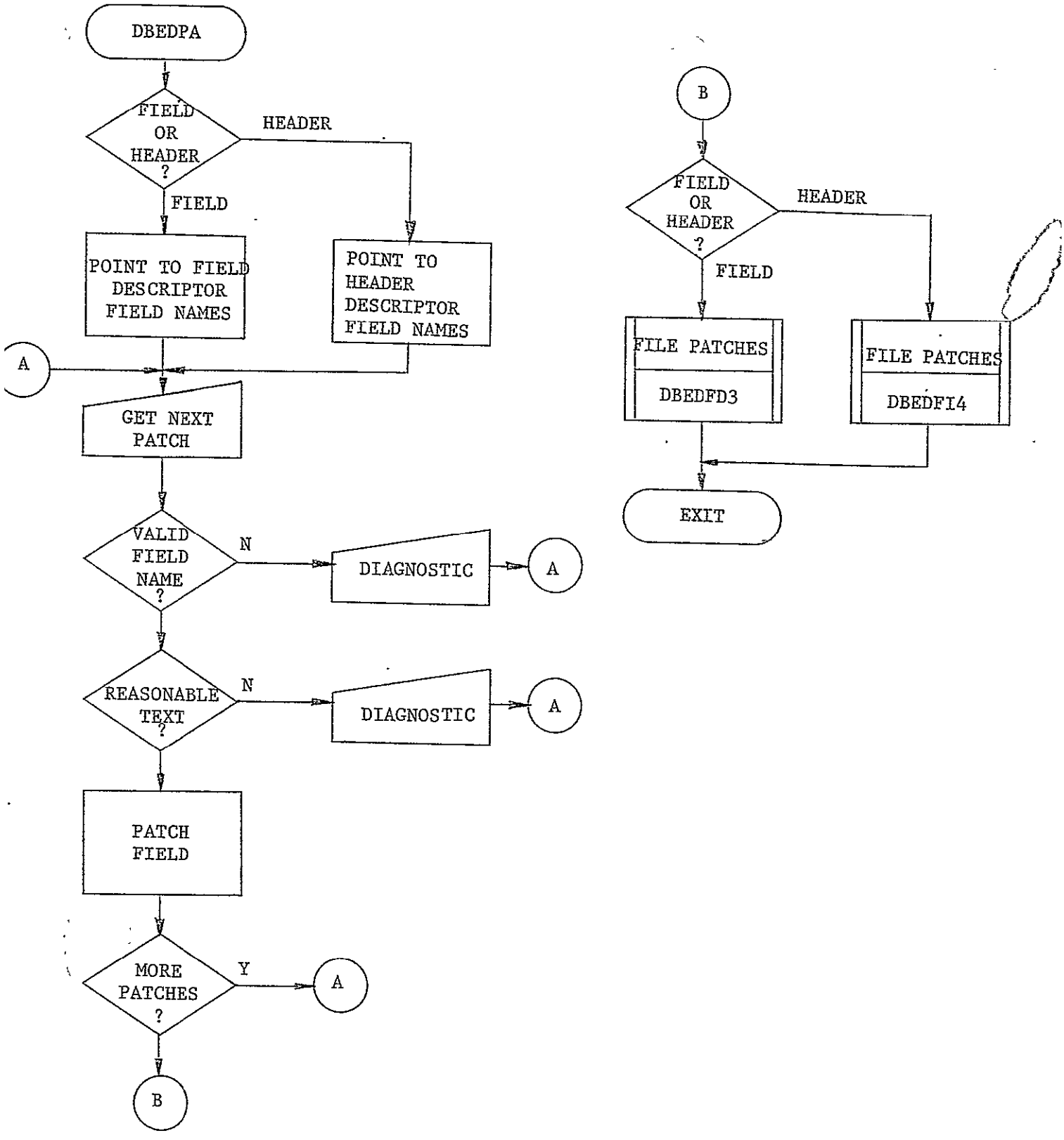
PL/I with TSFI/I statements.

2. Suggestions and Techniques

Not Applicable

Figure 1. I/O Block diagram

IV D23.1


DBEDPA
FIELD OR HEADER ?
HEADER
FIELD
POINT TO FIELD DESCRIPTOR FIELD NAMES
POINT TO HEADER DESCRIPTOR FIELD NAMES
A
GET NEXT PATCH
VALID FIELD NAME ?
N
DIAGNOSTIC
A
REASONABLE TEXT ?
N
DIAGNOSTIC
A
PATCH FIELD
MORE PATCHES ?
Y
A
B
B
FIELD OR HEADER ?
HEADER
FIELD
FILE PATCHES
DBEDFD3
FILE PATCHES
DBEDFI4
EXIT

TOPIC D.24 - DESCRIPTOR EDITOR - PRINT COMMAND

A. MODULE NAME

Program-ID - RDBEDPR
Module-ID -DBEDPR

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The PRINT command gives the user a formatted printout of the descriptor information as it exists in core at the time the print is issued.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

The output data from RDBEDPR is placed in the VSAM data set LIST.DESC(0) from where it is printed on the high speed printer by TSS. For the details of the data set refer to the dataset specifications.

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

The information stored in LIST.DESC(0) is printed using column one of each record as a carriage control.

4. Reference Tables

The following external tables are referenced by RDBEDPR:

1. FIELD
2. FLD
3. HDR
4. RECSEC
5. SECURITY
6. SUPER
7. VALID
8. X

A description of these tables can be found dataset specifications in the of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The generation data group LIST.DESC is created using the routine ASMCAT. The next generation is created using the routine ASMDD. A DCB is created for the output file by the routine ASMDCB, the JFCB set up by the routine ASMFNDS, and the dataset opened by the routine ASMOPEN.

The title lines for the data base name are output by the routine ASMPUT. The data base name is output followed by two trailing title lines.

The title lines for the field descriptors are output. The lines of field information for each field are built and output.

After the field information is processed, the title lines for the header descriptor information are written out. The lines of header information for each descriptor region are built and written out.

The LIST.DESC dataset is closed by calling the

routine ASMCLOS and a printer listing of the information is generated by using the routine ASMPR after which control is returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

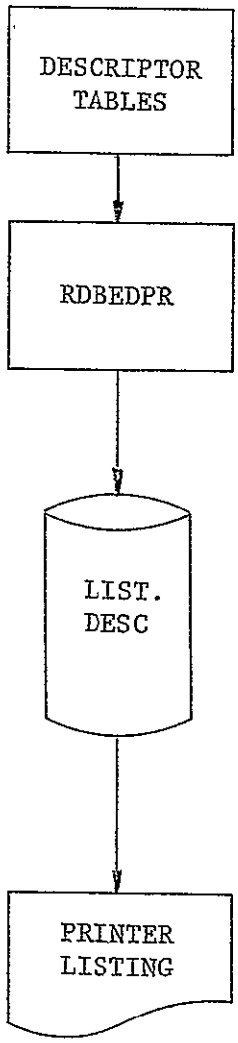


Figure 1. I/O Block diagram

IV D24.1

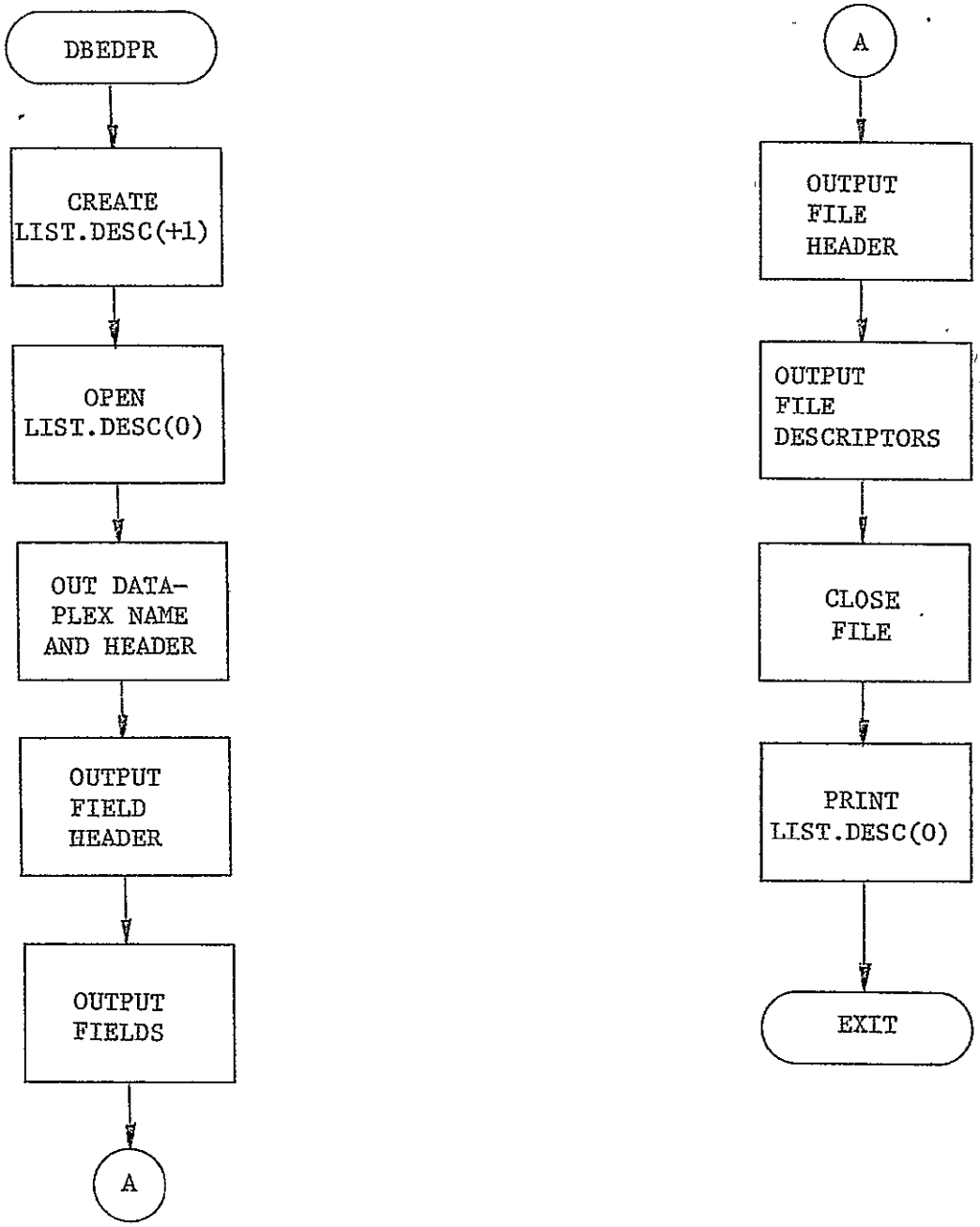


Figure 2. Top level flowchart

IV D24.2

TOPIC D.25 - DESCRIPTOR EDITOR - RECORD SECURITY COMMAND

A. MODULE NAME

Program-ID - RDBEDRS
Module-ID - DBERDRS

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This command is used to create and define record security for any data base file except for indicies..

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDRS:

1. FIELD
2. FLD
3. FLD_STRING
4. MDR
5. RECSEC
6. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Routine DBEDGF is called to obtain a fieldname used to define on which file record security is to be placed. If in update mode and the header record is not loaded, DBEDLD is called to load the header. If there is no record security currently defined for the file in UPDATE mode, the user is given a diagnostic and control is returned to the calling routine.

The user is prompted for a record security code. The add-delete indicator is removed from the code and validated. If it is invalid, the security code is rejected, the user is given a diagnostic and reprompted for the security coded. If no indicator is entered, "ADD" is assumed.

The security code is removed from the parameter. If this is not an alphanumeric character string, the security parameter is rejected, the user is given a diagnostic and reprompted for the security code.

The security mask to be valid must be a two digit hexadecimal character string. If it is invalid, the security parameter is rejected, the user is given a diagnostic and reprompted for the security parameter.

Once the security parameter is validated, it is saved in an internal work area. The user may enter a list of security parameters as a list in parentheses. Each security parameter is obtained

from the user and processed as above.

If record security has been previously defined for the file, a pcinter is set up to the file header and record security information. Otherwise a record security field is created and placed in the appropriate position in the fixed field list of the file. A record security save area is allocated and initialized.

A control loop is set up to process each entered security code. The existing security list if any is searched for the entered security code. If the security code exists and the new code is to be added, the two security masks are logically OR'ed together and the result posted in record security structure. If the code is to be deleted, the two security mask are logically exclusively OR'ed and the result placed in the record security structure. If the security code is not in the existing list and it is to be added, it is placed at the end of the existing list. If the code to be deleted and it does not appear in the list, it is ignored. Each security code is processed in this manner.

After all security code have been processed and the record security list is empty, the area is released and the record security field deleted from the file.

If in UPDATE routine DBEDFI is called to post the record security tc the descriptor file. The command string is saved in the current strategy and then control is returned to the calling module.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techninques

Not Applicable

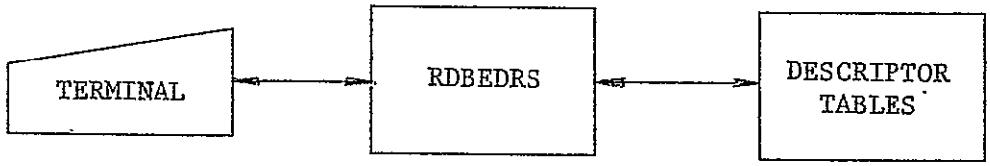
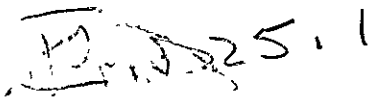


Figure 1. I/O Block diagram



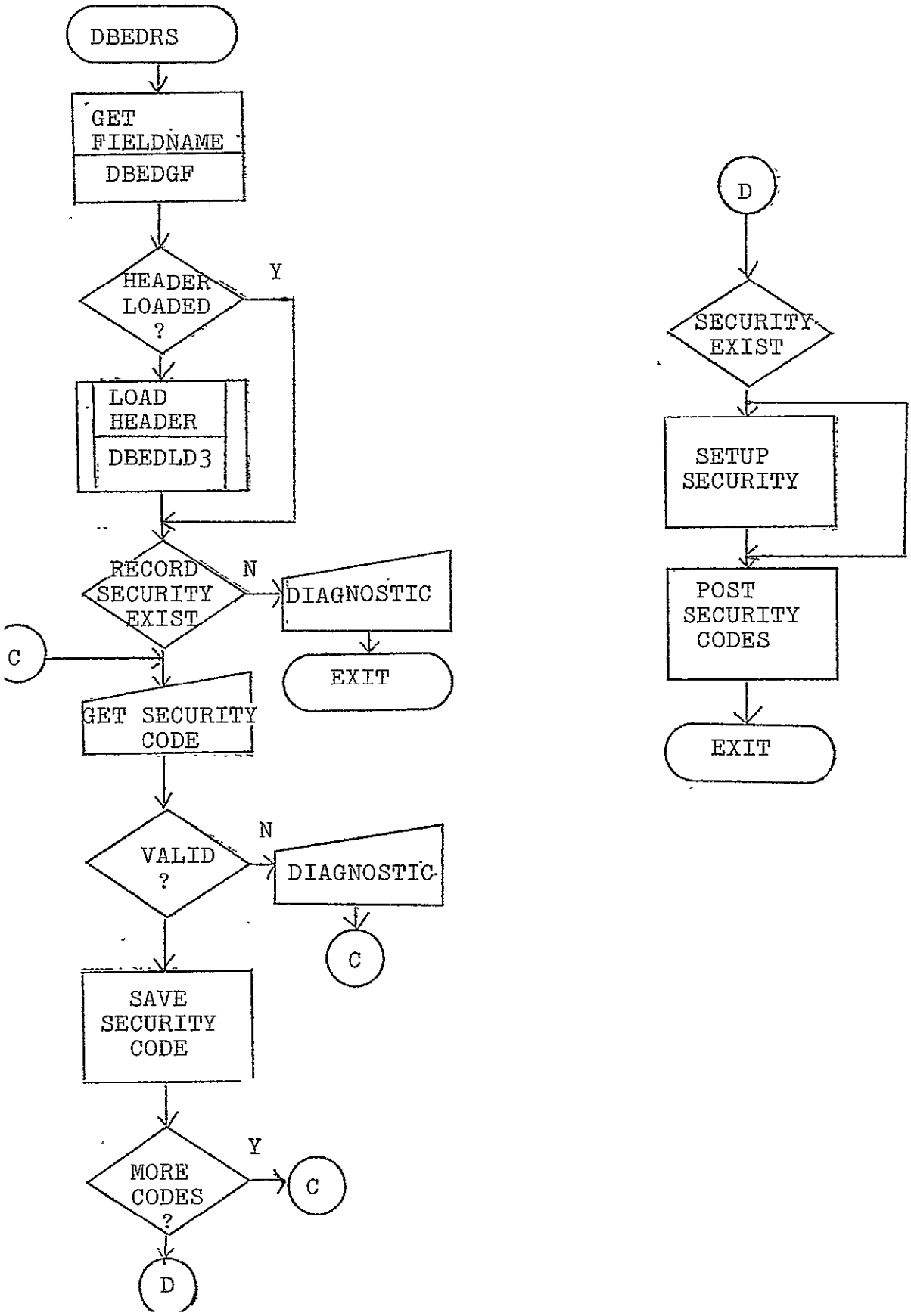


Figure 2. Top level flowchart

四十九号.2

TOPICS D.26 - DESCRIPTOR EDITOR - RESTORE COMMAND

A. MODULE NAME

Program-ID - RDBEDBT
Module-ID - DBEDRT

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This command is used to restore the descriptor tables from a VSAM data set to memory, so that the user may continue to create and/or modify the descriptors from their point of existence at the time the checkpoint was issued.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The input file is a TSS VSAM data set named DESCRP.CHKPOINT. Refer to the dataset specifications for a description of this data set.

3. Output Data Sets

a. Output Files

Not Applicable

b. On Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDRT:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. HDR_STRING
6. RECSEC_STR
7. SECURITY_STR
8. SUPER_STR
9. VALID
10. X

The description of these tables is in the specifications of the dataset DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into RDBEDRT a DCB is set up for the dataset DESCRP.CHKPOINT by calling the routine ASMDCE. ASMDD is then called to create a JFCB for the data set and ASMFNDS is called to setup the JFCB. Any and all existing field descriptors and file descriptors are released and then the FIELD structure itself is released.

ASMOPEN is called to open the dataset. That part of the X structure which was saved is read in over top of the same part of the existing X strucure.

The FIELD structure is allocated next. Note that the variable defining the size of FIELD structure is in that part of X which has just been restored. The FIELD structure is read in overlaying the just allocated existing FIELD structure.

A field descriptor is read into a work area. A FLD structure is allocated on the field information moved into it. If the FLD has field security, a validation argument, or is a super

field, the appropriate structures are allocated, the information moved into them, and the pointers in FLD setup accordingly. The changed flags in FLD are setup so that all of the field descriptor information will be forced out to disc when the FILE command is issued. Each field descriptor is processed in this manner.

A file descriptor is read into a work area. An HDR structure is allocated and the header information moved into it. If the file has record security, a RECSEC structure is allocated, the information moved into it, and the pointer posted into the HDR structure. The HDR pointer is posted into the proper slot in X.HEAD_TAB. Each header descriptor is processed in this manner.

The dataset DESCRP.CHKPOINT is then closed and control is returned to the calling program.

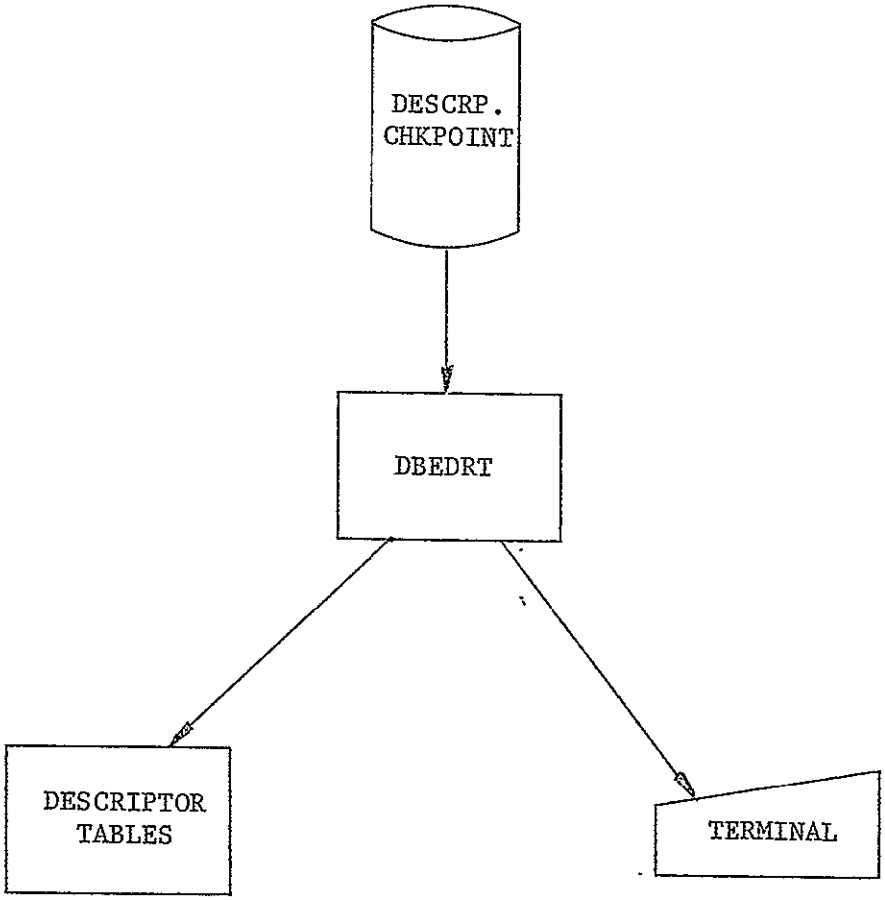
F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable


DESCRP.
CHKPOINT
DBEDRT
DESCRIPTOR
TABLES
TERMINAL

IV D526.1

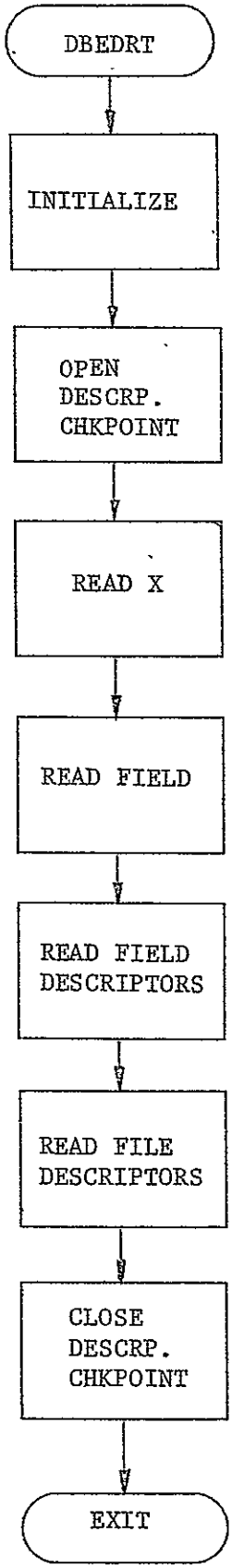


Figure 2. Top level flowchart

TOPIC D.27 - DESCRIPTOR EDITOR - REVIEW COMMAND

A. MODULE NAME

Program-ID - RDBEDRV
Module-ID - DBEDRV
Entry Points:
DBEDRV1 - Review Command
DBEDRV2 - Paging Entry

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This command is used to present the descriptor information to the user of any descriptor record in any descriptor region in the descriptor file. Review points to the record to be patched by means of the PATCH command.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input File

Not Applicable

c. Input files

The data base descriptor file is a TSS VISAM file maintained by DBPAC, containing the information defining and detailing the information contained in the data base.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The various pieces of information contained in the descriptor record are displayed on the screen preceeded by the descriptor descriptor field name. All fixed fields are displayed within a 20 character string. The number of items per line for fixed field items is determined by dividing the screen width by 20. The varying elements are display one per line, with continuation lines if necessary..

c. Formatted Print-Outs

Not Applicable

4. References

The following external tables are referenced by RDBEDRV:

1. FLD
2. HDR
3. RECSEC
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables is found in the dataset specifications of the dataset DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

At the command entry point the paging information structure is allocated and initialized. The user is prompted for the descriptor file a region id that he wishes to review from. If the region id is invalid, the user is given a diagnostic and prompted for a new region id.

The user is prompted for the name of the descriptor record he wishes to review from the descriptor region. If the descriptor name is invalid, the user is given a diagnostic and prompted for a new descriptor name. If the descriptor exists the routine DBEDLD is called to

load the descriptor data. If a loading error occured, the user is given a diagnostic and prompted for a new descriptor value.

The paging information structure is setup to point to the first page of information to be displayed. At which point the command entry and paging entry join in common code.

At the paging entry point, the paging information is set to point to the proper page to be displayed and then join common code with the command entry point.

At the start of the common code, the number of the next item to be displayed is retrieved from the paging information and a branch is taken to the approprate code to obtain the next piece of descriptor information. Seperate pieces of code exist for each field descriptor and file descriptor descriptor fields. After the piece of information is built, it is inserted in the output line. If there is sufficient room in the output line for more data, the next item of information is obtained as above. If the line is full, it is put into the TS screen buffer.

If there are more lines of screen available, they are built and processed as above. This continues until either the screen buffer is full or all of the information has been exhausted. If the screen is full and there is more information to output in the forward direction, a paging entry point is setup and the next page of information is posted in the paging information. The buffer is then flashed to the screen.

The X structure is posted as the descripter region and field name of the record REVIEW'ed so that the user may use the PATCH command if he desires, after which control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I and DBPL/I statements.

2. Suggestions and Techniques

Not Applicable

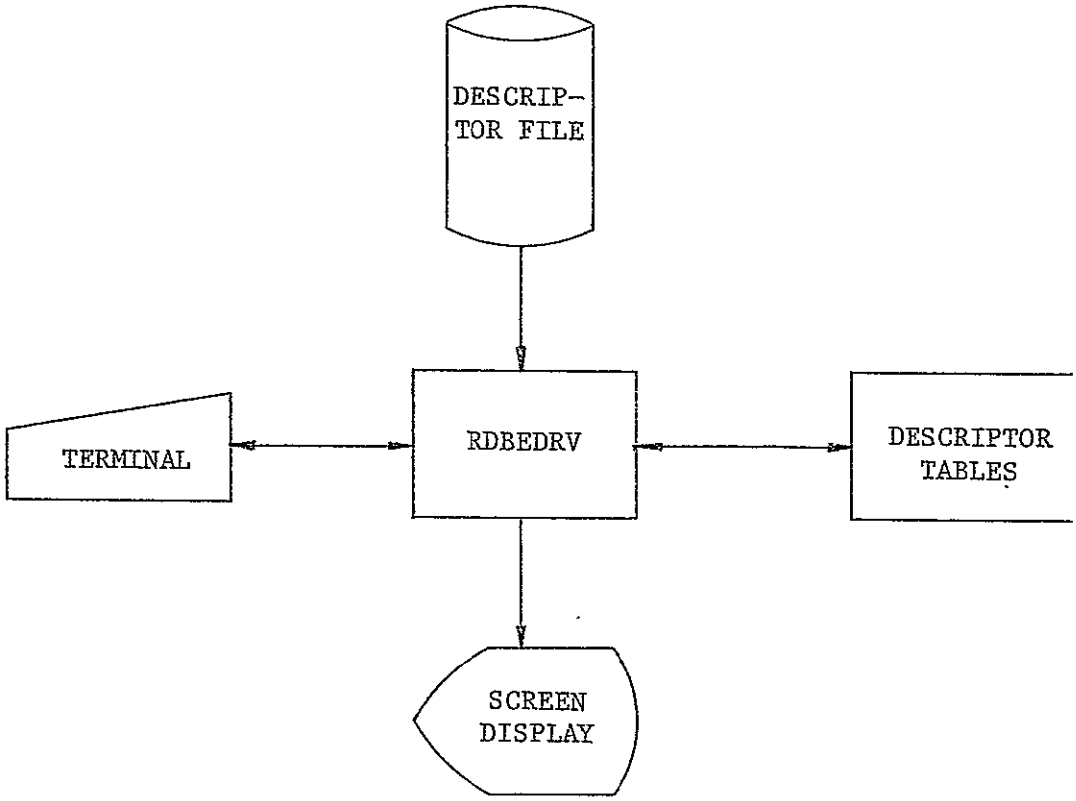


Figure 1. I/O Block diagram

IV. D27.1

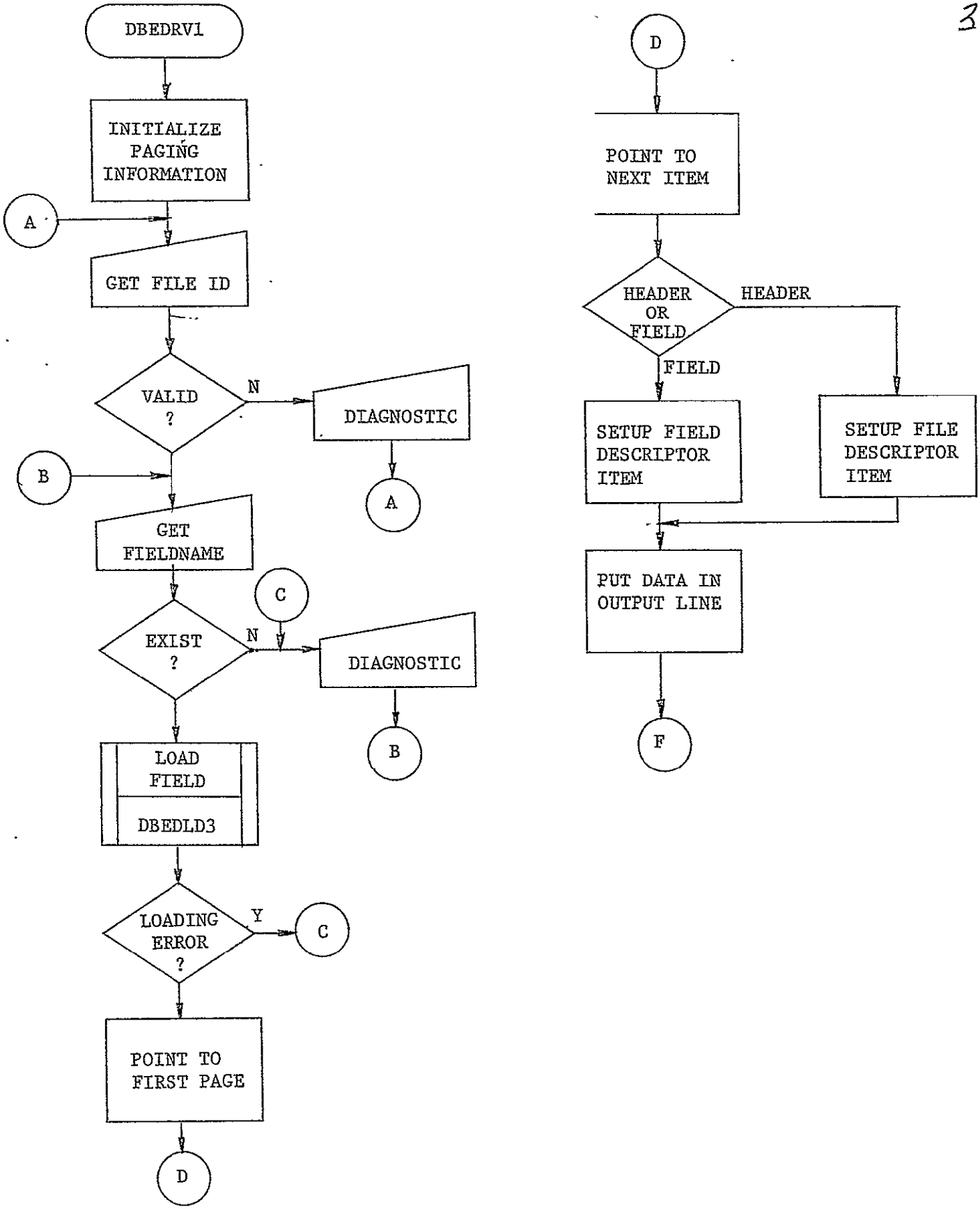


Figure 2a. Top level flowchart

IV 537.2a

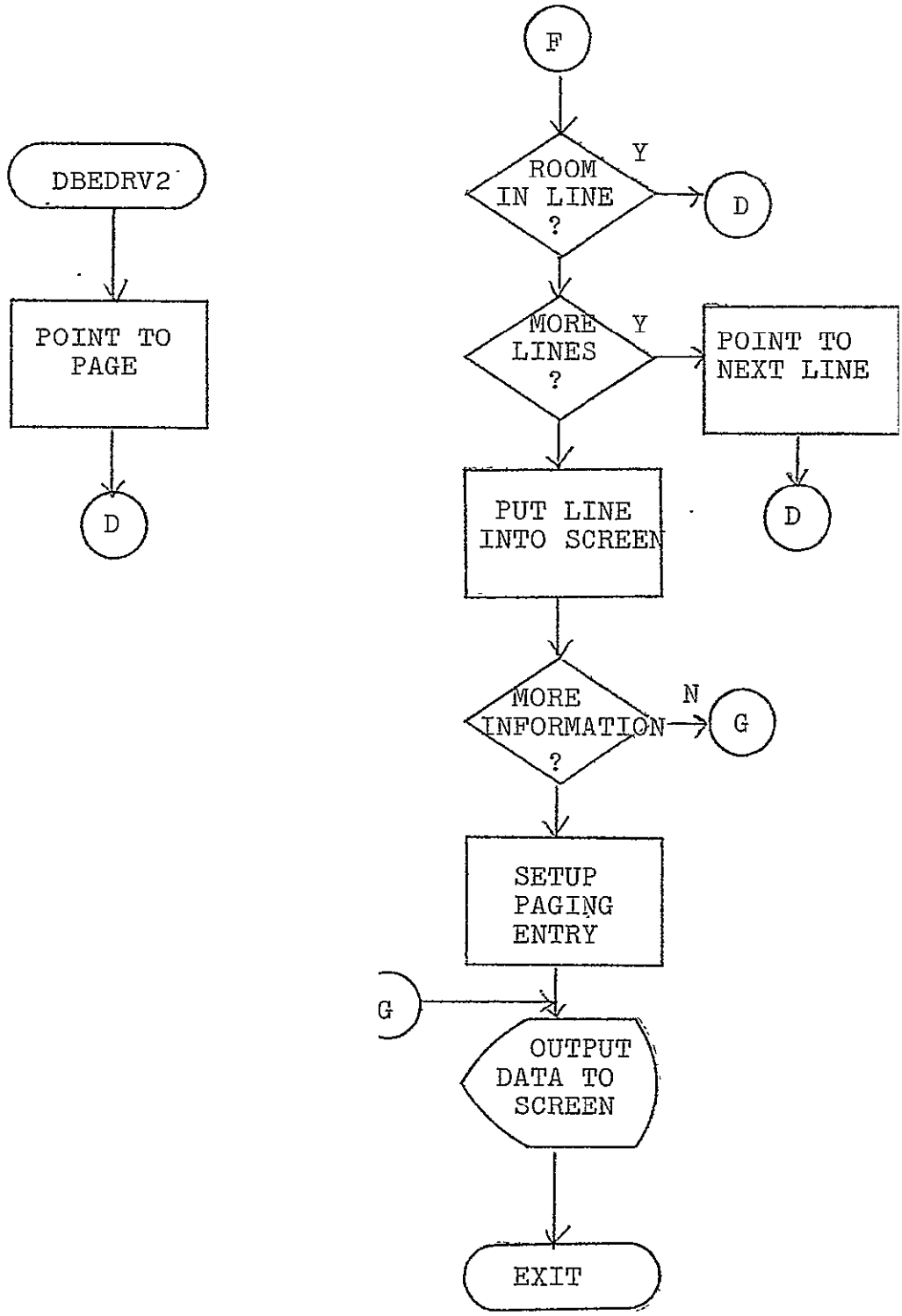


Figure 2b. Top level flowchart

27.23

TOPIC D.28 - DESCRIPTOR EDITOR - SAVE STRATEGY COMMAND

MODULE NAME

Program-ID - RDBEDSS
Module-ID - DBEDSS

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The command is used to create and save in the strategy data set, a list of Descriptor Editor commands which when executed at any future time will create a set of descriptors exactly like those that exist in core at the time the SAVSTRT command is issued.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEPSS:

1. FLD
2. HDR
3. RECSEC
4. SECURITY
5. SUPER
6. VALID
7. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into SAVSTRT, the user is prompted for the strategy name in which the Descriptor Editor commands are to be saved. If the name is not of the proper form or a strategy by that name already exists, the user is given a diagnostic and prompted for a new strategy name.

Once a valid strategy name is obtained, the MAINTAIN and EDIT command strings are saved in the strategy. This initializes the strategy. The internal subroutine SAVE_FLD is called to save the ADD command to create the anchor file key field.

A control loop is set up to process each of the 20 possible files in the order of anchor file, associate file and then subfiles. With each existing file each field list is processed in the order of packed bit fields, fixed fields, and then varying fields. Each field list is processed from the start of the list to the end of the list.

The SAVE_FLD command is called for each field to create and save the appropriate command string.

The fields COMMENTS, FREEFORM, the subfile key field, and the subfile parent key fields are

skipped as they are created thru the adding of the anchor key field or the CREATSUB command. The record security field is skipped if encountered and processed after all other fields on the file have been prccessed. All the fields on all of the files are processed in the manner and order.

If the RECLEN field has field security, the SAVE_FS is called to build and save the FLDSEC command.

After all of the fields and files have been processed, the FILE and END commands are saved in the strategy, after which time control is returned to the calling routine.

In the SAVE_FLD internal procedure, if the field is a subfile control field the CREATSUB command string is built else of the field is a superfield the SUPERFLD command string is built, otherwise the ADP command string is built. The appropriate command string is saved thru use of the routine TSPUTG.

The internal entry SAVE_FS is defined at this point to save the field security if any. This code is also part of the SAVE_FLD procedure. If the field has field security defined on it, a FLDSEC command string is built or saved in the strategy through use of the routine TSPUTG. Contrcl is then returned to the calling point in SAVSTRT.

F. CODING SPECIFICATICNS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Techniques

Not Applicable

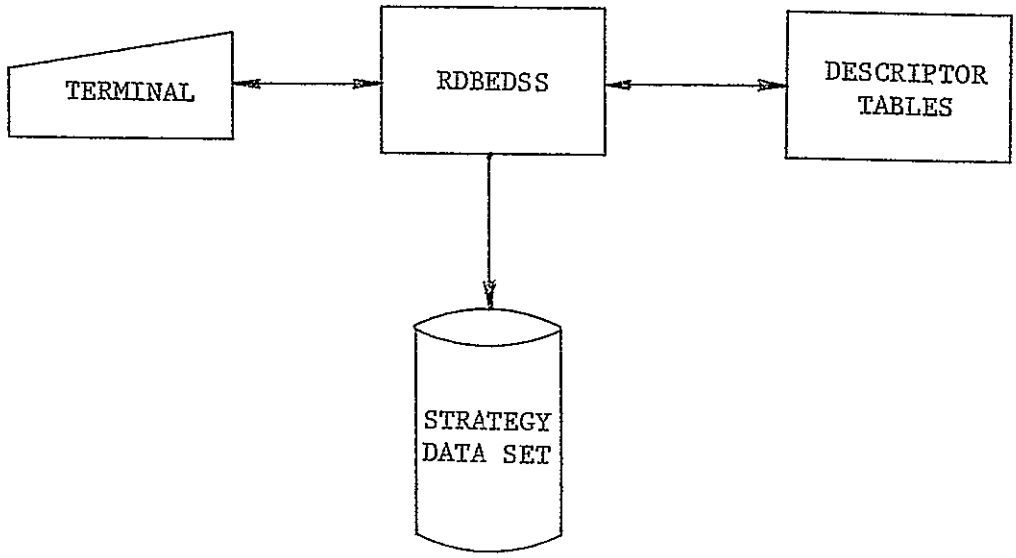


Figure 1. I/O Block diagram

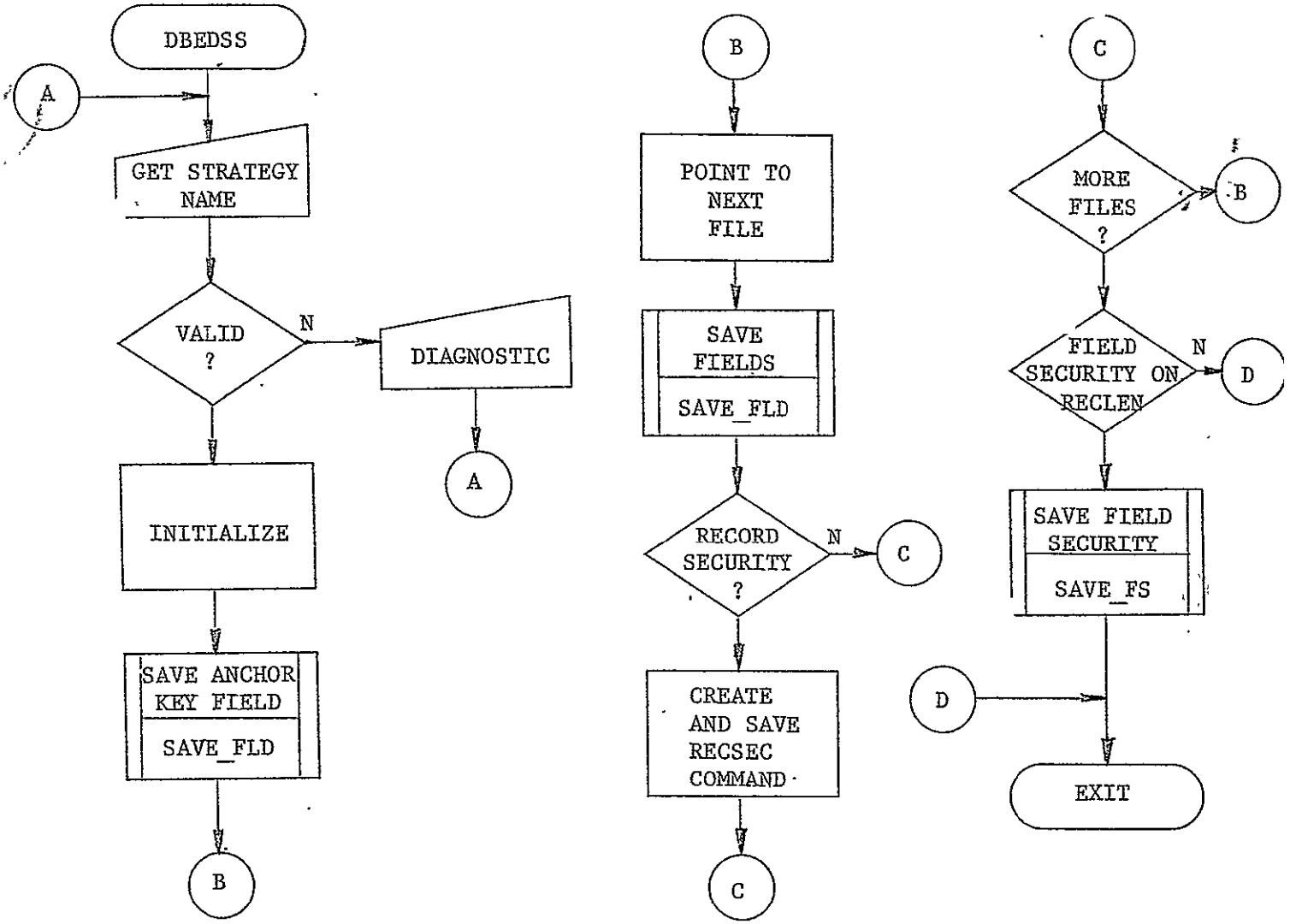


Figure 2a. Top level flowchart

28.20

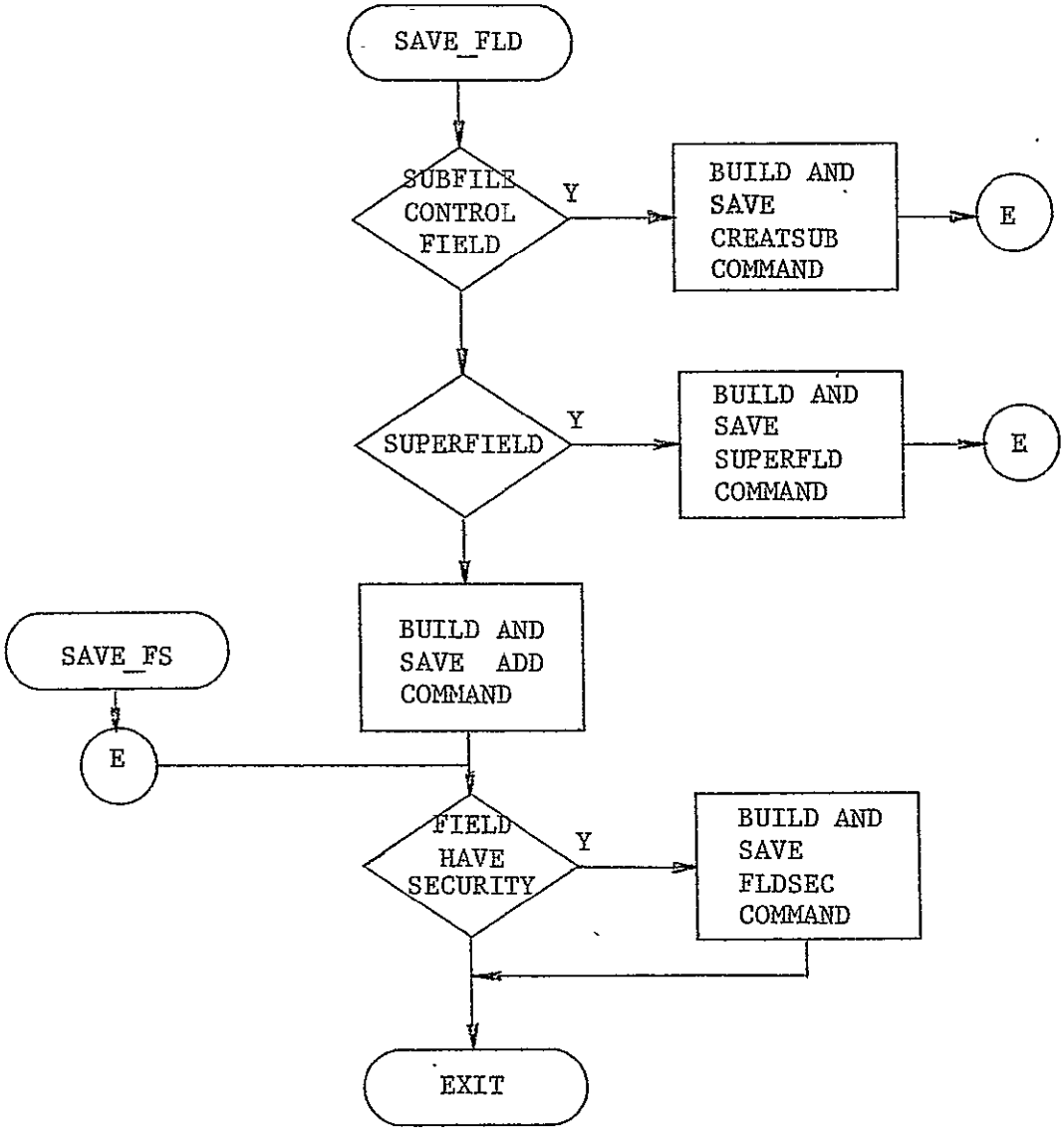


Figure 2b. Top level flowchart

IV D28.26

TOPIC D.29 - DESCRIPTOR EDITOR - SUPERFIELD COMMAND

A. MODULE NAME

Program-ID - RDBEDSU
Module-ID - DBEDSU

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

The SUPERFLD commands allow the user to define a superfield descriptor.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDSU:

1. FIELD
2. FLD
3. FLD_STRING
4. HDR
5. SUPER
6. VALID
7. X

A description of these tables can be found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into SUPERFLD, routine DBEDGF is called to obtain a new fieldname.

A FLD structure is allocated and initialized. Routine DBEDGR is called to obtain any conversion, formatting and validation routines and validation argument.

The user is prompted for a list of field names which are to be the superfield components. Each component is processed in the following manner. If no internal external indicator is present, external form is assumed. If an indicator is present, it is seperated from the field name. If the indicator is invalid, the user is given a diagnostic and prompted for a new component value. To be valid the component fieldname must be the name of an existing field. In addition, for a field having more than one component, the component list is limited to at most one multielement field and all if any subfile components must be from the same subfile. If the component fieldname is invalid, the user is given a diagnostic and prompted for a new component value.

After all the components are entered, and processed, they are saved in a SUPER structure and the pointer stored in the FLD structure.

Next it is determined on which descriptor file the

superfield is to be placed. If all the components are from one file, then the superfield descriptor is placed in the descriptor region. If the components are all from one associate file and one subfile and the subfile is defined off of that associate file, the superfied descriptor is placed in that associate descriptor region. All other superfield descriptors are placed in the anchor descriptor file.

The SUPERFLD command string is saved in the current strategy and control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with TSPL/I statements.

2. Suggestions and Technigues

Not Applicable

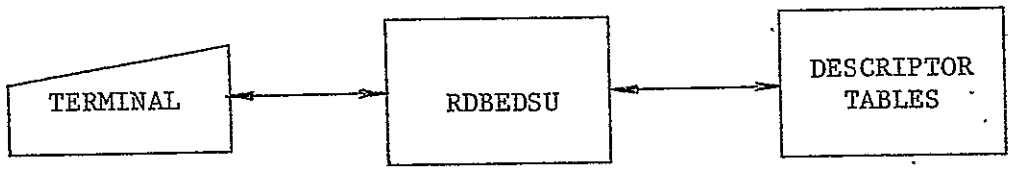


Figure 1. I/O Block diagram

329.1

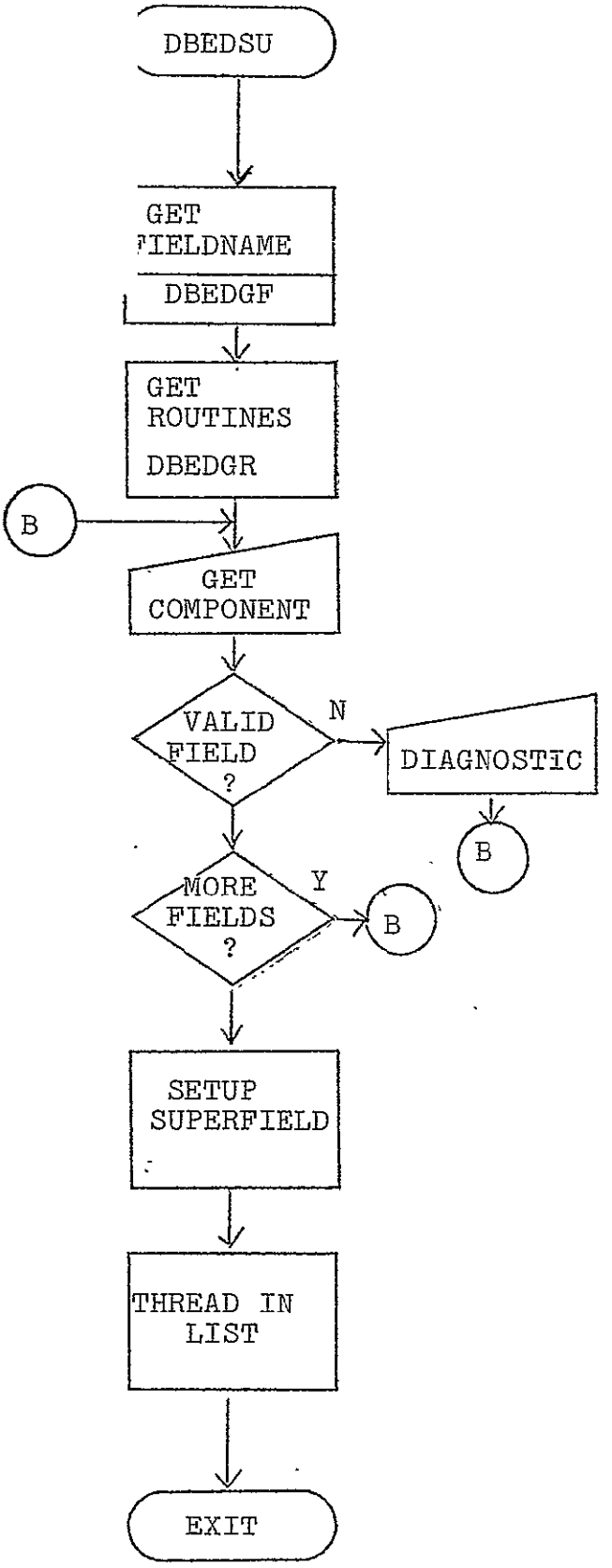
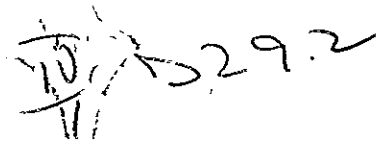


Figure 2. Top level flowchart



TOPIC D.30 - INDEX FILE MERGE

A. MODULE NAME

Maintenance - File Merger
Program-Id - RDBINDM
Module-Id - DBINDM

B. ANALYST

Edward McIntire
Neoterics, Inc.

C. MODULE FUNCTION

The merge module is an independent program whose function is to create an updated index file for a data base. The updating of the index file can be done in place or to a new index file. This new index file will be named'' 'INDMRG.' //FILE NAME//:'//FIELD NAME''. This module will also allow for the processing of duplicate records if deemed necessary.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

The merging utility is most often invoked from a terminal in conversational mode. However, it is possible to initiate the task in the non-conversational mode. In batch mode, the format of the punched card input is the same as when terminal input is used to invoke the routine.

c. Input Files

1. Index File: The primary input to the combine program is the current index file that is to be updated, and the update index file that is to be combined with the current index. The anchor

descriptor file is needed to provide field information.

2. Parameter File: If the user wishes to stop processing he may do so by pressing attention and responding 'y' to the prompt message. Thus a parameter file is created for input to further restarts. This file consists of the last key processed on the current index file. This file is a VSAM file used only in the combine program.

d. On-Line Terminal Entries

All of the terminal entries to the merge program are in the form of responses to prompting messages from the program itself. The one exception to this rule is the initial command with its parameter to invoke the procedure.

3. Output Data Sets

a. Output Files

The output data set is the index file created by the combine program. This data set can take two forms:

1. The current index file updated inplace.

2. A new file created by the merging of the current index with the update index.

b. On-Line Terminal Displays

All on-line terminal displays for the combine program follow the same format. The TSPL/I facility of the system is utilized to request entries at the terminal and display progress information.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Prompting

Prompt the user for first pass. If it is not the first pass, go and read the parameter file. Prompt the user for the anchor file name, for new file creation or merge in place, for the processing of duplicates or not, and for the inverted field name. The user must enter a valid response to all of the prompts or he will be reprompted. A read sequentially of the anchor descriptors is done until a fixed field with an offset of four (4) is found. That, in fact, is the key descriptor and its length is saved. The index field is also checked for validity, if it is not a valid index, then a reprompt is initiated. Following this the index descriptors are opened and read sequentially until the index field length is obtained, and the spanned indicator is checked and it's value is saved. In all of the above cases if a critical error is encountered an error message is displayed and the program is terminated.

b. Write Parameter File

If the user deems it necessary to stop processing during the combine operation, he can press the attention button and the total records processed will be displayed. Also, a message will be displayed asking if he wishes to quit processing. When the user replys with a quit processing command the following occurs:

1. Quit switch is set.

2. Processing is continued until a clean close can be carried out.

3. Parameter file is opened and the key of the last record written is written to

the parameter file.

4. Parameter file is closed.

5. Program branches to end of job routines.

c. Read Anchor Descriptors

DBPAC is utilized to read sequentially the anchor file descriptors and retrieve the field that is indexed and the anchor key length.

d. Read Index Descriptors

DBPAC is utilized to read sequentially the index file descriptors and retrieve the index key length and the spanned indicator.

e. Read Parameter File

If not the first pass, the parameter file is read to get the needed key for the restart. The restart key is used as the key and a get by key is done on the current index file. Also a read by key is done on the update file to find its starting position. From here we go to normal reads on the input files for further processing.

f. Write Index File

The writing of the index file can take two different forms.

1. Merge Inplace.

If the user decides to merge to the current index file the new records will be built in core and tabled there until a unique record is read. If a record is longer than the maximum allowable length, then create a spanned record. Then rewrite any existing records and write any new records that may have been created. When an update record does not match a current record, the update record and any with the same key are written to the current file.

2. Merge to New File.

The merge to a new file is much the same as the merge inplace. The differences are listed below:

a. An out put file is created with the same attributes as the current index file.

b. There will be no rewrites to the new file.

c. All current and update records will be written to the new file.

g. Attention Interrupts

Attention interrupt handling was discussed in section 'E', sub-section '2', Item 'B' (Write Parameter File).

F. CODING SPECIFICATIONS

1. Source Language

The combine program employs the IBM PL/I Programming Language. The special extensions of that language, called DBPL/I and TSPL/I, are utilized for access to file descriptors in the data base and for all terminal communication, respectively. Also, the merge program employs assembler routines to handle all I/O during the execution of this program, except for the writing of the parameter file which is handled exclusively by PL/I.

2. Suggestions and Techniques

Not Applicable

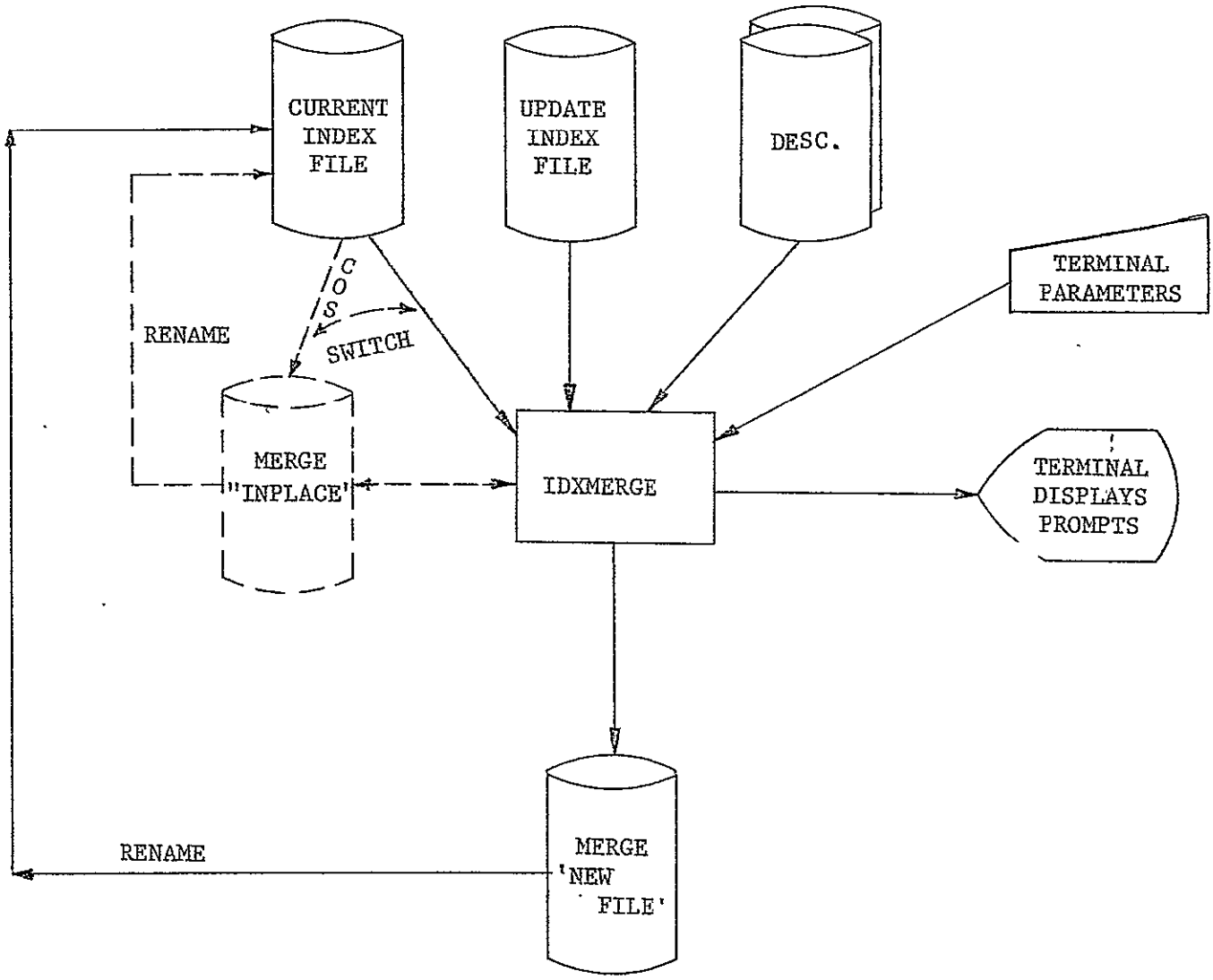


Figure 1. I/O Block Diagram

[illegible]

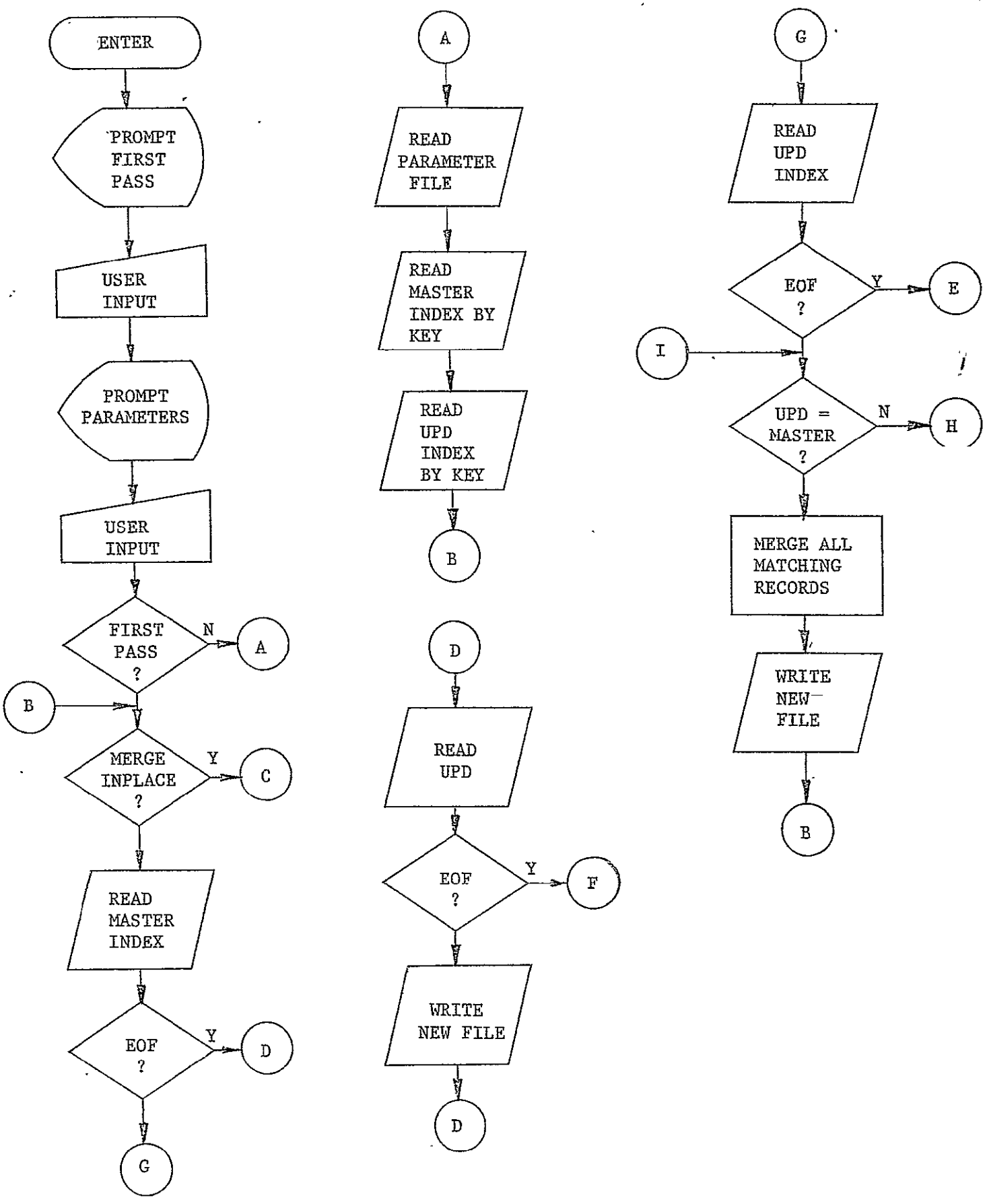


Figure 2A. Top Level Flowchart

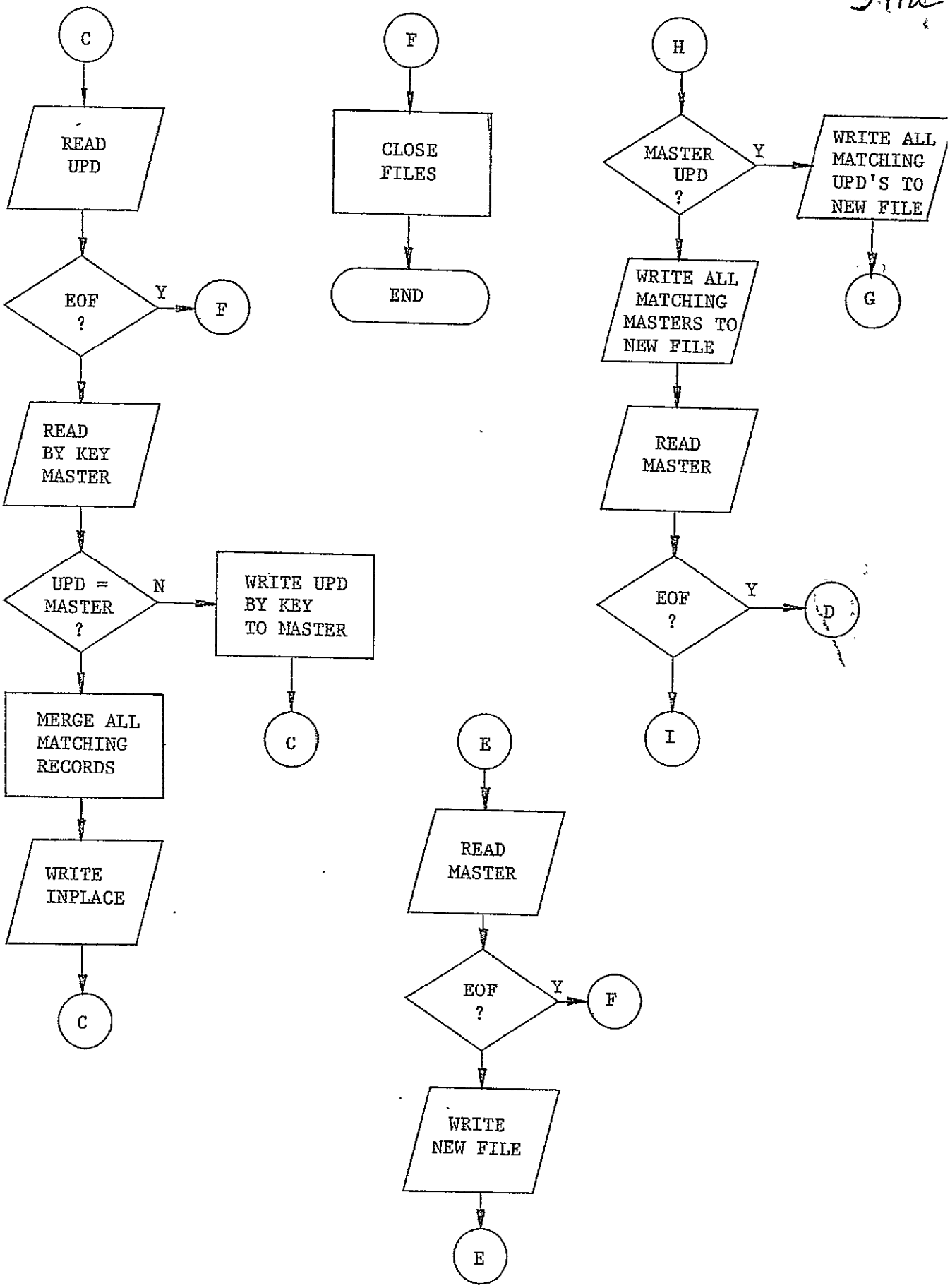


Figure 2B. Top Level Flowchart

30.2B

TOPIC D.31 - DESCRIPTOR EDITOR - INITIALI ATION

A. MODULE NAME

Program-ID - RDBEDIN

Module-ID - DBEDIN

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module performs all of the initialization necessary for the running of the Descriptor Editor. It is called by the Descriptor Editor director.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

There are no input files for the Descriptor Editor when in the CREATE mode and the user is creating a new set of descriptors. However, when in UPDATE mode, or when the user is continuing the creation of a previously entered set of descriptors, the previously created descriptor file is used as an input file. The description of this file is found in the dataset specifications of the DWB.

3. Output Data Sets

a. Output Files

Not Applicable

b. On Line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

4. Reference Tables

The following external tables are referenced by RDBEDIN:

1. FIELD

2. FLD

3. HDR

4. RECSEC

5. SECURITY

6. SUPER

7. VALID

8. X

A description of these tables is found in the dataset specifications of the DWB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The descriptor file indicated is opened for input to determine if the file exists. If the file exists and in CREATE mode, routine DBEDID1 is called to load the descriptors. If in UPDATE mode and the file does not exist, the user is given a diagnostic and prompted for a new file name.

The verb table is allocated and initialized to the proper verb table copy. The routine DBUSER is called to setup any additional user defined commands.

If in CREATE mode and no file exists, the user is

prompted for the anchor key field. The routine DBEDAC1 is called to process and setup the anchor key field.

The user is prompted for the descriptor mode. If the response is valid, flags are set indicating the mcde, and a pointer is set to the appropriate verb table copy. If the mode is invalid, the user is given a diagnostic and prompted for a new mode value.

The X structure is allocated and initialized. The initialization consists of setting the various pointers in X to NULL. he FIELD structure is allocated and its pointers set to NULL.

If RESTORE mode is indicated, DBEDRT is called to restore the checkpointed descriptor. If no restore errors occurred, the go setup the verb table. If there were restore errors, or CREATE or UPDATE mode were indicated, the file name is retrieved from the MFCB.

Control is then returned to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

PL/I with DEPL/I and TSPL/I statements

2. Suggestions and Techniques

Not Applicable

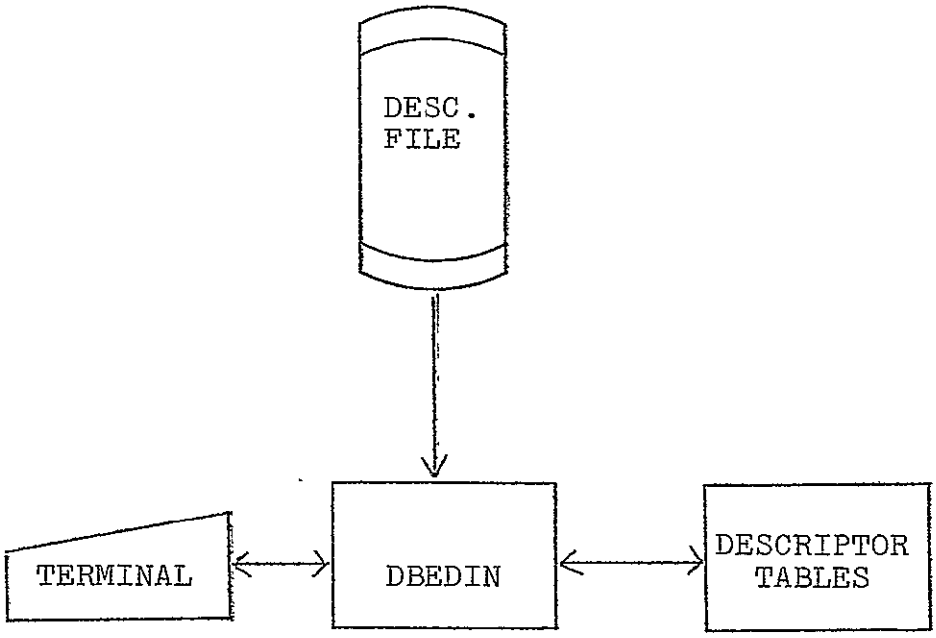


Figure 1. I/O Block Diagram

1531.1

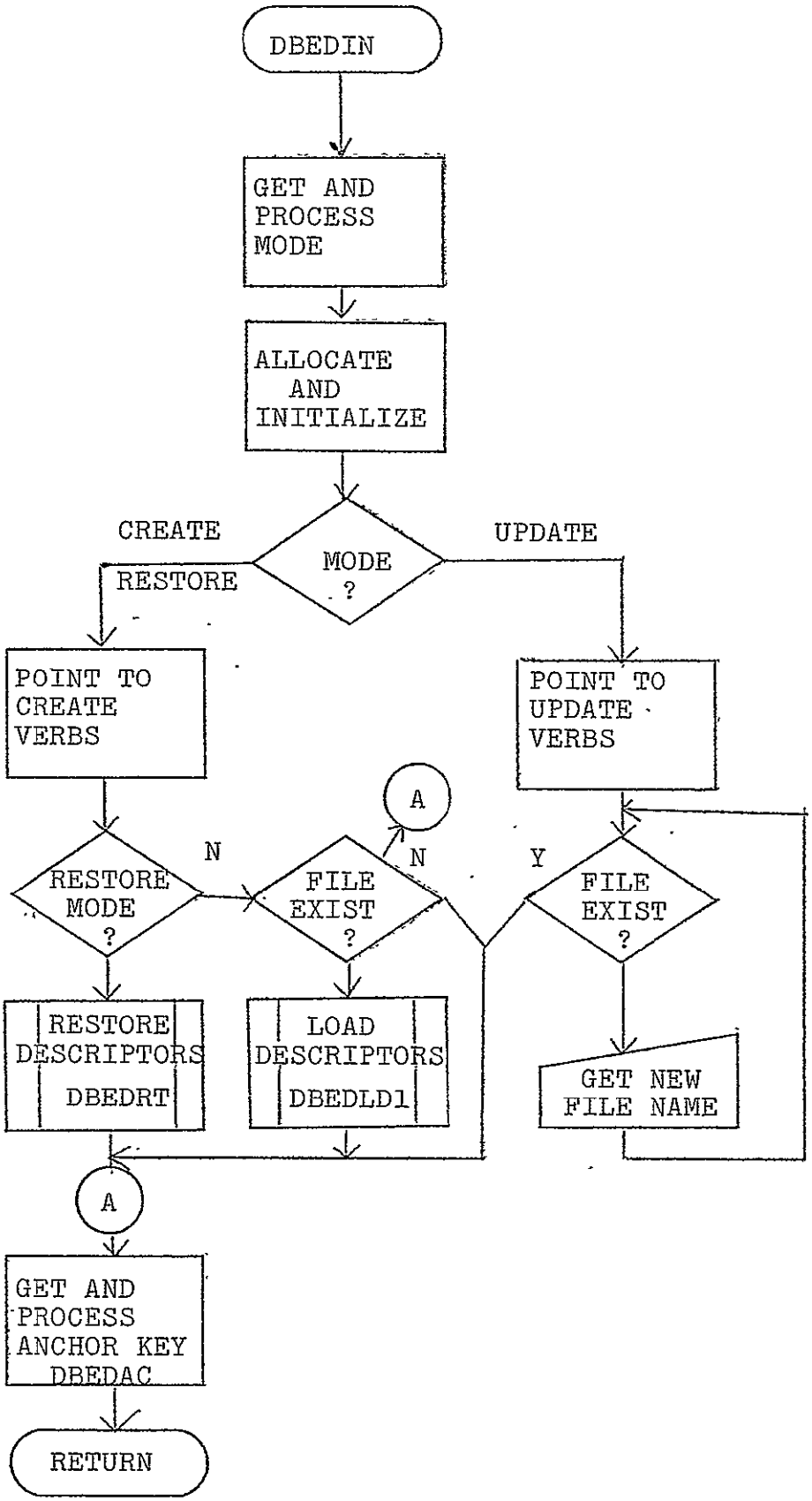


Figure 2. Top level flowchart

>31.2

TOPIC E.1 - TERMINAL SUPPORT - PREPROCESSOR

A. MODULE NAME

Terminal Support PL/I Preprocessor
Program-ID - TSPLI
Module-ID - TS

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

TSPLI analyzes terminal input/output PL/I language extension statements and produces statements acceptable to the PL/I compiler. These statements call the terminal support module allowing the program to communicate with the user's SYSIN and SYSOUT or, pending TSS support, an on-line display station. The user's SYSIN and SYSOUT are a terminal if the task is conversational, or data sets, if non-conversational. Diagnostic messages are generated for errors which can be detected by TSPL/I during preprocessing.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

The Job Control cards needed to invoke the PL/I compiler in TSS are described in the IBM TSS/Command System User's Guide, Form No. C28-2001. More detailed information will be provided in the IBM PL/I Programmer's Guide, Form No. C28-2049.

b. Punched Card Input Files

1. TSPLI Text

The TSPLI text deck is all text for insertion into the source program following a "% INCLUDE LISRMAC(TS);" statement in the source program. This

text consists of the source statements of the TSPLI preprocessor function and any PL/I statements to be inserted at the "% INCLUDE LISRMAC(TS);" statement in the source program. The TSPLI text is coded as specified in this report, formatted according to PL/I source language standards, and catalogued in a data set for compile time use by all programs using TSPLI.

2. Source Deck

The source deck is any PL/I source program using TSPLI statements to interface with the user's SYSIN and SYSOUT or any on-line display station. The statement formats and their use are described in the TSPLI User's Manual (Section 8, Topic E.2 of the DWB).

c. Input Files

The TSPLI text statements are catalogued as a member of a partitioned direct access data set for retrieval by the IBM PL/I precompiler. This data set is accessed via ddname LISRMAC.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The object module consists of the relocatable machine instructions and constants generated by the PL/I compiler for the source program. It is stored in a partitioned data set. This data set is the last DATADEFed JOBLIB. If the user has not DATADEFed any JOBLIBs, it is stored in the user's USERLIB data set. The module is loaded by TSS when called by the user.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

1. Precompiler Listings

Two precompiler listings are produced:

(a) A source listing before precompilation and

(b) Any precompiler diagnostics (i.e., errors in the use of preprocessor PL/I, not TSPLI error messages).

2. Compiler Listings

The compiler listings produced include an intermediate source listing (between precompiling and compiling) and any compiler diagnostics. Serious TSPLI PL/I errors may result in compiler diagnostics also.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. TC - terminal control block

b. TSPL/I - diagnostic comments.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Top Level

The mainline PL/I source program is required to have a "% INCLUDE LISRMAC(TS);" statement once in the program before all TSPLI preprocessor function references. This statement directs the PL/I precompiler to take text from member TS of the library accessed via ddname LISRMAC and incorporate it into the source program. (Refer to the TSPLI block diagram in Section D.1 of this write-up).

The TSPLI function receives one argument from a preprocessor function reference; i.e., a

variable length character string. It is TSPLI's function to scan and parse this input string to determine if it is in the correct format and then to generate a string called the "generated text." This string consists of valid PL/I statements and comments for communication with the terminal support modules.

The processing of TSPL/I is closely analogous to the processing of DBPL/I described in Section IV, Topic A of the DWB and is only summarized here. The TS text declares and activates the TS preprocessor function. Argument initialization, finding a subargument, passing labels and comments through, and finding the statement keyword to select the specific statement routine are all done analogously to the DBPL/I preprocessor function. Diagnostic comments are generated for any errors detected. (See Section III, Topic E.1 of the DWB.) There are no files to be analyzed.

In all programs a declaration of the entry point to the terminal support modules is generated and a declaration of TC - the Terminal Control block (See Section III, Topic E.2 of the DWB.)

b. Specific Statement Routines

Each specific statement routine examines the statement from left to right until the semicolon clause is found. The keywords are verified for correct spelling and order. If any error is detected, a diagnostic comment is generated and the statement abandoned by control being transferred to the inter-statement point. Following successful analysis, each specific statement routine generates PL/I statements for communication with the terminal support modules and loops back to the inter-statement point.

The ON PAGE statement routine generates the following statement:

```
TC.PAGING_ENTRY=expression;
```

Where "expression" is taken from the CALL clause of the TS ON PAGE statement.

The ENTRY statement routine generates the following statements:

```
TS_ENTRY_RETURN_POINT=TS_ENTRY_LABEL_n;
GO TO TS_ENTRY_CODING;
TS_ENTRY_LABEL_n:
```

Where "n" is a numeric value assigned sequentially to each ENTRY statement as it is encountered.

The ENABLE statement routine generates the following statements:

```
DCL TS_ENTRY_RETURN_POINT LABEL;
TS_ENTRY_RETURN_POINT=TS_ENTRY_LABEL_1;
TS_ENTRY_CODING:
ON CONDITION(END)
   GO TO TS_EXIT_CODING;
ON CONDITION (ATTN);
TC.FUNCTION='ENTRY';
CALL TSCNTRL(TC);
GO TO TS_ENTRY_RETURN_POINT;
TS_EXIT_CODING;
RETURN;
TS_ENTRY_LABEL_1;
```

Lines 4-6 and 10-11 of the above text are only generated when the user specifies the appropriate option on the ENABLE statement.

The TS logic is such that the ENABLE statement, if it appears, must appear before the first ENTRY statement, and in fact, implies an ENTRY statement. Likewise, the first ENTRY statement implies a default ENABLE statement, if none are present.

The PROMPT statement routine generates the following statements:

```
TC.FUNCTION='PROMPT-e';
TC.PROMPT.MESSAGE_KEY=expression;
TC.PROMPT.KEYWORD=value;
CALL TSPRMTe(TC,variable,list);
```

Where "expression" is taken from the MSG clause of the statement, "value" is taken from the KEYWORD clause (if present), "variable" is taken from the INTO clause (if present) and "list" is taken from the USING clause (if present). The value of "e" is generated according to the following table:

1. INTO clause - none
   "e"=M

2. KEYWORD clause - none
   "e"=C

3. KEYWORD clause - yes
   "e"=D

The READ statement routine generates the following statements:

```
TC.FUNCTION='READ';
CALL TSREAD(TC,variable);
```

Where "variable" is taken from the INTO clause of the TS READ statement.

The WRITE statement routine generates the following statements:

```
TC.FUNCTION='WRITE';
CALL TSWRITE(TC,variable);
```

Where "variable" is taken from the FROM clause of the TS WRITE statement.

The PUT statement routine generates the following statements:

```
TC.FUNCTION='PUT';
TC.OUTPUT.POSITION='a';
TC.OUTPUT.DIRECTION='b';
CALL TSPUT(TC,variable,value);
```

Where "variable" is taken from the FROM clause of the TS PUT statement and "value" is taken from the TAG clause (if present). The value for "a" will be generated according to the following table:

1. position clause - none
   "a"=0

2. position clause -LINE
   "a"=0

3. position clause - PAGE
   "a"=1

The value for "b" will be generated according to the following table:

1. direction clause - none
   "b"=0

2. direction clause - FORWARD
   "b"=0

3. direction clause - BACKWARD
   "b"=1

The FLUSH statement routine generates the following statements:

```
TC.FUNCTION='FLUSH';
CALL TSFLUSH(TC);
```

The FINISH routine sets a precompiler variable to indicate that a FINISH statement has been processed and to prevent the processing of any further TSPL/I statements. A diagnostic comment indicating the number of TSPL/I errors is generated. If there have been any errors detected, the following statement will be generated causing an IEM0512I PL/I error:

```
DCL TS_DUMMY_VARIABLE LABEL
INIT(TS_ERRS_nn);
```

Where "nn" is the number of TSPL/I errors detected.

All statements and comments generated will be aligned as seventy-one byte strings, for ease of analysis.

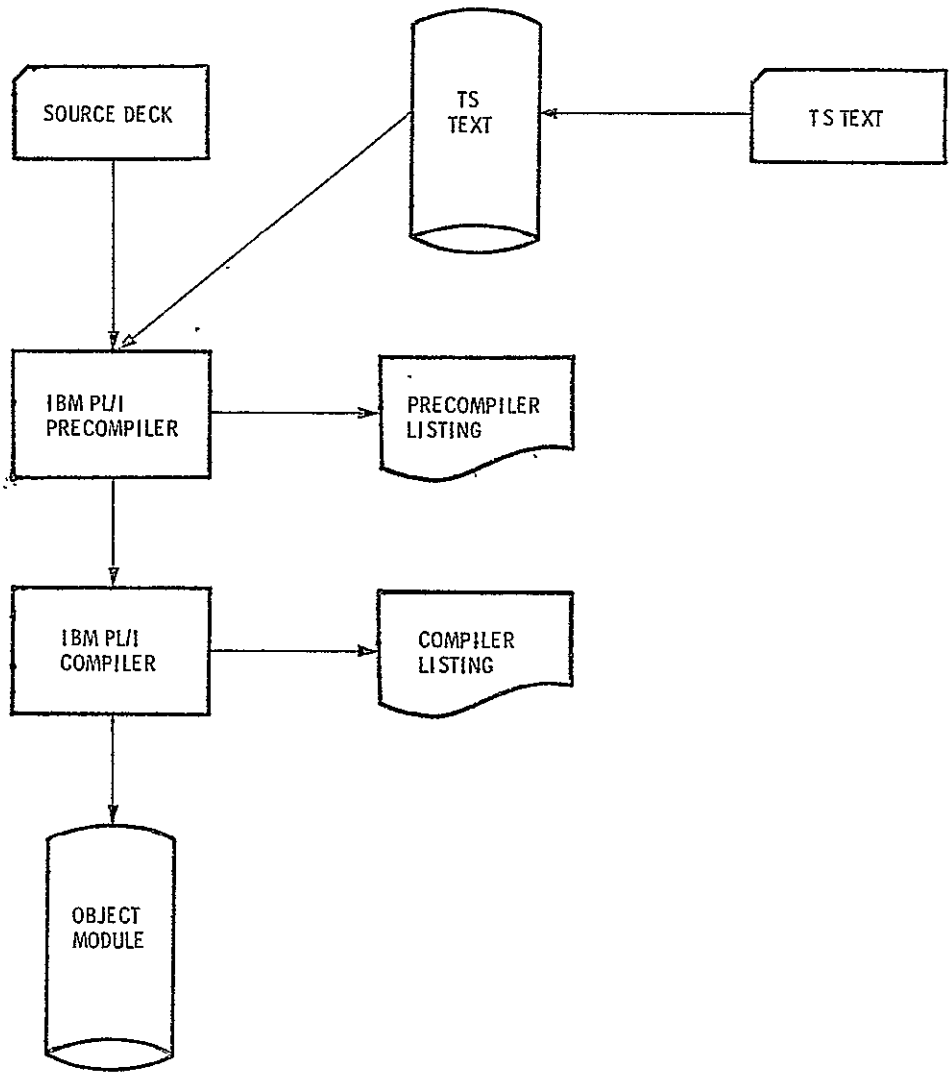


Figure 1. I/O Block diagram.

IV § 1.1

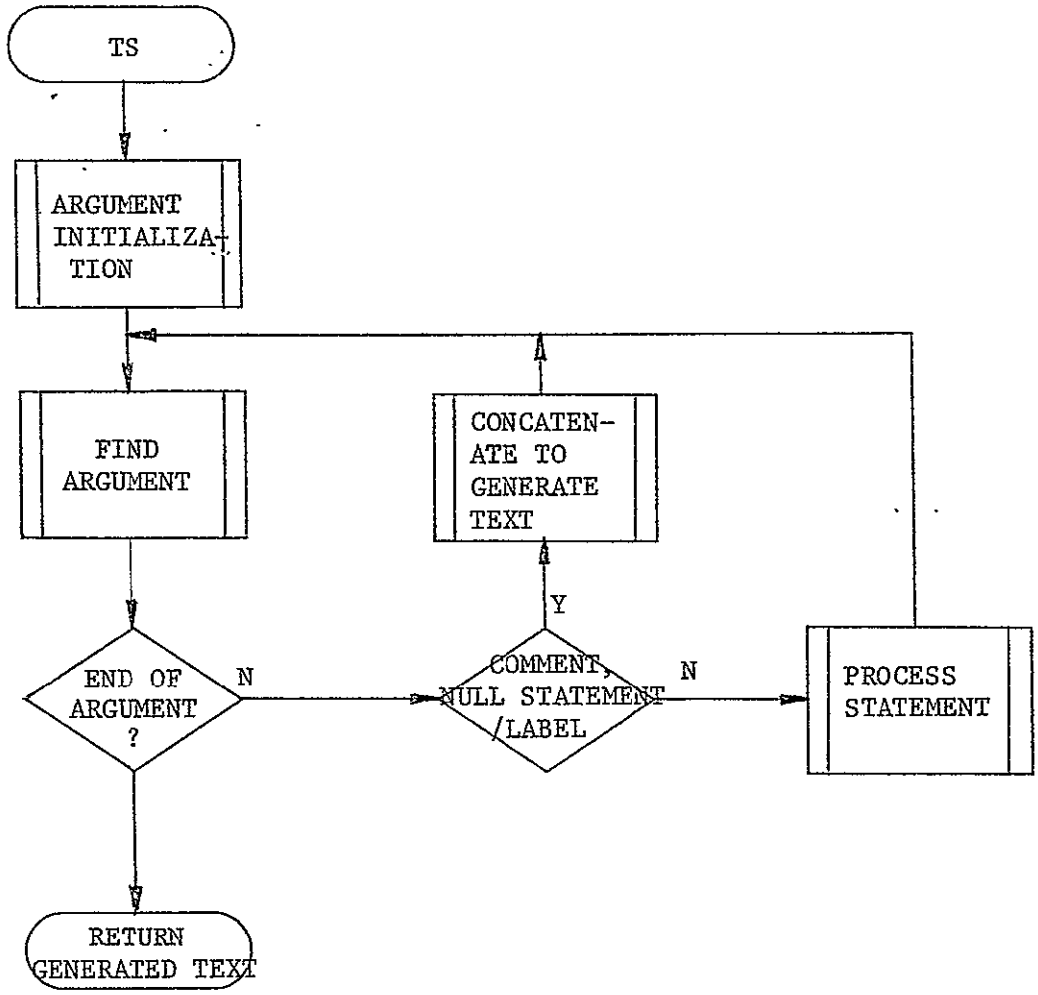


Figure 2. Top level flowchart

1.2

TOPIC E.2 - TERMINAL SUPPORT SUPERVISOR

A. MODULE NAME

Terminal Support - Terminal Support Supervisor
Program-ID - RTSUPER
Module-ID - TSUPER
Entry Points - TSATIN, TSCNTRL, TSFLUSH, TSGETKY, TSPRMTC, TSPRMTD, TSPRMTM, TSPUT, TSREAD, TSWRITE

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTIONS

1. Organization Chart

See Figure 1

2. Overview

RTSUPER is the primary vehicle of communications between the NASIS monitor (MTT or stand-alone) and the NASIS PL/I data Base programs. Among the functions RTSUPER performs are:

a. Issues I/O requests from data base programs. This includes command, data and message prompts and ordinary read and write requests.

b. Initializes the Terminal Control Block (TC) for each PL/I program. Supplies information about the current display area dimensions and resets all bit switches to zero.

c. Controls asynchronous interrupt processing. Detects APOFF, END and GO conditions and issues that asynchronous activities do not interfere with normal processing.

d. Maintains a push-down stack of message key references to support the EXPLAIN facility.

e. Scans and passes user input strings for commands and data. Information entered at the terminal is interpreted and passed to requesting programs in useful segments. The IC Block is utilized to enhance interprogram communication.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 2

2. Input Data Sets

a. Parameter Cards

Not applicable

b. Punched Card Input Files

Not applicable

c. Input Files

1. DBAMLF
2. LISRMLF

d. On-line Terminal Entries

All responses to command and data prompts by NASIS prcgrams pass through RTSUPER.

3. Output Data Sets

a. Output Files

Not applicable

b. On-line Terminal Displays

All output from NASIS data base programs passes through RTSUPER.

c. Formatted Print-Outs

Not applicable

d. Punched Card Output Files

Not applicable

4. Reference Tables

a. External Tables

1. TSCTL
2. USERTAB
3. TSCREEN
4. MTTUTAB

b. Internal Tables

1. EXELIST

An area in which a push-down list of message keys is saved.

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

a. MAINLINE: See Figure 3

b. Entry Pcints:

1. TSATIN - See Figure 4

2. TSCNTRL - See Figure 5

3. TSFLUSH - See Figure 6

4. TSGETKY - See Figure 7

5. TSPRMTC - See Figure 8

6. TSPRMTD - See Figure 8

7. TSPRMTM - See Figure 9

8. TSPUT - See Figure 10

9. TSWRITE - See Figure 11

c. Program Subroutines:

1. GETPR - See Figure 13

2. DELPR - See Figure 14

3. GETSYN and GETDFALT - See Figure 15

4. PULINEND - See Figure 16

5. SDGIVITD and SDGIVITC - See Figure 17

6. SDPASS and SDYSNCHK - See Figure 18

7. RESETBUF - See Figure 19

8. SDSTRIP and STRIP - See Figure 20

9. PRKEYSAV - See Figure 21

10. IRCHECK - See Figure 22

11. SIGNAL and SIGNALC - See Figure 23

12. SETLDAB - See Figure 24

13. GETMLF - See Figure 25

14. MOVE - See Figure 26

15. PROMPT - See Figure 28

16. EXIT - See Figure 29

2. Narrative

a. MAINLINE

All calls to RTSUPER entry points pass through the MAINLINE code. The purpose of this code is to insure that each user has the correct work areas, to initialize base registers and to restrict TS usage during APOFF and ATTENTION processing.

Execution proceeds by calling the PLI service routine IHESADA to obtain a Dynamic Storage Area (DSA). Next, registers are initialized and useful pointers are saved in unique locations. The PLI Pseudo Register Vector (PRV) draws special attention since it is not maintained in register 12, as is the norm for other programs.

Using the PRV, MAINLINE determines if a copy of the TS Psect has been allocated for this user. If not, the routine GETPR is invoked to obtain one. On return, data lifted from MTTUTAB is utilized to compute the user's logical and physical device dimensions and this information is saved for future reference.

The one-byte switch IOSW is checked to find out which entry point was entered. If entry was through TSATIN, control goes directly to the interrupt processing code. For any other entry, the contents of the user's TC Block (passed as a parameter) is moved to the DSA for easy addressability. If an APOFF has been requested by the user only calls to TSPRMTM and TSCNTRL are allowed to execute normally, all others being short circuited to

the routine which signals an END condition. If not in APOFF mode, control is passed to the routine specified at entry.

b. Entry Points

1. TSATIN

This entry point is called by module RTSATTN whenever it determines that a user attentions should be processed. If the user has previously entered APOFF, the attention is ignored. If an immediate command is currently processing, condition END is signaled which terminates the command. If this is the second successive attention and processing of the first is sufficiently advanced, condition END is signaled; otherwise, this interrupt is ignored.

On return, a second copy of the user psect is allocated, the string input buffer is initialized to null input and the PL/I routine RDBATTN is called to issue the ''-ATTN:'' prompt.

On return from RDBATTN, all user requests have been staisfied and the user is ready to continue. After closing the duplicate DCBs for the message files, the duplicate user psect is released. If the user entered END or APOFF in response to ''-ATTN:'', then pointers are set to cause execution to resume at the PL/I signal routine for END condition; otherwise, execution resumes at the point of interrupt.

2. TSCNTRL

Its function is to initialize the TC Block for use and pass the user's terminal dimensions. Terminal dimensions are obtained from the user's profile by repetitive calls to TSGDEF. If no defaults are specified, the necessary information is taken from MTTUTAB.

Control is returned to the caller through the EXIT routine.

3. TSFLUSH

TSFLUSH is the display output routine for terminal support. It is normally called after consecutive calls to TSPUT have caused an output buffer to be filled. If a buffer has overflowed and AUTOWRITE is indicated, this routine is called from TSPUT and a flag is set to cause the ''MORE:' message to replace the next prompt.

The name of the paging entry for the program doing a PUT or FLUSH should always be in the TC Block as it is saved by TSFLUSH just prior to the write. Data is output one line at a time for typewriters and in a block for screens. The most current display is saved in the external controlled storage named TSCREEN.

4. TSGETKY

This entry point is called with three parameters: (1) TC Block, (2) message key or list reference or list reference pointer in the range -7 < pointer < o , (3) varying length data area to hold the message text read from the file. On entry, pointers to these parameters are placed in registers and the type of request is determined (either key or pointer).

If it is a pointer, the key is obtained from EXPLIST (a push-down stack of keys). If the user wants just the key, control is returned to the caller. Otherwise, and if the second parameter is a key, the message file is searched for the Key.

If the key is not found, error flags are set and control returned to the caller. Else, the text of the message is read into the user's area and the message key is reset to point to the next record of the file, if any, and control is returned to the caller.

5. TSPRMTC

This is the entry point called by any data base program desiring to request a command from the user. On entry, and internal buffer is checked for the presence of a previously entered command. If one is there, it is returned to the caller as satisfying the prompt. If the buffer is empty, a message key passed as a calling parameter is used to access a message file to obtain the text of the message which describes the context of the prompt to the user. This message is displayed in the prompt area of the user's I/O device and the terminal is opened for input.

The response to this prompt must be a command. It may be the one requested by the calling program, in which case it is passed along. Or, alternatively, it may be any of the ''immediate'' commands which cause one of the immediate command processors to be invoked. After all activities associated with the immediate command are completed, the execution cycle beginning on enty to TSPRMTC is repeated until a satisfactory response is returned to the caller or until APOFF or END processing is initiated.

Consult the Command System User's Guide for details of command syntax.

6. TSPRMTD

This entry point is called by a data base program wishing to obtain user-entered data. On entry, the save internal buffer that holds commands is checked for a parameter string that may have been entered with a command. If data is there, it is passed out of the string (in accordance with the syntactic unless outlined in the Command System User's Guide) and returned to the calling program. If the buffer is empty or the next item in it is a command, a message key passed as a calling parameter is used to access a message file to obtain the text of the message which will explain to the user what data is requested. The message is displayed

in the prompt area of the user's I/O device and it is opened for input.

The response to this prompt may be data or any of the immediate commands. If it is data, it is passed as above and returned to the user. The terminal Control Block sewer as a center for communicating information about the data between TSPRMTD and its caller.

If the response is an immediate command, this command and its associated parameters are treated separately from any user input intended for a data base program. When processing of the immediate command is complete, the cycle beginning on entry to TSPRMTD is repeated until a satisfactory response is received or until APOFF or END processing is initialed.

Consult the Command System User's Guide for the details of parameter syntax.

7. TSPRMTM

This entry point is called to display a message from LISRMLF on the user's terminal. No reply is asked for. Auxiliary subroutine entry points are called from various locations in RTSUPER to perform prompting tasks.

The message filter MSGLEVEL in the user's profile determines wether or not informational (I-level) messages are displayed. Warning (W-level) messages are always transmitted.

The message ID filter MSGIDS specifies insertion of the message key between the message prefix and the text. MSGIDS=Y requests display of message keys. MSGIDS=N implies no keys.

If the last output to the display area left residual data undisplayed, the ''MORE'' message is substituted for any command or data prompt message. The key of every message (except explanations) is placed in the EXPLIST area for reference by the EXPLAIN command.

8. TSPUT

TSPUT may be called one or more times by data base programs to format data (passed as a parameter) in a buffer for output. The data consists of a string of characters to be displayed on the user's terminal and an optional tag field which is appended to the beginning of the string. Formatting consists of manipulating the data so that it appears in a consistent and logical pattern on the screen.

On entry, TSPUT initializes pointers and work areas based on whether a restart, continuation or backwards put is indicated. After insuring there is sufficient room in the buffer to insert new data, a subroutine is called to move the tag and data string to the output buffer. This step is repeated until all data is in the buffer or the buffer is filled. An attempt is made to terminate lines between words and at punctuation.

On buffer overflow, if the caller does not want overflowed records inserted, all pointers are reset and control is returned to the caller. If partial records are inserted, control characters are appended, a TC Block variable is set to indicate the number of characters taken and the AUTOWRITE switch is checked. If it's on control passes to the FLUSH routine, otherwise control is returned to the caller.

If all data is inserted with no overflow, the trailing position of the record is padded with blanks (to fill out a screen line) and control is returned to the caller.

9. TSWRITE

This routine is called to flush the contents of the external storage named TSCREEN. After locating the area control is passed to FLUSH, which outputs the data and returns control to the caller.

c. Subroutines

1. GETPR

This routine calls the PL/I controlled storage allocation routine ''IHESADD'' to obtain space into which the master psect may be copied. The caller's registers are saved in area common to the copy routine so after the area is obtained a branch is taken to MOVECOPY and from there control is returned to the caller.

2. DELPR

This routine simply deallocates the external controlled storage allocated by GETPR. The PL/I service routine ''IHESAFF'' is called to perform this function.

On return, register 12 is set to point to the next area in the chain and control is returned to the caller.

3. GETSYN and GETDFALT

These two subroutines primarily the save code, the differences being in the lengths of the parameter list used in the eventual call to an external program and the v-core which is posted in register 15 and points to the program which is called. GETSYN calls TSGSYN to obtain a synoym for a term. GETDFALT calls TSGDEF to obtain a default value for a parameter. On return from the respective calls, the length of the returned data is checked. If nothing came back, the data pointers are reset to point to the data used as a calling parameter.

4. PULINEND

This subroutine is called by TSPUT to insert the proper end-control characters on each line of display output as it is moved into the output buffer. Screen lines are padded with blanks to fill out the line. Typewriter lines are terminated with an interpretive hex

15.

5. GIVITD and GIVITC

These two subroutines are called by the prompting routine to pass data to the user. If the prompt processing is in skip mode or the call was inadvertently done before an item-was found, the pass is not done. Otherwise SDPASS is called to move the data to the user's area.

On return, the passed data is excised from the input buffer. If it came from a parenthesized list, the list flag is set in the terminal Control Block and control is returned to the user.

6. SDPASS

SDPASS compares the length of the data or command passed from the input string with the receive area. If the item will fit the area, it's moved, otherwise a syntax error is noted and error processing is begun.

7. SDSYNCHK

This routine is called to check for certain syntax errors. If an error is detected, control is transferred to SYNNER to initiate an error control sequence. Otherwise, control is returned to the point of call.

8. RESETBUF

Preparing a buffer for input and initializing all flags associated with input passing is performed here.

9. SDSTRIP and STRIP

SDSTRIP is called to delete leading and trailing quotes and leading and trailing blanks from an item passed as input to a calling program. If only blanks are to be deleted, entry is at STRIP.

10. PRKEYSAV

Inserts the key of a prompting message

into a push-down list of message keys for reference by the EXPLAIN command.

11. IMCHECK

Whenever the user enters an immediate command, it is discovered by this subroutine. Comparing the entered command against a table of valid immediate commands, a ''hit'' leads to either signalling ''END'' or calling an external program to initiate processing. On return, the prompt routine is informed of the occurrences and the prompting cycle begins again.

12. SIGNAL and SIGNALC

Entry at SIGNAL causes preparations to call the PL/I service routine IHEERRD. Control then falls through to SIGNALC, which calls a pre-indicated routine and, on return, itself returns.

13. SETLDBA

This routine opens and initializes the DCBs for the prompt message libraries. Also, it issues a SETL to find a particular message key—in the file. Member LISRMLF of DBALIB is searched first, followed by member LISRMLF of LISRLIB. If the Key is not found in either DBALIB or LISRLIB, a substitute message is written which indicates the message was not found.

14. GETMLF

Its function is to read the text of a message record pointed to by a message key. Each record read is check for the prescence of a minus sign (-) or plus sign (+) is its last character.

If there is a minus sign, the next record is read and appended to the first. If the last is a plus sign, the truncation bit in the TC Block is set to one(1) and control is returned to the caller.

15. MOVE

All extended data relocations are performed by this routine. In addition it is also used to blank-fill a data area and copy from one area to another.

16. PROMPT

On entry, if the user is in RESTART or RERUN mode the next record of input is obtained from the strategy dataset named in the external control block USERTAB. Otherwise pointer and constants are set in the I/O control block and MTT is called to do an I/D.

On return from MTT, the return code in register 15 is checked. If there was an error, attention interrupt or continuation the I/O is retried. Otherwise, the data is moved to a work area and control is returned to the caller.

17. EXIT

Returning to any program calling an RTSUPER entry point is accomplished by passing through this code. The caller's TC Block is updated by moving cur copy of it back into the caller's area. The PRV is restored in register 12 and control is returned by calling the PL/I service routine IHESAFA which releases our Dynamic Storage Area (DSA) and restores the callers registers.

F. CODING SPECIFICATIONS

1. Source Language

TSS/360 Assembler Language.

2. Suggestions and Techniques

Not Applicable

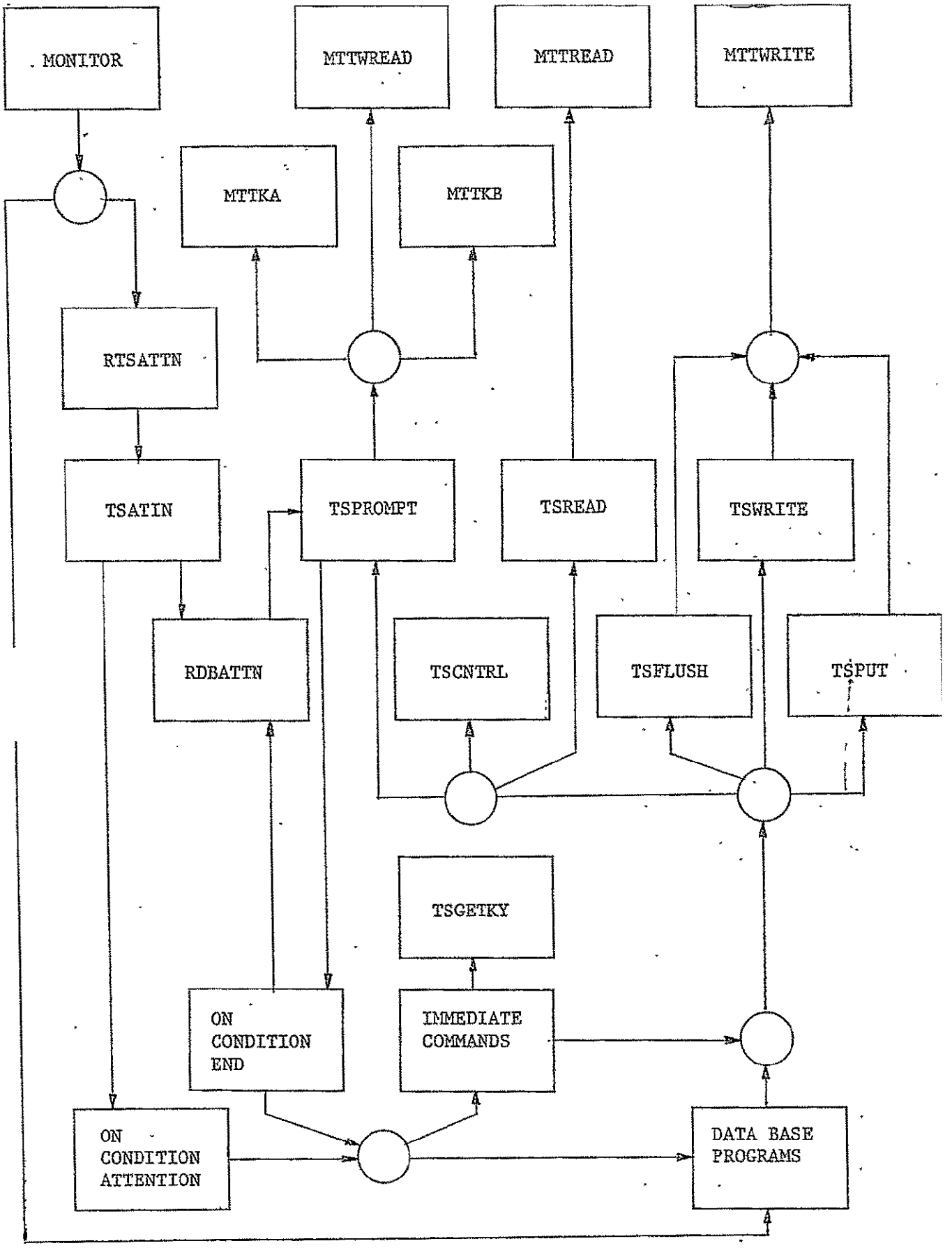


Figure 1. Terminal Support Organization Chart

IV.5.2.1

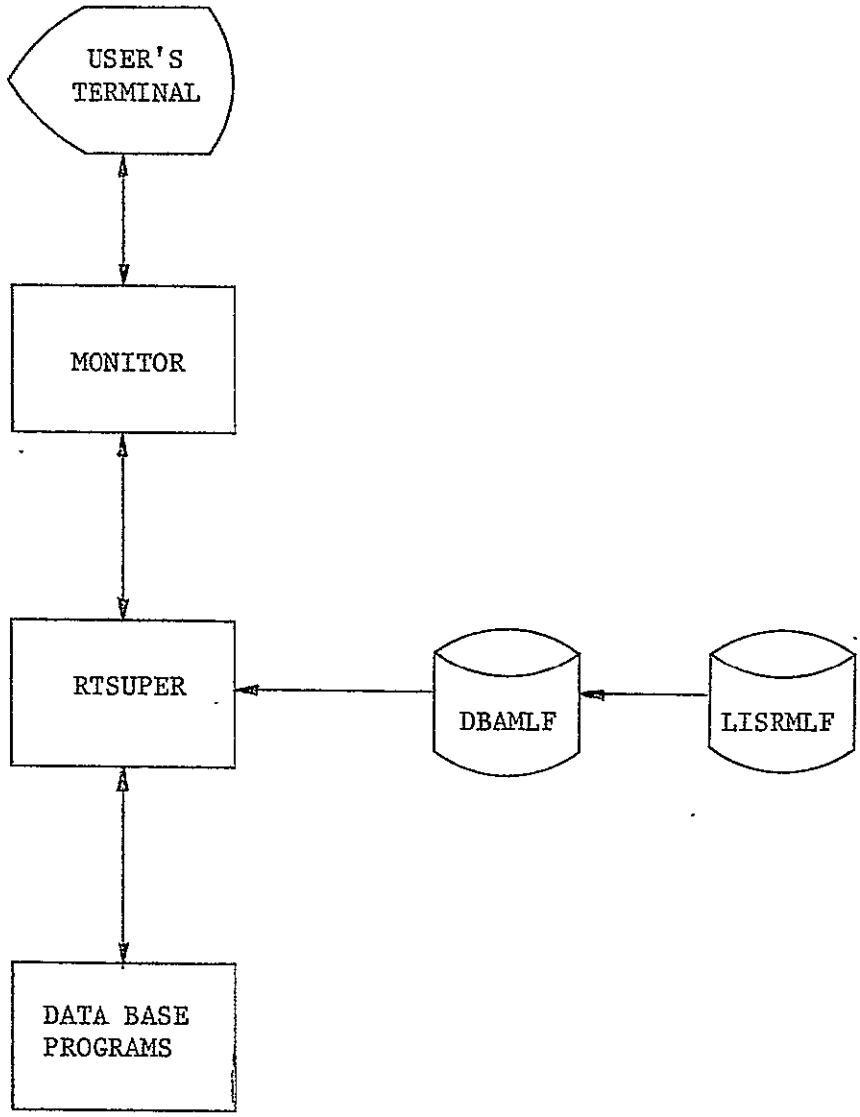


Figure 2. I/O Block diagram

IV 22.2

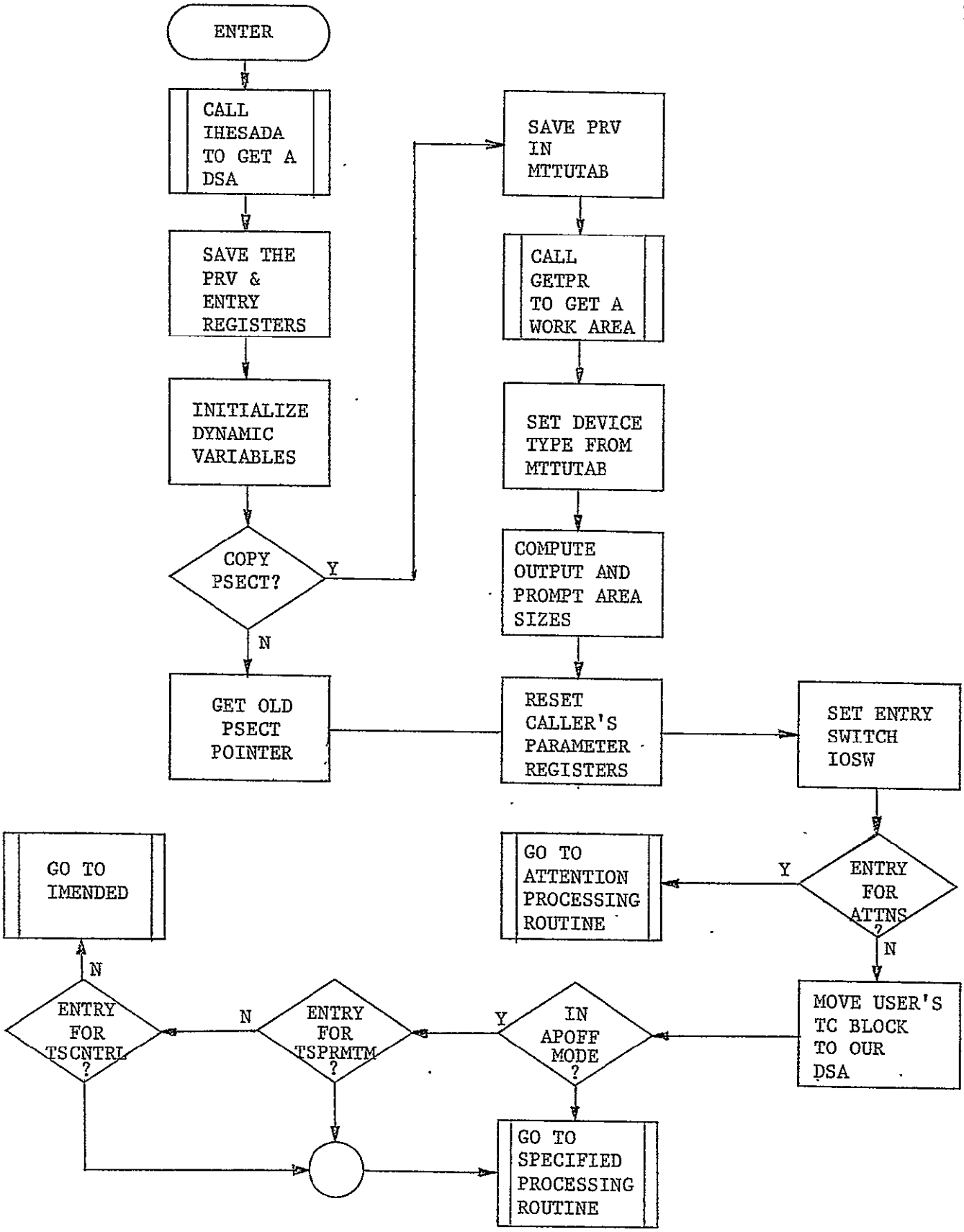


Figure 3. Terminal Support MAINLINE

IV, §2.3

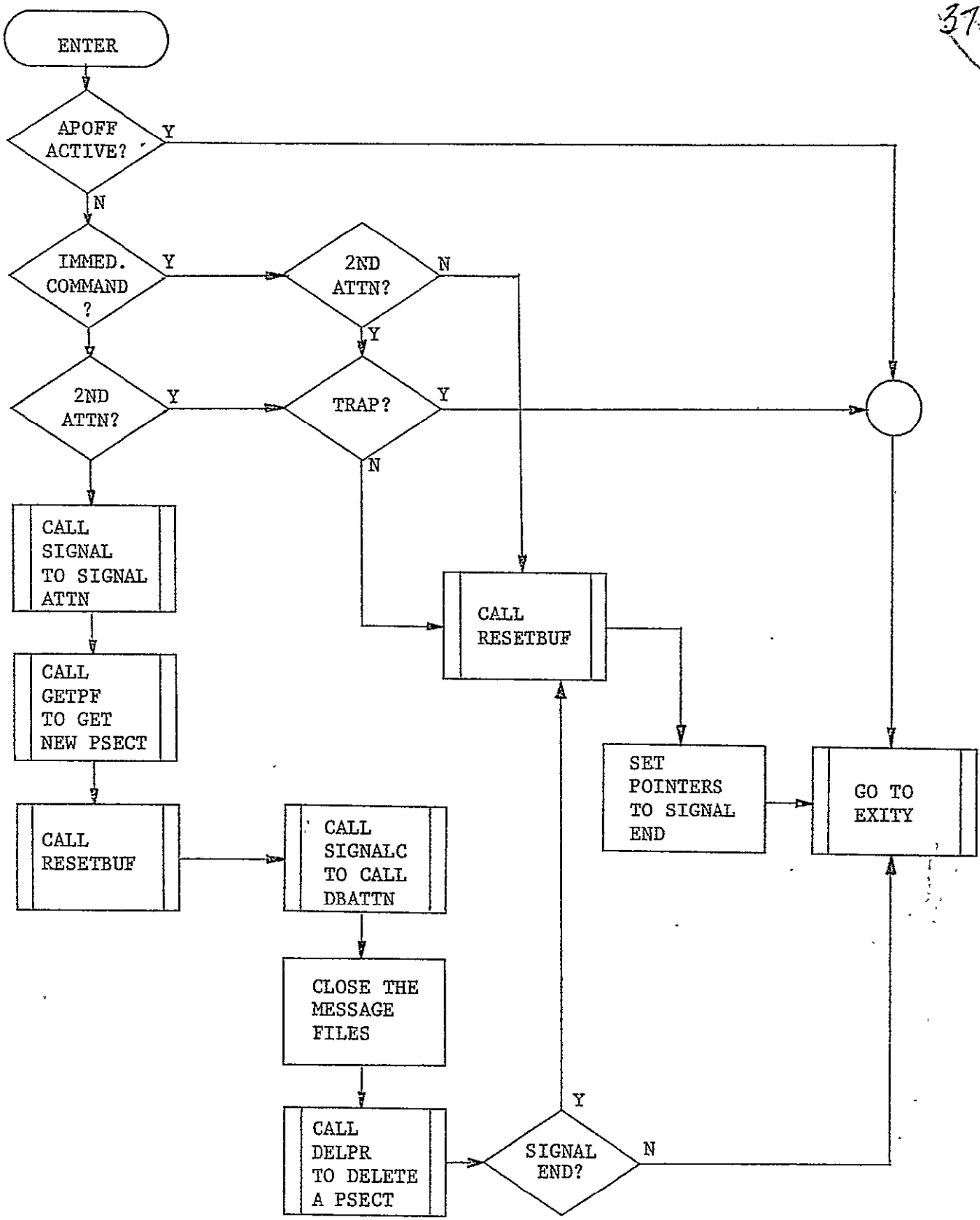


Figure 4. Entry Point TSATIN

IV §2.4

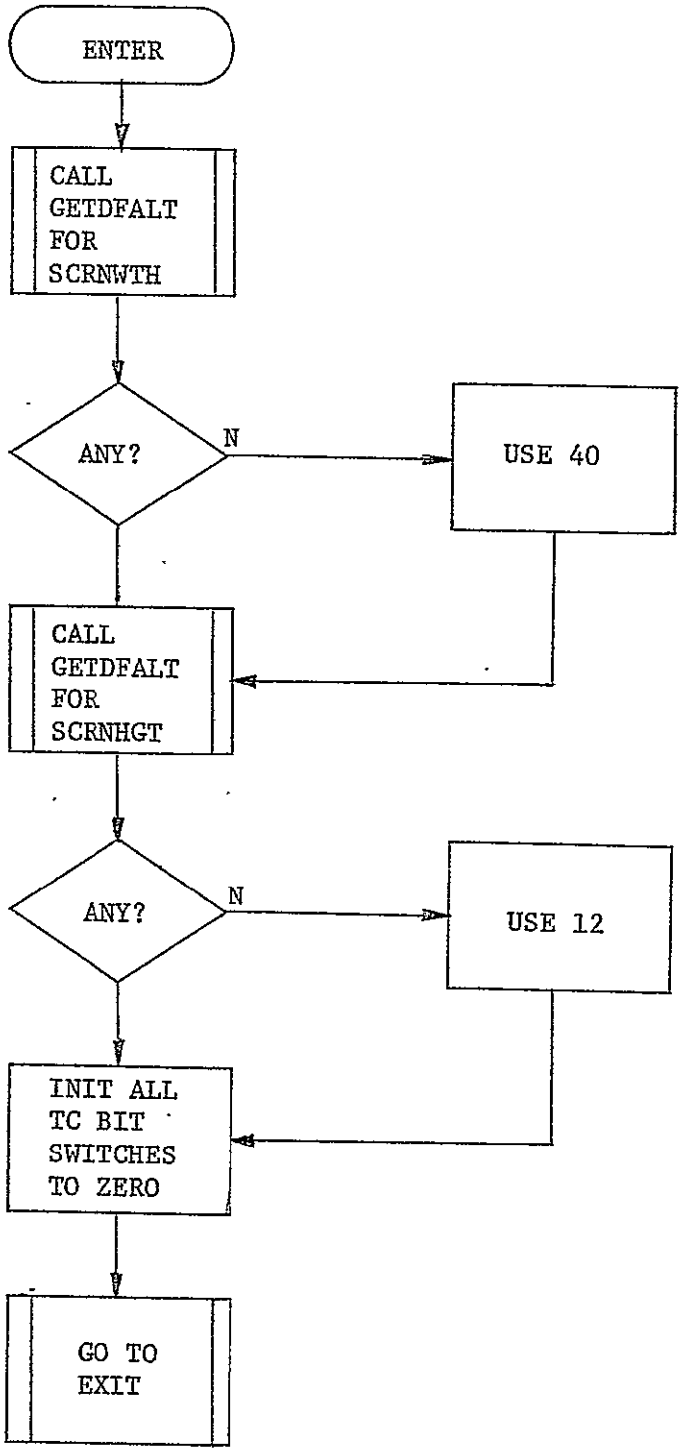


Figure 5. Entry Point TSCNTRL

IV §2.5

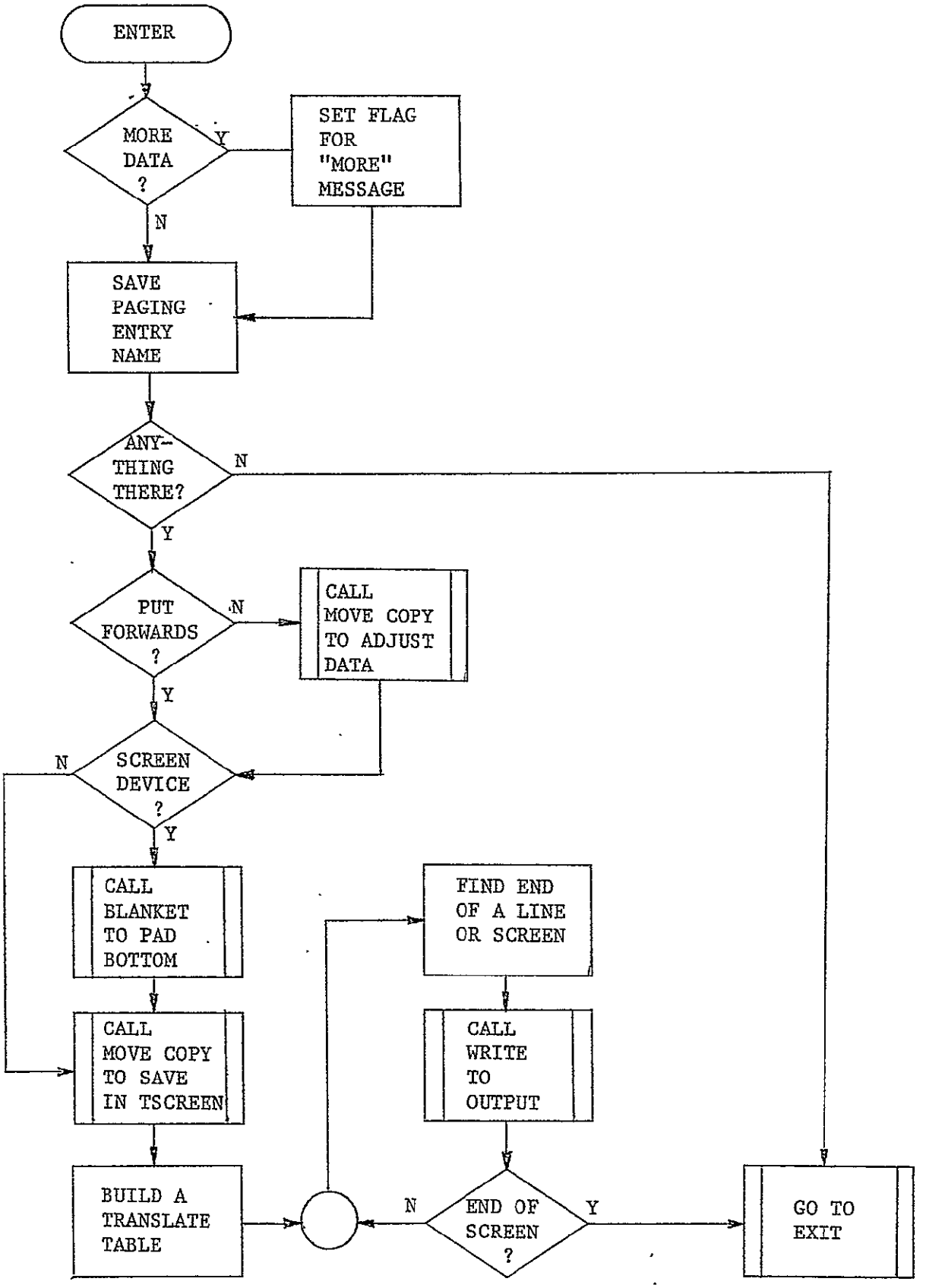


Figure 6. Entry Point TSFLUSH

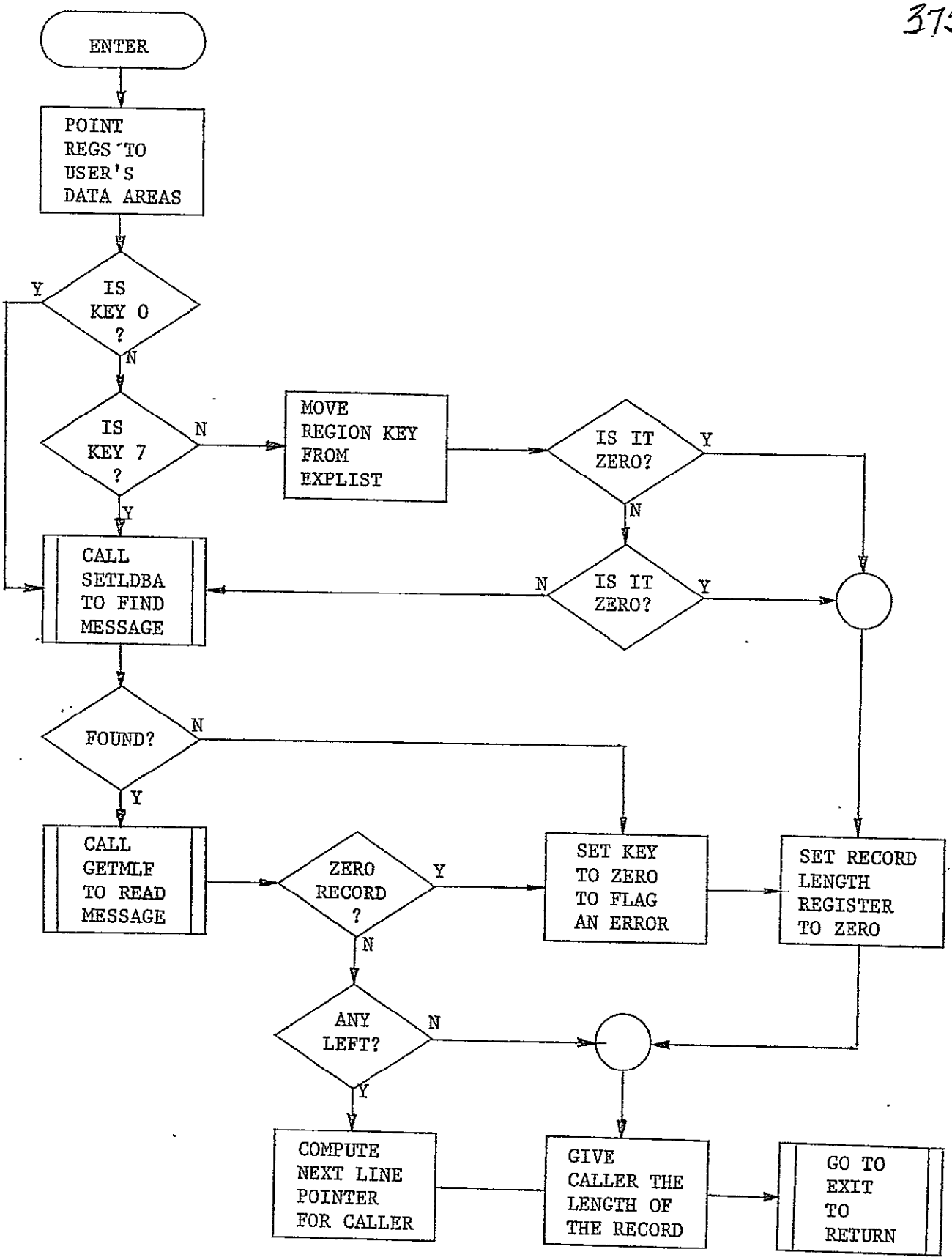


Figure 7. Entry Point TSGETKY

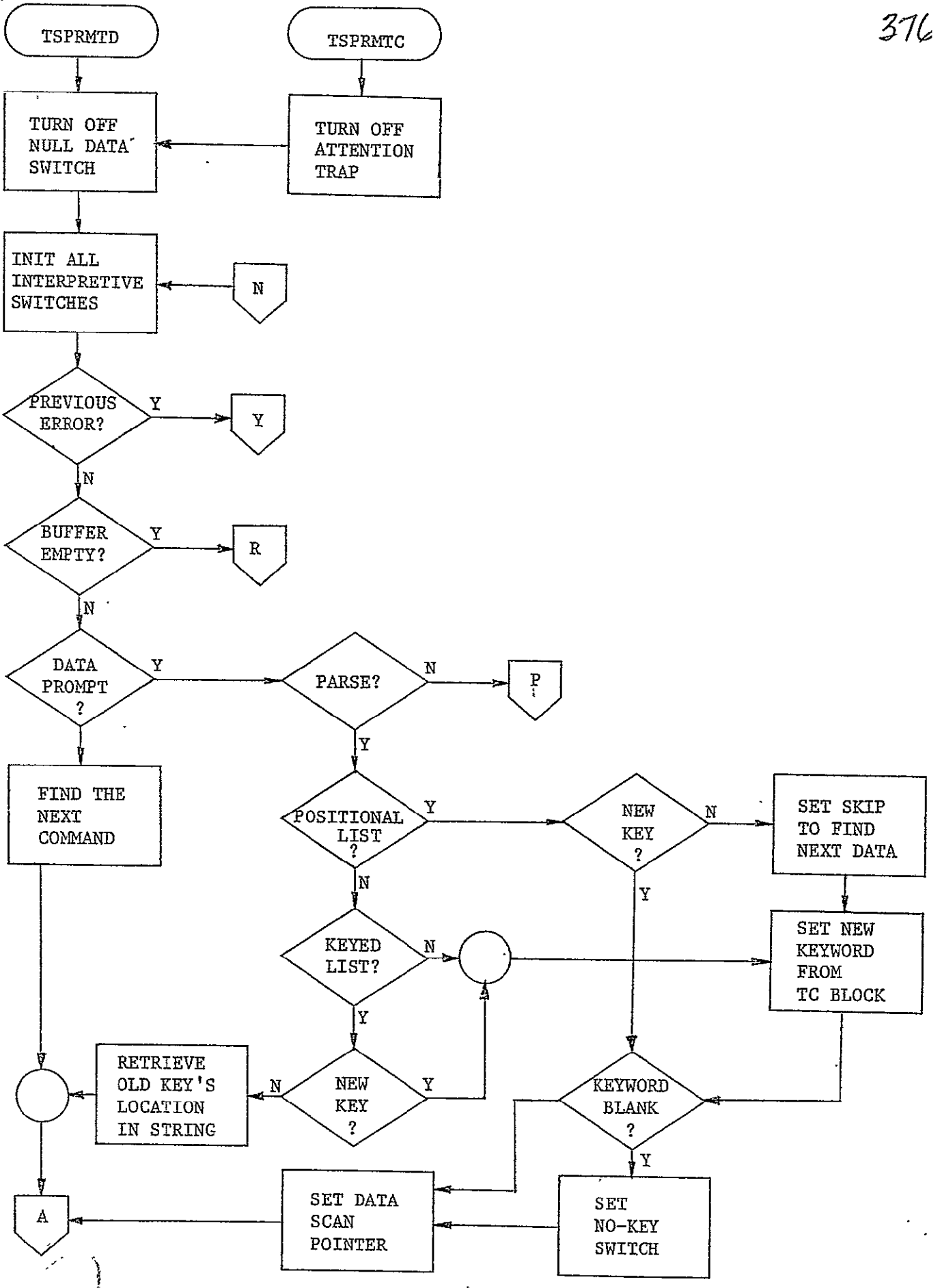


Figure 8. Entry Points TSPRMTC and TSPRMTD

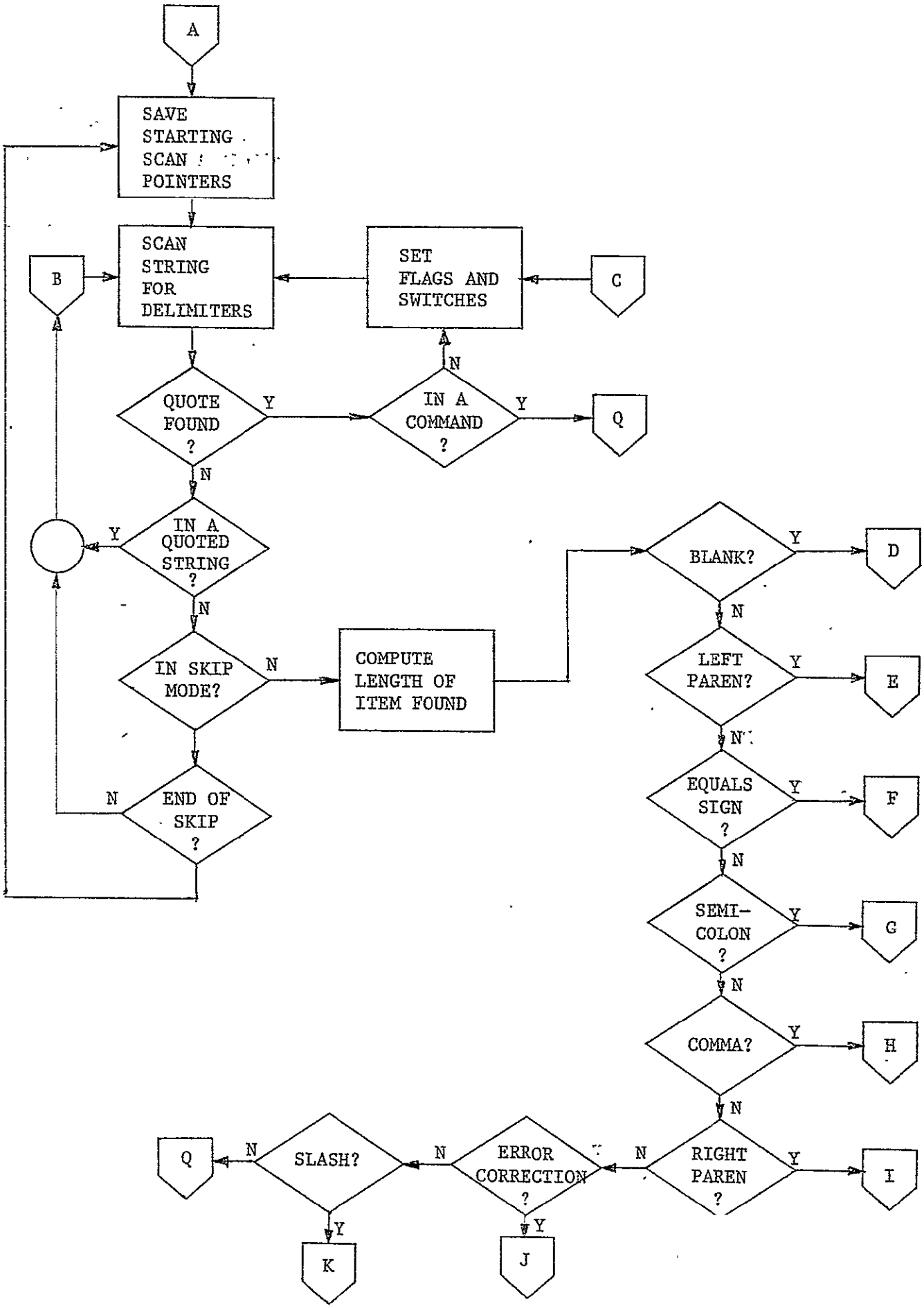


Figure 8A. Entry Points TSPRMTC and TSPRMTD

IV 22.8A

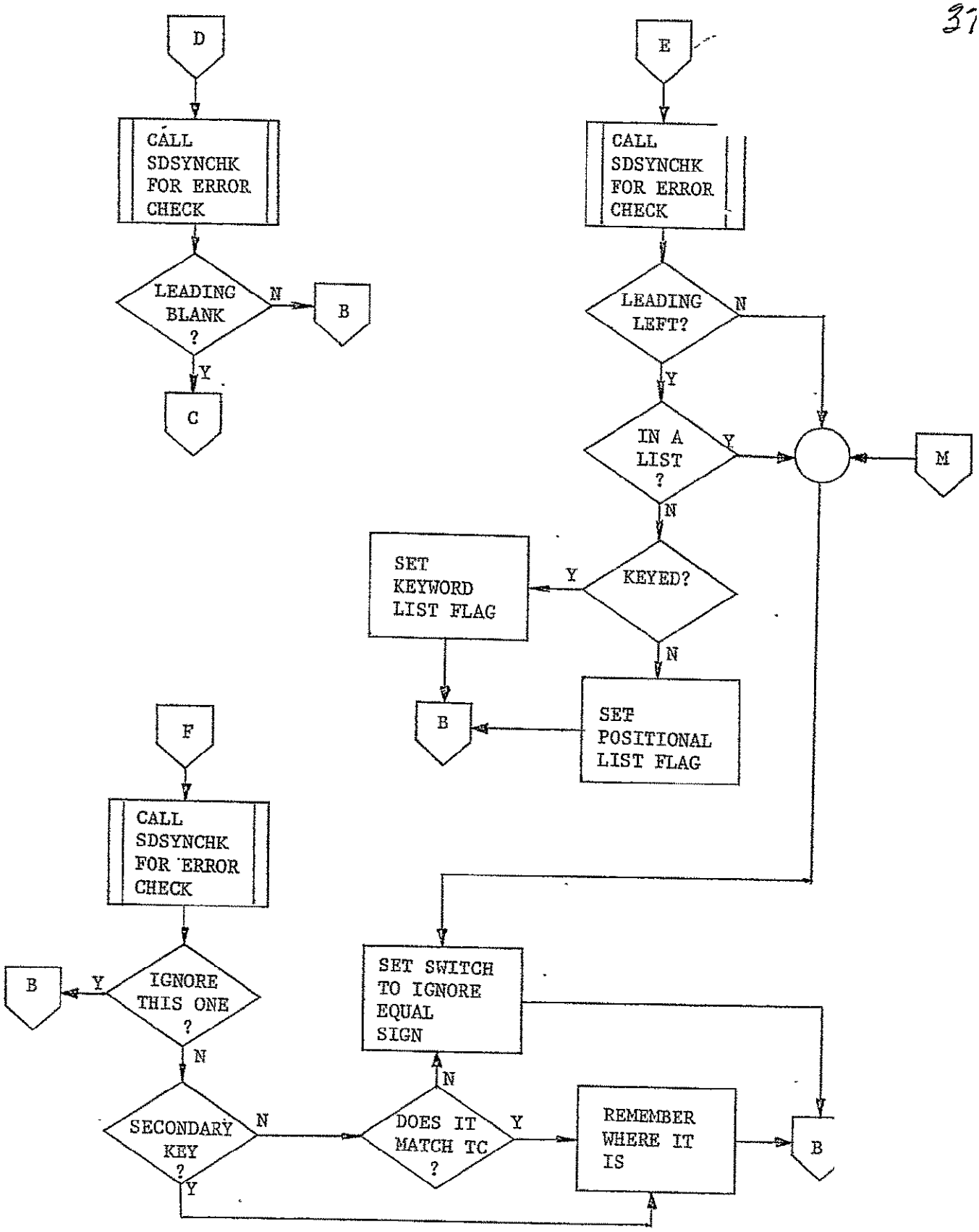


Figure 8B. Entry Points TSPRMTC and TSPRMTD

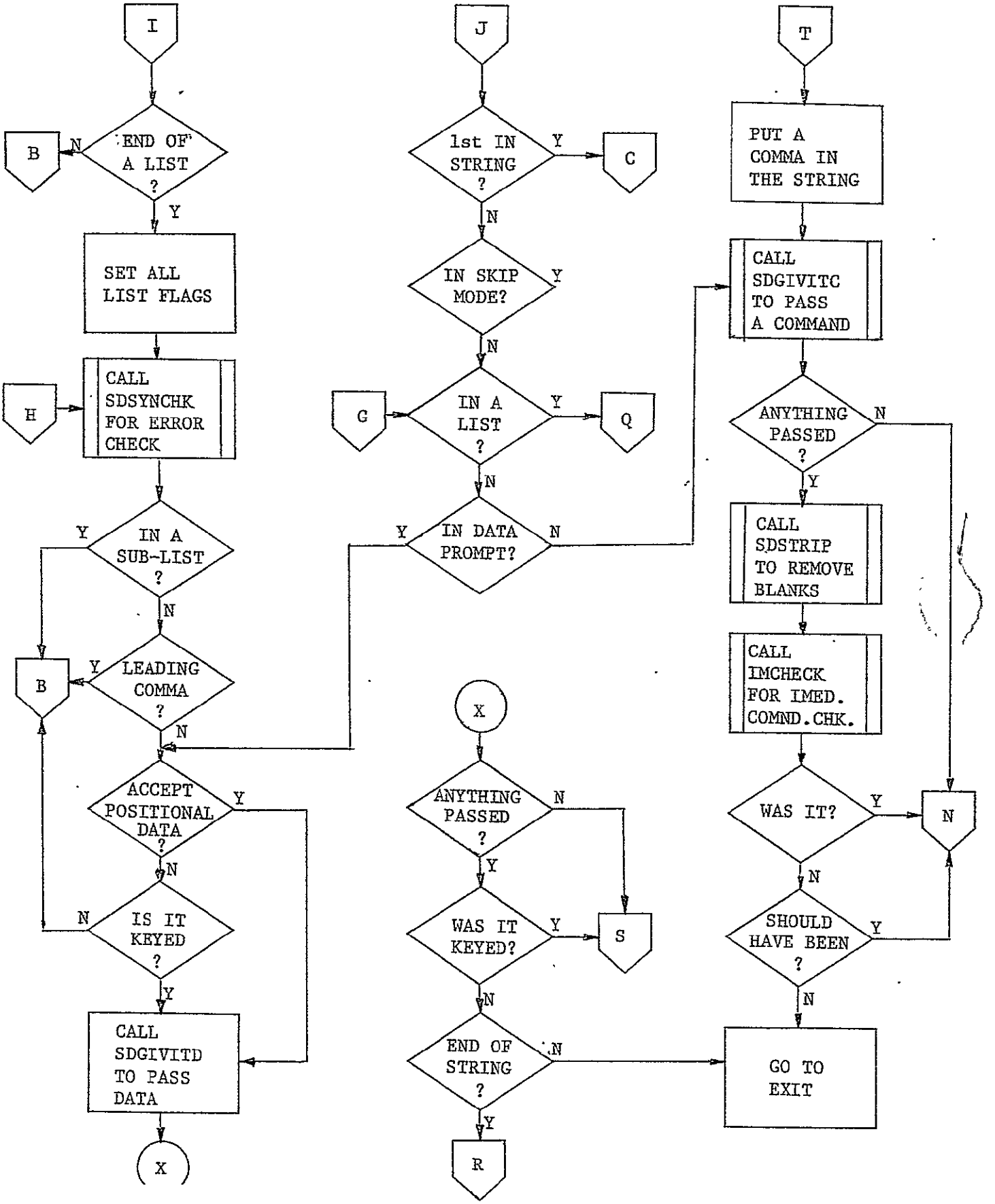


Figure 8C. Entry Points TSPRMTC and TSPRMTD

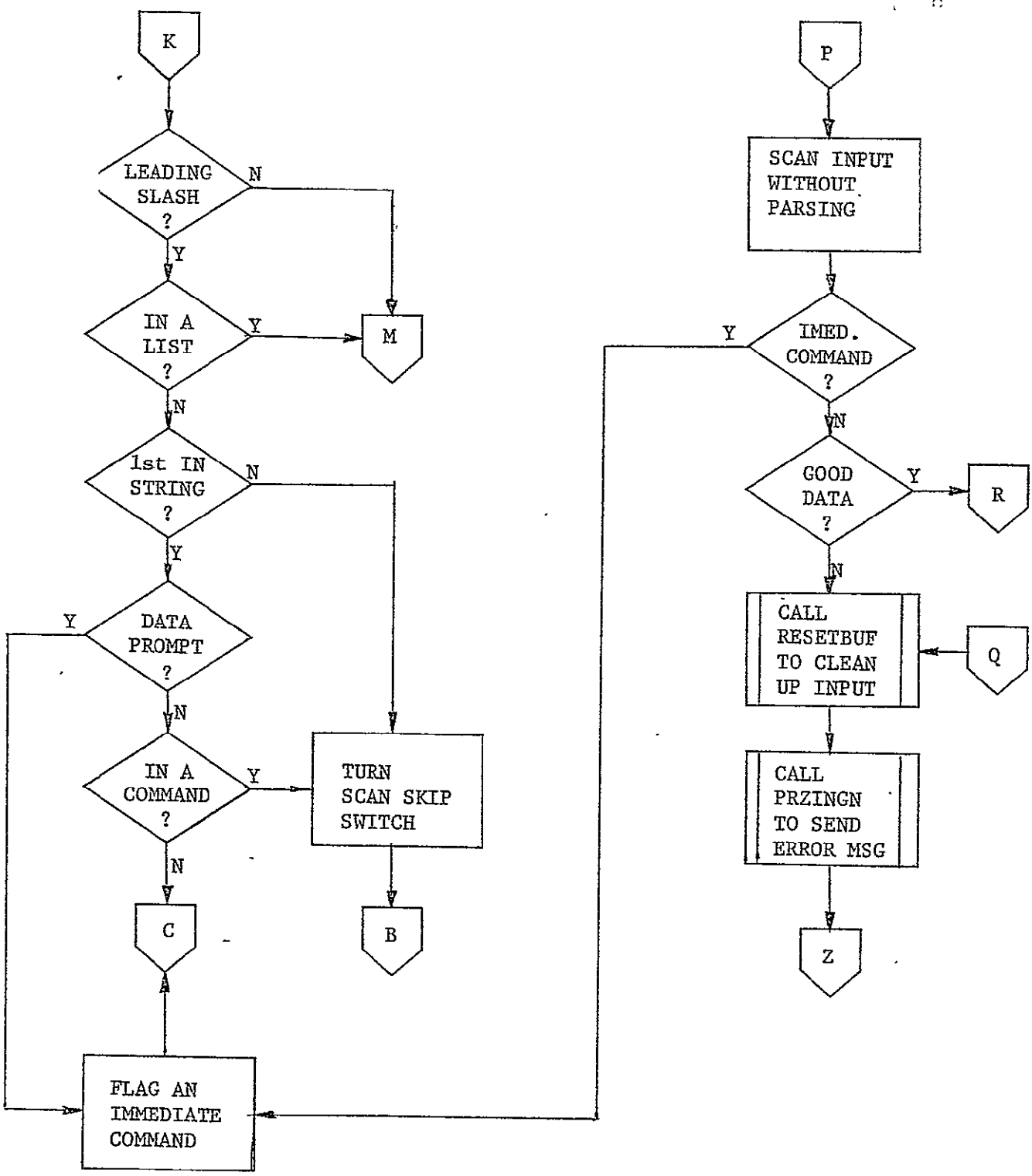


Figure 8D. Entry Points TSPRMTC and TSPRMTD

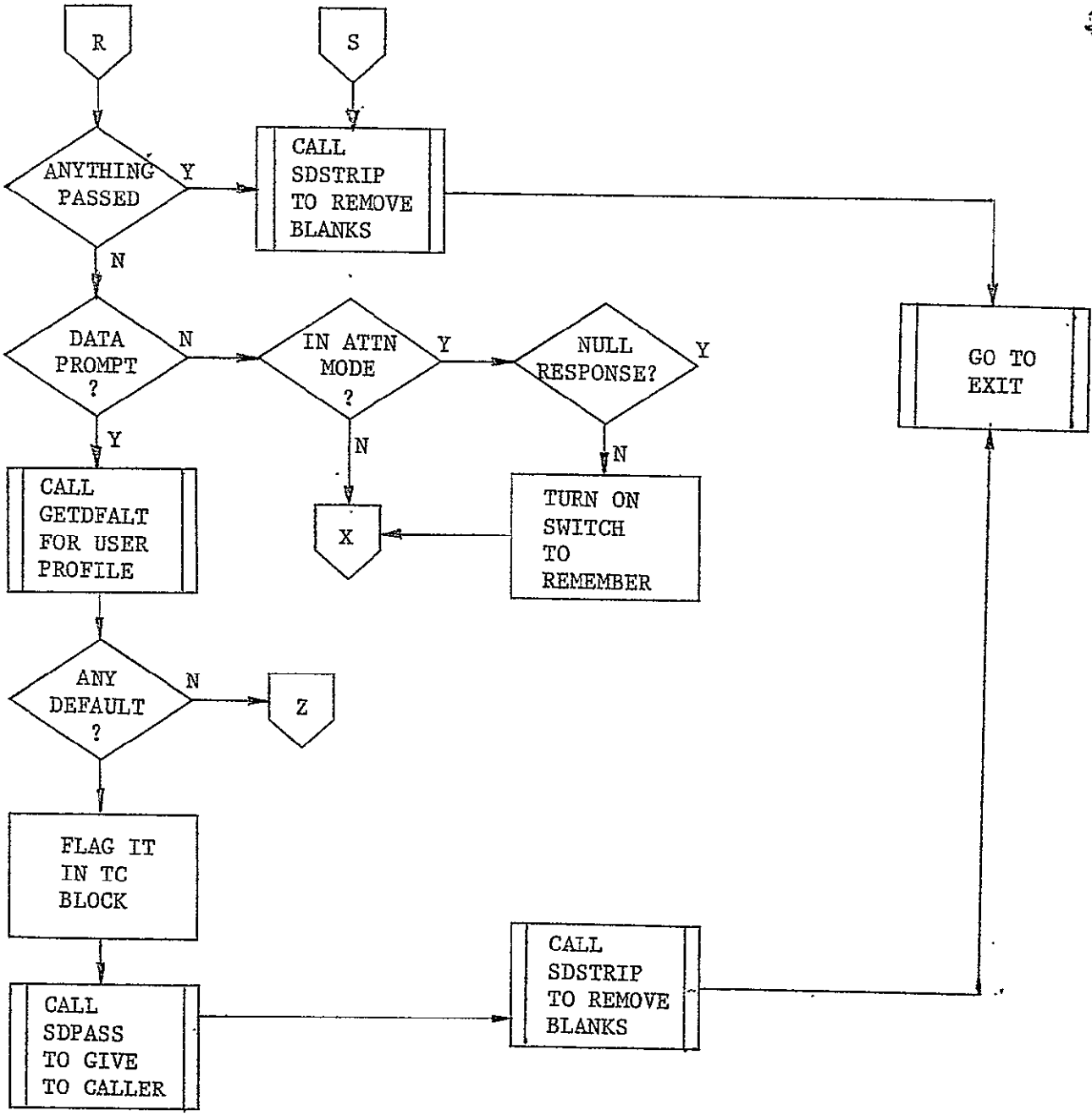


Figure 8E. Entry Points TSPRMTC and TSPRMTD

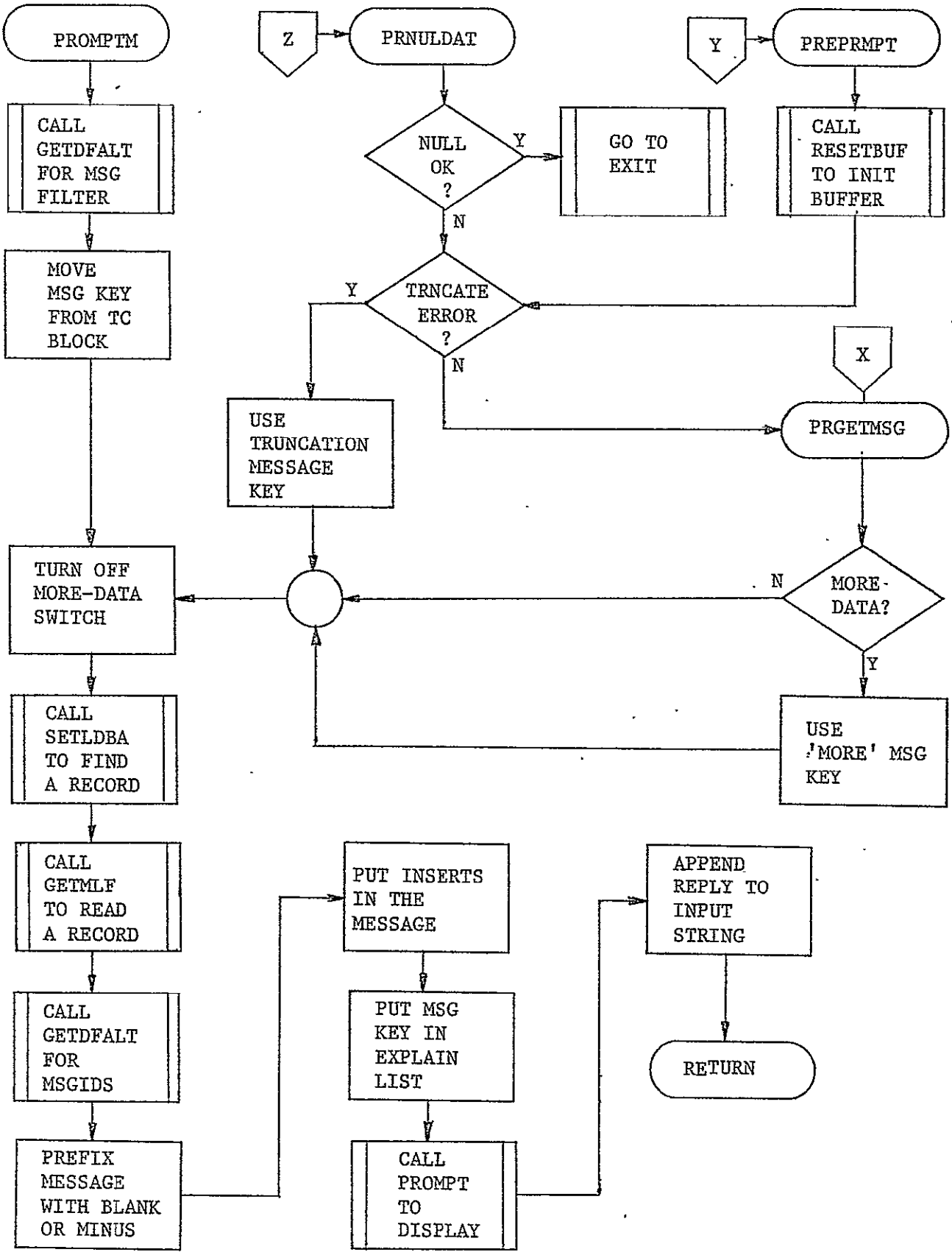


Figure 9. Entry Point TSPRMTM

6.23

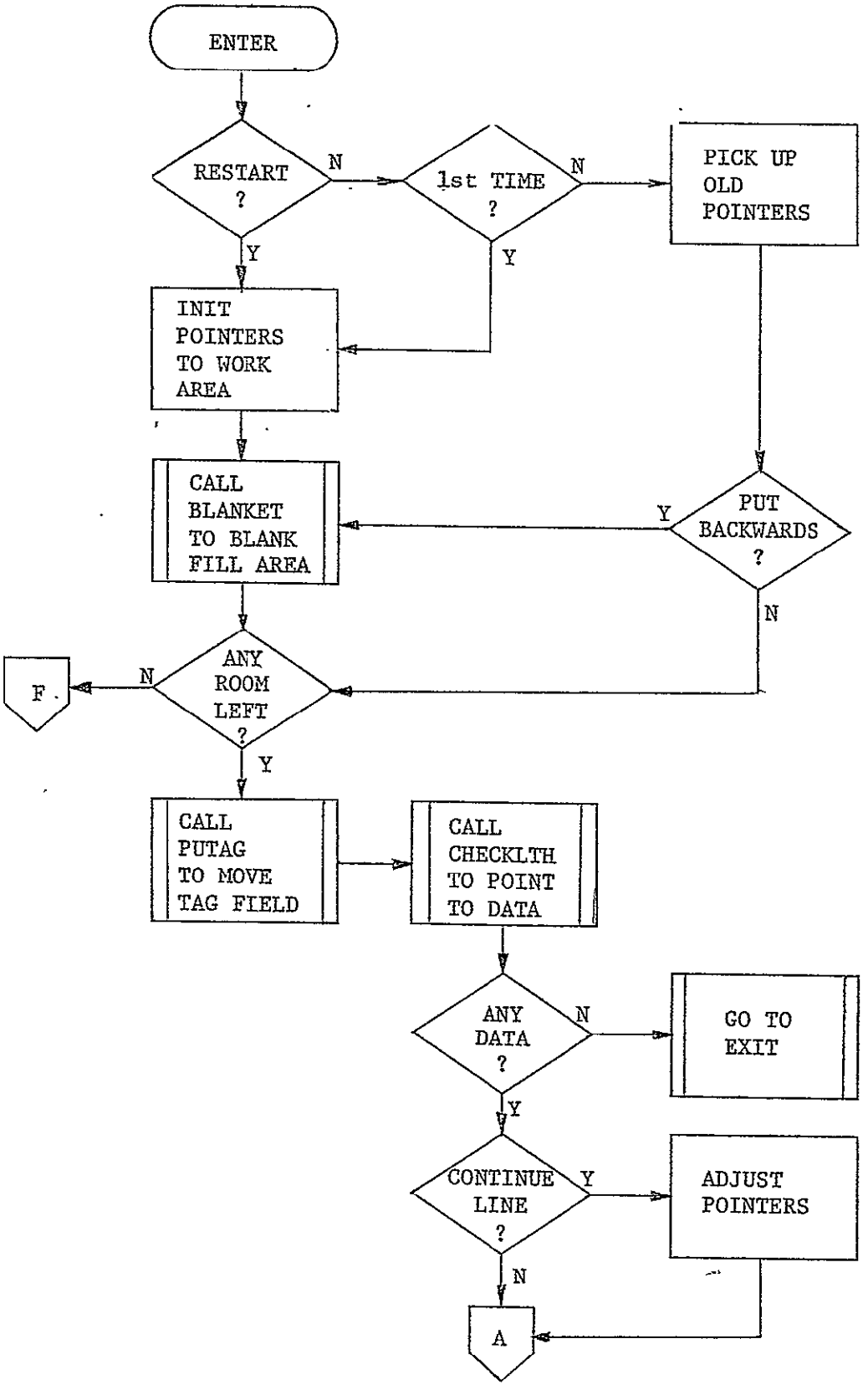


Figure 10. Entry Point TSPUT

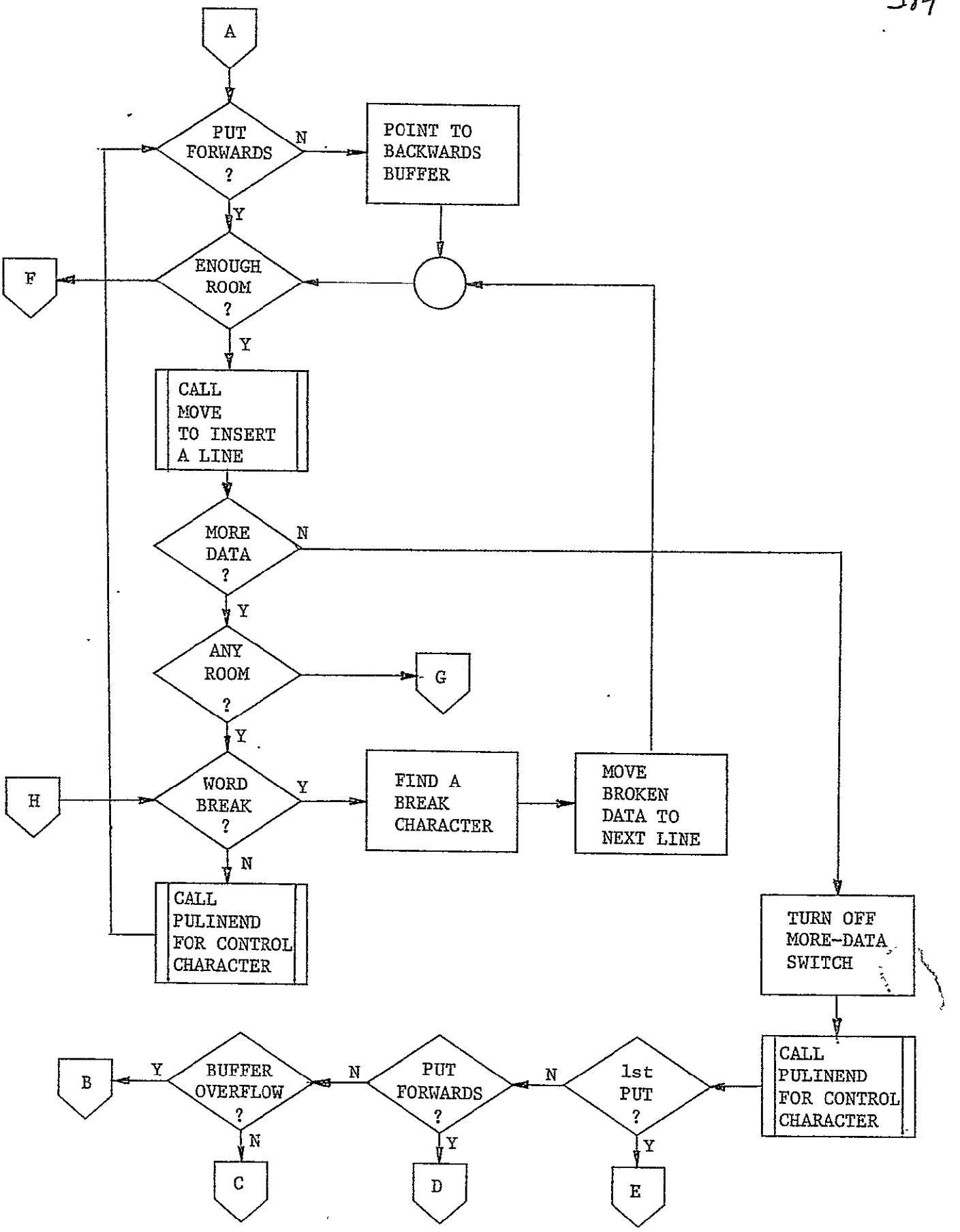


Figure 10A. Entry Point TSPUT

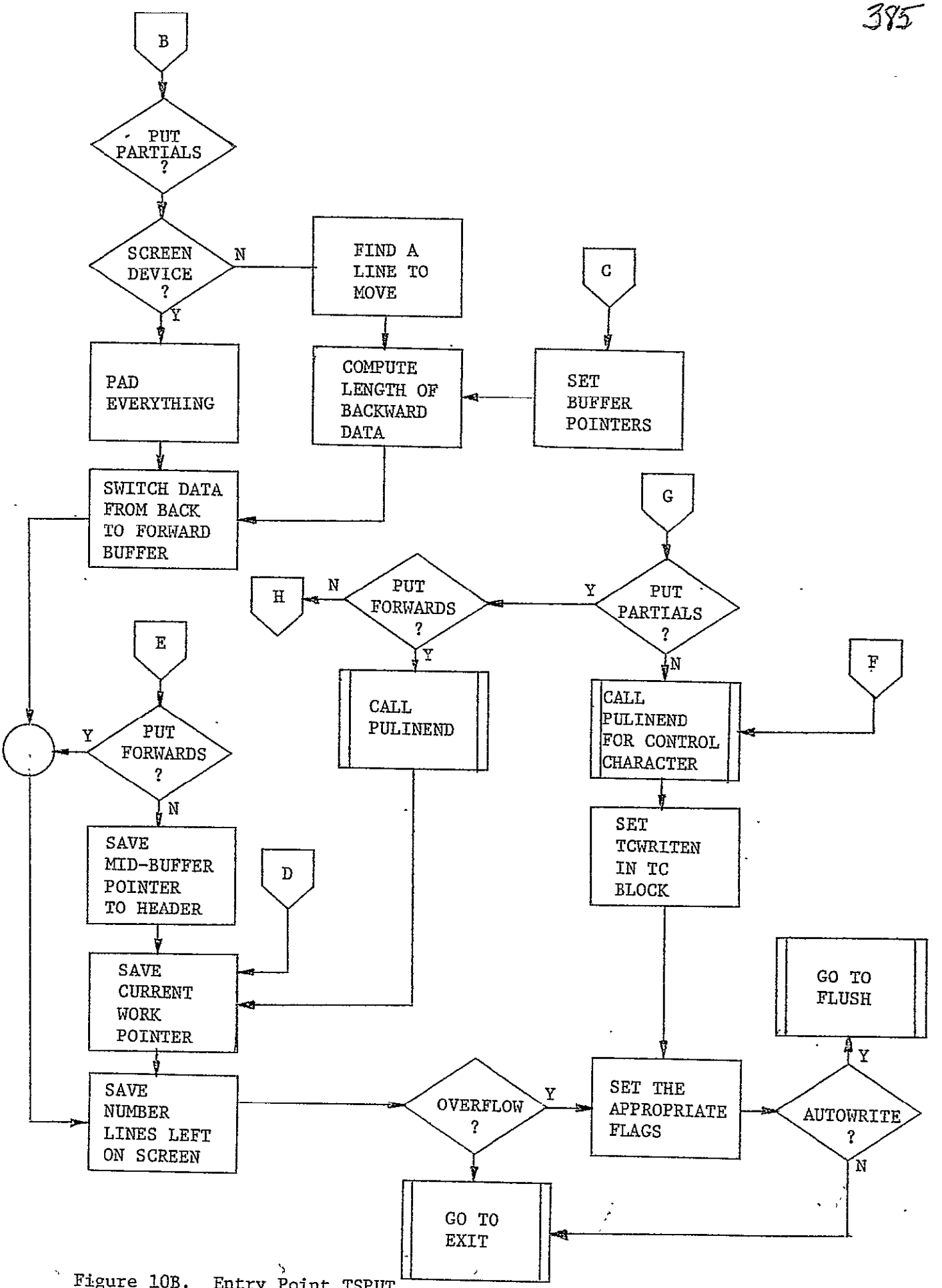


Figure 10B. Entry Point TSPUT

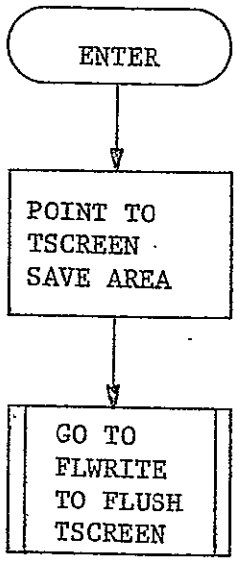


Figure 11. Entry Point WRITE

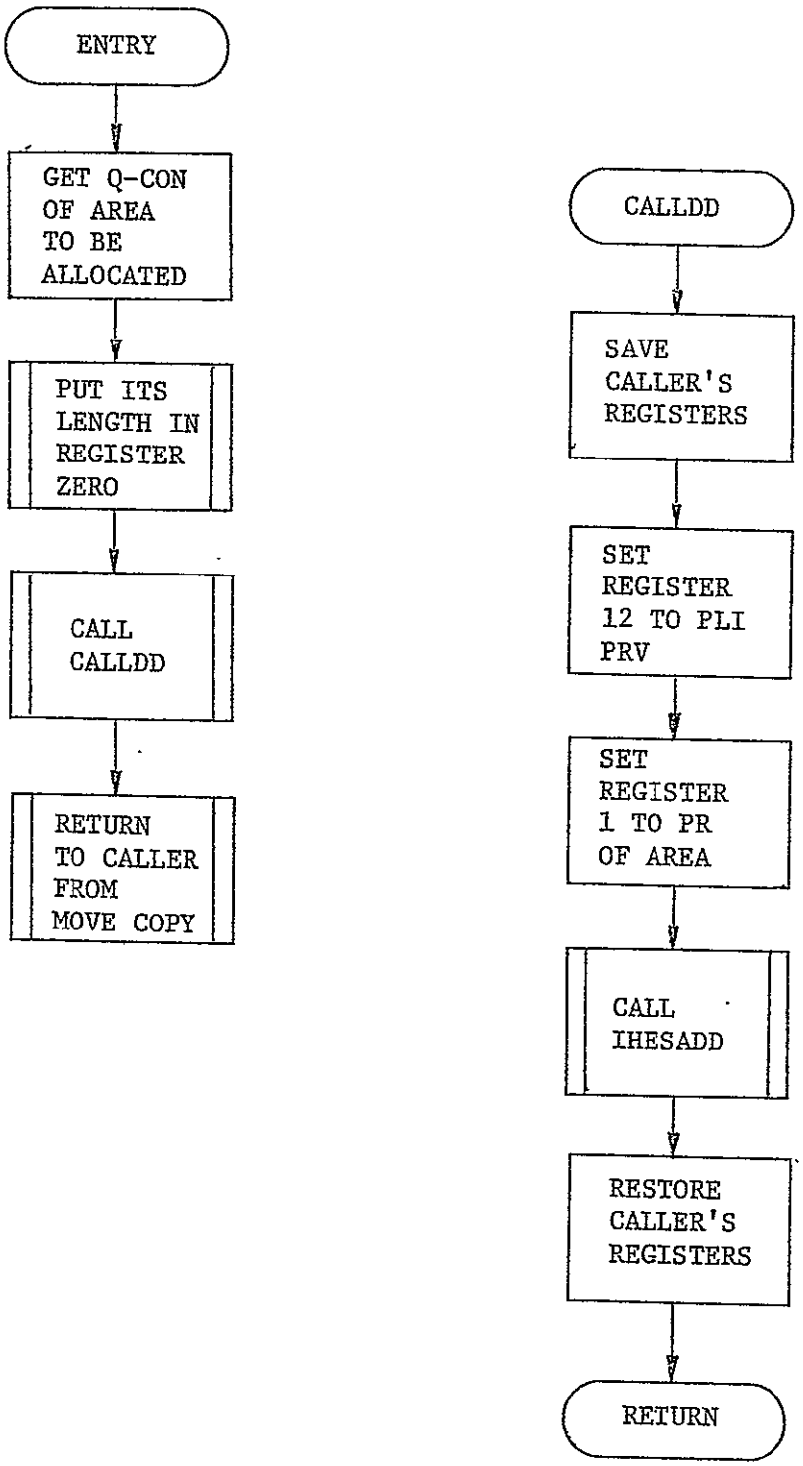


Figure 13. Subroutine GETPR

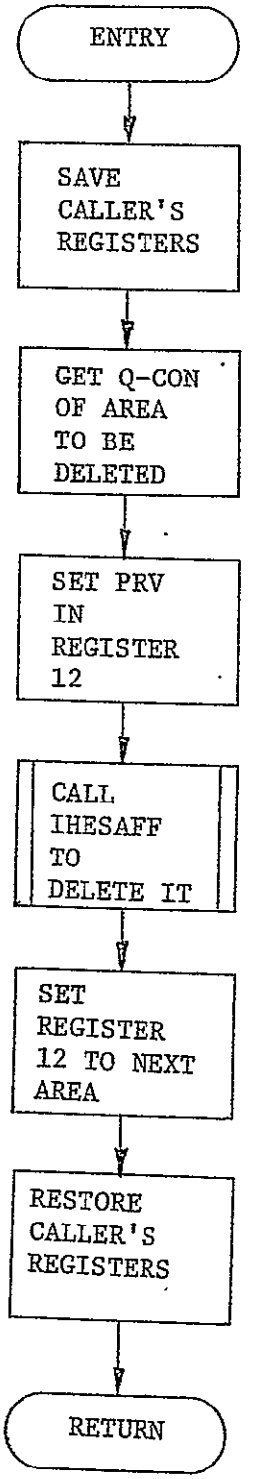


Figure 14. Subroutine DELPR

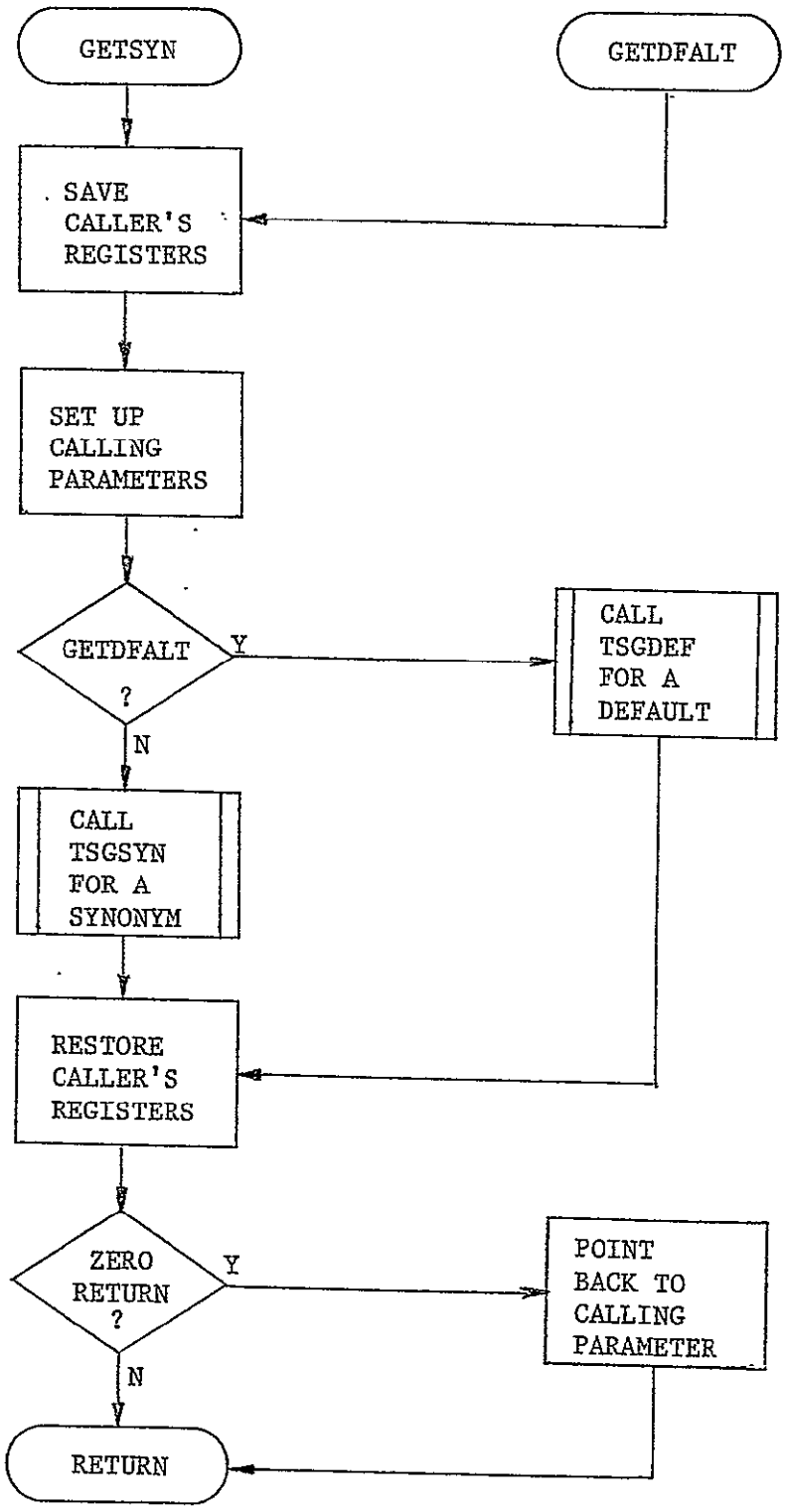


Figure 15. Subroutines GETSYN and GETDFALT

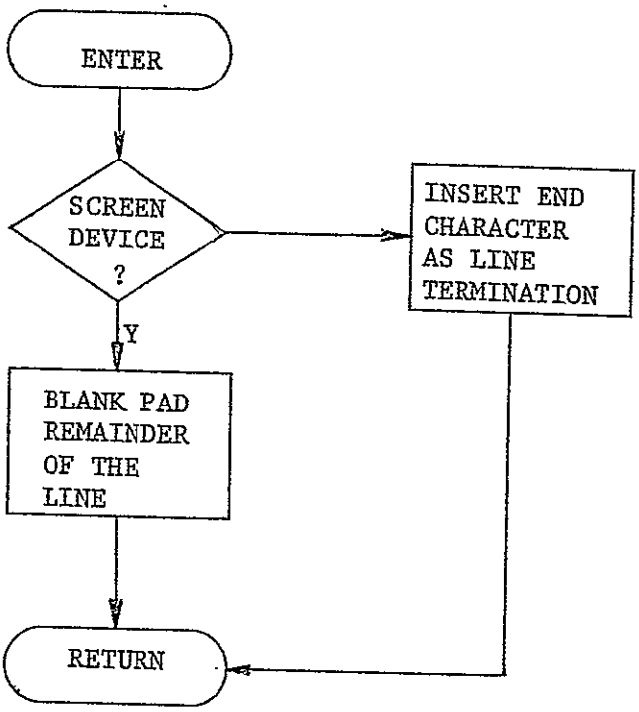


Figure 16. Subroutine PULINEND

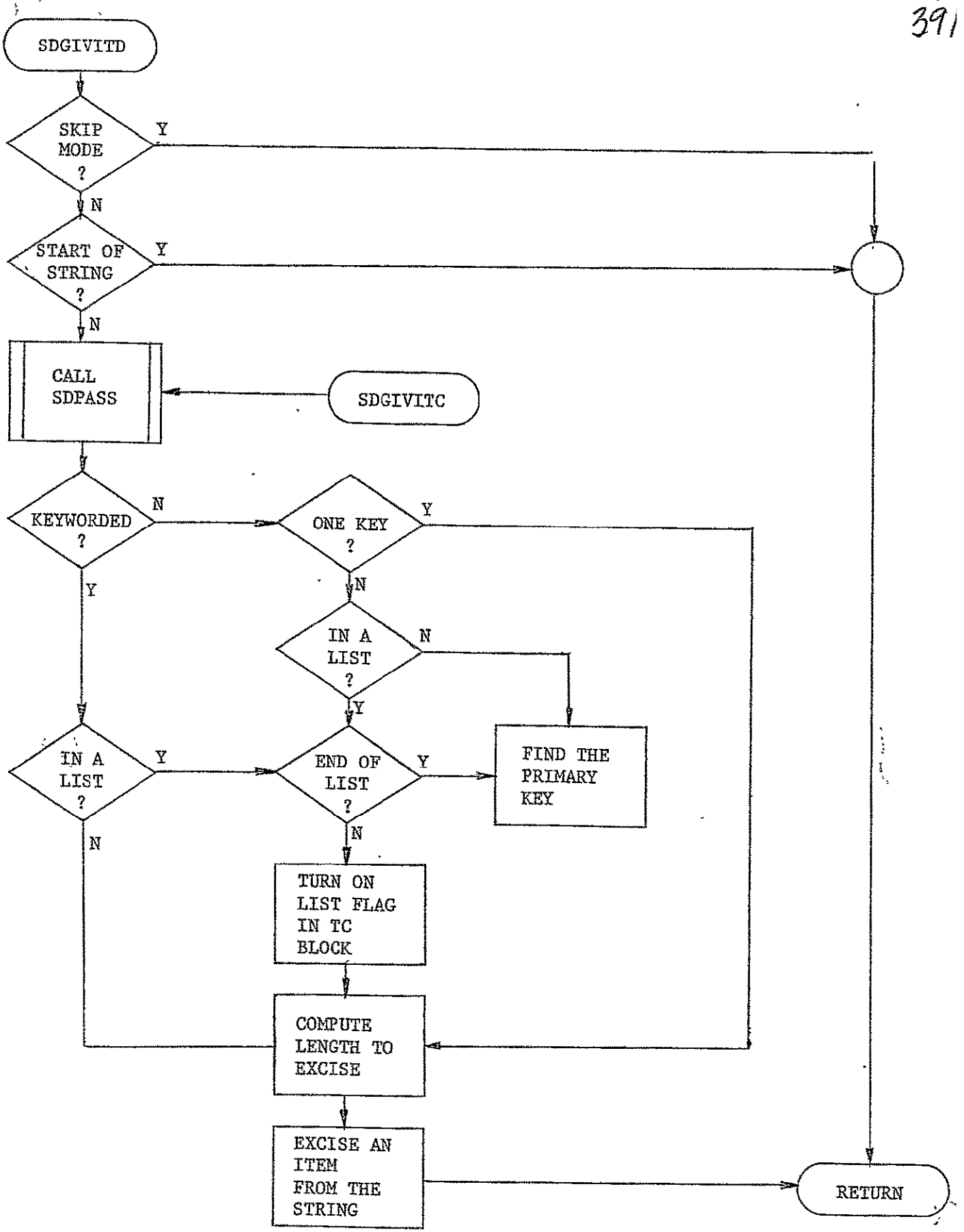


Figure 17. Subroutines SDGIVITD and SDGIVITC

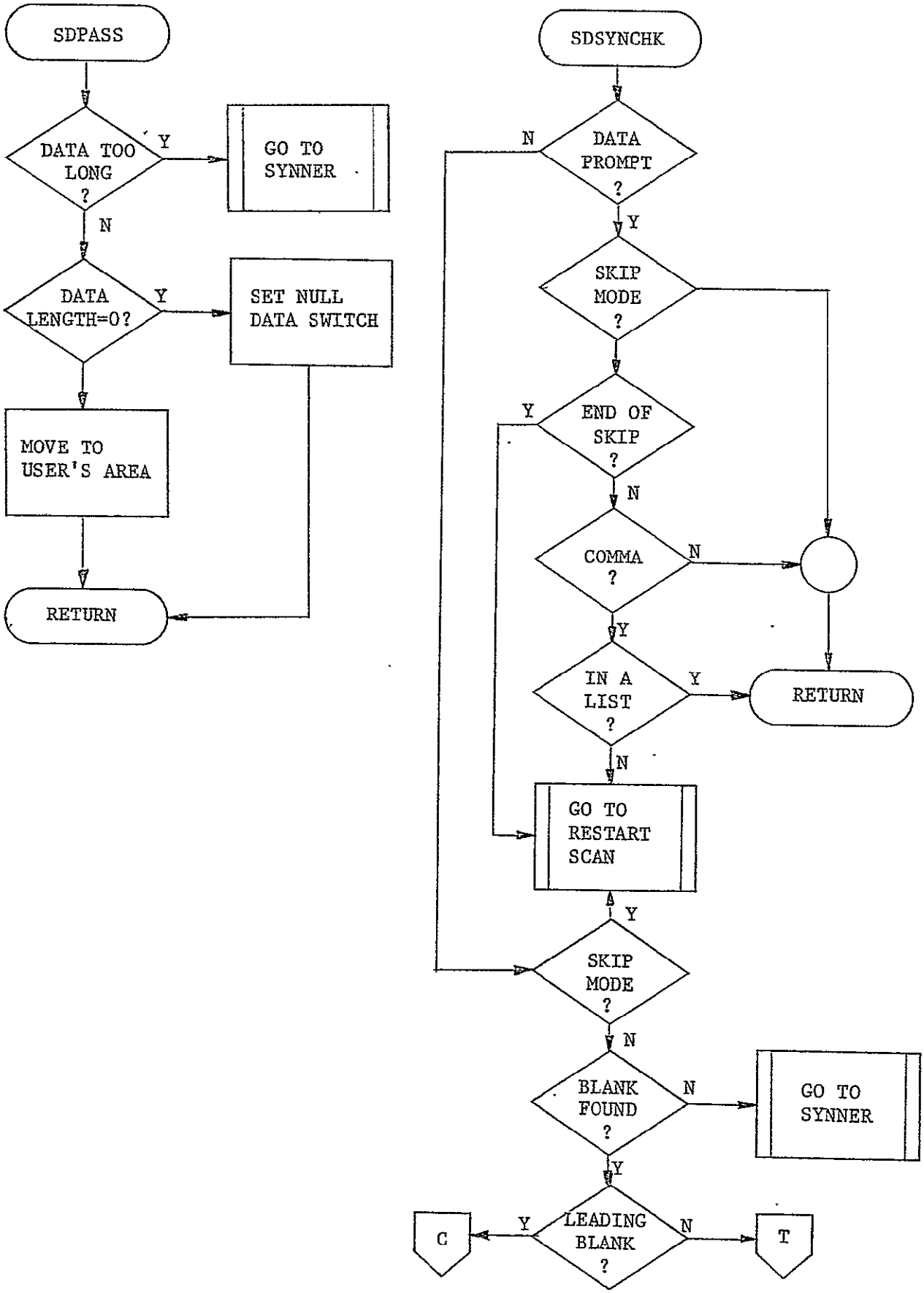


Figure 18. Subroutines SDPASS and SDSYNCHK

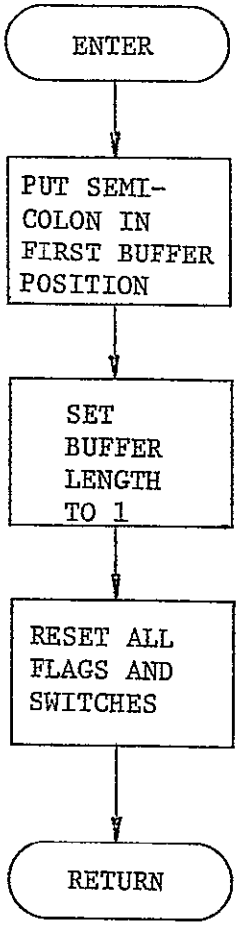


Figure 19. Subroutine RESETBUF

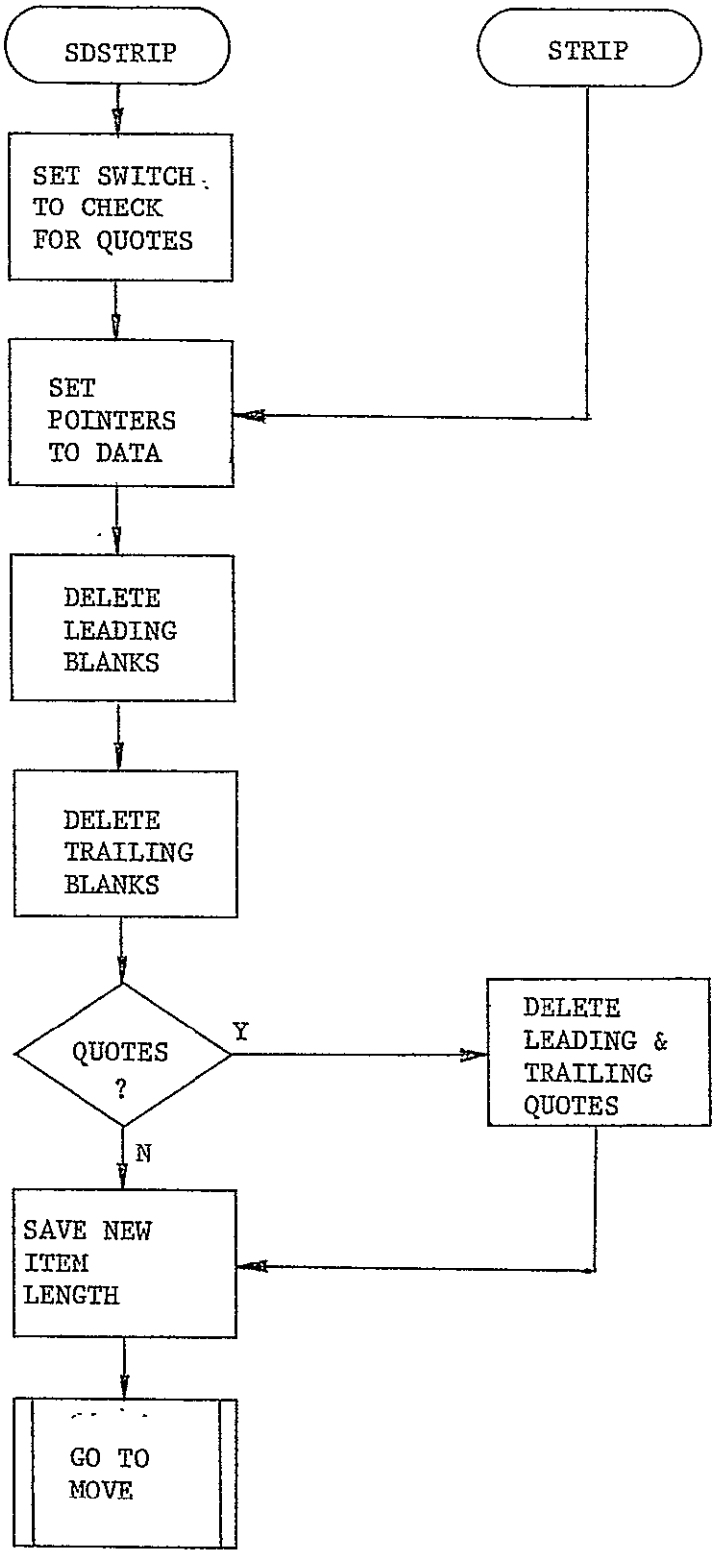


Figure 20. Subroutines SDSTRIP and STRIP

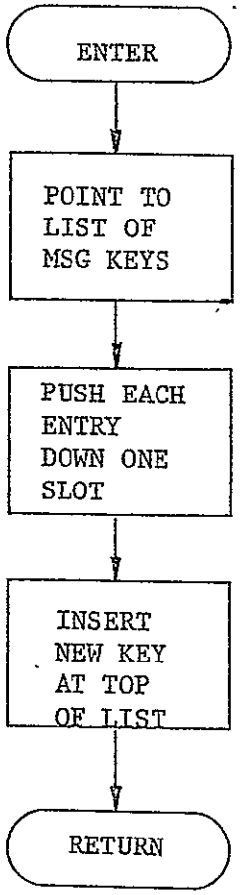


Figure 21. Subroutines PRKEYSAV

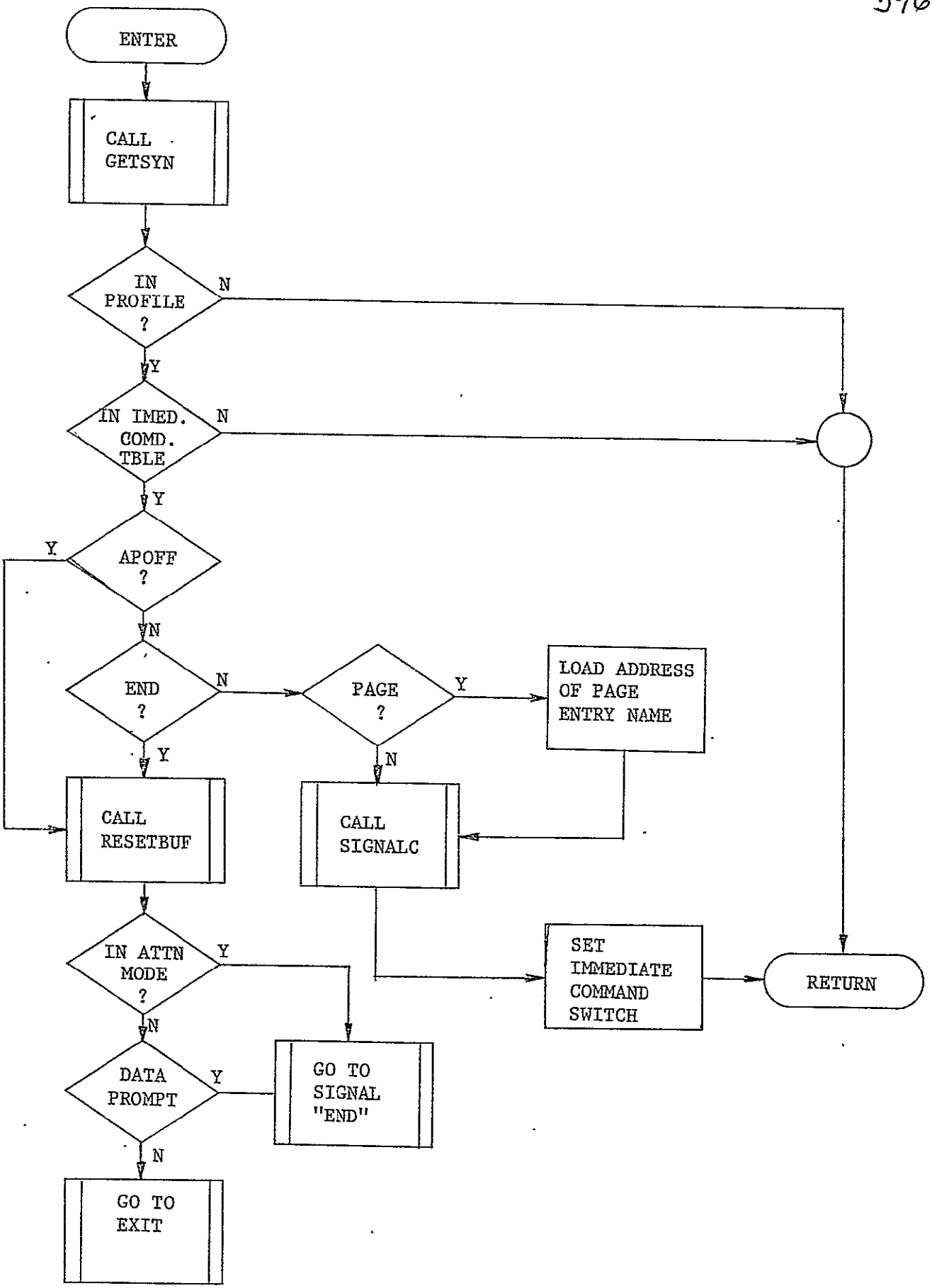


Figure 22. Subroutine IMCHECK

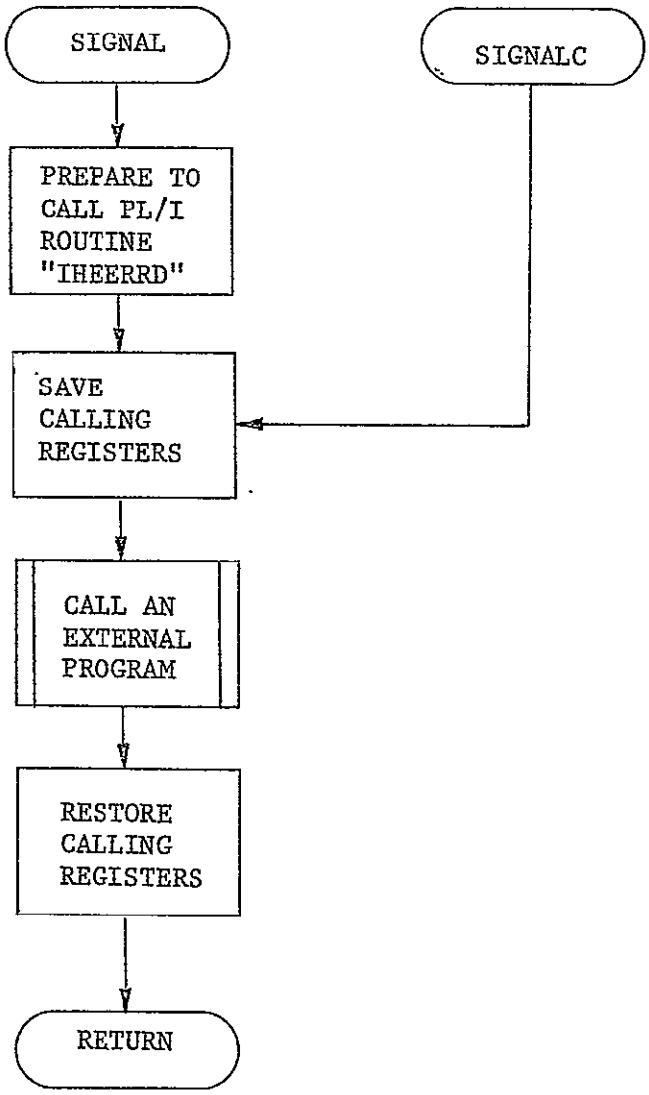


Figure 23. Subroutines SIGNAL and SIGNALC

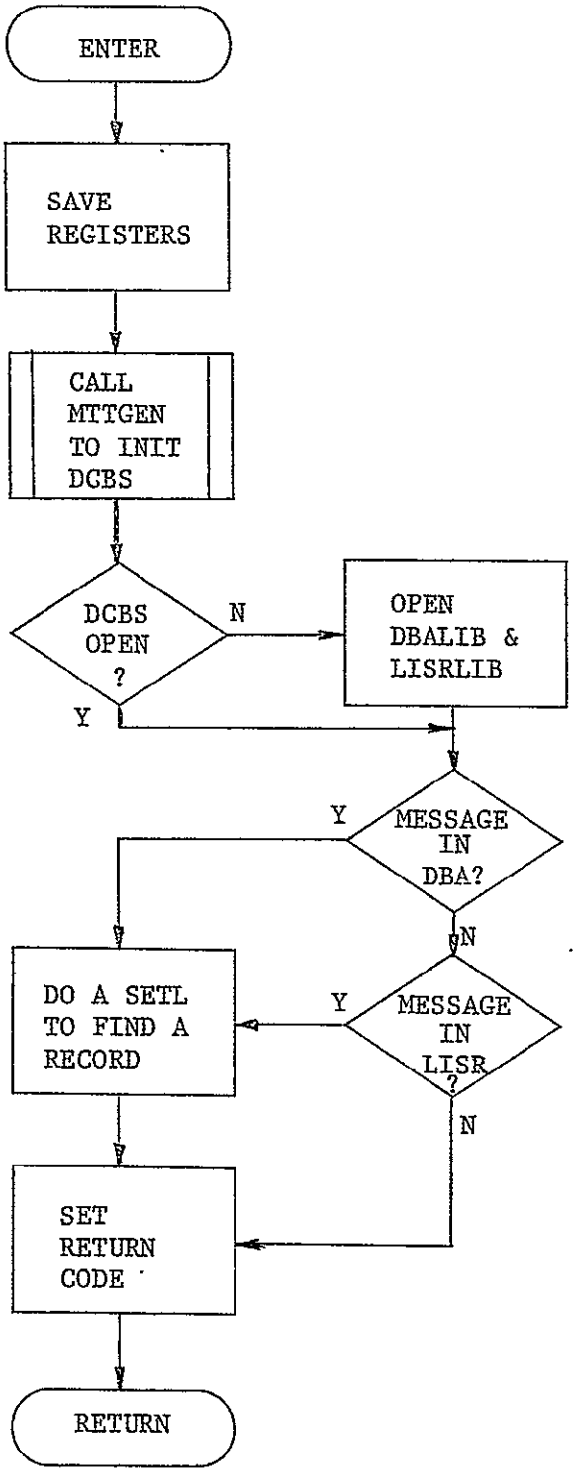


Figure 24. Subroutine SETLDBA

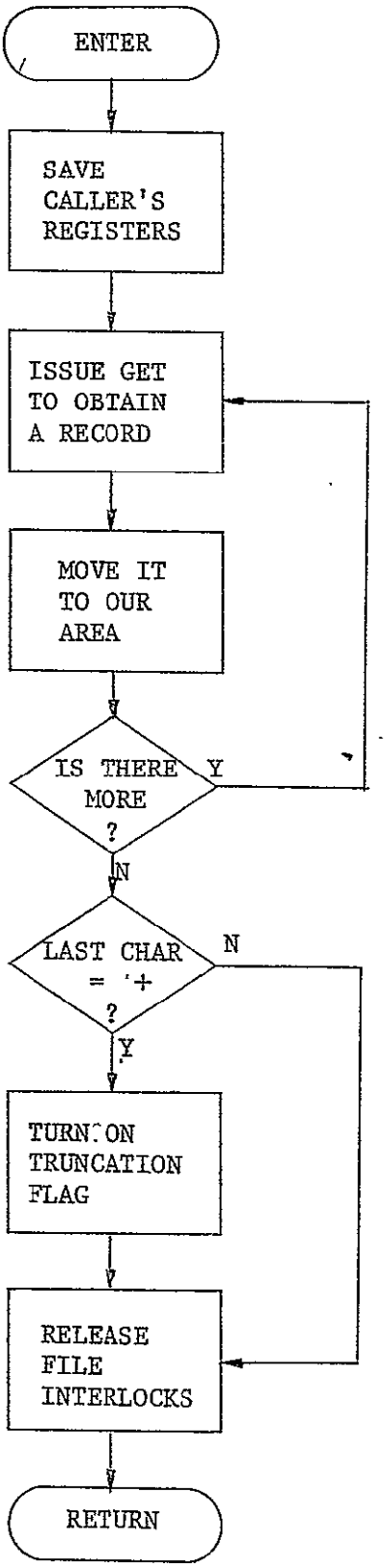


Figure 25. Subroutine GETMLF

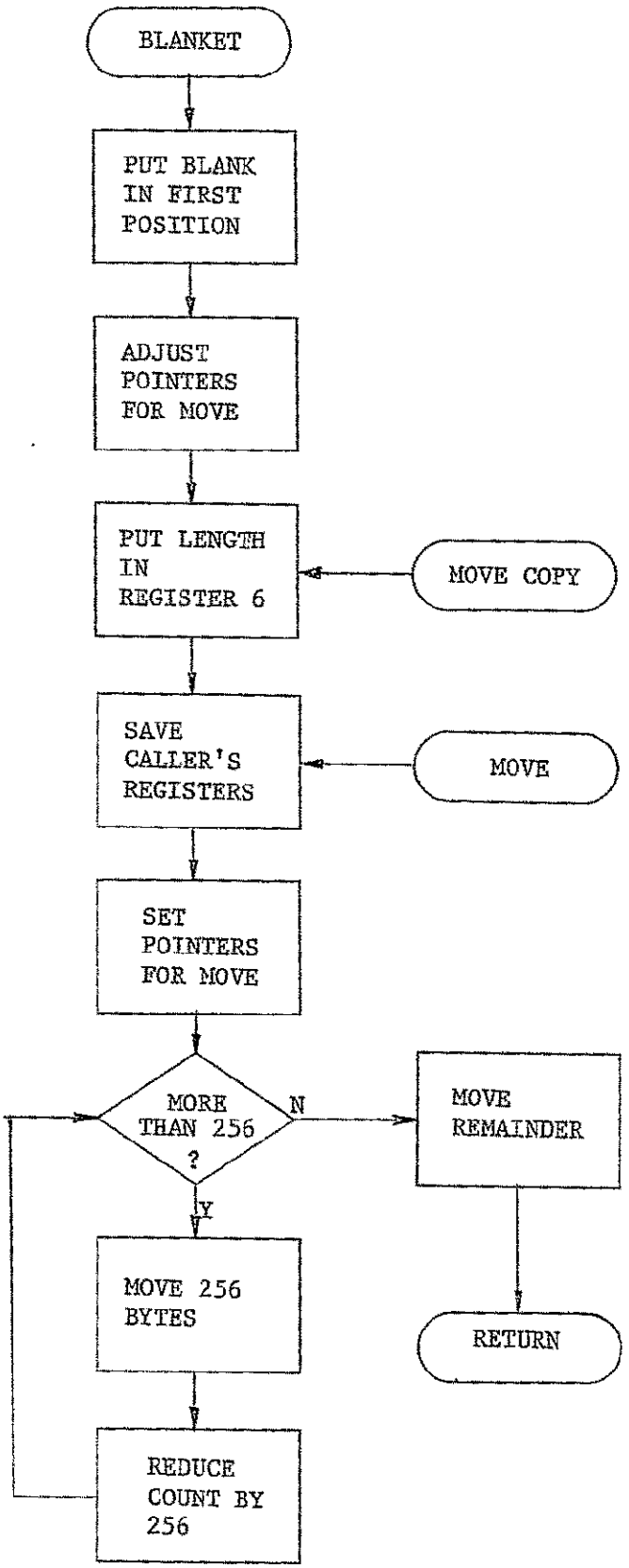


Figure 26. Subroutine MOVE

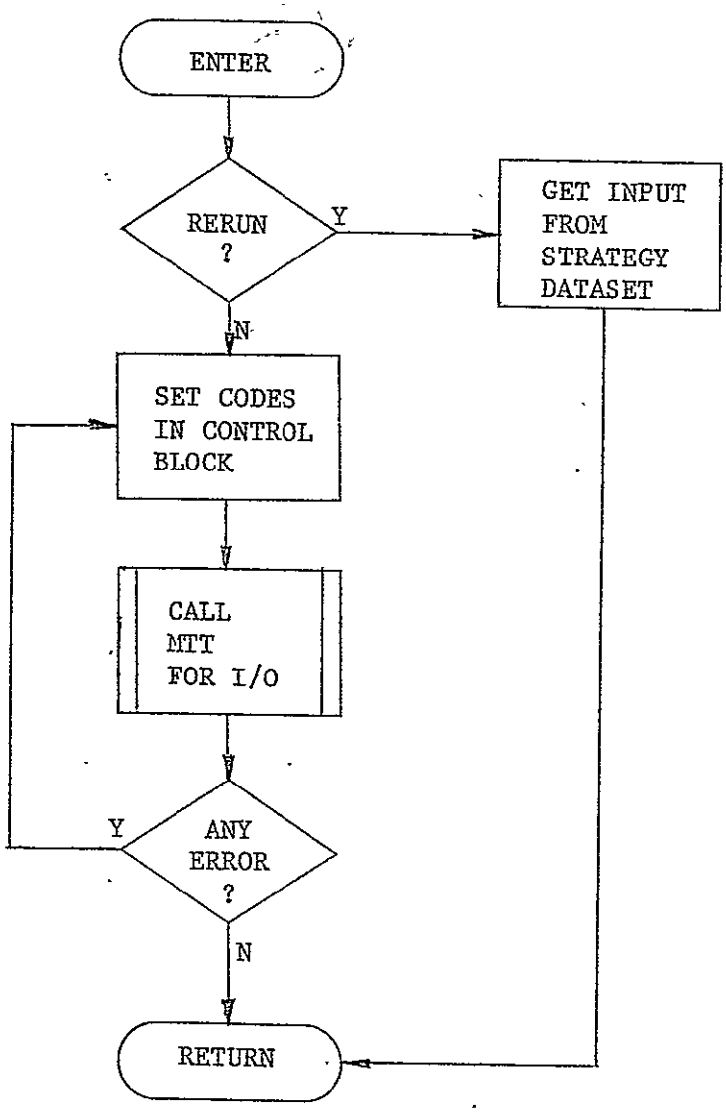


Figure 28. Subroutine PROMPT

TOPIC E.3 - PLI/ASSEMBLER LINKAGE MODULE

A. MODULE NAME

Program-ID - RDBPLINK
Module-ID - DBPLINK

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module completes the linkage between a PL/I program and an assembler subroutine. It does so in such a way that the assembler routine may in turn call a PL/I subroutine and yet maintain the continuity of control necessary for proper PL/I linkage and communication. Another aspect of the linkage method is that it not only makes the module reentrant, from an MTT standpoint, but also recursive.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

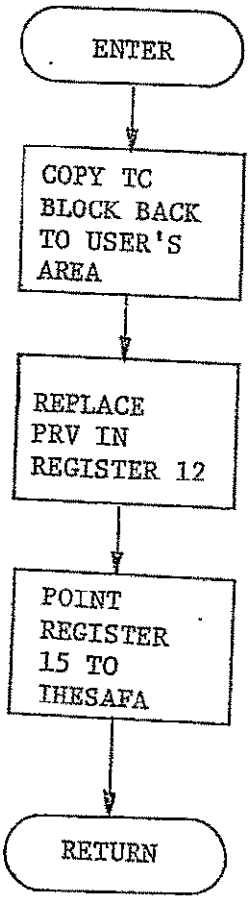


Figure 29. Subroutine EXIT

IV 82-29

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1

2. Narrative

Upon entry, the program initializes the variables it needs from the parameter list passed by the calling module. This data is used to obtain from PL/I library routine IHESADA a dynamic storage area (DSA) large enough to contain the register save area and a copy of the calling routine's psect.

Once this has been done, the program copies the calling programs PSECT to the DSA, chains the DSA into the pseudo register vector (PRV) and posts the DSA address in register 13. The program then initializes all of the base registers required.

Before exiting the program restores the remaining registers from the calling programs caller's savearea. It then chains the DSA into the savearea chain and returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 Assembler language.

2. Suggestions and Techniques

Extreme care must be taken to ensure the fact that

this program is completely reentrant and recursive. All operations should be performed in registers, or in the DSA obtained from PL/I.

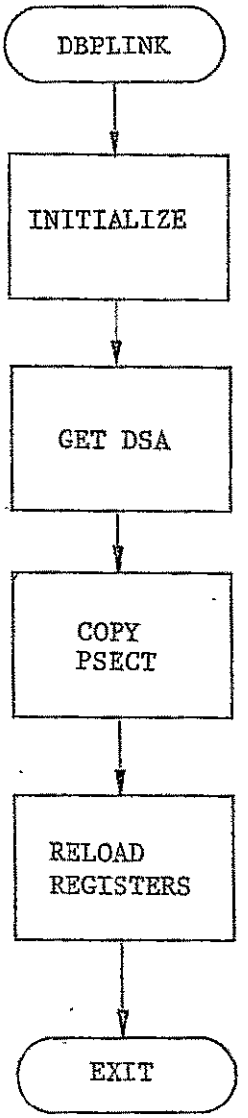


Figure 1. Top Level Flowchart - DBPLINK

TOPIC E.4 - ASYNCHRONOUS INTERRUPT PROCESSOR

A. MODULE NAME

Terminal Support - Attention Interface
Program-ID - RTSATTN
Module-ID - TSATTN
Entry Point - TSMATTN

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTIONS

1. Organization Chart

See Figure 1

2. Overview

RTSATTN is the interface between whatever monitor is running (NASIS or NASISX) and the terminal support supervisor RTSUPER. Its function is to link the monitor to the RTSUPER attention routine TSATIN. RTSATTN is only called after are asynchronous interrupt resulting from the user depressing the attention key at his terminal.

D. DATA REQUIREMENTS

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

On entry, RTSATTN performs TSS standard linkage except that the address it picks up as its PSECT register points to a table of r-cons which are (in order): TSATIN and MTTUTAB. TSATIN is the entry point to terminal Supports' attention processing routine MTTUTAB is a table which holds the user's pseudo-register vector (PRV).

After linking, RTSATTN checks the interrupted register 13 to determine if it points to a PL/I

Dynamic Storage Area (DBA). If not, no further attempt is made to process the attention. That is, RTSATTN returns to the monitor, effectively ignoring the interrupt.

When a valid DSA is found, the PRV is checked and if it is OK the DSA registers are saved in an area provided by the monitor. RTSATTN next calls TSATIN using the interrupted DSA as a savearea.

On return frcm TSATIN, the DSA regs are restored, the caller's registers are restored and control is returned to the monitor.

F. CODING SPECIFICATIONS

1. Source Language

TSS/360 Assembler Language.

2. Suggestions and Techniques

The NASIS assembler macro library RSOURCE must be used to reference the User Information Table (RTSUTAB). Also, entry linkage is standard TSS/360 while calling linkage is standard PL/I.

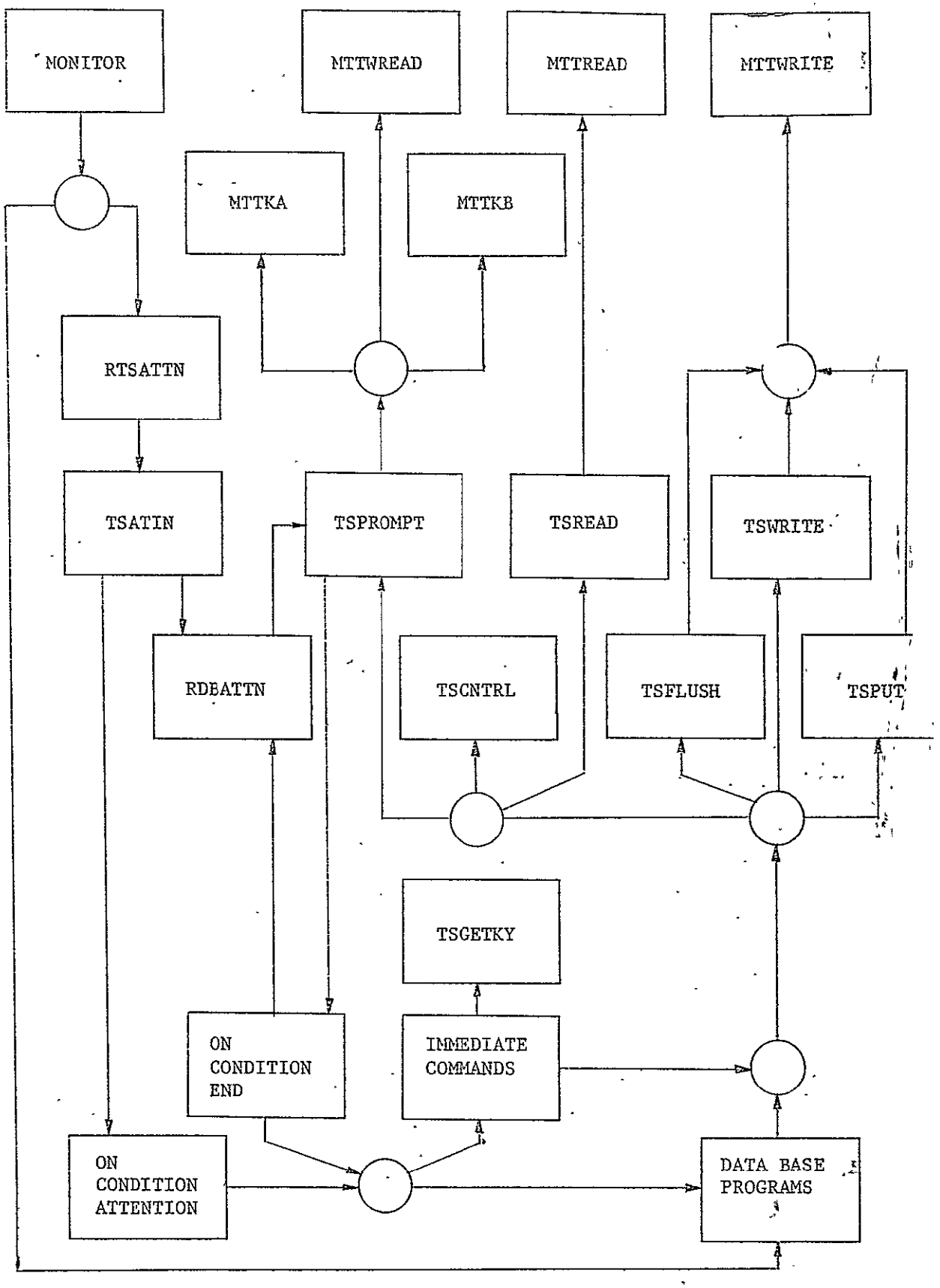


Figure 1. Terminal Support Organization Chart

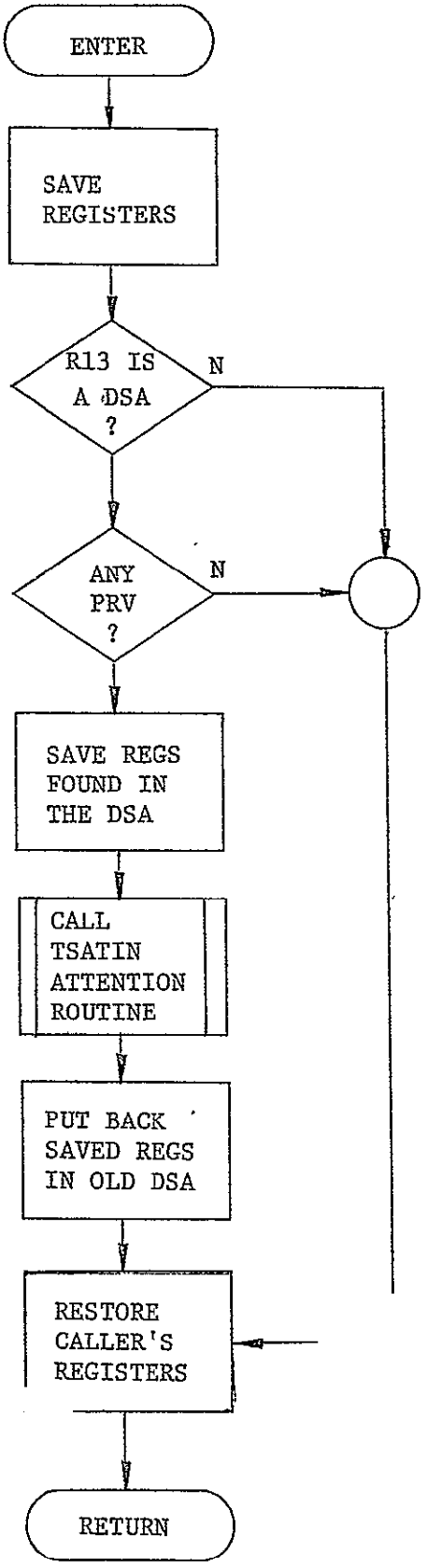


Figure 2. Top-Level Flowchart - RTSATTN

TOPIC E.5 - ATTENTION PROMPTING PROGRAM

A. MODULE NAME

Terminal Support-Attention Prompting Program
Program-ID - RDBATTN
Module-ID - DBATTN
Entry Point - DBATTN

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTIONS

1. Organization Chart

See Figure 1

2. Overview

DBATTN is called by RTSUPER to issue the command prompt ''-ATTN:'' and check the user's response thereto.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

Not Applicable

3. Output Data Sets

Not Applicable

4. Reference Tables

a. External tables

LISRMAC(USERTAB)

b. Internal Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   On entry, DBATTN checks the DISABLED switch in USERTAB. If attentions have been disabled, TSPRMTM is called to inform the user at the terminal and execution returns to the caller. If attentions are enabled, DBATTN sends a blank character out to insure that the carriage is in its home position, then issues a command prompt with the message ''-ATTN:'' to allow asynchronous commands to be entered by the user.

   RTSUPER intercepts all ''immediate'' commands except GO and calls the appropriate routine. If the user enters GO, null or any non-immediate command, DBATTN takes the following action:

   a. GO or null - returns control to the caller, thus signifying the end of the prompting sequence.

   b. Non-immediate command - ignores the user's response and reprompts as above.

   If the END condition is raised while executing this module, execution control is returned to the caller.

F. CODING SPECIFICATIONS

1. Source Language

   TSS/360 PL/I

2. Suggestions and Techniques

   Not Applicable

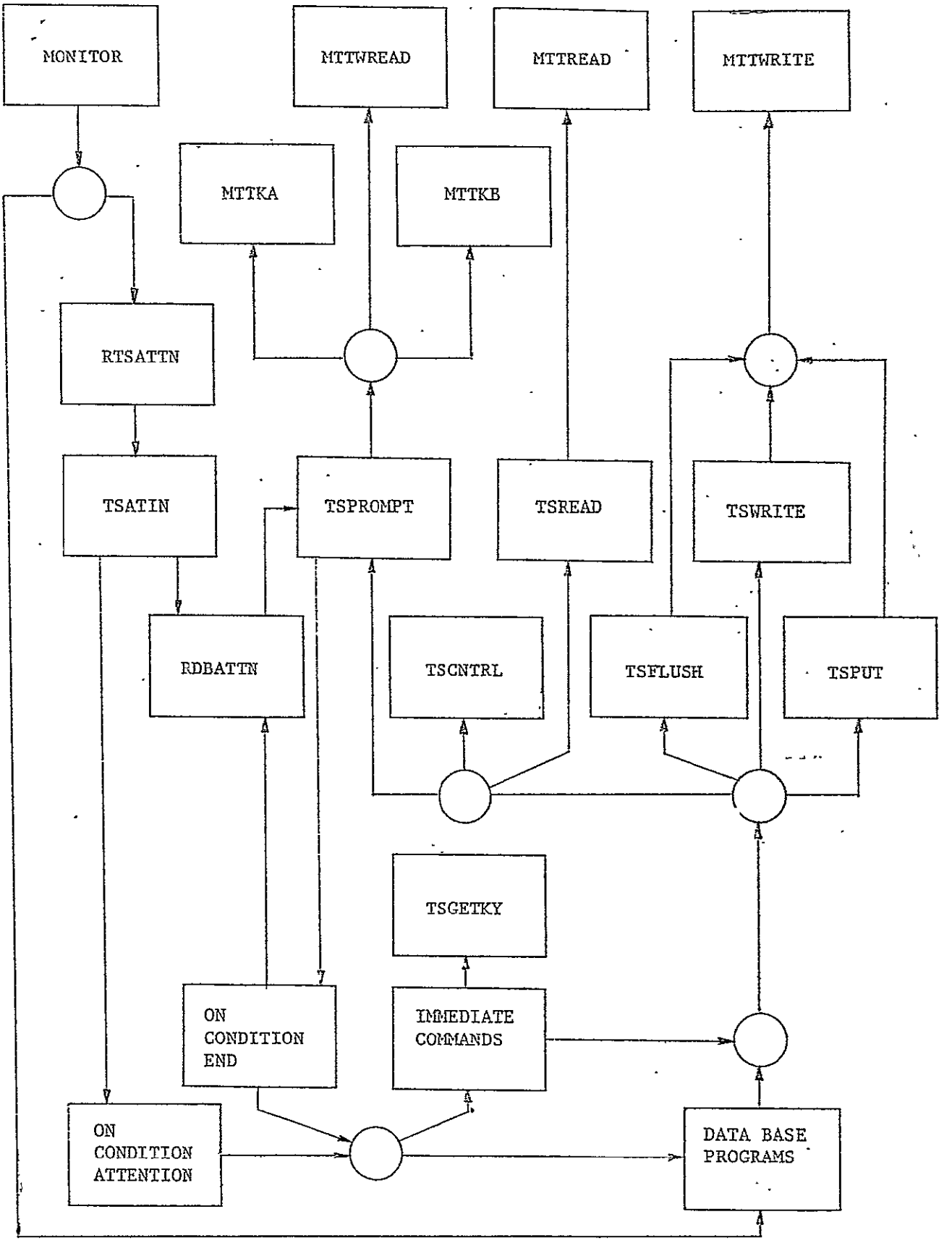


Figure 1. Terminal Support Organization Chart

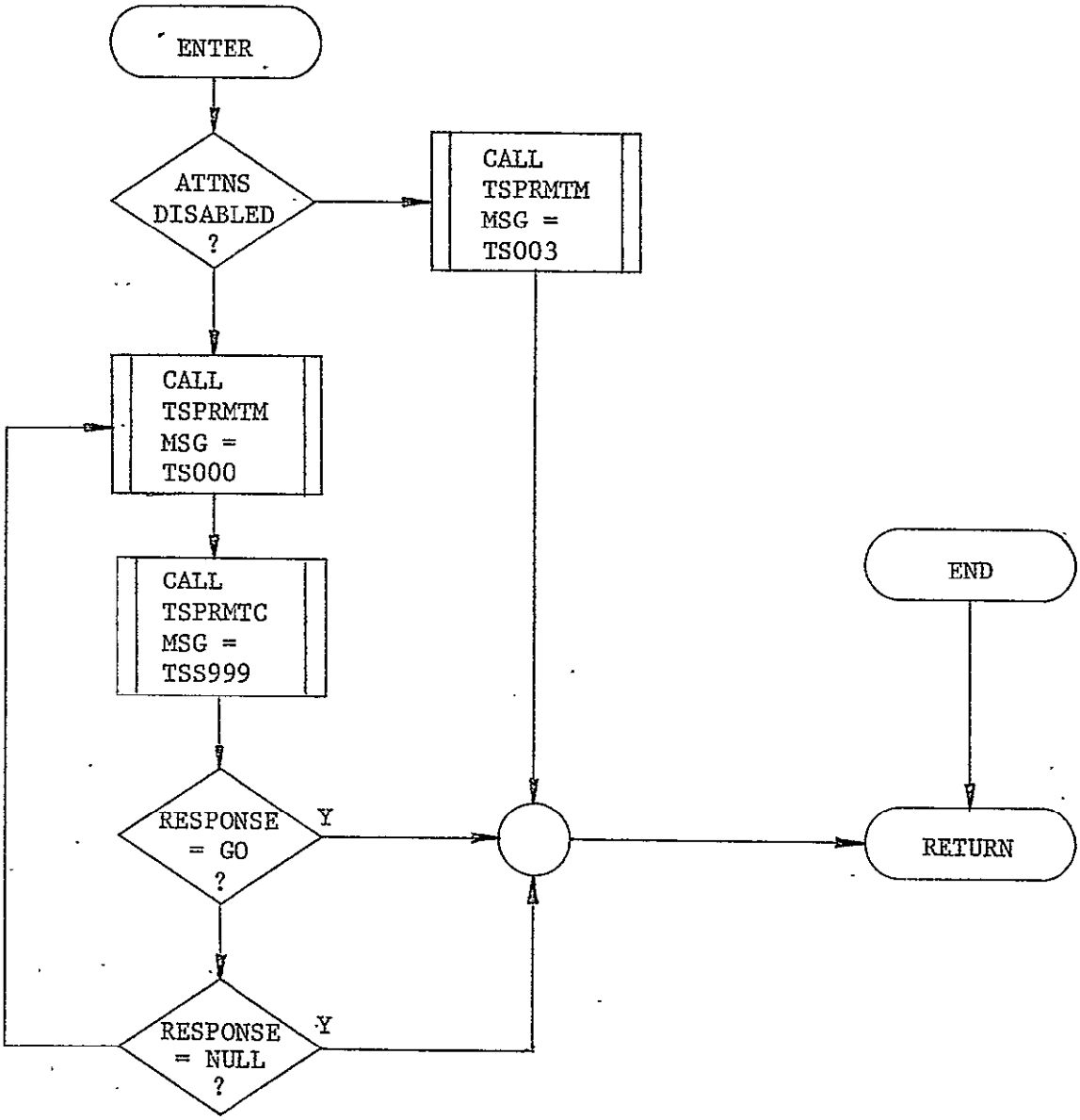


Figure 2. Top Level Flowchart

TOPIC F.1 - RETRIEVAL INITIALIZATION

A. MODULE NAME:

Program-ID - RDBINIT
Module-ID - DBINIT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module performs the initialization functions for the retrieval system and is the command director (prompting module) for retrieval.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

C. Input Files

Not Applicable

d. On-Line Terminal Entries

The program initially prompts for the FILE, NAME and ADDRESS parameters, and later, prompts for the retrieval commands.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program issues various diagnostic messages, where appropriate.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program references and optionally initializes the following tables,

USERTAB
FLDTAB
COLFORM
SEQFORM
SRCHTAB
VERETAB
RETDATA
SETAB

E. PROCESSING REQUIREMENTS

1. TOP LEVEL FLOWCHART

See Figure 2

2. Narrative

It calls DBJOIND to process the file parameter and prompts for the NAME and ADDRESS parameters. The parameters are all verified and saved for later reference.

The program then defines the print data set and the save data set and initializes the retrieval data table, RETDATA. The set table, SETAB is then initialized, the data base is opened for input and the field table, FLDTAB, is initialized.

The program then initializes its verb table, including the addition of any user defined commands. Now the program prompts the user for a retrieval command.

If the command entered was not END or BEGIN, the program calls the entry point specified for that command and than branches back to prompt the user for his next command. If the user entered END or

BEGIN, the retrieval session is terminated by closing the data base, erasing the sets, the formats and the save data set. The print data set is printed. All searches are cancelled. If the user entered BEGIN, the program branches back to initialize itself for a new retrieval session. Otherwise, the program is terminated.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and Techniques

Not Applicable

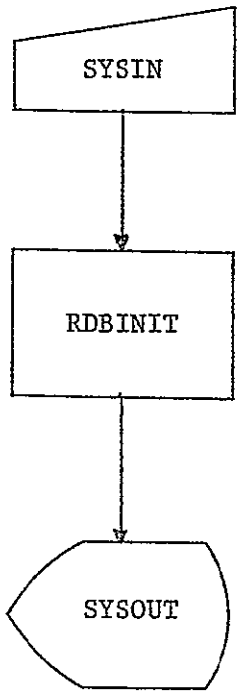


Figure 1. I/O Block diagram

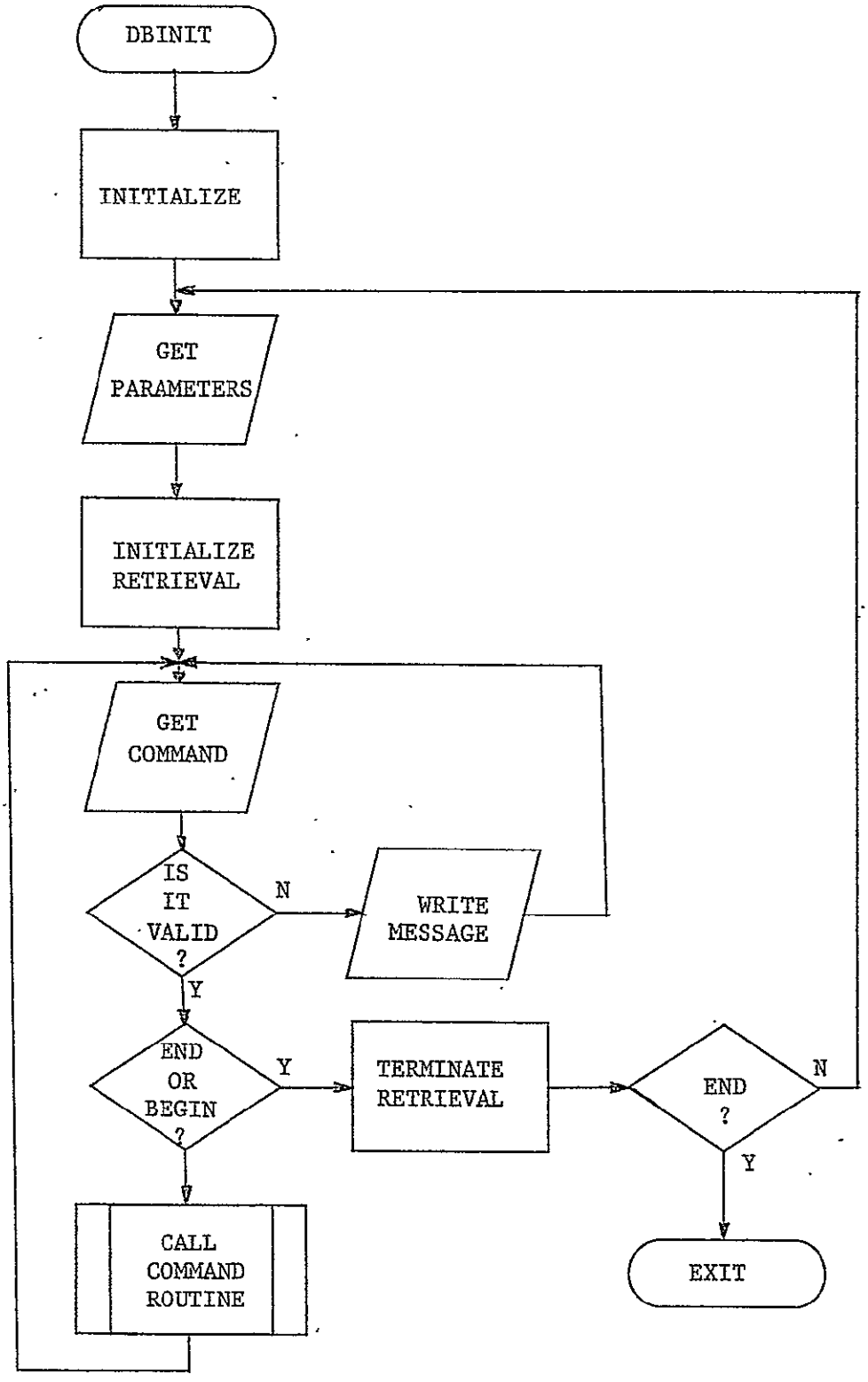


Figure 2. Top level flowchart

TOPIC F.2 - RETRIEVAL FIELDS COMMAND

A. MODULE NAME

Program-ID - RDBFLDS
Module-ID - DBFLDS

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module displays a formatted listing of the field names of the file currently being accessed by the user.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

The routine prompts for the parameter associated with a PAGE command.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program produces a formatted list of

field name.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

FLDTAB-The program extracts all of its information from FLDTAB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBFLDS

At this entry point the program initializes the screen and paging status data. It extracts the data base name from FLDTAB. The program then, repetitively, extracts the field names from FLDTAB. It flags each field that has an inverted index. It posts the field names to the screen. When the list of names has been exhausted, or the screen has been filled, the screen is displayed to the user, the paging status data is posted and the program is terminated.

b. DBFLDSP

At this entry point the program is re-initialized using the paging status data. If more data remains, the program branches to the proper routine to build the next screen image. Otherwise, a diagnostic message is written to the user and the program is terminated.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I Language.

2. Suggestions and Techniques

Not Applicable

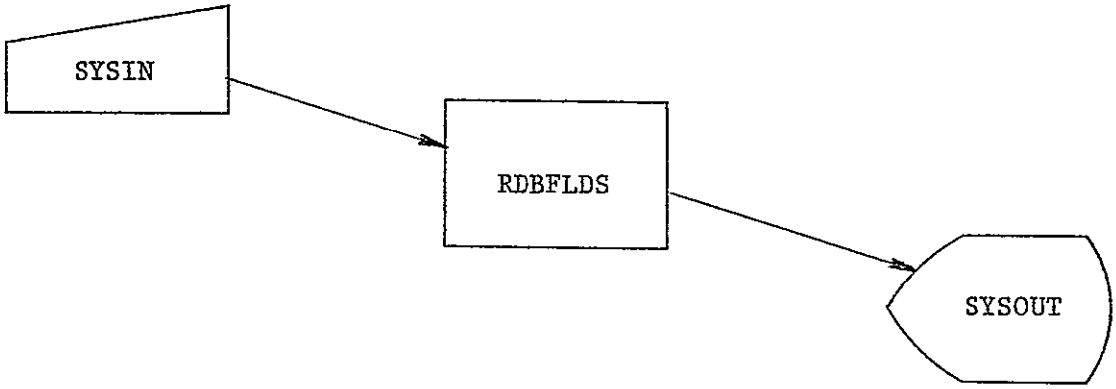


Figure 1. I/O Block Diagram

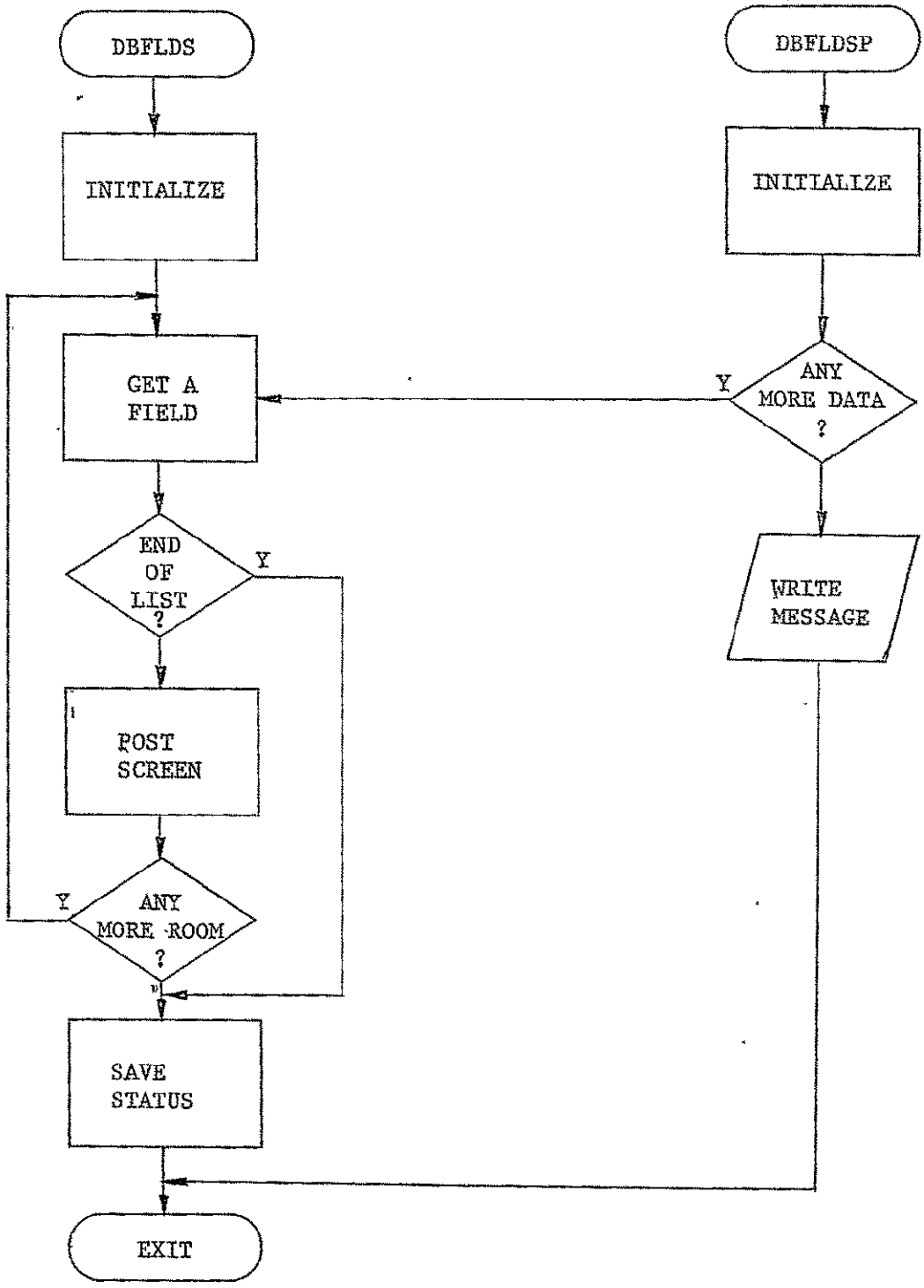


Figure 2. Top Level Flowchart - DBFLDS

IV 5272

TOPIC F.3 - RETRIEVAL EXPAND COMMAND

A. MODULE NAME

Program-ID - RDBXPND
Module-ID - DBXPND

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module displays to the retrieval user, a formatted listing of a cross section of an inverted index surrounding a specified term.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The inverted index files of a dataplex are used as a source of data by the program.

d. On-Line Terminal Entries

The program prompts for the TERM and INDEX parameters.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program produces a formatted listing of the index records read.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program uses the following tables as a source of data and as a means of data control,

USERTAB
FLDTAB
EXPTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBSPND

At this entry point the program initializes itself to perform a new expansion of an index file. The program initializes the screen and the data storage table EXPTAB.

The program then prompts the user for the TERM and INDEX parameters. The parameters are validated and the program gets ready to read the index (or anchor) file specified. The first read of the file is for positioning, based upon the term entered by the user. The program then attempts to read the previous three records in the file. As each record is read the term is posted into EXPTAB along with the number of cross references. The relative E-number is computed and is also posted.

If more space remains on the screen and more data remains on the file, the program positions itself and begins reading records, saving the data in EXPTAB and posting them on the screen. If an end-of-file is

encountered, an indication is posted on the screen. At this point, or when the screen is filled, it is displayed to the user, the paging status data is posted and the program terminates.

If any errors are encountered, a diagnostic message is written to the user and the program is terminated.

b. DBXPNDP

At this entry point the program re-initializes itself using the paging status data. If more data remains to be displayed the program branches to the appropriate point to begin reading the index and building the new screen image. If no more data remains, a diagnostic message is written to the user and the program is terminated.

c. DBXPNDE

At this entry point the program initializes itself to decode an E-number reference. If the E-number parameter is valid, the data associated with it is passed back to the caller. Otherwise, an error indicator is passed back to the caller and the program is terminated.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and Techniques

The AREA facilities of PL/I should be used to organize the term data stored in EXPTAB to optimize file access and data storage.

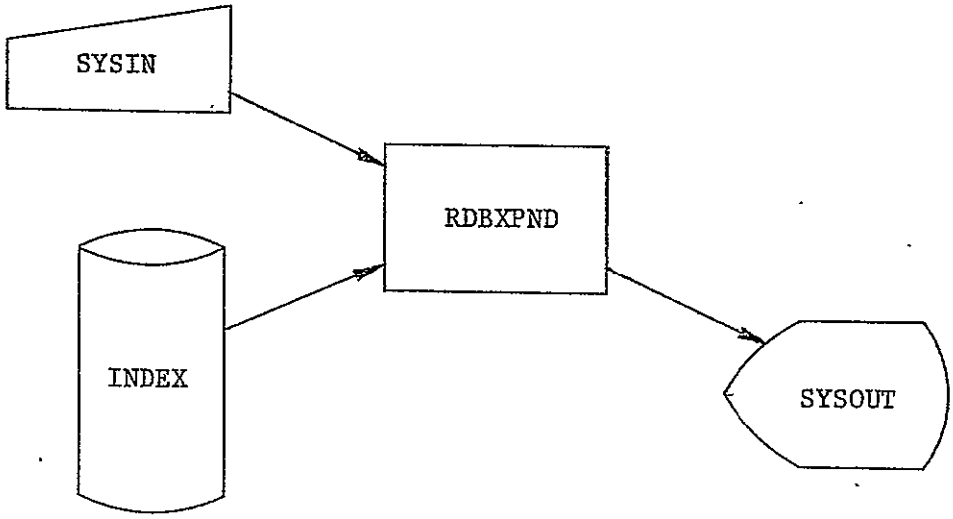


Figure 1. I/O Block Diagram

(1) F3.1

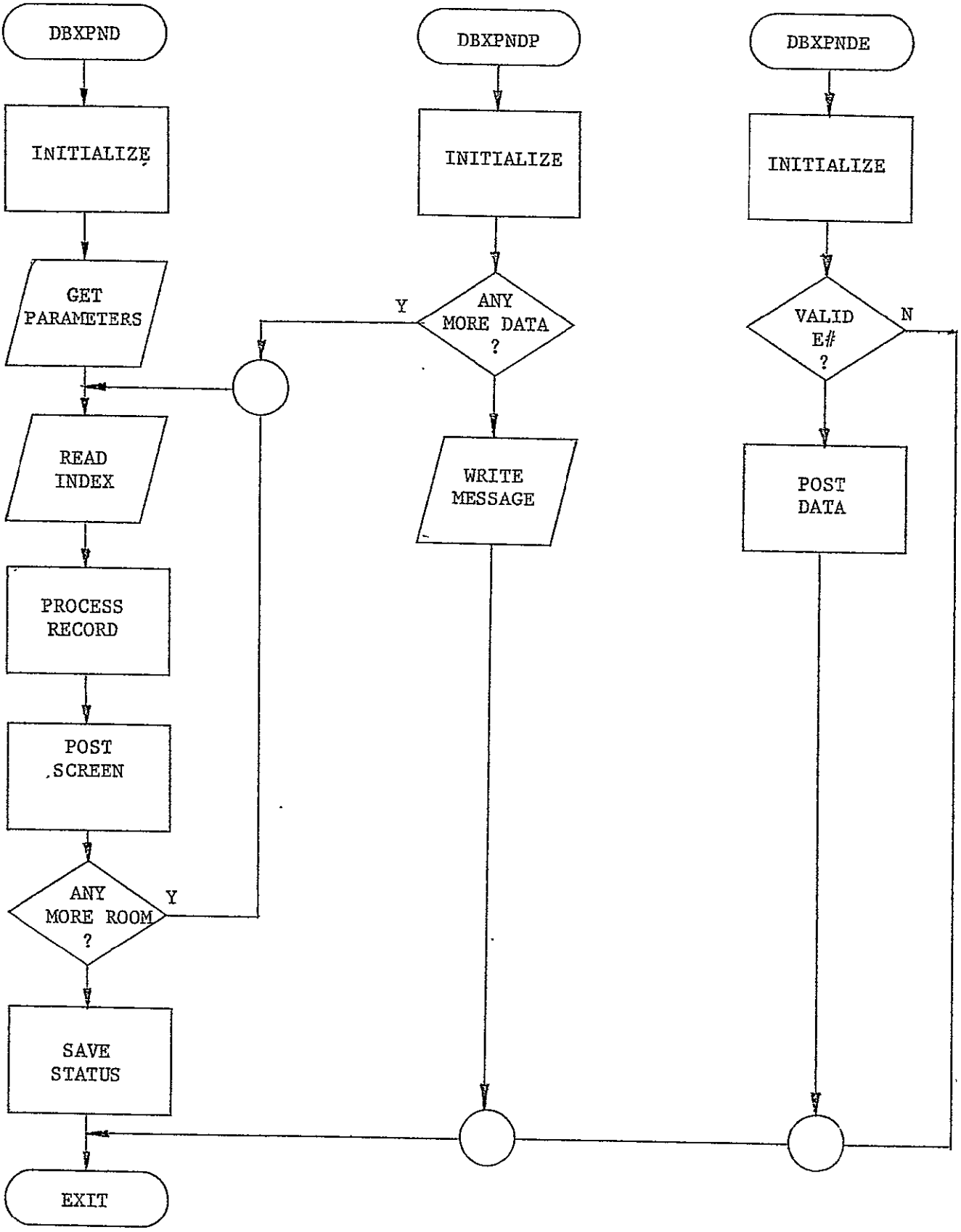


Figure 2. Top Level Flowchart

TOPIC F.4 - RETRIEVAL, Select Command

I. SELECT

A. MODULE NAME

Retrieval, SELECT Command
Program - ID - RDBSLCT
Module - ID - DBSLCT
Entry Points (DBSICT0,DBSLCT1,DBSLCT2)

B. ANALYST

O. Kirt Hearne
Neotericsf Inc.

C. MODULE FUNCTION

The SELECT command format is:

SELECT expression,field,replace,method

The SELECT command outputs the expression and the number of citations (record keys) for which the expression applies. A set number or S-number is assigned to the expression, and the command string is entered into the next available line in the current search strategy.

The expression parameter (keyword=EXPR) is a boolean combination of terms which define a set. If all fields referenced are indexed, the expression is evaluated immediately and a set-number assigned. If a field in the expression is not indexed or a previous S-number is referenced, a search entry is constructed and saved, and an S-number assigned.

Only a single non-indexed field is allowed in a single SELECT expression.

The field parameter (keyword=FIELD) is used by SELECT to resolve any values in the expression which are not directly related to a fieldname within the expression.

The replace parameter (keyword=REPLACE) is a previously defined S-number which is to have its expression replaced by the current expression.

The method parameter (keyword=METHOD) is used to force a search on indexed fields. To do this, "SEARCH" must be entered as the method parameter.

Note that only a single field may be referenced in this case.

SELECT will prompt the user if the expression is missing, or the field parameter is missing and found to be needed.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The descriptor files and the index files may be referenced by the SELECT command. The descriptor file is used to obtain the data set name of the subject term index file. The index files are used to obtain a list of accession numbers associated with a particular subject term.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The command string, as it is entered, is saved in the region containing the current strategy of the VISAM member DBSTRAT of the VPAM data set USERLIB. This action is accomplished using the routine PSTRAT.

b. On-line Terminal Displays.

The following is displayed if a set is

successfully produced from the expression:

(1). A unique set number or S-number.

(2.) The number of citations (or keys) in the set

(3.) The expression, with:

(a.) E-numbers replaced with the corresponding "fieldname=value".

(b.) Values which return a null are notated with special symbols, as: AGE =>>'9999'<<.

(c.) If the resultant set consists of subfile keys, the expression will be displayed with the subfile name, as:

(FROM:subfilename) expression

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. EXPTAB

b. FLDTAB

c. MFCB

d. PARSED

e. SETAB

f. SRCHTAB

g. TC

h. USERTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. NARRATIVE

   The SELECT command outputs the expression and the number of citations (record keys) associated with that expression. A unique set-number or S-number is assigned.

   The input expression is a boolean expression made up of set-numbers, S-numbers, values, E-numbers or range forms of these terms.

   The SELECT command is processed in three phases:

   1. Parsing
   2. Expression analysis
   3. Execution of SELECT "instructions"

   SELECT parses the expression in three passes. The first pass recognizes and marks as such, letter strings, digit strings, operators, special characters and delimiters. Quoted strings are recopied to remove any double quotes.

   The second pass recognises primary elements such as S-numbers, E-numbers, set-numbers, values, and field names. Field names are marked as indexed or non-indexed.

   The third pass recognises groupings of elements such as range forms and associates each value in the expression with the proper field name. If necessary a prompt with the keyword "FIELD" is done to obtain the field name. This pass also sets up SELECT execute phase instructions for the creation of sets from basic terms such as a set-number.

   Also, during the third pass, a non-indexed field name appears in the expression, the proper entries are made in SRCHTAB to provide for the search to be executed later.

   All information found during the first three passes is entered into PARS_TAB and PTAB_INFO. The original expression, recopied quoted strings, and other necessary character strings are all contained in WAS. Each

element in PARS_TAB contains an index (IDX) into was to note the position of the item described.

The next phase of SELECT analyses the expression algebraically and builds execute phase instructions to perform the proper operations. If a search is required instructions are built to post final entries in SRCHTAB, before the search, and to retrieve information from SRCHTAB, after the search, for final evaluation of the expression.

During expression analysis, the ANDing of a search term with another set is noted, and instructions are created to cause the search to occur only within the set ANDed with the search term.

After the second phase all is ready for final evaluation of the expression by execution of the previously created instructions. At this time the input command, with parameters, is reconstructed and posted in the CURRENT_STRATEGY data set.

If a search is required, all SELECT tables and instructions are stored for use at the time of search execution. An S-number is assigned and this number, with the expression, is output to the terminal.

If no search is required, the execution phases of SELECT is invoked. The instructions built earlier are now executed. Sets are created, combined, and a altered as the expression dictated, until the final resultant set is obtained. This set is assigned a unigue number and posted into SETAB through the use of the DBPSET routine which also sends a line describing the set (set- number, size, expression) to the terminal.

When the user enters the EXECUTE command to invoke the search, the DBEXSR program is given control. This routine contains all of the actual search logic, however repetitive calls to SELECT (DBSLCT2 entry point) are made. The execute phase instructions are used by SELECT to control the search.

During a search each previously defined S-number has associated with it an instruction list. The first instruction in the list for each S-number is a "branch" initialized to point to the second instruction in the list. When SELECT is first given control, each instruction list is executed until an S-number or a search term instruction is encountered. The search instruction posts proper final information to SRCHTAB and in both cases execution of the instruction list is suspended. A new branch point indicating where to resume execution is stored in the "branch" instruction at the top of the list.

When all instruction lists have been executed as far as possible, control is returned to DBEXSR for the actual search to take place. After this SELECT is called again and instruction execution is restarted. Some S-numbers and searches may now be evaluated. Again each instruction list is executed until an undefined S-number or search term is encountered or an actual set is created and posted. Again control returns to DBEXSR. This process continues until all instruction lists terminate by posting a set.

The SEARCH is implemented simply as an additional entry (DBSLCT1) into SELECT. The command format is the same as that for the SELECT command, thus a valid SELECT expression may be used.

DBSLCT1 is the entry point for the SEARCH. This command first gets and verifies the set number or S-number on which a linear search is to be performed. SEARCH then prompts the user for the rest of the search expression to be performed to the specified set. Once the search expression is entered, then SEARCH passes this information to the search option part of the SELECT command. When control is returned to SEARCH, it then prompts the user for another search to be performed on the same set as before. This loop continues until the user enters a null response to the search expression prompt, at which time control is passed to the calling routine.

F. CODING SPECIFICATIONS

1. Source Language

   The SELECT command module is written in the IBM/360 TSS PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the data base and the TSPL/I language extension is used to handle all communication with the terminal.

2. Suggestions and Techniques

   Not Applicable

II. SELECT, THE SEARCH OPTION

A. MODULE NAME

Retrieval, SELECT Search Option
Program - ID - RDBSLCT
Module - ID - DBSLCT

B. ANALYST

O. Kirt Hearne
Neoterics, Inc.

C. Module Function

The SELECT search option is a feature of the SELECT command which guides the user through a search strategy. The SEARCH command is used to define a set or pseudo-set to be used as the search universe.

The user is then prompted for linear search expressions with the phrase:

SELECT (Set-number S-number) IF:

The reply is of the same format as the SELECT command itself:

expression,field,replace,method

where the parameters have the same meaning as with the SELECT Ccmmand.

The set-number or S-number defined by the SEARCH command is added along with an AND boolean operator to the left end of the expression entered in response to the SELECT IF prompt. The resultant expression is then sent directly to the SELECT command processor.

1. Reference Tables

   a. EXPTAB

   b. FLDTAB

   c. MFCB

   d. PARSED

e. SETAB

f. SRCHTAB

g. TC

h. USERTAB

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

If a terminal is the source of search parameters as previously defined, the TSS parameters as previously defined, the TSS system will apply default values, if available, to the parameters when no values are entered.

3. Output Data Sets

a. Output Files

Using the PSTRAT routine, the command string, as it is entered and validated, will be saved in the region CURRENT_STRATEGY.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The SELECT Search command format is:

```
SEARCH expression,field,replace,method
```

which results in a set-number or S-number. The user is then prompted for a linear search:

```
SELECT (Set-number S-Number) IF:
     expression
     field,replace,method
```

The set-number or S-number is added, along with an AND operator to the expression and the result is sent to the SELECT command processor. Thereafter all processing is the same as for any SELECT expression.

After the expression is processed, the user is again prompted with the SELECT IF prompt. This continues until a null is entered.

F. CODING SPECIFICATIONS

1. Source Language

The SELECT Search command is written in the IBM/360 TSS PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the data base, and the TSPL/I language extension is used to handle all communications with the terminal.

2. Suggestions and Techniques

Not Applicable

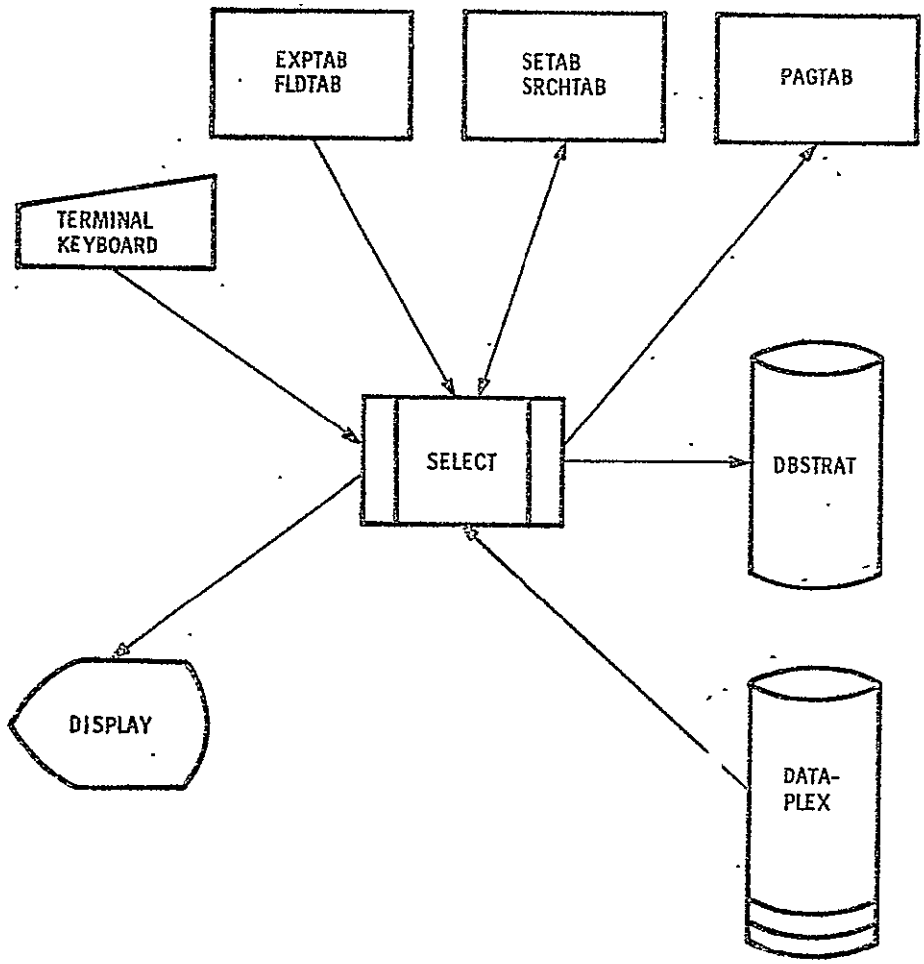


Figure 1. Block diagram.

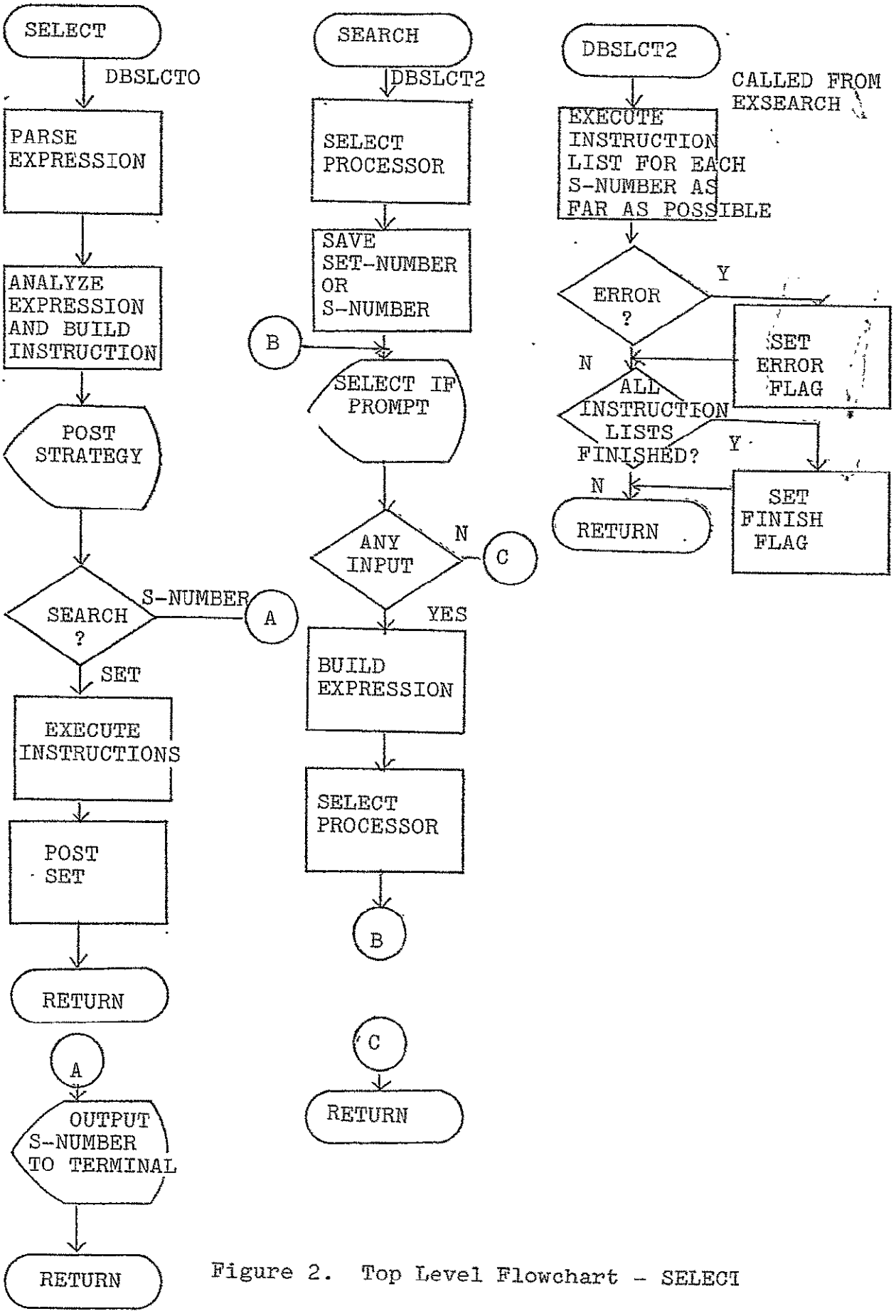


Figure 2. Top Level Flowchart - SELECT

TOPIC F.5 RETRIEVAL DISPLAY COMMAND

A. MODULE NAME

Retrieval, DISPLAY Command
Program-ID - RDBDSPL
Module-ID - DBDSPL
Primary Entry Point (DBDSPL)
Secondary Entry Point (DBDSPLP)

B. ANALYSTS

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The DISPLAY command is a routine, called by the TSS system, whose purpose is to allow the retrieval system user to have designated data for a given set to be displayed on a terminal. Like the PRINT command, the user may specify the format of the output as the citation number, the citation, the abstract, or the full text for any item contained in a set which has been previously selected. Optionally, the user may prespecify a format of his own, using the FORMAT command, to govern the DISPLAY command. One set-number is reserved for special purposes in the system. Set-number 0 is a logical reference to the entire anchor file. The PAGE command also calls the DISPLAY command in order to create additional displays, logically, before and beyond the current one. The calling sequence is: DISPLAY set-number, format, item, type or, alternately, DISPLAY citation#, format.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The anchor and associated files of a data base will be input to the DISPLAY command.

d. On-line Terminal Entries

The parameters available to the DISPLAY command are set or citation number, format, items, and type. The NASIS system will apply default values to the parameters, if they are available, when no original values are entered.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

The DISPLAY command will output a partially-formatted display of the items in a set or for a specific citation number. The content of the display depends upon the format code entered as the second parameter.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COLFORM

The DISPLAY command refers to a COLFORM table when a columnar format is referenced.

b. USERTAB

This table contains user-oriented and status information.

c. FLDTAB

The DISPLAY command refers to FLDTAB to locate the appropiate sequential (SEQFORM) of columnar (COLFORM) format table and for a table of data base field names ordered

according to format numbers 1-4.

d. RETDATA

This table contains data fields unique to the retrieval sub-system.

e. PLEX

The DISPLAY command uses a DBPL/I file called PLEX for all of its retrievals from the data base.

f. SEQFORM

The DISPLAY command refers to a SEQFORM table when a sequential format is referenced.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Display

The DISPLAY command is called by the NASIS system by the director.

b. Accept Parameters

Since the parameters are not passed to the DISPLAY command, by the director, they are retrieved via Terminal Support (TS). The first parameter is either a "set-number", or a "citation #". The second parameter is "format" code, the third is an "item" number and the fourth is the "type" code. The last three parameters are optional. The "set-number" is a one or two digit number and is not likely a default value since it will change for every command. The "citation #" is a character string, which is not likely to have a default. If no entry is made and no default exists, then an error is reported and control passed back to the calling routine. The "format" code is a value of 1 to 25 designating a sequential format or F1 to F25 designating a columnar format, or a format name representing one of the above format values or a fieldname. If no entry or

default is present, the value "2" is provided for anchor key sets or "5" for subfile sets. The "item" parameter designates the member of the specified set. The entry is a character string having a numeric value. If no entry or default is given for this parameter, the first item in the set is displayed. The "type" code indicates whether the user wants subfile information to be displayed continually following the anchor data, and if so, whether the data fields of each subfile record are to be exhausted sequentially or the data field values are to be exhausted across subfiles before proceeding to the next field. An invalid entry is reported before returning control to the calling routine. If all parameters have valid values, then execution continues with the next section.

The DISPLAY command is placed as the next record in the strategy data set by a call to the save strategy routine. The parameters to this subroutine are the word DISPLAY and its parameters in their normal order.

c. First Page Initialization

Depending on the "class" of the first parameter, certain specific initialization is necessary. If the parameter is a data base key (class 1), e.g., a citation number, then the anchor record is read and a heading prepared. If the parameter is a set number (class 2), the relative key is taken from the set and used to read the anchor record and a heading prepared. Control is transferred to Section (f) below.

d. Page DISPLAY

The DISPLAY command is entered at this secondary entry point from the PAGE command. The paging direction and mode are indicated by the PAGE parameters.

e. Validate Next Page

If the page requested has been seen before, it need not be regenerated, but may be retrieved from based storage, where it was saved, and control can be transferred to Section(g) below to display it. When non-contiguous skip paging is being done, the

relative key is taken from the set and the anchor record read.

f. Build Screen Image

This is a common routine for building a DISPLAY screen image either for an original DISPLAY command or for a PAGE command.

For a sequential format, field names are taken successively from the SEQFORM down to the number of field names.

In the most general case, each field consists of multiple elements and each element value is so long as to require multiple lines of a buffer. The first line for the first element of a field is tagged with the fieldname and a colon. The first line for an element after the first of a field is tagged with only the colon. Successive lines after the first for an element have their tag entirely suppressed. The degenerate cases of a single element field and/or an element short enough to fit on one line of the buffer are handled. And if the field is null (no data present), nothing is posted to the buffer at all for that field name.

Subfile resident fields are displayed similar to multiple elements; however, the first element of the field per subfile record has the field name tag duplicated, and a special heading is displayed (depending on the "type" parameter) as each new subfile record is processed.

If the field names are not all processed before the bottom line of the buffer is reached, the routine is left in such a state that it will resume where it left off if normal forward paging is attempted. But if the field names are all used, then the remaining lines are cleared.

For a columnar format, the optional page number, title, and header lines are copied into the buffer. Then field names are taken successively from the COLFORM, and used to retrieve the field values which are arranged across a line of the buffer. If there are any multiple element fields, futher lines of the FLDTAB buffer are used for remaining

elements until the record's desired fields have all been retrieved. If there are any further records in the set, the next record is read and the process repeated. When the buffer is full, the routine is left in such a state that it will resume where it left off if normal or skip paging is attempted. But if the data is exhausted, then the remaining lines are cleared.

g. Write Screen

Using the full screen mode of output, the current screen image is displayed on the terminal.

h. Return

Do a normal return to the calling routine.

3. Submodules Required

a. DB - data base package

b. PSTRAT - save strategy

c. TS - terminal support package

d. SETS - set information package

F. CODING SPECIFICATIONS

1. Source language

The DISPLAY command is coded entirely with the IBM PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the data base. The TSPL/I language extension handles all instances of communication with the terminal.

2. Suggestions and Techniques

Not Applicable

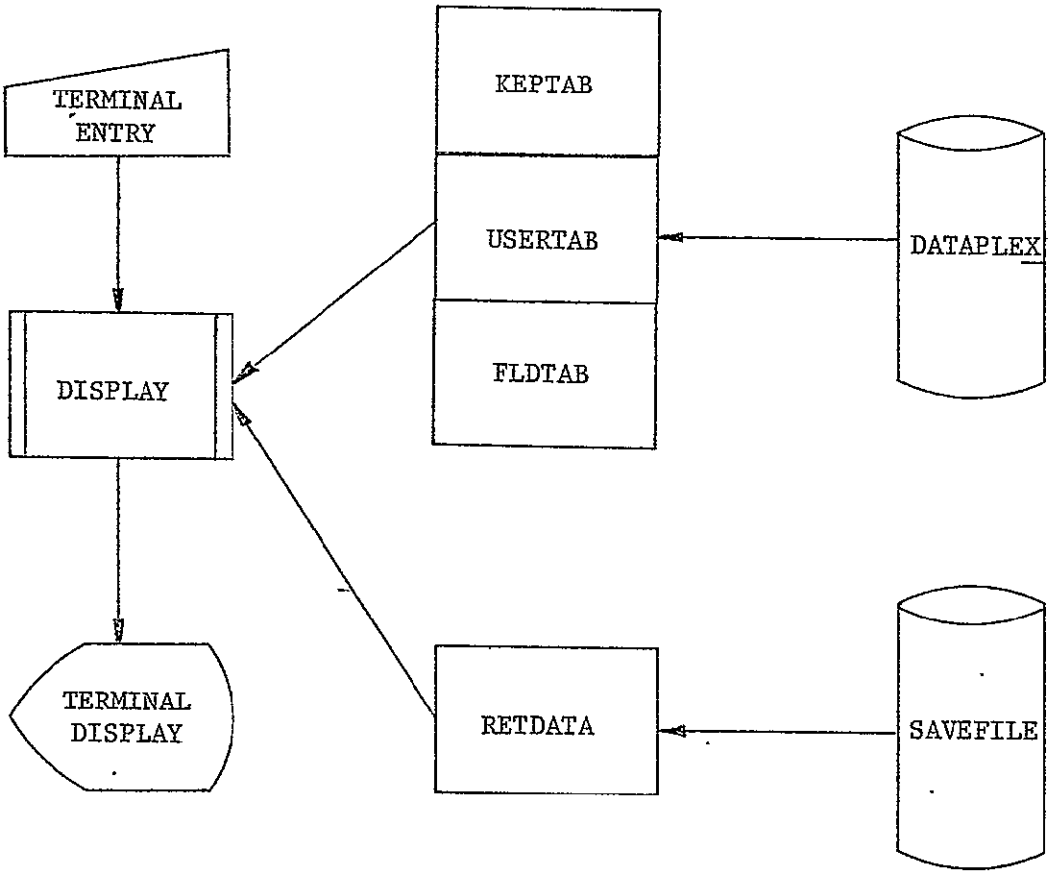


Figure 1. I/O Block diagram

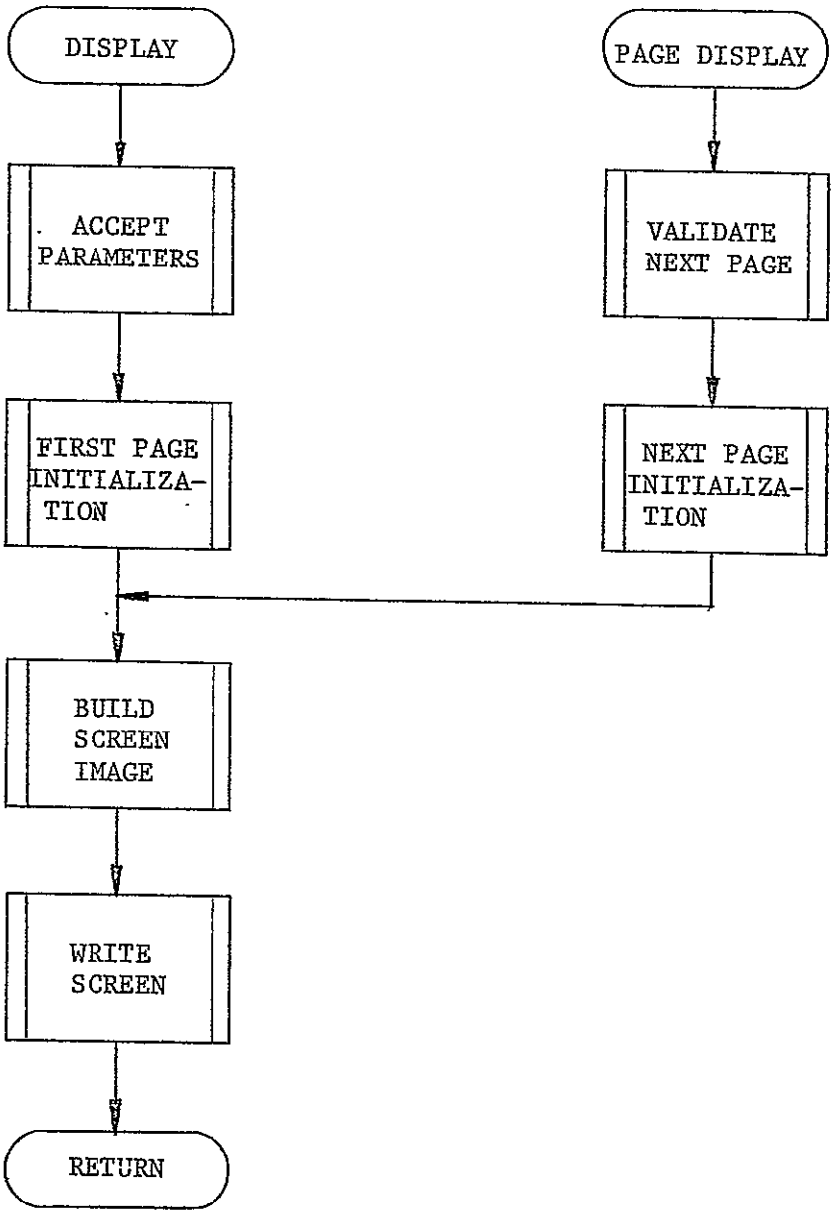


Figure 2. Top level flowchart

TOPIC F.6 - RETRIEVAL PRINT COMMAND

A. MODULE NAME

Retrieval, PRINT command
Program-ID - RDBPRNT
Module-ID - DBPRNT
Entry Point (DBPRNT)

B. ANALYSTS

Garth B. Wyman
William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

The PRINT command is a routine whose purpose is to allow the retrieval system user to have designated data for a given set listed on a high-speed printer. Like the DISPLAY command, the user may specify the format of the output as the citation number, the citation, the abstract, or the full text for any item or range of items contained in a set which has been previously selected. Optionally, the user may prespecify a format of his own, using the FORMAT command, to govern the PRINT command. All of the uses of the PRINT command during a single terminal session will be accumulated and printed out as one continuous output for the user to pick up at a later time. Three set numbers are reserved for special purposes in the retrieval system. Set-number 99 is an array in core used by the KEEP command to store parameter lists for the DISPLAY and PRINT commands. Set-number 98 is a data set used by the SAVE command to store screen images for later processing by the DISPLAY and PRINT commands. Set-number 0 is a logical reference to the entire anchor file. The calling sequence is: PRINT set-number, format, item(s) or, alternately, PRINT citation#, format.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The anchor and associated files of a data base are input to the PRINT command. The SAVFILE containing display screen images stored by the SAVE command is also input to the PRINT command.

d. On-line Terminal Entries

The parameters available to the PRINT command are "set-number", or "citation number", "format", and "items." NASIS will apply default values to the parameters if they are available, when no original values are entered.

3. Output Data Sets

a. Output Files

The output of the PRINT command consists of a data set containing the line images to be sent to the high-speed printer at the conclusion of the current terminal session. The line images consist of up to 132 characters, preceded by a carriage control character.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

When the current strategy is terminated, then the actual printing process is initiated. The description of this output is contained in the Data Set Specifications Section of the Development Workbook.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COLFORM

The PRINT command refers to a COLFORM table when a columnar format is referenced.

b. FLDTAB

The PRINT command refers to FORMTAB to locate the appropiate sequential (SEQFORM) or columnar (COLFORM) format table and for a table of data base field names ordered according to format numbers 1-4.

c. USERTAB

This table contains user-oriented and status information.

d. KEPTAB

The PRINT command refers to KEPTAB when the special set 99 is used for a PRINT specification that was previously stored by the KEEP command.

e. PLEX

The PRINT command uses a DBPL/I file called PLEX for all of its retrievals from the data base.

f. PRINTER

The PRINT command uses a PL/I file called PRINTER to write all of its printer line images for ultimate off-line printing.

g. PRTUSED in RETDATA

The PRINT command tests this switch to determine whether to write line images for a lead page identifying the report and then sets the PRTUSED switch to indicate that there are line images for off-line printing.

h. SRCHTAB

This table contains S-number information.

i. SAVFILE

The PRINT command uses a PL/I file called SAVFILE for all of its retrievals from the special set 98 of screen images previously

stored by the SAVE command.

j. SEQFORM

The PRINT command refers to a SEQFORM table when a sequential format is referenced.

k. RETDATA

This table contains data fields unique to the retrieval system.

l. ADDRESS in RETDATA

The PRINT command refers to ADDRESS when writing line images for the lead page of the report.

m. NAME in RETDATA

The PRINT command refers to NAME when writing line images for the lead page of the report.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Record

The PRINT command is called by the director.

b. Accept Parameters

Since the parameters are passed to the PRINT command through Terminal Support, they are arranged in a keyword or predefined order. The first parameter is either a "set-number", as defined by a SELECT or LIMIT command, or a "citation #" or an "S-number" as defined by a SELECT-IF command. The second parameter is a "format" code, and the third is an "item" number or range of numbers. The latter two parameters are optional. The set number will be a one- or two-digit number and will not likely have a default value since it changes for every command. The "citation #" is a character string which also will not likely

have a default. If no entry is made and no default exists, then the error is reported and control passed back to the calling routine. The "format" code only applies to a PRINT of a "citation#" or of set 0 - 97. It is a value of 1 to 25 designating a sequential format or F1 to F25 designating a columnar format or a format name representing one of the above format numbers. If no entry or default is present, the value of two is provided for anchor key sets or four for subfile sets. The "item" parameter is not required when the "citation #" is entered as the first parameter; otherwise, it designates the member or range of members of the specified set. The entry is a character string of one to eleven positions. When a range of items is entered, the two values are separated by a hyphen. If no entry or default is given for this parameter, all of the items in the set are printed. An invalid entry will be reported before control is returned to the calling routine. If all parameters have a valid value, then execution continues with the next section.

The PRINT command is placed as the next record in the strategy data set by a call to the save strategy routine. The parameters to this subroutine are the word PRINT and its parameters in their normal order.

If the first parameter was an S-number, processing continues with Section (g) below.

c. Initialization

The data set for the printer file will have been defined in a procdef before the PRINT command is called. The first time the PRINT command is used, it writes line images for a leader page identifying the user's name and mail stop for distribution.

If the first parameter specifies the special set 98 of saved screen images, processing continues at Section (f) below. If it specifies a data base key, e.g., a citation number, then the anchor record is read and processing continues at Section (D) below. If the parameter is a set number, then it is examined for validation and the first

relative key specified by the item range parameter is taken from the set and used to read the first anchor record. If the parameter refers to the special set 99 of kept items, then KEPTAB is examined to find the first relative item specified by the item range parameter and the first relative key specified by the item range in KEPTAB is taken from the set and used to read the first anchor record.

d. Process from Data Base

For a sequential format, field names are taken successively from the SEQFORM beginning with the key field name down to the number of field names.

In the most general case, each field consists of multiple elements and each element value is so long as to require multiple lines on the printer file. The first line for the first element of a field is tagged with the fieldname and a colon. The first line for an element after the first of a field is tagged with only the colon. The first subfile field value per subfile record retains the field name as a tag; however, subsequent elements are nevertheless tagged with only a colon. Successive lines after the first for an element will be automatically wrapped around to new lines by the PL/I stream output and are tagged. If the field is null (no data present), nothing is written at all for that field name. The PL/I stream output will detect the logical end of page condition so that page heading lines can be inserted and then normal outputting resumed.

e. Re-initialization

If a data base key is specified by the user, this Section is bypassed and control is returned at Section (g) below.

If there are more keys in the set within the range specified by the user or in the KEPTAB item, then the new key is taken and used to read another anchor record and control loops back to Section (d). When this loop is completed, control is returned at Section (g) below unless the user had specified a range of kept items from set 99. In that case, the

next kept item is taken from KEPTAB and the first relative key specified by the item range in KEPTAB is taken from the set and used to read another anchor record and control loops back to Section (d). Finally, control is returned at Section (g) below.

For a columnar format, the optional page number, title, and header lines are put out at the top of each page of output. Field names are taken successively from the COLFORM and used to retrieve the field values which are arranged across the output line. If there are any multiple element fields, futher lines are put out until the record's desired fields have been retrieved.

f. Process from SAVFILE

One or a contiguous range of saved screen images are successively retrieved from SAVFILE (set 98). For each screen image, a page heading line is written followed by the screen image subdivided into lines the same length as the display screen width so that the appearance is identical.

g. Return

When all processing for the PRINT command has been completed, control is returned to the calling routine.

3. Subroutines Required

a. DB - data base package

b. PSTRAT - save strategy

c. TS - terminal support package

d. SETS - set information package

F. CODING SPECIFICATIGNS

1. Source Language

The PRINT command is coded entirely with the IBM PL/I programming language. The DBPL/I language extension is used to handle all access to the files in the data base, and the TSPL/I extension handles all instances of communication with the terminal.

2. Suggestions and Techniques

   a. Normal PL/I statements are used to write the line images to the print data set.

   b. The many external variables required in the PRINT command are combined into external data structures, in many cases. This requires only one name to be an external symbol.

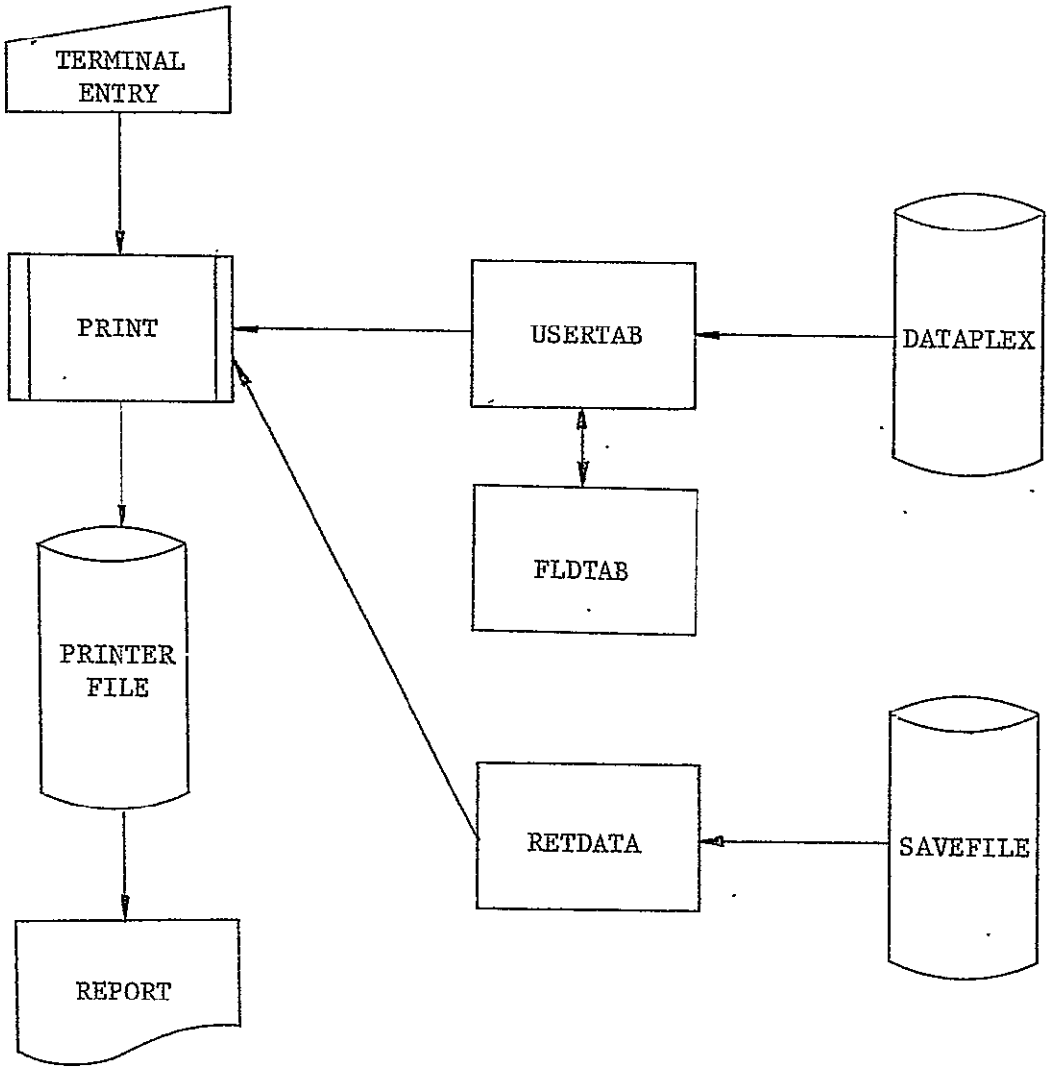


Figure 1. I/O Block diagram

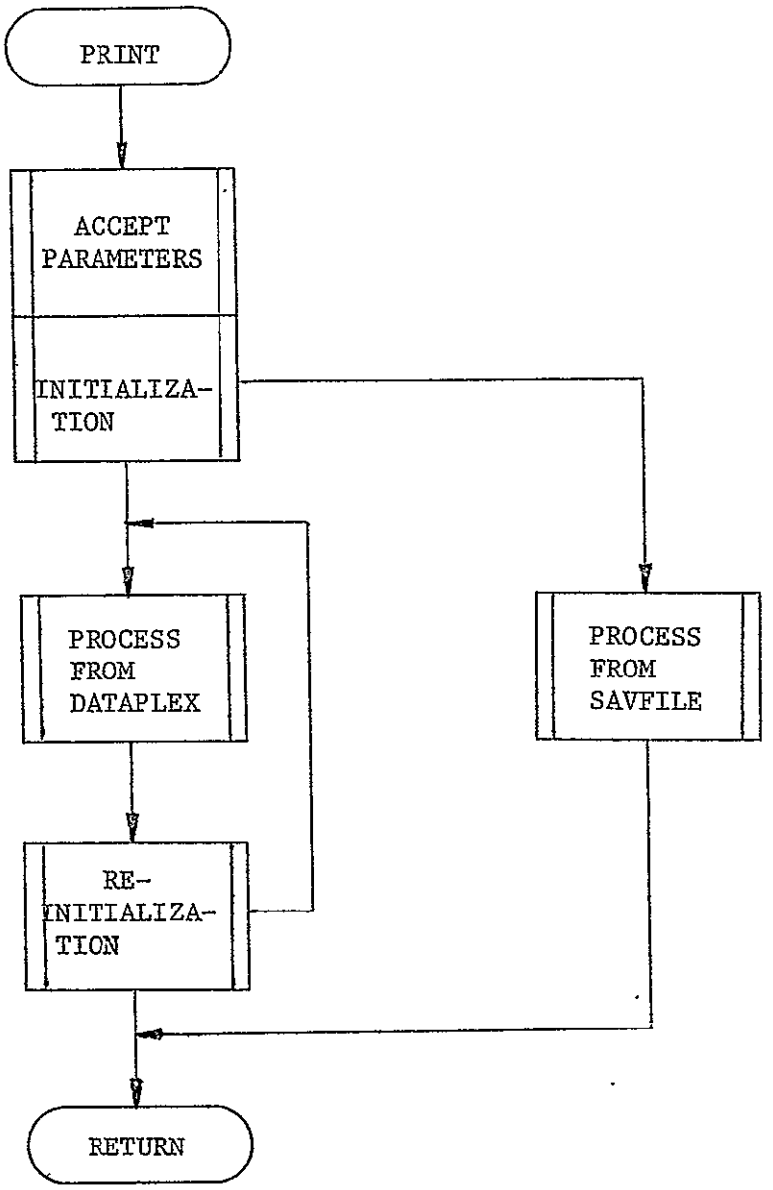


Figure 2. Top level flowchart

TOPIC F.7 - RETRIEVAL EXECUTE COMMAND

A. MODULE NAME

Retrieval, EXECUTE Command
Program-ID - RDBEXSR
Module-ID - DBEXSR

B. ANALYSTS

Barry G. Hazlett
William H. Petrarca
Neoterics, Inc.

C. MODULE FUNCTION

Use of the EXECUTE command informs the NASIS system that user has specified all of his SELECT-IF and/or PRINT commands for his linear search and is now ready to have them executed.

The format of the Execute-Search command is as follows:

EXECUTE

Use of the EXECUTE command informs the NASIS system that the user has specified all of his search requests on a set and is now ready to have them executed. When an attention interrupt is made, the EXECUTE command will return the user with its current status; i.e., the number of processed records and the number of records to be processed. To continue any further in the execution of the linear search, the user must then enter:

GO — which will resume the search at the point of execution, or

END — which will cancel the search in progress, returning the user to the point of his strategy immediately before the last EXECUTE.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input files

The data base anchor file is accessed to obtain the records for the linear search.

d. On-Line Terminal Entries

If a Terminal is the source of EXECUTE parameters.

3. Output Data Sets

a. Output Files

Using the PSTRAT routine, the command string, as it is entered (modified if any by prompt responses) and validated, is saved in the region CURRENT-STRATEGY of the VISAM member DBSTRAT of the VPAM data set USERLIB. For a complete description of the data set DBSTRAT, refer to the Specifications for the module DBGPS (DWB, Section IV, Topic F.8).

b. On-Line Terminal Displays

The following is displayed at the output interface:

1. new set number,
2. items contained in a new set, and
3. the (combined) expression describing the new set.

for each set created as the result of the linear search.

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. FLDTAB is the descriptor field table referenced to determine whether a search field name is an inverted index. The anchor file key field name is used from FLDTAB.

b. SRCHTAB is the search table referenced to maintain search testing criteria, pseudo-set information, and search list pointers.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The EXECUTE calling sequence is as follows:

CALL DBEXSR

Search processing will follow the following steps:

1. Notify STATISTICS of the search and what data base.

2. Identify a search set; group tests on that set.

3. Read in records of the the search set one at a time.

4. For each record test each field against its corresponding test criterion as defined in the LS strategy.

5. Each successful record is added to a search list associated with the pertinent test pseudo-set.

6. After all records have been tested, new sets are made with the lists for pseudo-sets defined by the SELECT-IF command.

7. Each pseudo-set defined by a Booolean SELECT is made into a set via a call to a special entry point in the SELECT command.

8. If there is another set to search continue at step 2.

9. All pseudo-sets requiring a "PRINT" are printed via a call to the PRINT command (DBPRNTS).

At the search termination all unnecessary dynamic storage will be freed. In addition a special entry point into the EXECUTE module for the CANCEL command will accomplish the same function.

F. CODING SPECIFICATIONS

1. Source Language

The EXECUTE command is written in the IBM/360 TSS PL/I programming language. The DBPL/I and TSPL/I language extensions are used for data base file accessing and terminal communication, respectively.

2. Suggestions and Techniques

It is suggested that considerable analysis be made of search universes to determine the final search universe for the EXECUTE command due to the rather large data bases that may exist. The success of reducing a search universe to its minimal size is reflected to the user in response time.

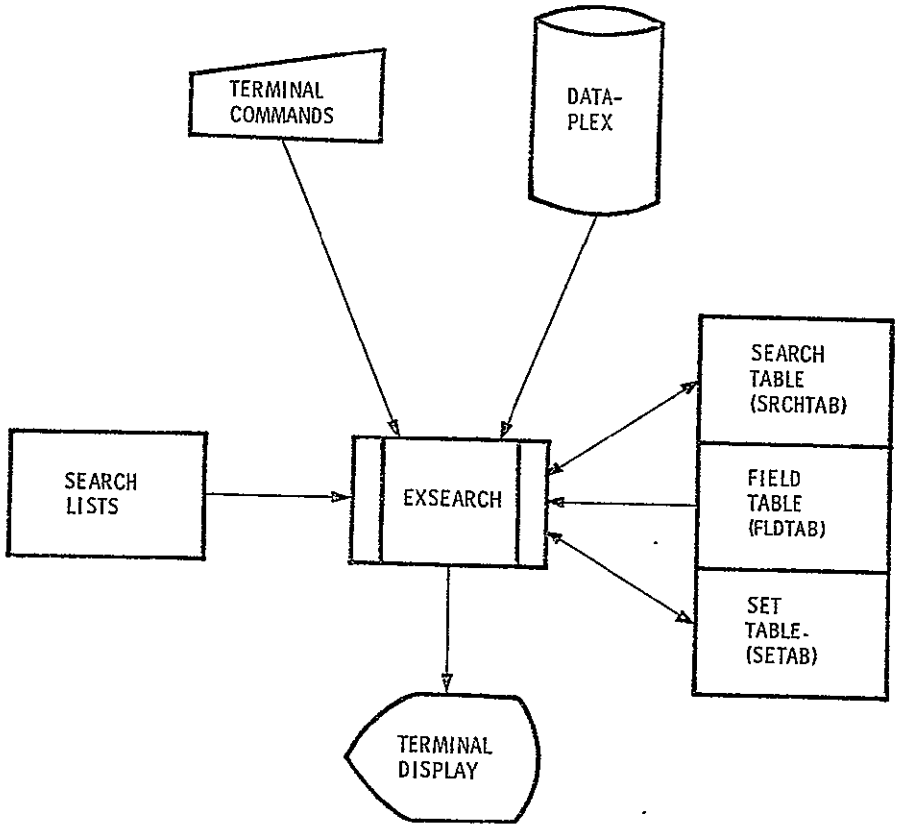


Figure 1. I/O Block diagram

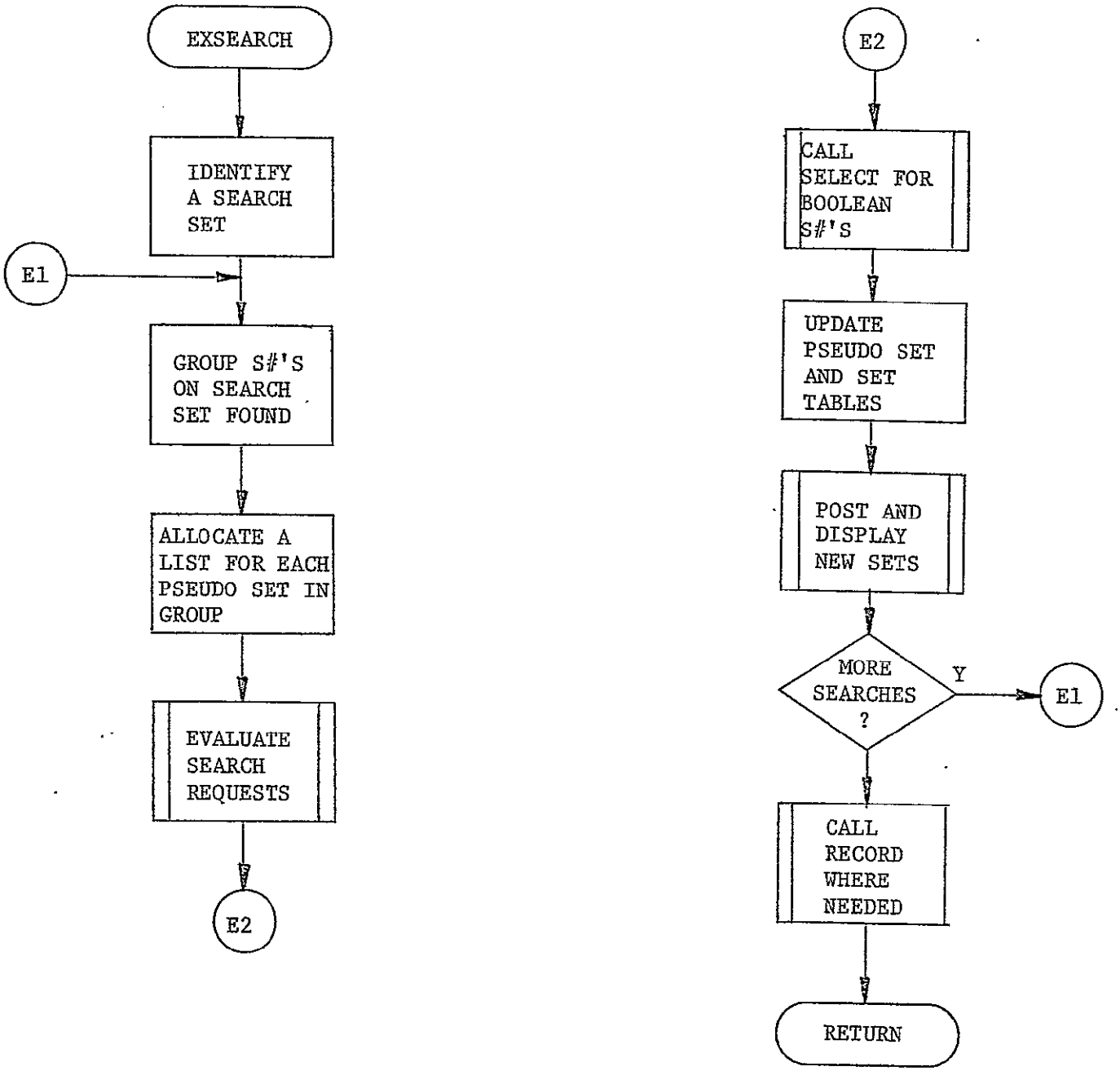


Figure 2. Top level flowchart

TOPIC F.8 RETRIEVAL SETS COMMAND

A. MODULE NAME

SETS Command and Sets Management
Program-ID - RDBSETS
Module-ID - DBSETS
Entry Points - (DBSETS, DBGSET, DBPSET, DBPAGST)

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

The primary function of the DBSETS module is to display to the NASIS Retrieval Sub-system user a list of the sets or s-numbers he has formed during the current strategy session. The list is displayed in the form; set number or s-number (including the subfile suffix, if present), the number of items in the set, and the expression that formed the set.

The entry points DBGSET and DBPSET are called from application programs to GET SETS and POST SETS, respectively.

The entry point DBPAGST is called by the PAGE command to display the next 'page' of sets in the user's current strategy.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

SETAB and STRATEGY.LIBRARY

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

SETAB and STRATEGY.LIBRARY

b. On-line Terminal Displays

The terminal display from this module will consist of a list of the set numbers or s-numbers (including the subfile suffix, if applicable), the number of items in the set and the expression that formed the set.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. SETAB

b. TS

c. DB

d. STRATEGY.LIBRARY

e. SRCHTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBSETS

Upon entry at the DBSETS entry point, it allocates and initializes a controlled area for the current user to keep track of his paging operations.

The module looks for any parameters that were passed with the command. If there are none, the module will default to set numbers and start displaying at the beginning of SETAB.

If a number between 1 and 97 is passed, the module verifies that as a valid set number and starts the displaying with that number.

If the parameter is an 'S' the module will display s-numbers (pseudo-sets). A second parameter may be included here to indicate starting at a specific s-number.

This processing continues until the bottom of SETAB or SRCHTAB is encountered or the TS supervisor indicates the output screen is full and automatically writes the screen.

DBSETS saves the set number or s-number, that would have caused the screen overflow, in the user control table. This set number or s-number is then used as the first number to appear on the next page forward.

b. DBPAGST

This entry point is called by the TS supervisor when the user wishes to page in either a forward or backward direction through his list of sets.

Upon entry DBPAGST vaidates the command and (re)constructs a page in the appropriate direction. Only the letter 'B' will cause a backwards page operation; anything else defaults to forward.

c. DBPSET

This entry point is available to the application programmer who wishes to post a new set and its corresponding data to SETAB. The calling sequence follows:

```
CALL DBPSET(POINTER,EXPRESSION,SET#);
```

Where:

POINTER - is a pointer variable passed by the user. It points to the list to be posted.

EXPRESSION - is a varying length character

string, maximum 256 bytes. It is passed by the user as the expression that formed the set to be posted.

SET# - is a varying character string, maximum 2 bytes long. It is passed by the user as the one byte subfile suffix character for the set being posted and is returned by DBPSET as the 2 byte set number on a successful posting or a null string to indicate an I/O error or no more sets available.

This entry point first checks for a slot in SETAB; if none are available, it sets the set number variable to null and returns to the user.

If a set number is available,it verifies the suffix as being between Q and Z or it assigns a blank suffix. DBPSET then collects and posts the data to SETAB and the STRATEGY LIBRARY. It posts the set number for the user and returns.

d. DBGSET

This entry point is available to the application programmer who wishes to get and verify a given set number. The calling sequence follows:

```
CALL DBGSET(SET#,POINTER,#LIST,SUFFIX);
```

Where:

SET# - is a varying length character string, maximum 3 bytes long. The user passes this variable as the set number, and optionally the subfile suffix, to be gotten and verified. If either the set number or the suffix is invalid, that is, a non-existent set number or a wrong suffix, this variable is returned as null.

POINTER - is a pointer variable. It is returned by DBGSET as a pointer to the set (list).

#LIST - is a integer full word. It is returned by DBGSET as the number of XREFS in the set.

SUFFIX - is a single character. It is always

returned as the correct suffix for the set requested. In the event an invalid suffix is specified in the set number, the set number is returned as null and the correct suffix is returned here.

DBGSET first separates the set number from the suffix and verifies both. If either is invalid, set number is returned as null and the correct suffix, if available, is put in SUFFIX. If the validation is successful, the set number, the list pointer, the number of XREFS and the suffix are returned to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The RDBSETS command module is written in the IBM/360 TSS PL/I programming language. The DBPL/I and TSPL/I language extensions are used for data base access and terminal I/O, respectively.

2. Suggestions and Techniques

Not Applicable

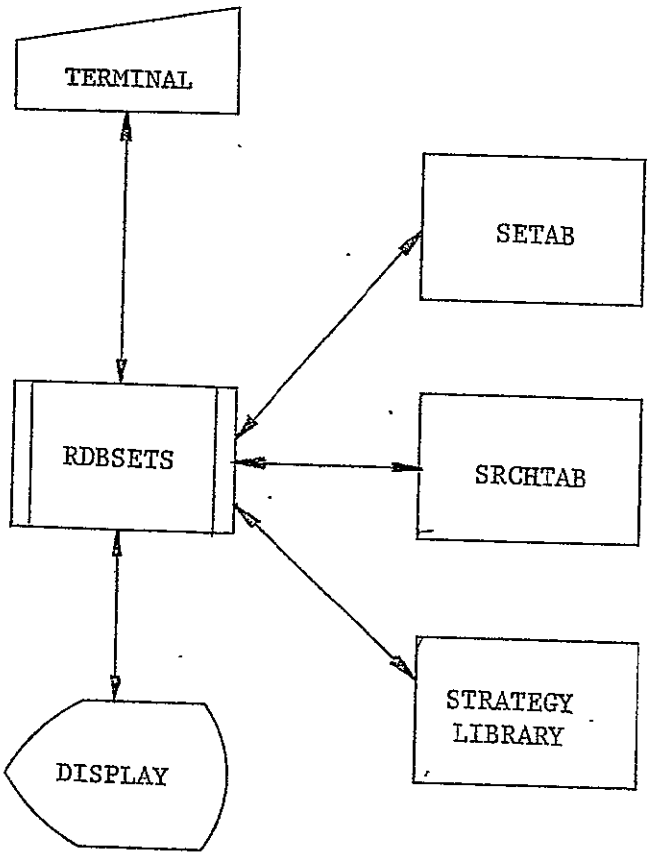


Figure 1. I/O Block Diagram

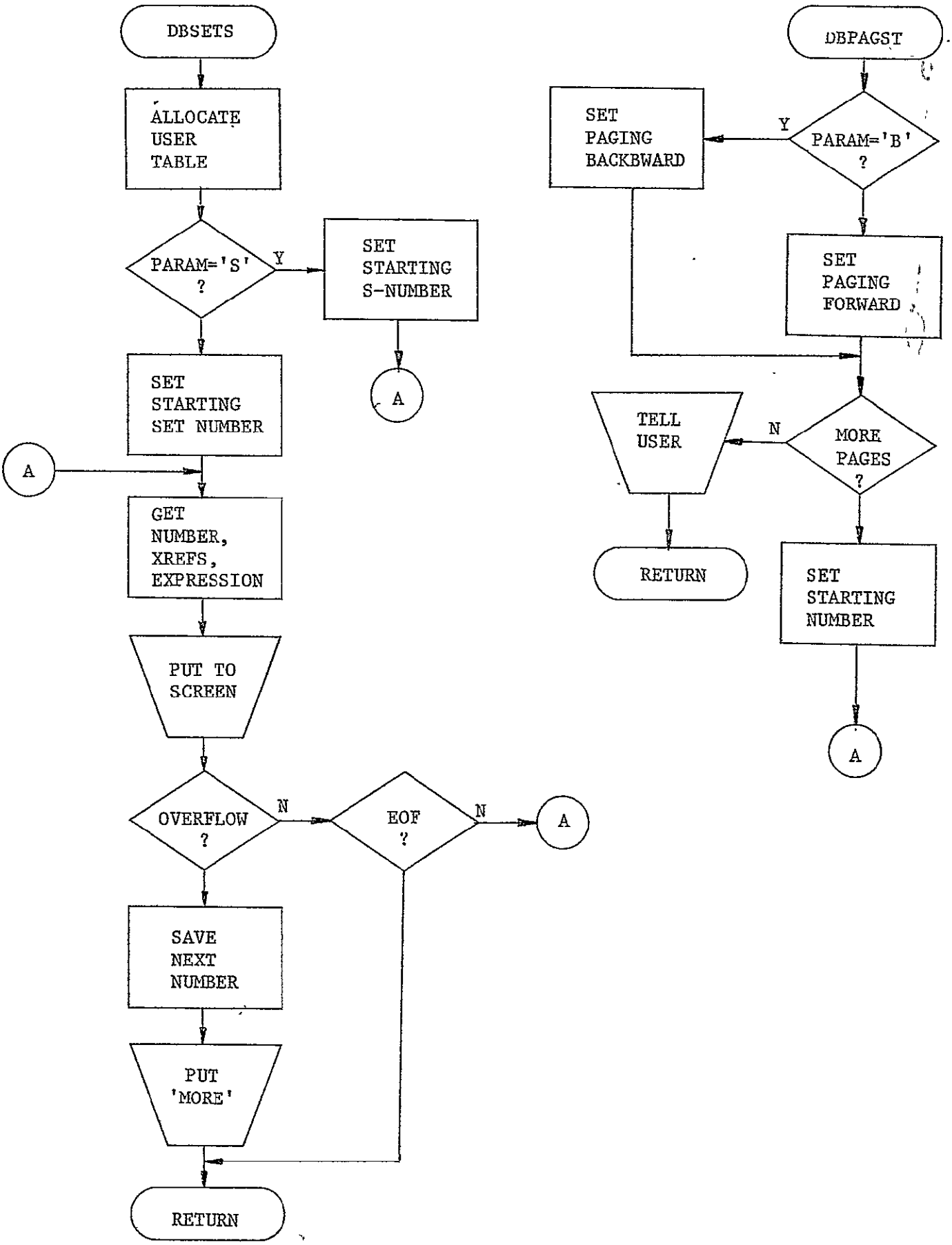


Figure 2A. Top Level Flowchart

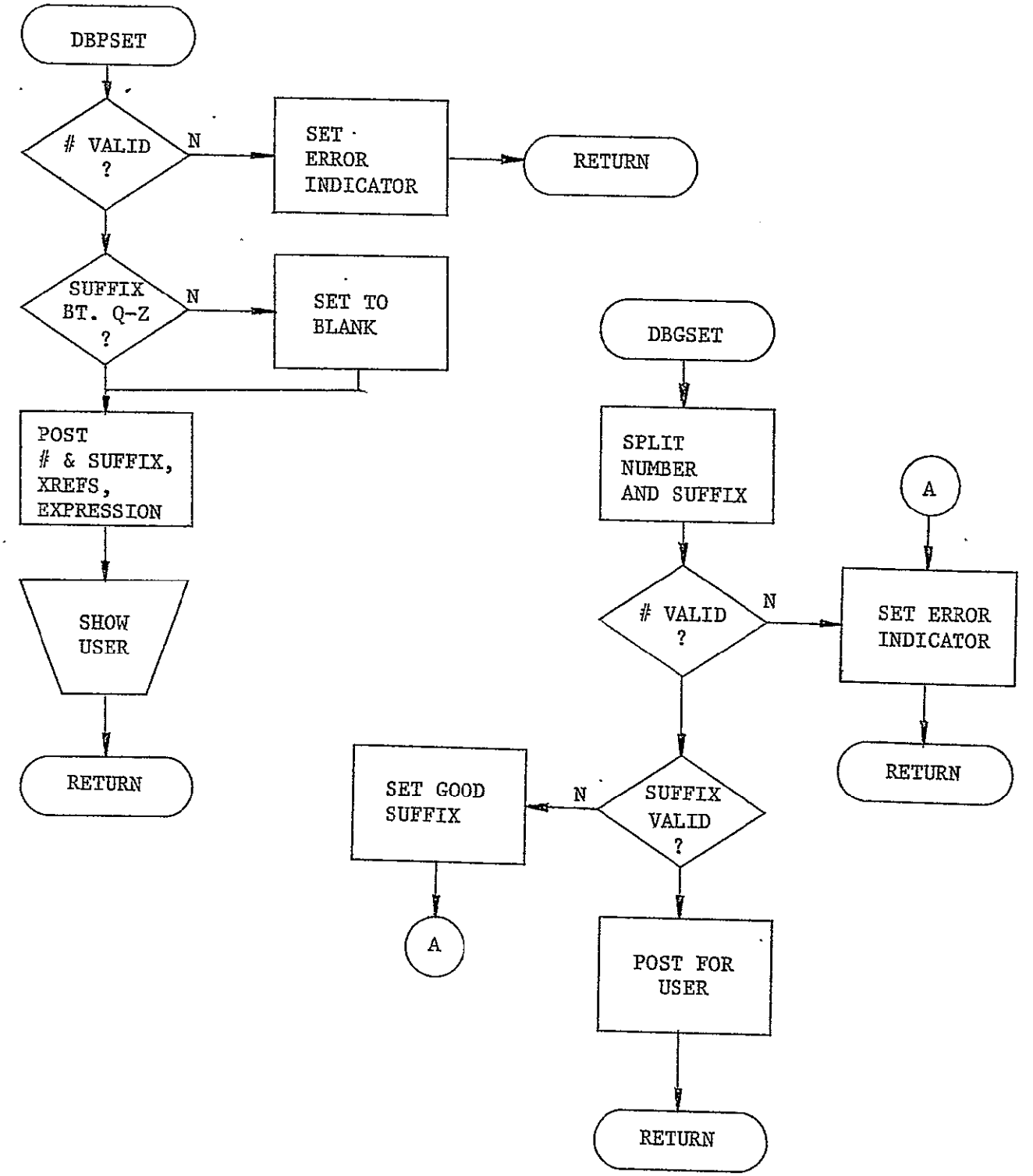


Figure 2B. Top Level Flowchart

TOPIC F.9 - GENERIC KEY LISTS

A. MODULE NAME

Program-ID - RDBGENR
Module-ID - DBGENR

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module expands or contracts lists of generic keys based upon the user's specifications and the generic key description table.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The anchor and index files of a data base may be used for input by the program.

d. On-Line Terminal Entries

The program prompts the user for the FIELD (sub-level name) and SET parameters.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program writes diagnostic messages to the user for any errors encountered.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program uses the following tables to obtain information necessary to perform its function,

FLDTAB
GENERIC

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBGENR1

At this entry point the program initializes itself to process data passed by another program. A switch is set to indicate this fact, so that parameter prompting and the posting of SETAB can be bypassed.

b. DBGENR

At this entry point the program initializes itself to process the user's GENERATE command. It extracts the current file name from FLDTAB and calls the generic key routine to obtain the generic key description table. The program then prompts for and verifies the two parameters. If the SET parameter is not a valid set number, the program uses it as a key and reads the anchor file to verify it. If any errors are detected during the above operations, the program terminates with an appropriate diagnostic message.

The list described by the second parameter is analysed to determine the generic sub-level represented by its keys. This result is

compared against the sub-level defined by the first parameter to determine whether this is a request for parent or children processing.

For parent processing, the list of keys is analyzed, one at a time, and the unique parent or root records are derived and posted to a resultant list. This processing is done by the structural analysis of the keys, based upon the sub-levels determined above.

For children processing, the generic key index field name is extracted from the generic key description table. The input list of keys is used to read this index file by key. As each record is read, the list of cross references is ''or'ed'' logically to the previous list of cross references creating an aggregate list. When the end of the input list is reached, the sub-levels are compared, and if more sub-levels remain to be processed, the resultant cross reference list is used as the new input list and the process is repeated.

At the completion of list processing, for either parent or children lists, the program posts the resultant list. If entry was to DBGENR, this involves a call to DBPSET to port SETAB. If entry was to DBGENR1 this involves the posting of the caller's parameter list. The program then returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

This module is written using the TSS 360 PL/I Language.

2. Suggestions and Techniques

Not Applicable

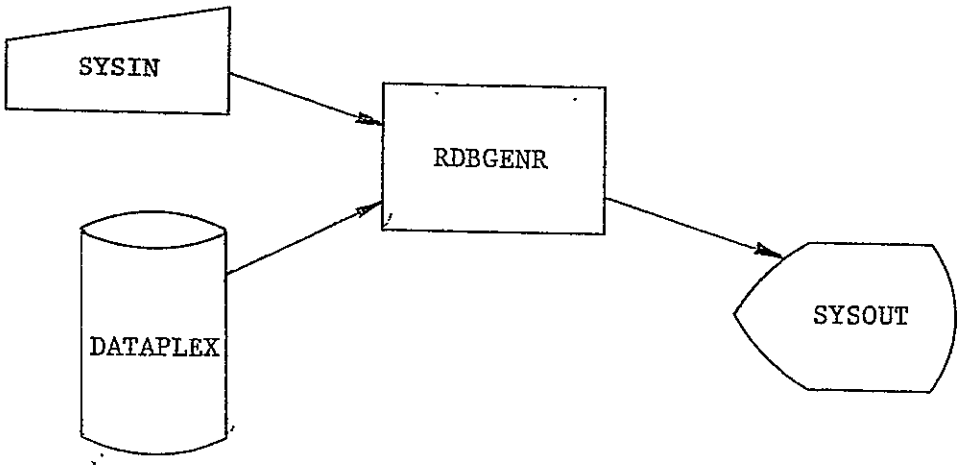


Figure 1. I/O Block Diagram

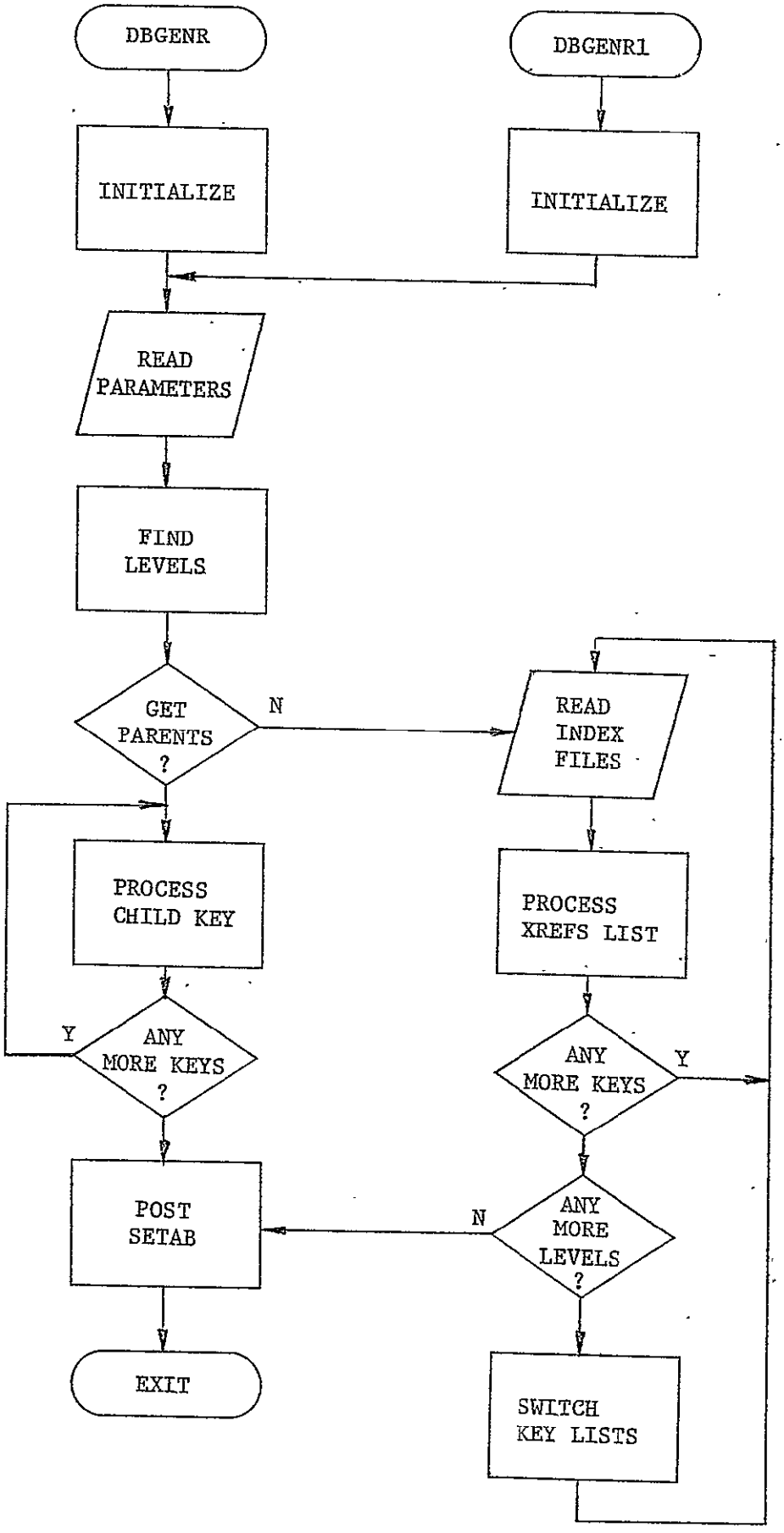


Figure 2. Top Level Flowchart - DBGENR, DBGENR1

IV. E.9.4

TOPIC F.10 - RETRIEVAL FORMAT COMMAND

A. MODULE NAME

Retrieval, FORMAT Command
Program-ID - RDBFORM
Module-ID - DBFORM. Entry points: DBFORM (primary) and DEFORM (for PAGEing a format DISPLAY).

B. ANALYST

Garth B. Wyman
Neoterics, Inc.

C. MODULE FUNCTION

The DBFORM module is the FORMAT command routine, called by the RETRIEVEal system, whose purpose is to allow the retrieval system user to define, revise and/or display the content and format for subsequent information retrievals using the DISPLAY or PRINT retrieval commands. Sequential and columnar formats may be defined.

Sequential formats extend the series of predefined formats 1-4 by allowing the user to select a set of fields to be displayed one under another with no more than one record's fields per output page.

Columnar formats are a separate series allowing the user to select a set of fields to be displayed in tabular format. Optionally, the user may define screen or printer output, page numbering, titles, column headers, column positions, and element tallying, summing and averaging.

After a current format has been established, the DBFORM module functions as a command director processing the FIELD, FIELDS, NAME, STORE, FORMATS, DISPLAY, PAGE, TITLE, HEADER, FORMAT and END subcommands of the FORMAT command.

The user may review the appearance of the ultimate display (paging through screen-width portions, if necessary). The user has complete revision and storing capability.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

   a. Parameter Cards

      Not Applicable

   b. Punched Card Input Files

      Not Applicable

   c. Input Files

      Not Applicable

   d. On-line Terminal Entries

      A terminal is the most likely source of the parameters which are passed to the FORMAT command by the Terminal Support system. The fundamental parameters are the format number and the field names. Default values for the fundamental parameters are unlikely. The FORMAT command then accepts the FORMAT subcommands and their parameters.

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      For sequential formats, the DISPLAY subcommand will display the field names vertically in the order they will ultimately be displayed. The PAGE subcommand will display any field names that do not appear on the first screen.

      For columnar formats, the DISPLAY subcommand will display the title and header values and field column positions as they will ultimately be displayed. In the case of printer formats wider than the display screen, the left-most portion will be displayed initially. The PAGE sub-command will display subsequent portions. These displays will show the positioning and length of the field values for the first data line; otherwise, they have the same format as the DISPLAY and PRINT retrieval commands produce (see Section III, Topic F.4 of the DWB).

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

a. COL_FORM

When the FORMAT command processes a new columnar format, it allocates and initializes a COL_FORM structure and posts its base address in the COL_FORMAT array in FLDTAB. When the FORMAT command processes a TITLE or HEADER sub-command or any other revision to a columnar format, it updates the appropriate COL_FORM structure. Thus, a COL_FORM structure specifies a columnar format for use by the DISPLAY and PRINT commands.

b. FLDTAB

The FORMAT command refers to the DATA BASE and FIEID portions of FLDTAB for descriptor information previously posted by RDBINIT. The FORMAT command posts the SEQ_FORMAT and COL_FORMAT arrays as it processes new formats.

c. SEQ_FORM

When the FORMAT command processes a new sequential format, it allocates and initializes a SEQ_FORM structure and posts its base address and field name count in the SEQ_FORMAT array in FLDTAB. Thus, a SEQ_FORM structure specifies a sequential format for use by the DISPLAY and PRINT commands.

d. USERTAB

The FORMAT command checks the USERTAB.RETRIEVE switch to verify that it is being called properly.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. Format

The FORMAT command is recognized by the retrieval system director module RDBINIT which calls the DBFORM entry point.

b. Process FNUMBER parameter

If null or blanks are entered, the FORMAT command is cancelled. The value is checked for proper syntax and for range and duplication of the number; errors are diagnosed and the user allowed to re-enter. If the value is a name, the external GETSFMT routine is called to obtain the stored format. For a new format, a SEQ_FORM or COL_FORM structure is allocated and initialized according to given and default options and the structure's base address posted in FLDTAB. For a revised columnar format, any options given will result in the COL_FORM structure being modified or re-allocated and initialized accordingly. Removal of page numbering may be specified and/or expansion to printer width or contraction to screen width. If the width changes, any titles are re-centered. If the width changes and the columns are proportional, they are re-proportioned and their headers (if any) re-centered. If the width expands and the columns are explicit, the rightmost column will have its width expanded and its headers (if any) re-centered. If the width contracts and the columns are explicit, columns to the right of a screen width are dropped from the format with their headers (if any) and the remaining rightmost column will have its width reduced and its headers (if any) re-centered.

If a FLDSPEC parameter was entered explicitly by the user with the FORMAT command, control passes to (d.) below where the parameter is processed. Otherwise, processing continues at (c.).

c. Process subcommand

A command is obtained from the Terminal Support system. If it is a valid FORMAT subcommand, it is processed by one of the

routines (d.) through (k.) below. Otherwise, it is diagnosed as an invalid subcommand and the user allowed to re-enter..

d. Process FIELD command

The field names are checked for existence in the current data base by lookup in the FIELD portion of FLDTAB. If a field name or porition is invalid, a diagnostic is issued and the keyboard unlocked for re-entry of that field name with any options or default for that field to be ignored. Normally, for suequential formats, the field name is posted in SEQ_FORM, or for columnar formats the field name, position (proportioned, if not specified by the user) and options are posted or updated in COL_FORM.

e. Process FIELDS or FORMATS command

These commands are recognized as a convenience to the user to save him having to leave FORMAT and later re-enter it. Processing consists only of a call to the external entry point DBFLDS or DBSTRT2 respectively.

f. Process NAME or STORE command

An FMTNAME parameter value is obtained from the Terminal Support system, validated syntactically by calling the external DBUCHEK routine, and checked for duplication of the name of any other current format. For a NAME command, the value is simply posted in FLDTAB. For a STORE command, the value is posted in FLDTAB or it is verified that a name value was posted there previously and the external PUTSFMT routine is called to store the format for availability in later sessions. If the FMTNAME value is invalid or missing or if PUTSFMT returns an error code, a diagnostic is issued and the user allowed to re-enter it.

g. Process TITLE command

If the current format is not columnar, the TITLE command is cancelled with a diagnostic message.

A TTLLINE parameter value is obtained from

the Terminal Support system, if the user entered it explicitly, or by assuming the next relative title line number. The value is checked for syntax, range, duplication, and space in COL_FORM.TOP. Any error is diagnosed and the user allowed to re-enter the parameter. For a title line deletion, any lower title and header line images are shifted up and COL_FORM.TOP.#TITLES is decremented and control branches to (c.). For a new title line, any lower title and header line images are shifted down and intervening lines blanked in COL_FORM.TOP.LINE and COL_FORM.TOP.#TITLES is posted.

A TTLSPEC parameter value is obtained from the Terminal Support system, if the user entered it explicitly, or by taking the FLDTAB.DATA BASE name value and stripping any trailing dollar sign characters. The value is posted centered in the particular COL_FORM.TOP.LINE.

h. Process HEADER command

If the current format is not columnar, the HEADER command is cancelled with a diagnostic message.

A HDRLINE parameter value is obtained from the Terminal Support system, if the user entered it explicitly, or by assuming the next relative header line number. The value is checked for syntax, range, duplication, and space in COL_FORM.TOP. Any error is diangosed and the user allowed to re-enter the parameter. For a header line deletion, any lower header line images are shifted up and COL_FORM.TOP#HEADERS is decremented and control branches to (c.). For a new header line, any lower header line images are shifted down and intervening lines blanked in COL_FORM.TOP.LINE and COL_FORM.TOP.#HEADERS is posted. Thus a current header line is determined for the following processing.

If no HDRSPEC parameter values were entered explicitly by the user, every column accross the current header line has its field name value centered over it and control branches to (c.).

Otherwise, HDRSPEC parameter values are obtained one by one from the Terminal Support system and processed individually. If only a literal value is given, it is centered over the next column to the right. If only a parenthesized field name is given, it is centered over the column for the field name. If both a literal value and a parenthesized field name are given, the value is centered over the column for the specified field name. Any syntax, field name, or past rightmost column error results in a diagnostic message allowing the user to re-enter one value or to default for that value to be ignored.

i. Process DISPLAY command

The display simulates the appearance produced by the retrieval system DISPLAY command if it was used with the current format.

If a sequential format display overflows the screen at the bottom or if a columnar format display overflows the screen at the right side, "MORE" is indicated and the Terminal Support system is requested to call DBFORMP if the user enters the PAGE immediate command.

When the module is entered at the DBFORMP paging entry a DIRECTON parameter value is obtained from the Terminal Support system, if the user entered it explicitly, or by assuming forward paging. If the value starts with "B" the previous display screen image is re-composed, otherwise the next display screen image (down or to the right) is composed. Screen overflow is rechecked to reset the "MORE" indication and the Terminal Support system transmits the screen image to the user's terminal.

j. FORMAT command

If "FORMAT" is detected as a sub-command, control simply branches up to (b.) where its parameters are obtained and it is processed. (This is more efficient than "END;FORMAT" because the DBFORM module stays active.)

k. END command: RETURN

If the END condition is raised by the user entering the END immediate command in blocks (a.) or (b.), control returns to the RDBINIT module. If it is raised after block (b.) control branches to block (c.), that is, the subcommand is aborted and another taken.

3. Submodules required

DBFLDS - FIELDS command
DBSTRT2 - FORMATS command
DBUCHEK - check name routine
GETSFMT - get stored format
PUTSFMT - put stored format
PSTRAT - save strategy
TS - terminal support package

F. CODING SPECIFICATIONS

1. Source Language

The FORMAT command is coded in TSS PL/I. The TSPL/I language extension is used for all communication with the terminal.

2. Suggestions and Techniques

The PSTRAT external routine shall be called whenever a valid command or subcommand with valid parameters is detected.

Subroutine facilities shall be coded to handle the general case of re-proportioning columns and re-centering headers. (DUP_COL, RE_PRO_COL, RE_HEAD).

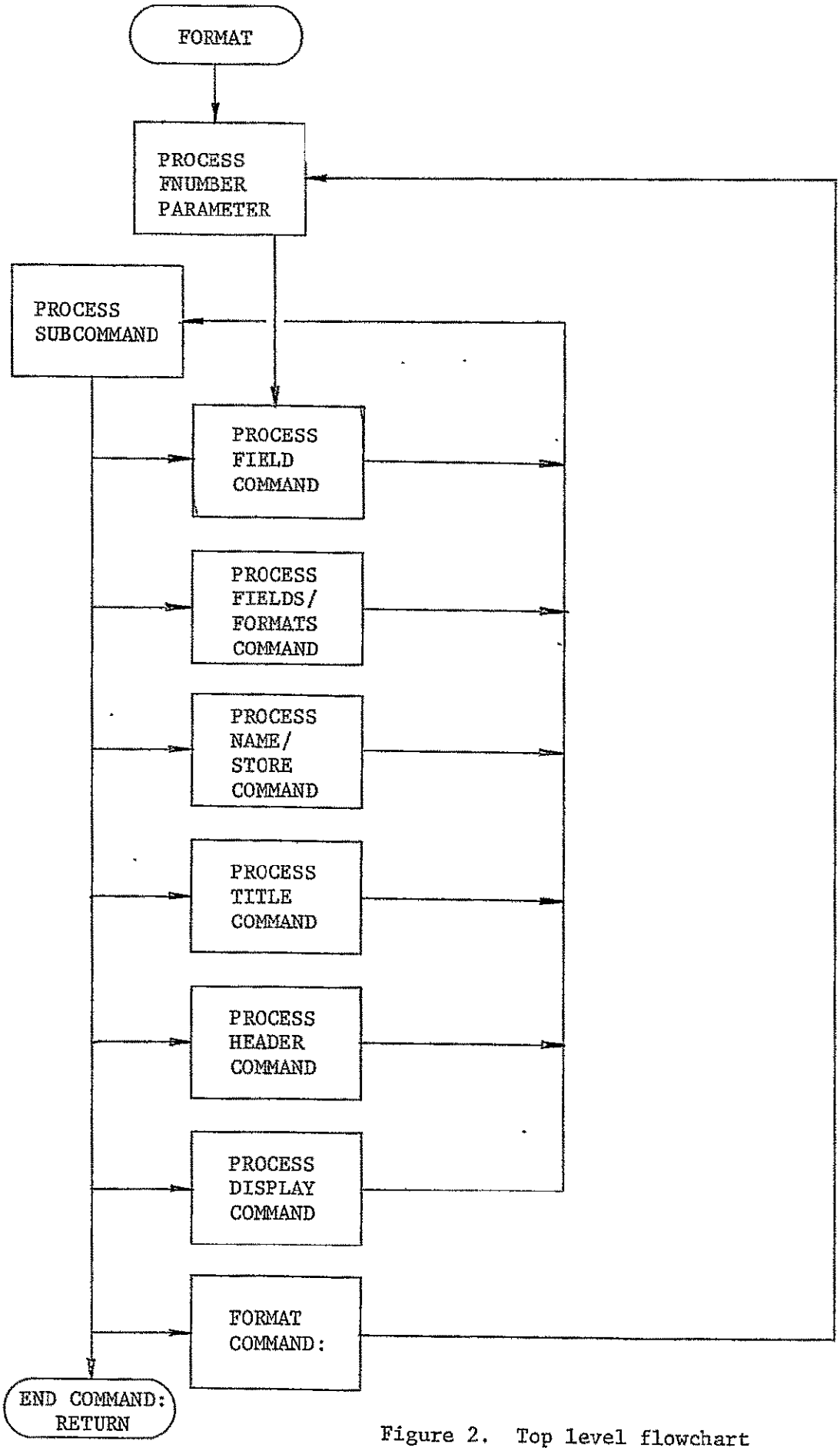


Figure 2. Top level flowchart

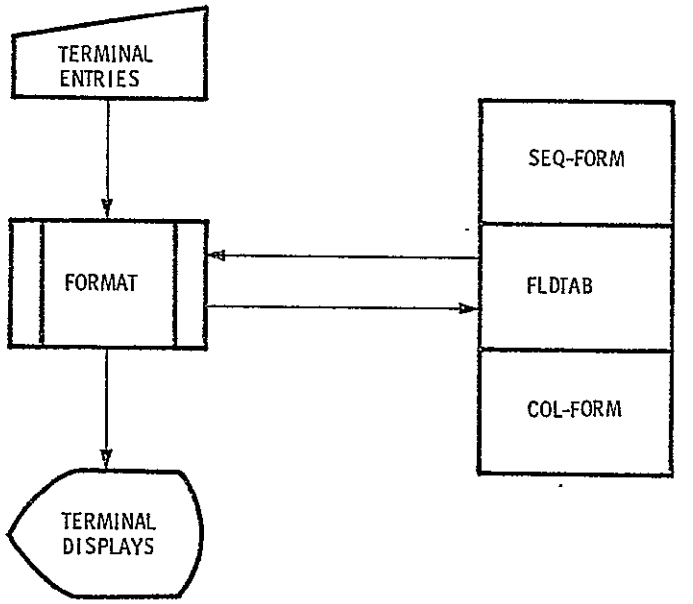


Figure 1. I/O Block diagram

TOPIC F.11 - STORED FORMATS

A. MODULE NAME

Program-ID - RDBSFMT
Module-ID - DBSFMT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

The function of this module is to provide generalized GET/PUT routines for the processing of stored formats.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input File

Not Applicable

c. Input Files

Not Applicable

d. On-time terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program produces diagnostic messages for the varicus errors that may occur.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The following tables are referenced, used in the construction of new formats and used to output exiting formats.

FLDTAE
SEQFORM
COLFORM

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. GETSFMT

At this entry point the program initializes itself to read in a previously-stored format. It verifies the name of the format and checks to see if the format is already in the format table. If so, the program returns immediately with the appropriate information.

If the format must be read, the first record of the format is obtained by calling TSGETRG. This record is analyzed to determine if the format is columnar or sequential. The appropriate format tables are then searched for a slot into which the format can be placed and the format is allocated and initialized.

The program then obtains the remaining format records and posts the data obtained into the appropriate locations within the format entry. If any errors are encountered, an appropriate diagnostic message is written to the user and the partial format is freed. After an error, or when the format has been completed, the required information is

updated and the program returns to the caller.

b. PUTSFMT

At this entry point the program initializes itself to write out one of the currently defined formats. It verifies the name of the format and checks to see of the format exists in the format tables. If not, the program terminates with a diagnostic.

If everything is in order, the program constructs the first format record (FORMAT), indicating the format name, type, the intended file name and other descriptive information and writes it to the data set by calling TSPUTRG.

The remaining format data is organized into TITLE, READER and FIELDS records and written to the data SET in the same fashion as the FORMAT record. If any errors are encountered, an appropriate diagnostic message is written to the user and the partially stored format is erased. After an error, or when the format has been completely written out, the required information is posted and the program returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I Language.

2. Suggestions and Techniques

Not Applicable

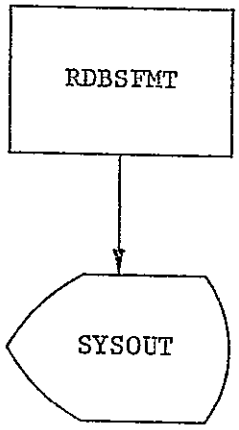


Figure 1. I/O Block Diagram

IV

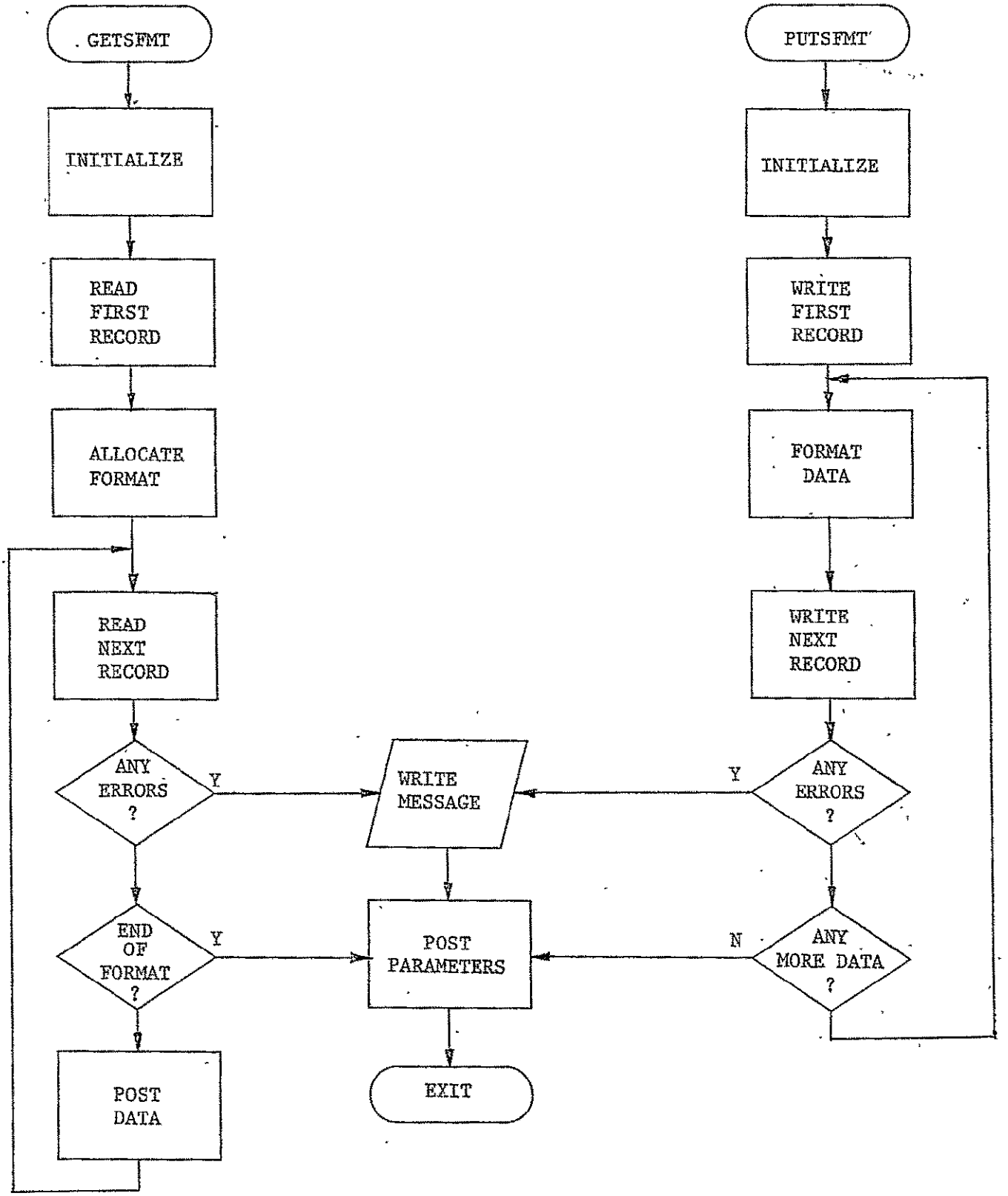


Figure 2. Top Level Flowchart - GETSFMT, PUTSFMT

TOPIC F.12 - GENERIC KEY DISPLAY

A. MODULE NAME

Program-ID - RDBGFLDS
Module-ID - DBGFLDS

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module displays a formatted listing of the names assigned to the sub-levels of the key for a generic key file.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program produces a formatted list of the sub-level names.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program references the following tables to obtain the information which it displays,

FLDTAB
GENERIC

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry the program initializes the screen and other data necessary to construct the display. It extracts the current file name from FLDTAB and uses it to construct the generic table definition routine (XXXXXXX, where xxxxxx is the file name). It calls this routine to obtain the generic key description. If any errors are indicated, a diagnostic message is written to the user and the program is terminated.

The program then extracts each name from the generic key description table and posts is to the screen. When the list is exhausted, the screen is displayed to the user and the program returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and Techniques

Not Applicable

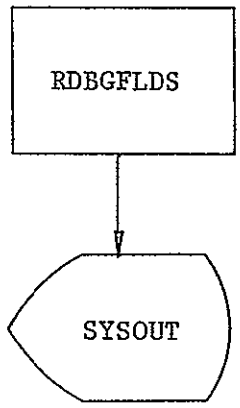


Figure 1. I/O Block Diagram

10 Fn. 11.2

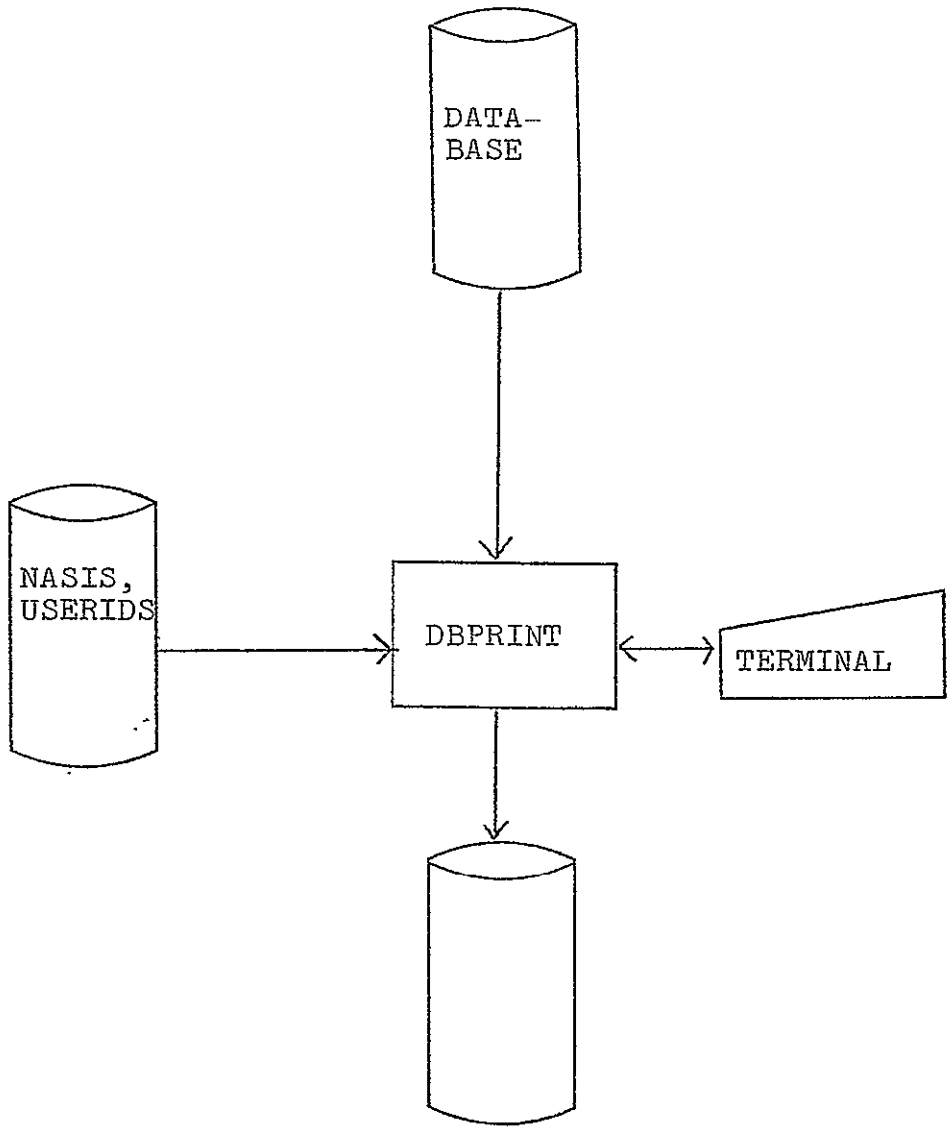


Fig. 1 I/O Block Diagram

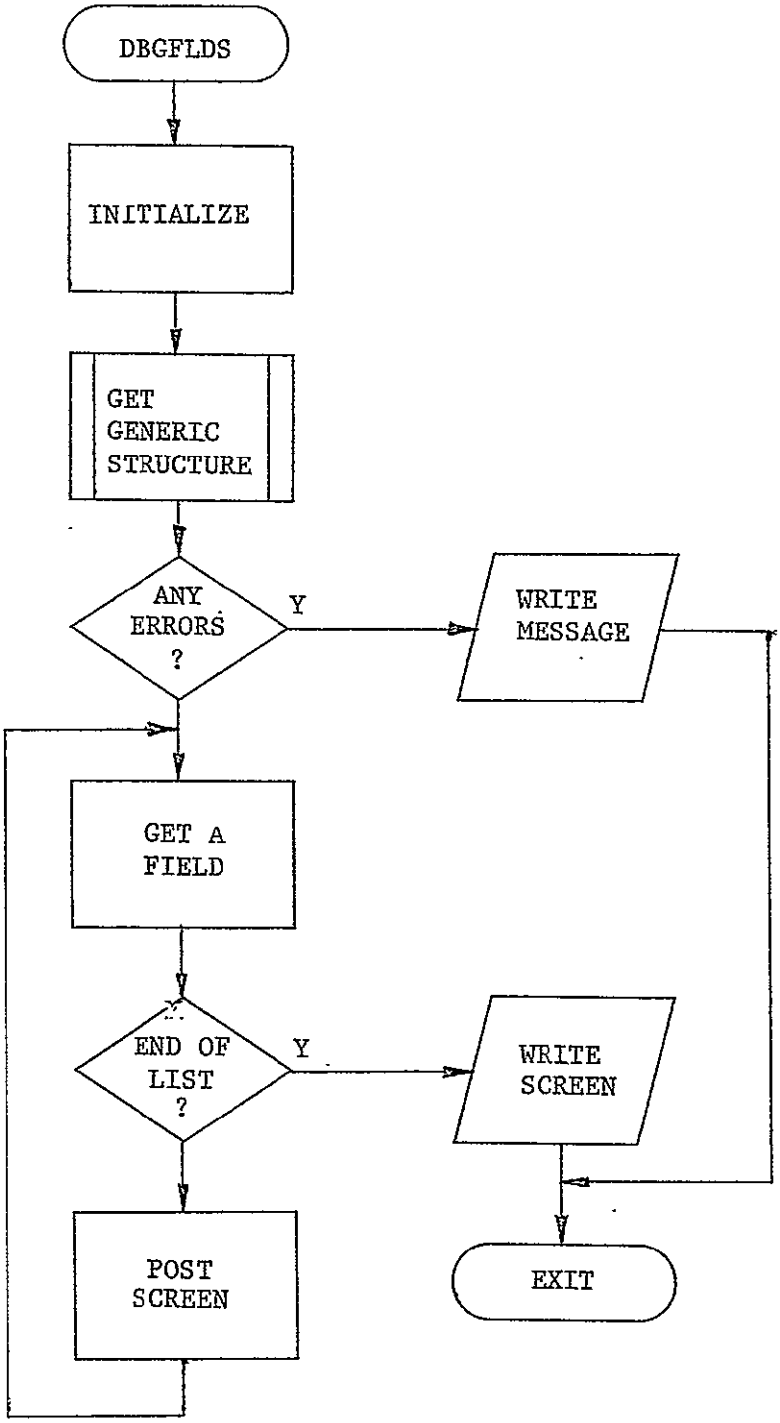


Figure 2. Top Level Flowchart - DBGFLDS

IV F.12.2

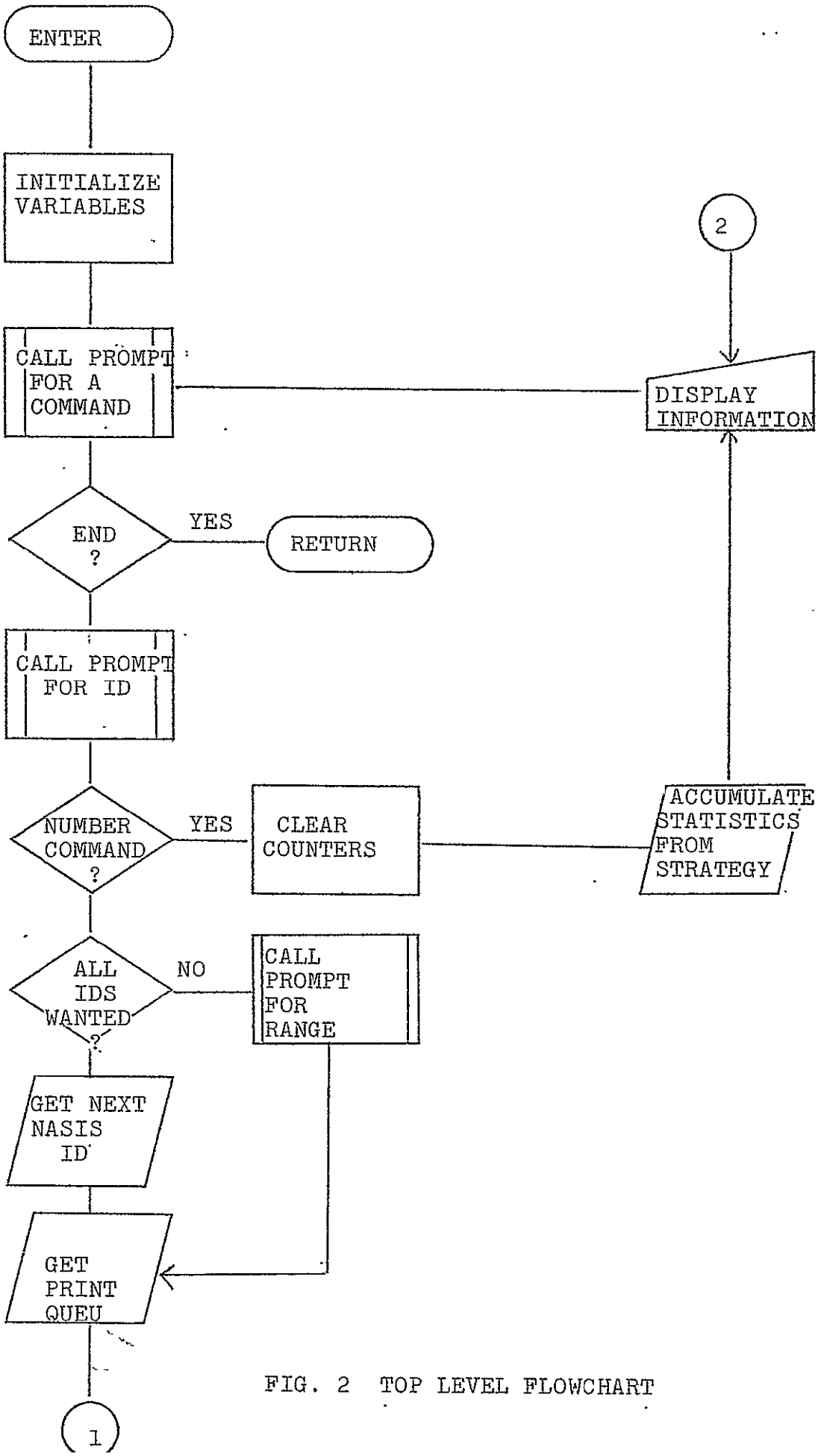
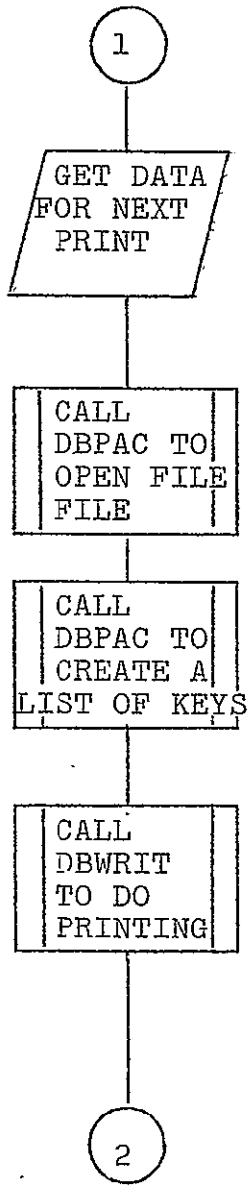


FIG. 2 TOP LEVEL FLOWCHART

IV. E 13:2a


1
GET DATA FOR NEXT PRINT
CALL DBPAC TO OPEN FILE FILE
CALL DBPAC TO CREATE A LIST OF KEYS
CALL DBWRIT TO DO PRINTING
2

IV F, 3.2b

TOPIC - BATCH PRINT MONITOR

A. MODULE NAME

Program - ID - RDBPRINT
Module - II - DBPRINT

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTION

This program controls the execution of the batch print system in much the same way that RDBINIT controls the retrieval system. That is, it initializes file-related tables and issues command prompts to activate batch sub-system operations.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1.

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

NASIS.USERIDS

d. On-line Terminal Entries

The user of the batch print system communicates with the system through a series of command and data prompts. The commands and parameters are:

1. END

Terminate the terminal session

2. PRINT NASISID=,BSN=

Produce a formatted print-out of data from a file utilizing information saved in the print queue for Nasis ID with Batch Sequence Number (BSN) specified.

3. HOLD NASISID=,BSN=

   Place a print job in "hold" status.

4. RELEASE NASISID=,BSN=

   Place a print job in "active" status so that it can be executed.

5. EXHIBIT NASISID=,BSN=

   Display a formatted description of the contents of the batch print queue at the user's terminal.

6. NUMBER NASISID=

   Tally the number of print tasks in the queue.

7. CANCEL NASISID=,BSN=

   Remove a print task from the queue.

8. KEYS NASISID=,BSN=

   Display the file name and record keys recorded for a print task.

9. COPIES NASISID=,BSN=,COPIES=

   Overide the user specified value for number of copies of a printed report.

3. Output Data Sets

   a. Output Files

      Not Applicable

   b. On-line Terminal Displays

      Not Applicable

   c. Formatted Print-outs

      Not Applicable

   d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

See Figure 2.

2. Narrative

DBPRINT gets control from DBMTT, then prompts for one of the commands outlined in section 2D. If the command is PRINT, the information relating to the user's print queue is retrieved from the strategy data set and used to open the file from which data is to be printed. After all initialization is complete, control is passed to DBWRIT to perform the actual data retrieval and printing.

All other commands provide various operations on the user's print queue as described above, except END, which returns control to DBMTT.

F. CODING SPECIFICATIONS

1. Source Language

PL/I

2. Suggestions and Techniques

Not Applicable

TOPIC - BATCH PRINT WRITER

A. MODULE NAME

Program - ID - RDBWRIT
Module - ID - DBWRIT

B. ANALYST

Frank Reed
Neoterics, Inc.

C. MODULE FUNCTION

This program retrieves data from a user - specified data base and prints a listing in either a predefined sequential format or a user-defined sequential or columnar format.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1.

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Any NASIS data base.

d. On-line Terminal Entries

None

3. Output Data Sets

a. Output Files

Print file (PRINTER)

b. On-line Terminal Displays

Not Applicable

c. Formatted Print Outs

User - defined sequential or columnar prints.

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowcharts

See Figure 2.

2. Narrative

DBWRIT gets control from DBPRINT, then opens the PRINTER output file and creates the title page. Next, a record from the data base being retrieved from is read and either sequential or columnar formatting is begun based on a table of field names specified by the user. For sequential formats, the field names and associated data are displayed on successive lines with the field names to the left of the data. Columnar formats require the printing of header and title information (saved by the PRINT and FORMAT functions) along with the field names or other identifier for each column of data across the top of each page. The data for each field is presented under the appropriate column heading until the list of record keys is exhausted.

When all printing of data is completed, a summary of information contained therein is displayed. For sequential prints this is simply a count of the number of records displayed. For columnar prints, this can be, optionally, a tally, sum, and average of the numerical values of items occurring in one or more of the columns.

After closing the PRINTER file, control is returned to DBPRINT with a return code of 'X' for a print terminated by the operator of '0' for a print terminated by a data base error. The return code is unchanged if the print completes successfully.

F. CODING SPECIFICATIONS

1. Source Language

PL/I

2. Suggestions and Techniques

Not Applicable

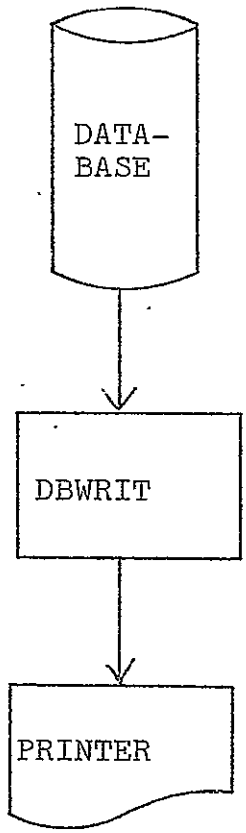


FIGURE 1 I/O BLOCK DIAGRAM

IV. F. 4.1

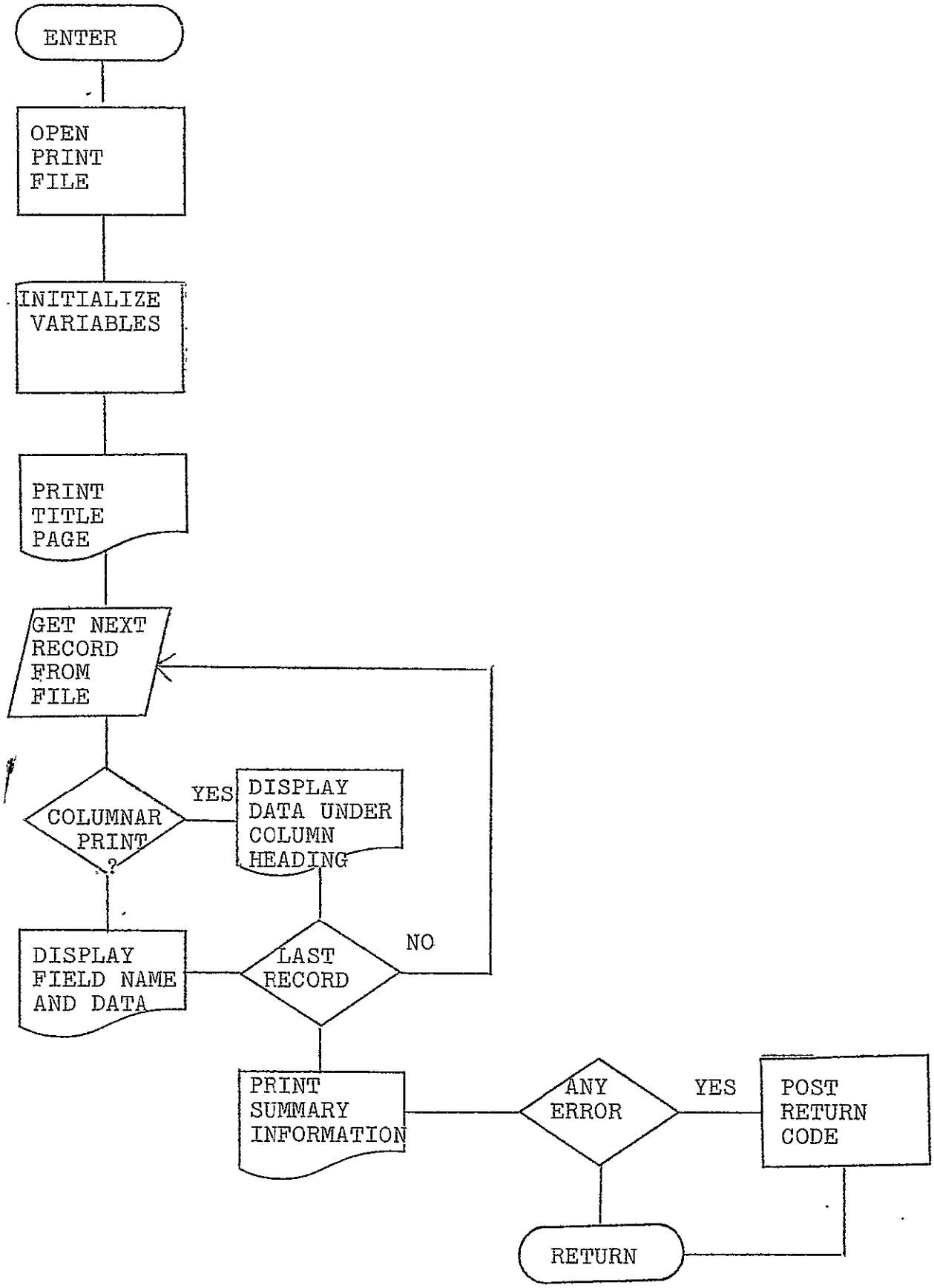


FIGURE 2 TOP LEVEL FLOWCHART

TJ F.14.2

TOPIC F.15 - LIMIT Command

A. MODULE NAME

Retrieval LIMIT Command
Program-ID - RDBLMT
Module-ID - DBLMT

B. ANALYST

Barry G. Hazlett
Neoterics, Inc.

C. MODULE FUNCTION

This module limits an existing set of anchor file keys according to the specified criteria thereby creating a new set.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The new set created by the LIMIT command is displayed on the output screen through use of

the routine DBPSET. Refer to the dataset specification section of the DWB for a writeup of this display.

c. Formatted Print-Outs

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry into DBLMT, of the LIMIT structure has not been allocated, a module name is derived by concatenating "L" to the data base name. If the module does not exist, the user is given a diagnostic and control is returned to the calling routine.

After determining a valid LIMIT structure exists, the user is prompted for the set to be limited. To be valid the set must exist and must consist of anchor file keys. If the set number is invalid, the user is given a diagnostic and prompted for a new set number.

After obtaining a valid set number, the user is prompted for a list of limits to apply against the set. To be valid the specified fieldname must be present in the LIMIT table and the values must be less than 51 characters long and the two values must be separated by a colon. If the limit is invalid the user is given a diagnostic and reprompted for the limit. If the limit is valid, a flag is set in LIMIT indicating which subfield to test along with the two values indicating the value range. If more limits are in the input streem, they are prompted for and processed as above.

Once all of the limit criteria have been established, a control loop is setup to obtain the keys one by one from the input set. Each subfield to be tested is extracted from the key and compared against the acceptable values for this field. If a key fails any of the specified tests, it is ignored and the next key from the list is obtained and processed as above. If the key is acceptable, it is posted in a new list.

After all the keys in the input list have been processed, the new set is posted in SETAB and the results posted to the user screen through use of the routine DBPSET, after which control is returned to the calling program.

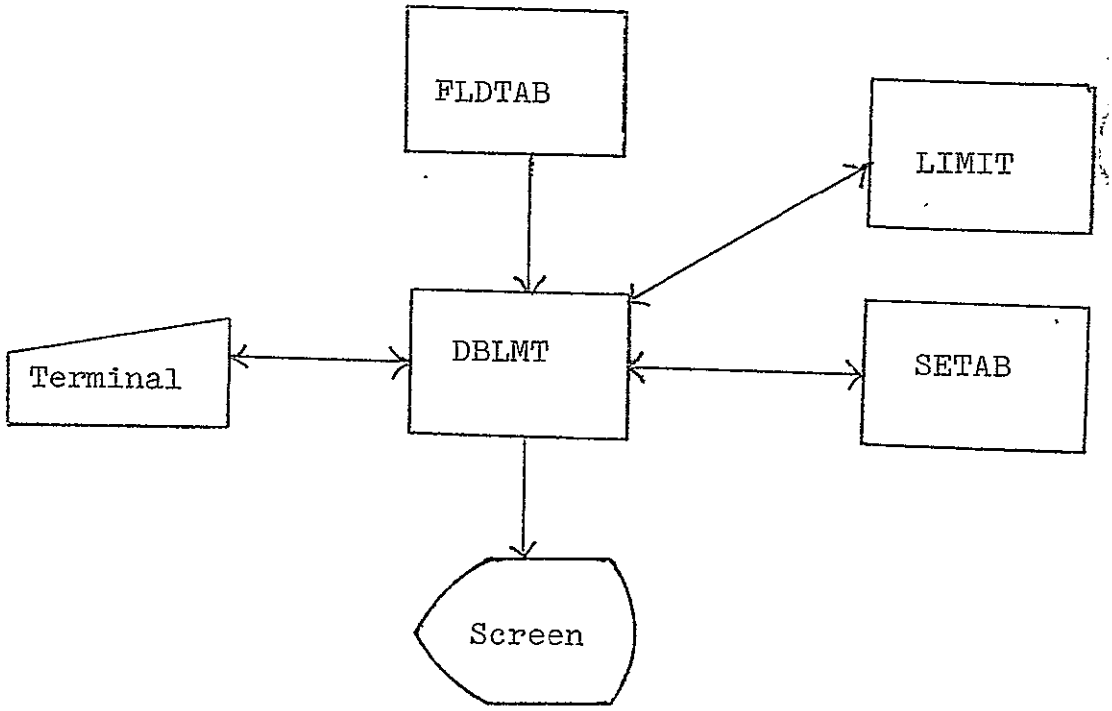
F. CODING SPECIFICATIONS

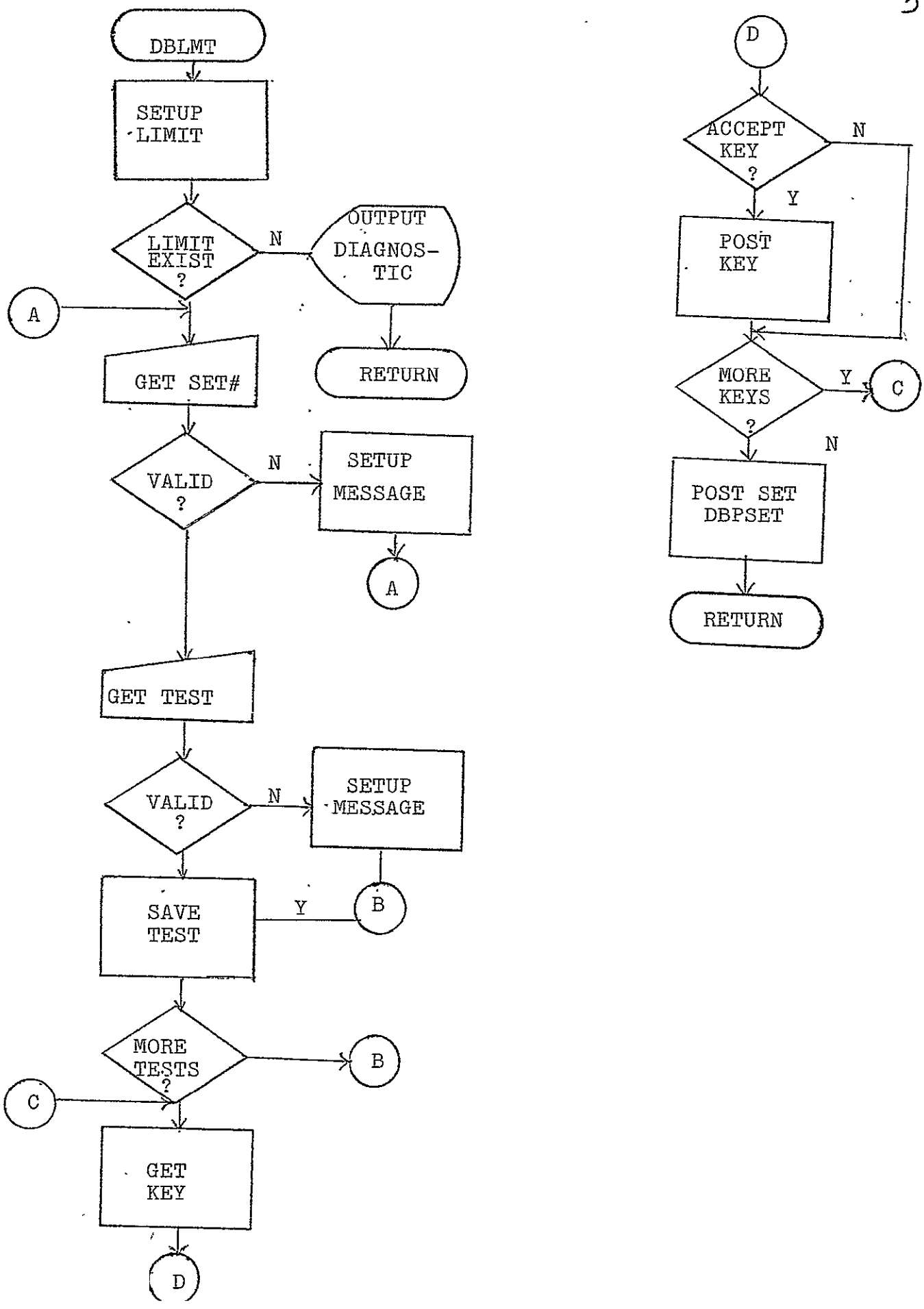
1. Source Language

IBM/360 PL/I Language

2. Suggestions and Techniques

Not Applicable


FLDTAB
LIMIT
Terminal
DBLMT
SETAB
Screen


DBLMT
SETUP LIMIT
LIMIT EXIST ?
N
OUTPUT DIAGNOS-TIC
RETURN
A
GET SET#
VALID ?
N
SETUP MESSAGE
A
GET TEST
VALID ?
N
SETUP MESSAGE
B
SAVE TEST
Y
MORE TESTS ?
B
C
GET KEY
D
D
ACCEPT KEY ?
N
Y
POST KEY
MORE KEYS ?
Y
C
N
POST SET DBPSET
RETURN

TOPIC G.1 - ACCUMULATION

A. MODULE NAME

Statistics Accumulator
Program-ID - RDBACCUM
Module-ID - DBACCUM

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

Primarily, this module is used to accumulate the maintenance statistics on those data bases which have already been loaded.

This program reads an existing data base anchor file and accumulates the number of records on it. Then, it posts this record count to the STATIC data base.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The data base which is to have the statistics accumulated, and the STATIC dataplex.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

   a. Output Files

      The STATIC Dataplex

   b. On-line Terminal Displays

      Not Applicable

   c. Formatted Print-outs

      Not Applicable

4. Reference Tables

   Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

   See Figure 2

2. Narrative

   Error Message:

      ERROR ON $01 OF $02.

      Where:
         $01 is the ONFILE,
         $02 is the ONCODE.

   The program will accept the data base name as a parameter and will proceed to count the anchor files records. When this task is completed, it will open the STATIC data base for update and post the record count.

   The posting of the STATIC data base assumes that no record for this data base currently exists. Therefore, if an error occurs on the LOCATE Statement for the posting, the job is terminated. The key's value for the locate statement is as follows:

      A value of '0' concatenated to the data base name and filled with $'s to 24 characters.

      The field 'ANCOUNT' is posted with the number of anchor records.

The field 'MAINDATE(1)' is posted with the jobs run date, i.e., this is assumed to be the creation date for statistics.

The field 'TOTAL RUN' is posted with a '1'.

The field 'TRANCNEW' is posted with the number of anchor records.

The following fields are posted to '0': 'TOTALTRN', 'TRANCDEL', 'TRANCUPD', 'TRINVNEW', 'TRINVDEL', and 'TRINVUPL'.

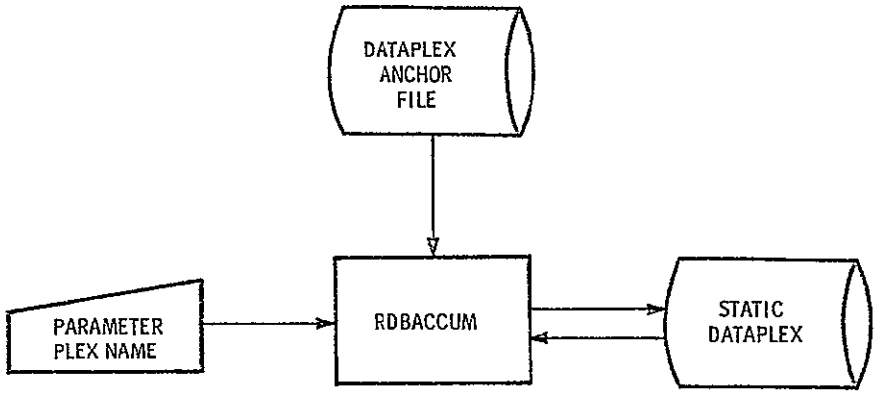
F. CODING SPEICIFCATIONS

1. Source Language

The RDBACCUM module is coded in the IBM programming language PL/I. The DBPL/I and TSPL/I language extensions are used for data base access and terminal I/O, respectively.

2. Suggestions and Technigues

It is important to remember that the executive error '99' indicates an end of file condition. Special attention is made for the handling of the data base executive errors.


DATAPLEX
ANCHOR
FILE
PARAMETER
PLEX NAME
RDBACCUM
STATIC
DATAPLEX

IR 1.1

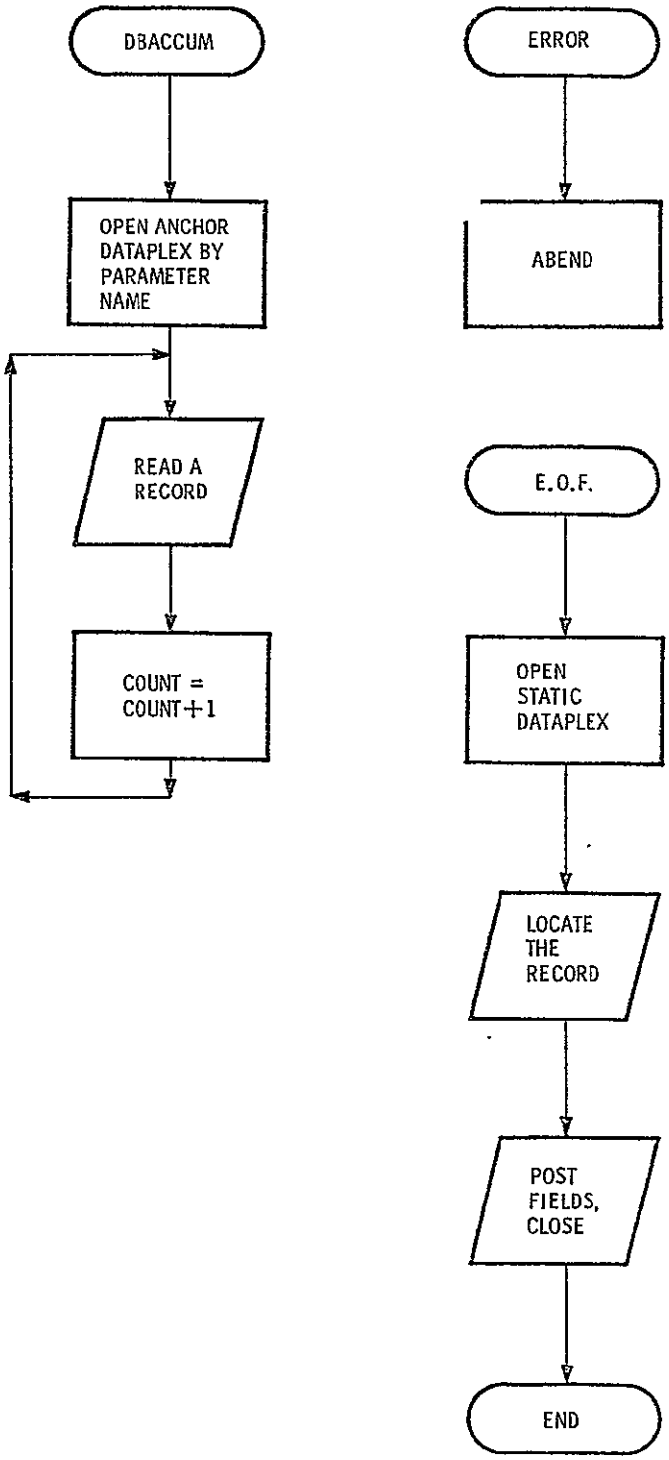


Figure 2. Top level flowchart

TOPIC G.2 - REPORT PRINT

A. MODULE NAME

Print the Retrieval Statistics
Program-ID - RDMPRNTR
Module-ID - DBPRNTR

B. ANALYST

Edward J. Scheboth, Jr.
James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

The purpose of this program is to present a detailed listing of the contents of the STATIC data base pertaining to retrieval statistics.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The STATIC data base, (for full details on this data base see Section III of the Development Workbook).

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

The retrieval statistics' report, (for full details of this report (listing) see Section III of the Development Workbook).

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Flowchart

See Figure 2

2. Narrative

This module performs the following logic in order to produce the retrieval statistics' report

a. Open the STATIC data base for sequential input (use DBPL/I).

b. Read the STATIC file sequentially record by record and while reading, construct from the current information on the STATIC data base the required listing.

c. Output the print file required to produce the retrieval statistics' report.

d. Close all files: Terminate.

Note: It will be necessary for this program to accumulate various information so that it can output the summary of retrieval statistics, representing all of the statistics on the STATIC data base.

F. CODING SPECIFICATIONS

1. Source Language

The RDBPRNTR module is coded in the IBM programming language PL/I. The DBPL/I and TSPL/I language extensions are used for data base access and terminal I/O, respectively.

2. Suggestions and Techniques

Refer to Section III of the Development Workbook for all data set specifications and all data base executive errors.

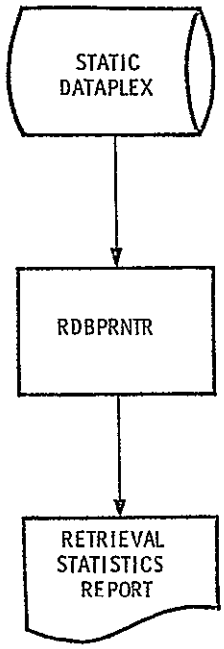


Figure 1. I/O Block diagram

TVGR.1

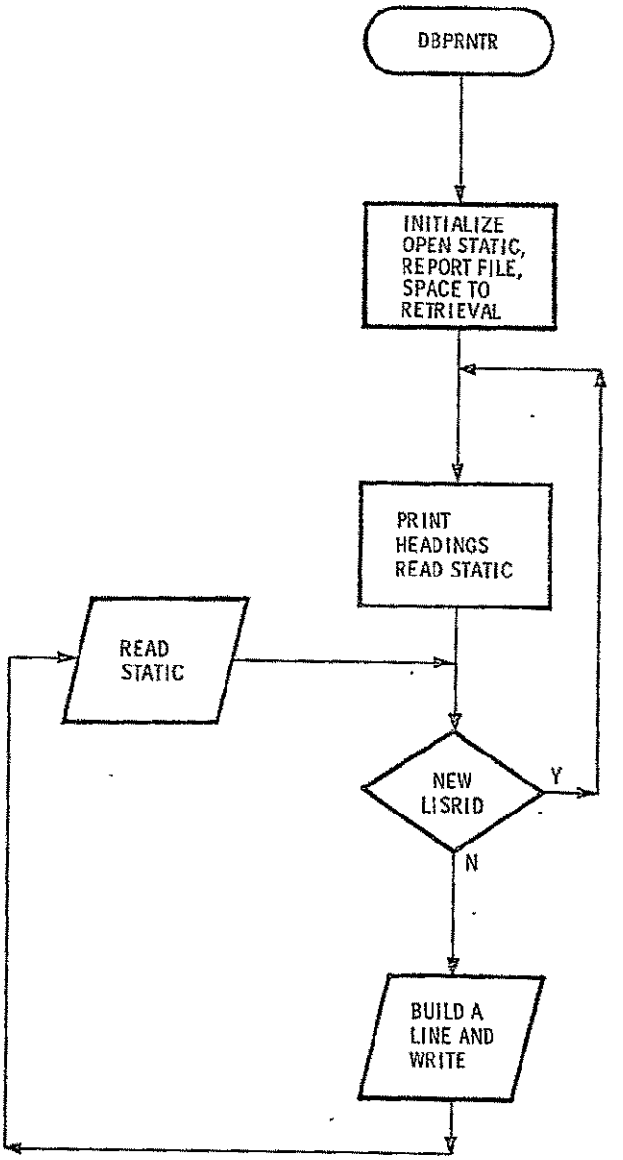


Figure 2. Top level flowchart

TOPIC G.3 - USAGE STATISTICS UPDATE

A. MODULE NAME

Update Maintenance Statistics
Program-ID - RDBUPDST
Module-ID - DBUPDSI

B. ANALYST

Edward J. Scheboth, Jr.
James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

This program updates the statistics data base (STATIC) with the maintenance statistics from the load/create program (RDBLOAD) or from the maintenance mainline (RDBMNTN).

D. DATA REQUIREMENTS

1. I/O Blcck Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The STAIIC Dataplex

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Cutput Files

The STAIIC Data Base

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The parameters are passed via standard PL/I procedure/procedure linkage key calls from RDBMNTN and RDBLOAD.

The parameters which are passed are as follows:

a. Calling program identifier character 2.
First Character
C = first call.
M = subsequent call.
Second character.
L = called from LOAD.
anything else signifies - called from elsewhere.

b. File being updated.

c. Number of new anchor records, character 6.

d. Number of deleted anchor records, character 6.

e. Number of updated anchor records, character 6.

f. Number of new subfield records, character 6.

g. Number of deleted subfile records, character 6.

h. Number of updated subfile records, character 6.

i. Number of new inverted records, character 6.

j. Number of deleted inverted records, character 6.

k. Number of updated inverted records, character 6.

The load/create module (RDBLOAD) invokes this module only once, and this is at the end of the create run. Therefore, this module opens the STATIC data base for direct (update or output) and locates the new record. The data is put and the file closed.

The maintenance mainline (RDBMNTN) is calling the module continuously while processing (this is to preclude the possibility of a system crash causing a loss of statistics). Therefore, upon the first invocation from the maintenance mainline, the STATIC data base is opened for direct update. The proper record is read and written, and control is returned to the maintenance mainline.

The final call from the maintenance mainline will have an 'F' posted to the calling program identifier.

If the updating of the STATIC data base is successful, a 'G' is posted to the calling program identifier upon return; whereas, if the results are not successful, a 'B' is posted.

If the results of the attempted posting are bad, the calling programs will resolve the disposition of the non-posted data.

The details of the contents of the STATIC data base can be found in Section III of the Development Workbook.

The following illustrates the parameters passed and the associated fields which are updated; they are in the form "parameter - static field name":

a. Maintenance date - MAINDATE

b. Number of new anchor records - TRANCNEW

c. Number of deleted - TRANCDEL

d. Number of update - TRANCUPD

e. Number of new subfile records - TRSUBNEW

f. Number of deleted - TRSUBDEL

g. Number of updated - TRSUBUPD

h. Number of new inverted records - TRINVNEW

i. Number of deleted - TRINVDEL

j. Number of update - TRINVUPD.

k. Calling program identifier - *-none-*

It is important to remember that there is a one for one correspondence between all of the previously mentioned STATIC data base fields. For Example:

> If MAINDATE = '03/16/70' and this is the actual date of the maintenance run, then if the MAINDATE value of '03/16/70' is the third element in the variable length field, then all updates to the other elemental fields of the record are made to the third element.
>
> The table which follows will help to illustrate this more clearly.

MAINDATE	01/16/70	02/16/70	03/16/70	null
TRANCNEW	9	3	1	
TRANCDEL	18	4	1	
TRANCUPD	3	7	1	
TRSUBNEW	7	9	1	
TRSUBDEL	3	12	6	
TRSUBUPD	1	9	2	
TRINVNEW	16	3	1	
TRINVDEL	4	4	1	
TRINVUPD	12	7	1	

The fields we are concerned with are: MAINDATE, TRANCNEW, TRANCDEL, TRSUBNEW, TRSUBDEL, TRSUBUPD, TRANCUPD, TRINVNEW, TRINVDEL, TRINVUPD.

These fields are all variable length fields with multiple fixed length elements. The maximum number of elements is 13. The first element in the array is used as an accumulator. Elements 2 through 13 are used to represent individual maintenance runs.

This is simple enough--when this module is called

from RDBMNTN, it simply locates the maindate which is the same as the parameter and does the posting into that given element.

The question is what does this module do when it's called for the first time from the maintenance program (RDBMNTN) and the date is not equal to any of the posted dates and all 13 elements have data so that there is no additional elemental slot where the data can be placed.

The solution is as follows: First, the second elemental slot is 'REPUT' to null. Which causes the data base executive to automatically slide all of the other elements (logically). Then, the new maintenance data will be 'PUT' as the thirteenth element.

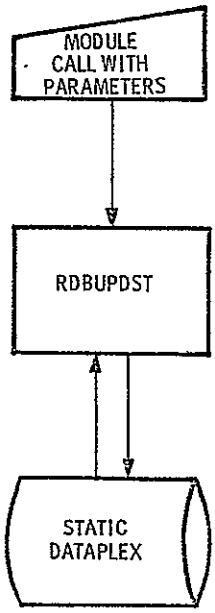
F. CODING SPECIFICATIONS

1. Source Language

The RDBUPDST module is coded in the IBM programming language PL/I. The DBPL/I and TSPL/I language extensions are used for data base access and terminal I/O, respectively.

2. Suggestions and Techniques

Refer to Section III of the Development Workbook for all data set specifications and all data base executive errors.


MODULE
CALL WITH
PARAMETERS
RDBUPDST
STATIC
DATAPLEX

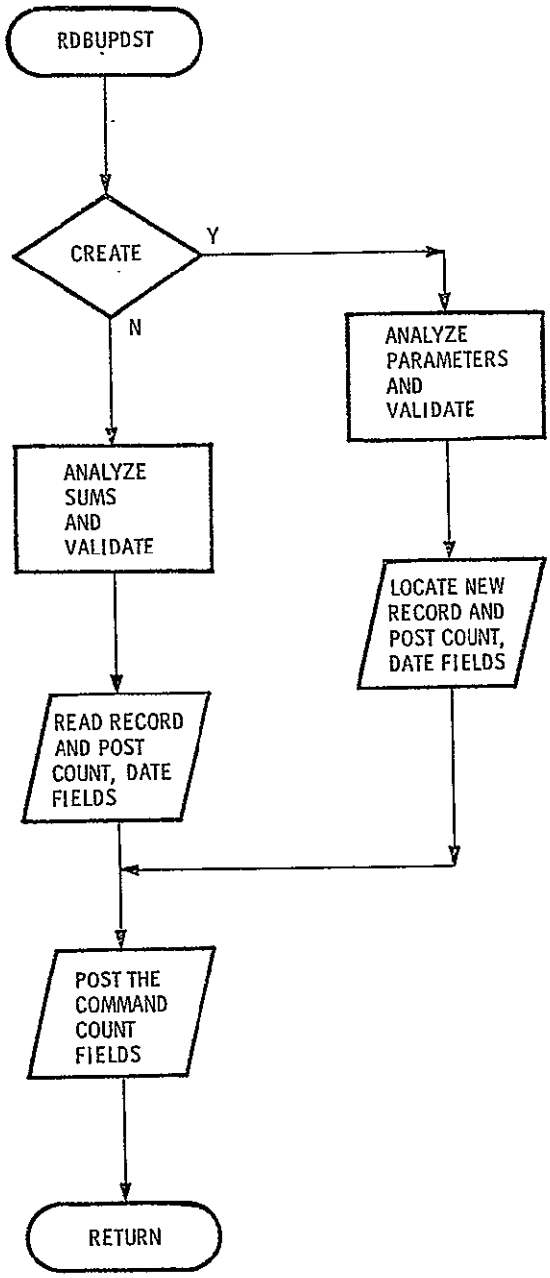
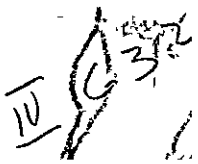


Figure 2. Top level flowchart



TOPIC G.4 - CLOCK ROUTINES

A. MODULE NAME

Clock Routines
Program-ID - RTIMERS
Module-ID - RTIMERS

B. ANALYST

Edward J. Scheboth, Jr.
Neoterics, Inc.

C. MODULE FUNCTION

This module initializes two TSS clocks, one for CPU time and the other for CONNECT time. These clocks may be read at a later time to provide the elapsed time plus initial values.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1

2. Narrative

In the START entry, the initial values are assigned to the total clock value and an even odd pair of clocks are started even (0) with task option ODD(1) with real option and two counters are set with these values.

In the READ entry, a flag is set to on at entry. The clocks are read and the initialized totals are updated. The clocks are stopped and restarted to prevent expiration, the values are provided to caller the C-1 pair of clocks started, the indicator turned off and return made to caller.

In the STOP entry, the two counters of clock numbers are deducted by 2 and each pair of active clocks stopped.

If either clock should expire, the expiration routing post full values to total and starts a new clock with value +2 and returns.

F. CODING SPECIFICATIONS

1. Source Language

Assembler

2. Suggestions and Techniques

Not Applicable

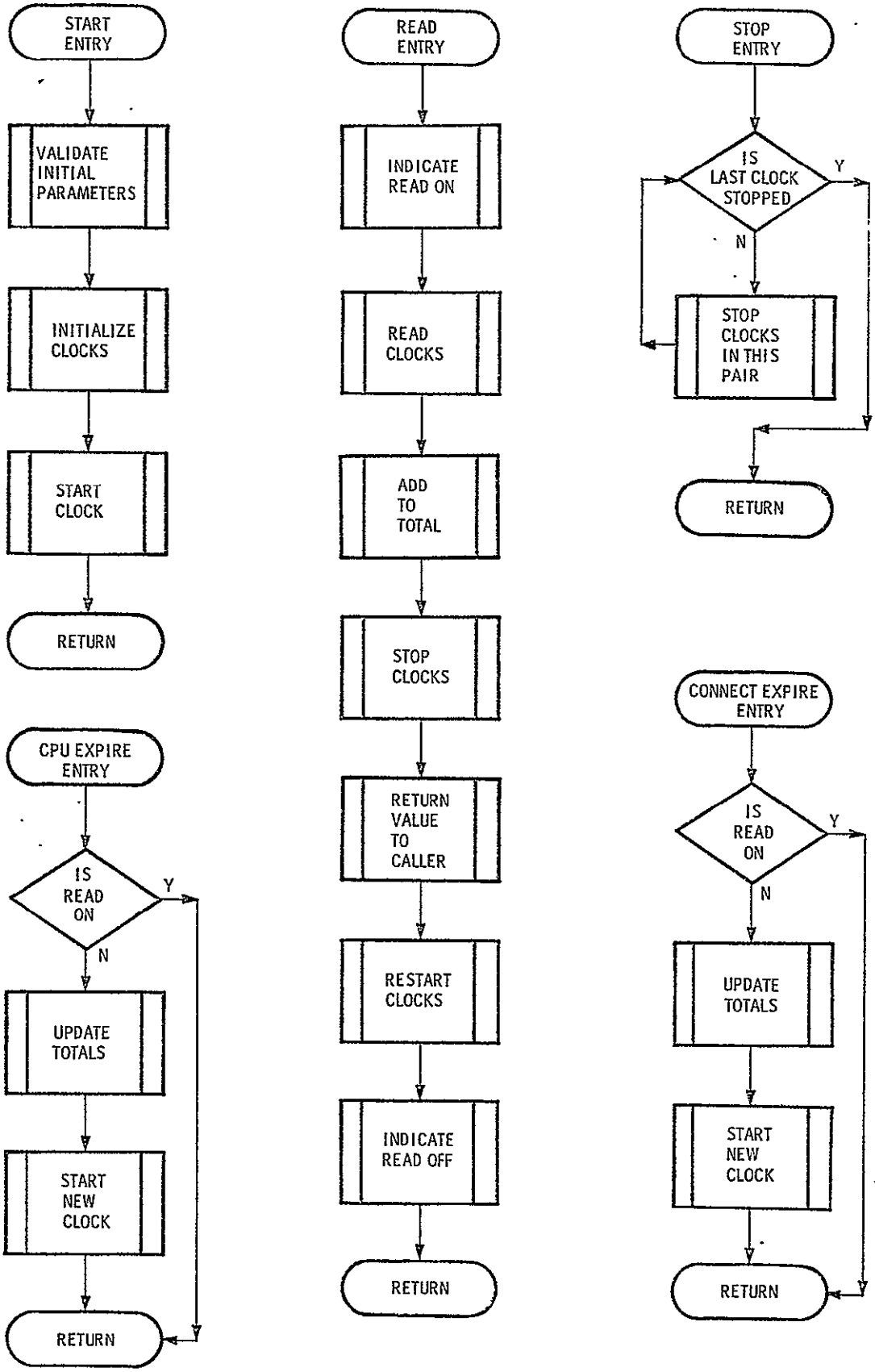


Figure 1. Top level flowchart

IV GAT

TOPIC G.5 - STATIC REPORT

A. MODULE NAME

Maintenance Statistics' Report
Program-ID - RDBPRNTM
Module-ID - DBPRNTM

B. ANALYST

Edward J. Scheboth, Jr.
James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

This program opens and reads the STATIC data base (sequential input); analyzing, accumulating and formatting (for printing) the maintenance statistics' information which is currently posted. The end result is a maintenance statistics' report. It has the added function of snapshot dump and re-initializing the seven variable element fields which are the running totals of the maintenance statistics.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

The STATIC data base (for detailed and complete information on this data base refer to Section III of the Development Workbook).

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

The maintenance statistics report (for complete detailed information on this listing refer to Section III of the Development Workbook).

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

This module performs the following logic in order to produce the maintenance statistics' report:

a. Opens the STATIC data base for sequential input (use DBPL/I).

b. Read the STATIC file sequentially, record by record, and while reading constructs from the current information on the STATIC data base, the required listing.

c. Outputs the print file required to produce the maintenance statistics' report.

d. Snapshots the ten variable element fields if they are full.

e. Close All Files: Terminate.

Note: It is necessary for this program to accumulate various information so that it can output the summary of maintenance statistics.

F. CODING SPECIFICATIONS

1. Source Language

The RDBPRNTM module is coded in the IBM programming language PL/I. The DBPL/I and TSPL/I language extensions are used for data base access and terminal I/O, respectively.

2. Suggestions and Techniques

Refer to Section III of the Development Workbook for all data set specifications and all data base executive errors.

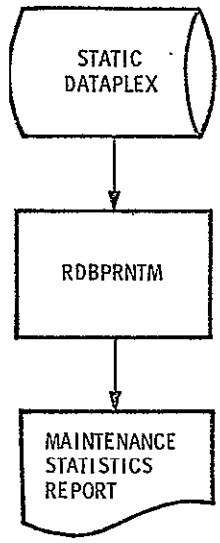


Figure 1. I/O Block diagram

١٠ كم

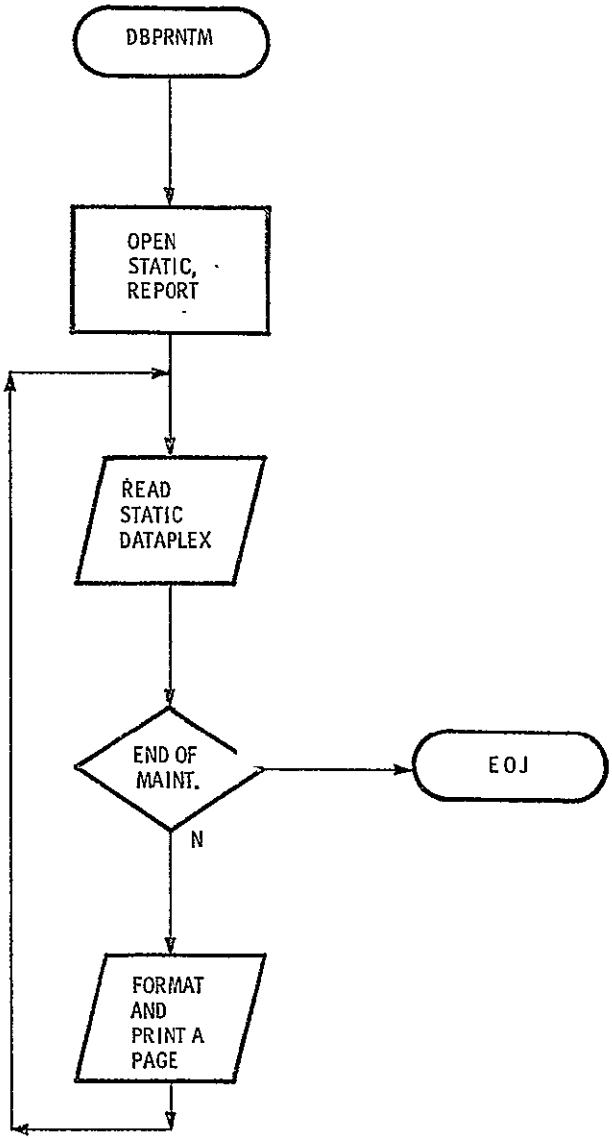
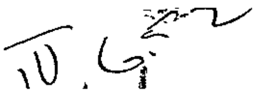


Figure 2. Top level flowchart



TOPIC G.6 - RETRIEVAL STATISTICS DIRECTOR

A. MODULE NAME

Retrieval Statistics
Program-ID - RDBSTAT
Module-ID - DBSTAT

B. ANALYST

James A. Wesley
Neoterics, Inc.

C. MODULE FUNCTION

This module is the heart of the retrieval statistics. It has an entry point for each retrieval module included in the statistics.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

The Static data base.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print-outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

'FLDTAB' is used to convert inverted indices to data base file names.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

The INIT entry checks to see if there was a crash during the last session by the existence of the ONES record and then write one if there wasn't one or after RDBCHKPT deletes it. INIT initializes statistics that like INIT in the command system setting up the necessary tables or pointers for later use.

Each command entry, one each for EXPAND, SELECT, SEARCH and CORRECT, pushes its information, command type, NASISID OWNERID and fill, into the stack and then checks to see if it is time to update the statistics by checking the command count and entry count for critical level.

The DBSTATF entry call on termination of a session, just indicates that this is to be the end and provides strategy information and branches to the PUTSTAT routine.

The DBSTATD entry deletes this strategy from the statistics if it is there.

The PUTSTAT routine always updates the CPU and connect time by calling the RTIMERS routines for their values. It also always pops the command stack and updates each command count and the set-date for the specified file. The stack is a FIFO stack, a one dimensional structured array. If this is a DBSTATF entry, then the strategy information 'STRATNME', 'STRATSTR,' and 'STRATLEN' and usage information 'LASTDATE' are complete

updated. Finally, for the DBSTATF entry to update the storage allocation is freed and the ones record deleted from STATIC.

F. CODING SPECIFICATIONS

1. Source Language

PL/I and DBPL/I

2. Suggestions and Techniques

Not Applicable

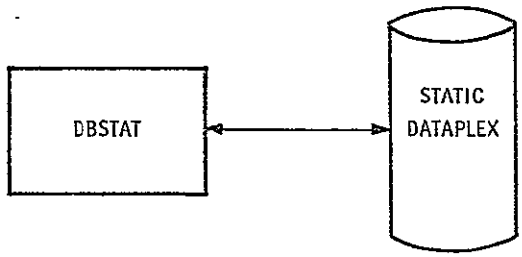


Figure 1. I/O Block diagram

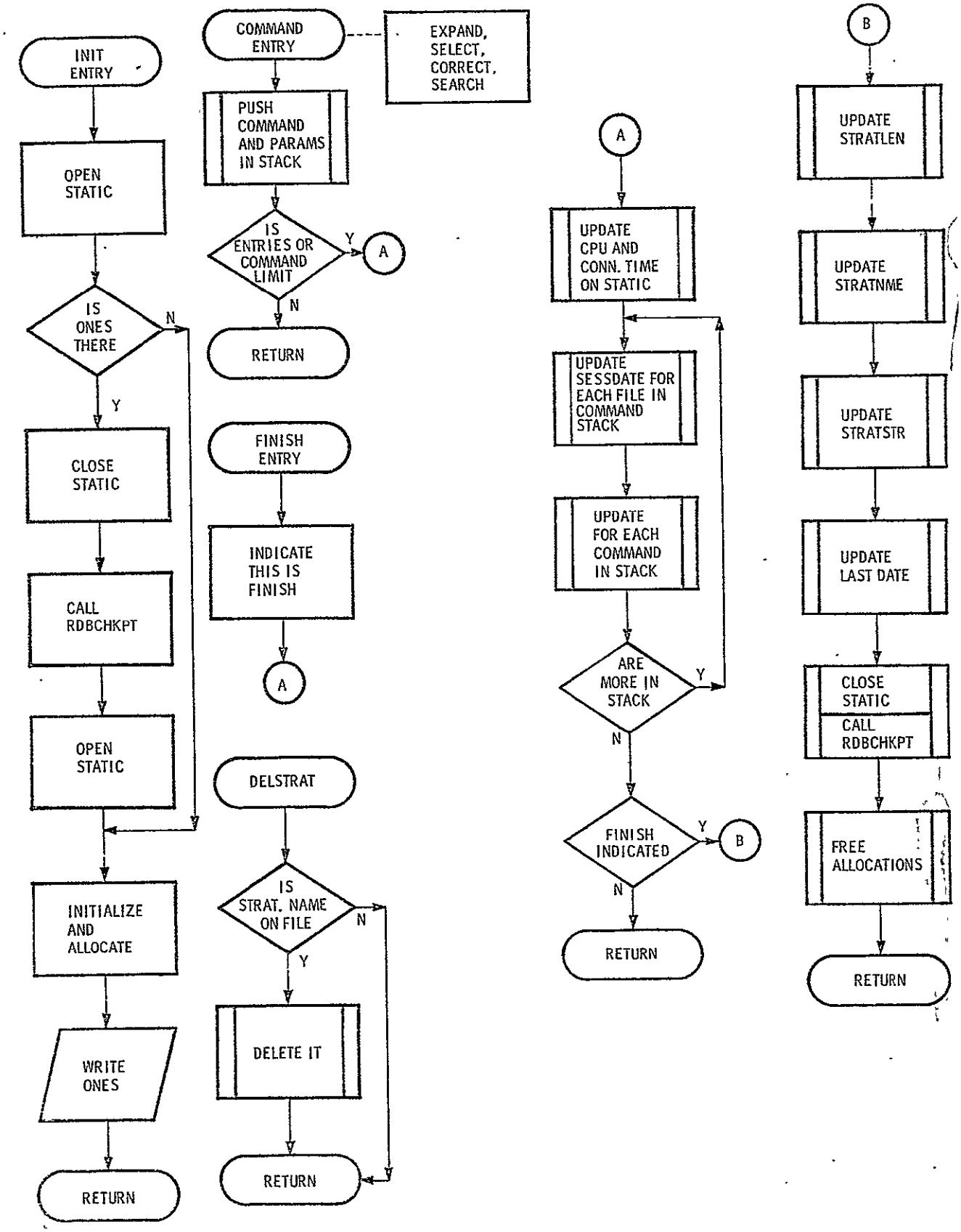
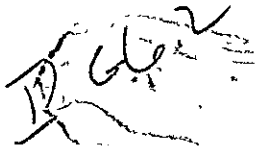


Figure 2. Top level flowchart



TOPIC H.1 - EXPLAIN FACILITY

A. MODULE NAME

Program-ID - RDBEXPL
MODULE-ID - DBEXPL

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module allows the user to display the explanation of a message or term, the origin of a message or the responses to a prompt, that has appeared on the screen, or, the text of any of the standard prompting messages on the message file.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

This module receives its input in the form of parameters passed with the EXPLAIN or PROMPT commands.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

This module displays the requested information for the user on the terminal.

c. Formatted Print-Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Not Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. IBEXPL1

Upon entry, the program initializes the variables that control execution and the displaying of data to the user. It also sets up the mechanism by which paging is to be accomplished.

Next the program prompts for the OPTION and MESSAGE parameters required for the EXPLAIN function. It verifies that the option selected is valid, and if so, branches to the appropriate routine.

For simple explains, i.e., message explanations, the OPTION is treated as an index, verified, and the line number set to a value of 100. If the OPTION is not a valid index, the request is treated as a term explanation. The OPTION is then treated as a qualified term and used to construct the message key which is used to locate the term's explanation. For response explanations the live number is set to a value of 400.

In each of the above instances, control is passed to a routine which attempts to read a

data record. If successful, the record is written to the screen and the process repeated, until the data has been exhausted, or the screen filled. At this time, the paging controls are set, the screen is displayed to the user and the program is terminated. If no data was found, the routine branches to an error routine which displays a message to the user and terminates the program.

If the original request was for a message origin, the OPTION is treated as an index, and if valid, the appropriate message key is obtained, displayed to the user, and the program is terminated.

b. DBEXPL2

At this entry point, the program initializes the varriables that control execution and prompts for the MESSAGE parameter. It then prompts for the INSERTS parameter list.

Once complete, the program attempts to display the message indicated with the specified inserts.

c. DBEXPLP

At this entry point the program re-initializes the variables that control execution and the displaying of data to the user. If the paging status data indicates that more data remains, the program uses this data to restore the proper program status and then branches to the routine which posts data to the screen. If no data remains to be displayed, the program simply terminates.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and Techniques.

Not Applicable.

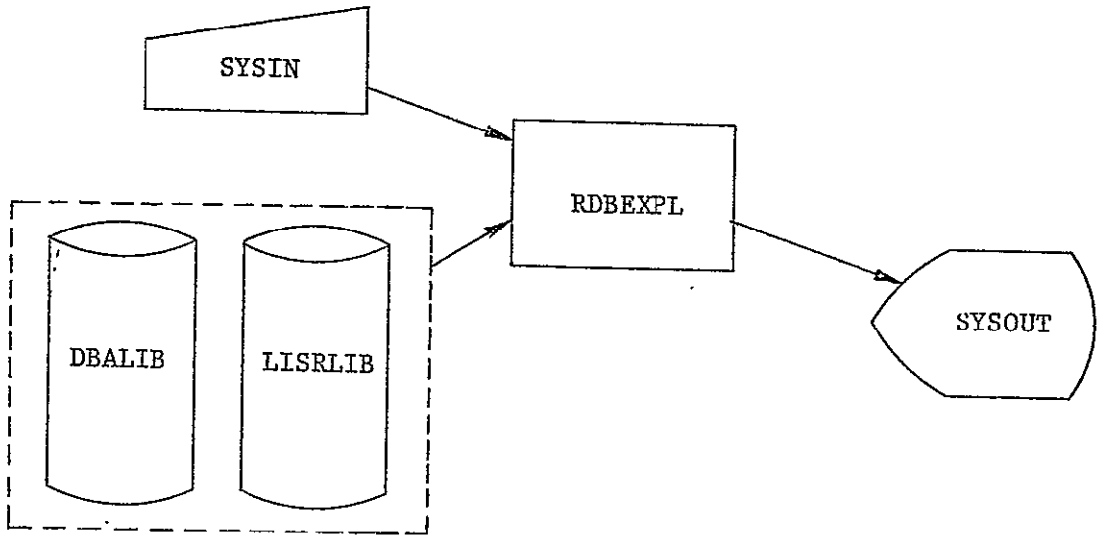


Figure 1. I/O Block Diagram

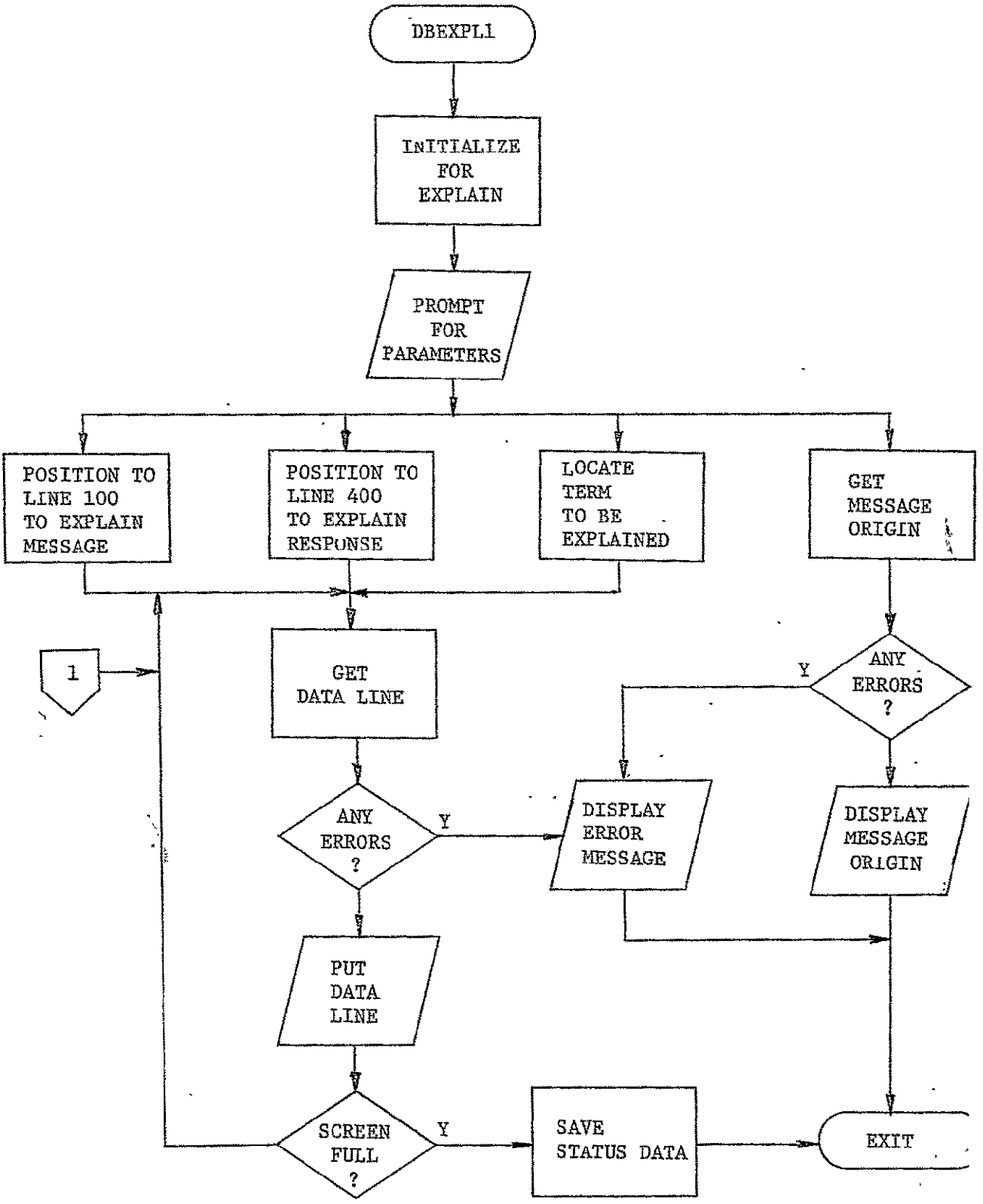


Figure 2A. Top Level Flowchart - DBEXPL1

$\frac{1}{4}$ 1.? (5)

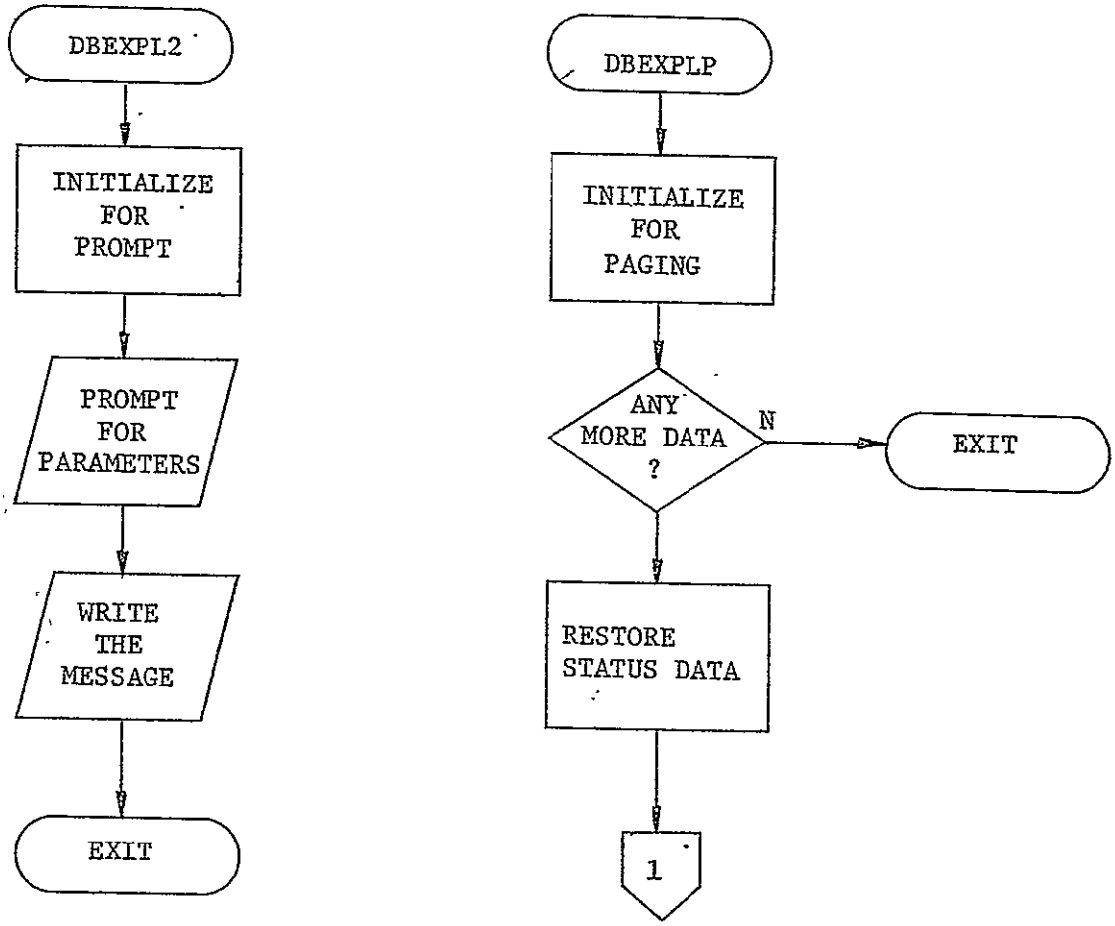


Figure 2B. Top Level Flowchart - DBEXPL2, DBEXPLP

IV H 1.2 (2)

TOPIC H.2 - STRATEGY INTERFACE

A. MODULE NAME

Program-ID - RDBSTFT
Module-ID - DBSTFT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module serves as an interface between the strategy data set service routines and the rest of the NASIS system. In addition, it is the module which performs the functions specified by the FORMAT? and STRATEGY commands, i.e., the listing of format and strategy names, the listing of strategies and the deletion of strategies.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-line Terminal Entries

When serving as the processor for the FORMAT? and STRATEGY commands, the program reads in the command and parameters specified by the user to invoke those commands.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

When serving as the processor for the FORMAT? and STRATEGY commands, the program produces the follcwing formatted screen images,

1. Format names display

2. Strategy names display

3. Strategy display

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

USERTAB-is used to obtain the NASIS-id and to test the task status as represented by the various bit switches.

FLDTAB -is used to reference the formats currently defined for this user.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. GSTRAT and GFORM

At these entry points the program initializes the parameter lists necessary to obtain a line frcm the strategy data set, and calls TSGRTRG to do it. If an error occurs, and it is the first error for that region, a diagnostic message will be written to the user. Ctherwise, the program simply returns to the caller.

b. PSTRAT and PFORM

At these entry points the paragram

initializes the parameter lists necessary to write a line to the strategy data set, including the generation of the strategy or format name. It then calls TSPUTRG to perform the write. If PSTRAT is called and the TESTMODE, RERUN or RESTACT flags are set, the program immediately returns to the caller. If an error occurs while writing out the record, a diagnostic message is written to the user and the TESTMODE switch is turned on. The program then returns to the caller.

c. CSTRAT and CFORM

At these entry points the program initializes the parameter lists necessary to change the name of a region. It then calls TSCM6R6 to accomplish the change. If any errors are encountered, a diagnostic message is written to the user. The program then returns to the caller.

d. DSTRAT and DFORM

At these entry points the program initializes the parameter lists necessary to delete a region of the strategy data set. It then calls ISDELR6 to perform the deletion. If any errors are detected, a diagnostic message is written to the user. The program then returns to the caller.

e. DBSTRT1

At this entry point the program initializes itself to process the strategy command. It reads in the OPTION and STRATEGY parameters. The program then branches to the routine used to process the type of request specified by the OPTION parameter. If that parameter is not valid, the program writes a diagnostic message and terminates immediately.

If the user requested a strategy deletion, the program calls TSDELRG to delete the strategy specified. If an error occurs, a diagnostic message is written to the user. The program then checks for any additional names, and processes each in the same way. When all processing has been completed the program terminates.

If the user requested a listing of the strategy names, the program initializes the screen and paging control data. It then repetitively calls TSGETSN to retrieve the names of the strategies. As each name is obtained, it is added to the output line and the line is written to the screen. When the screen is filled or when the strategies names are exhausted, the screen is displayed to the user, the paging status data is posted and the program is terminated.

If the user requested a listing of a particular strategy, the program initializes the screen and paging control data. The first strategy name specified is selected, and TSGRTRG is repetitively called to obtain the lines comprising the strategy. Each line is posted to the screen. When the screen is filled or when the last line has been written, the screen is displayed to the user, the paging staters data is posted and the program is terminated. The paging status data must indicate when a strategy has been completely listed, so that the next name from the list can be used.

f. DBSTRT2

At this entry point the program initializes itself to display the names of the formats available to the user. It initializes the screen and the paging status data. The program then extracts the identifiers for all of the formats currently specified in the format tables. It then calls TSGETFN to retrieve the name of a stored format. It places the names of the formats on a line and writes the line out to the screen. The names are processed alphabetically, and as each stored format name is processed, a new one is read in. Stored formats that are also present in the format tables are only shown once. When the screen is filled, or when the list of names is exhausted, the screen is displayed to the user, the paging status data is posted and the program is terminated.

g. DBSTRTP

At this entry point the program re-initializes itself to the status saved before the last termination. If more data

remains to be displayed, the program branches to the proper routine to produce the next display screen. If no more data remains, a diagnostic message is written to the user and the program is terminated.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and Techniques

Not Applicable

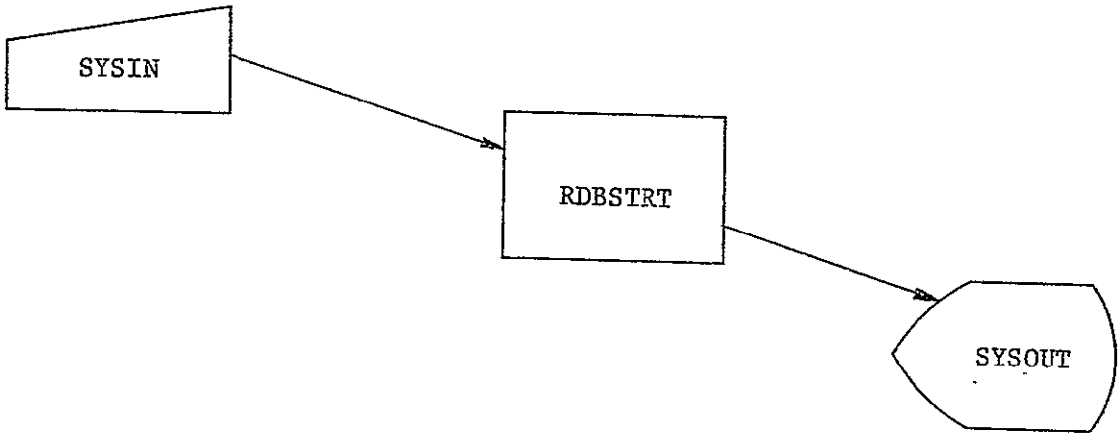


Figure 1. I/O Block Diagram

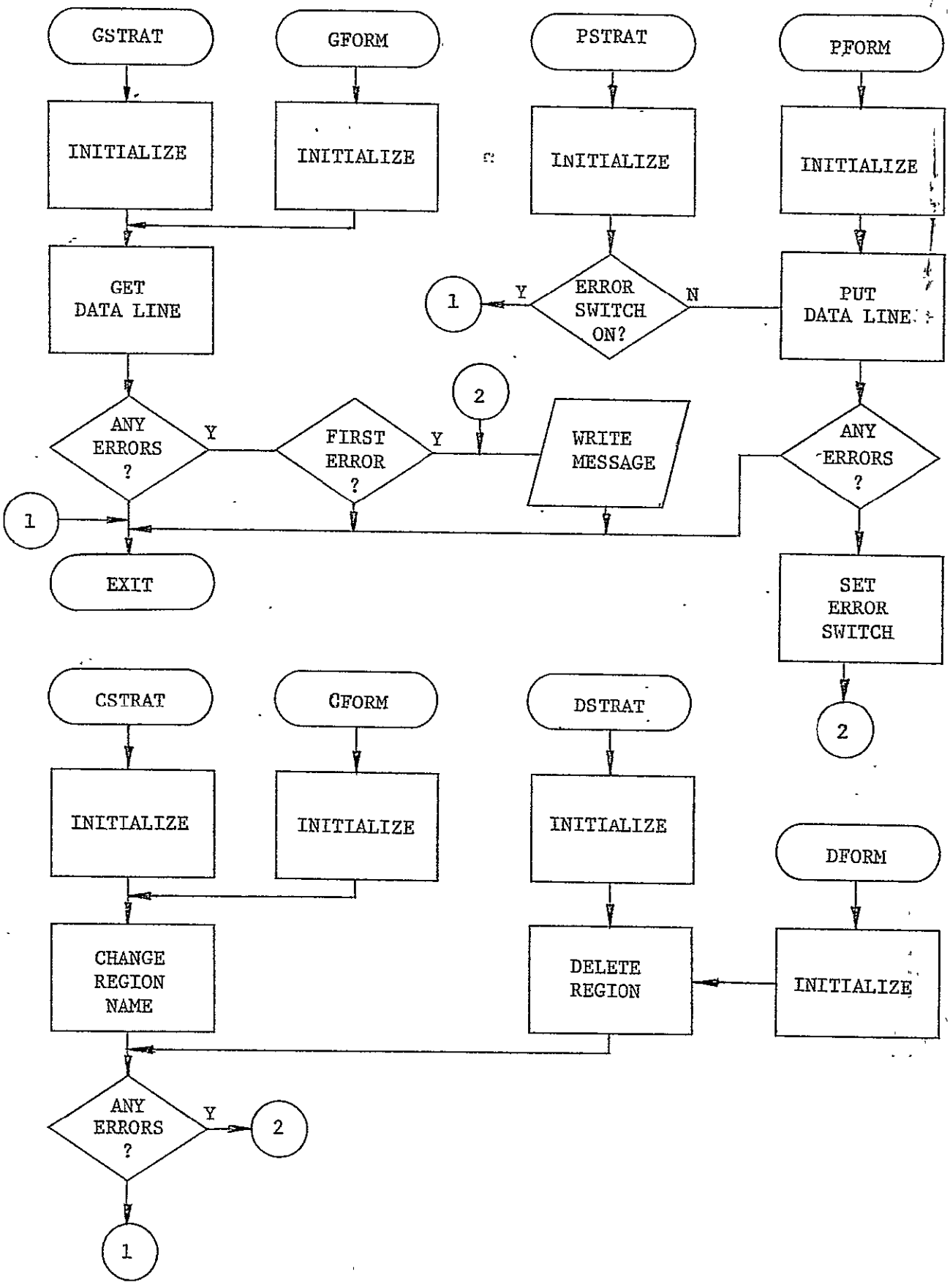


Figure 2A. Top Level Flowchart

비고: [illegible](A)

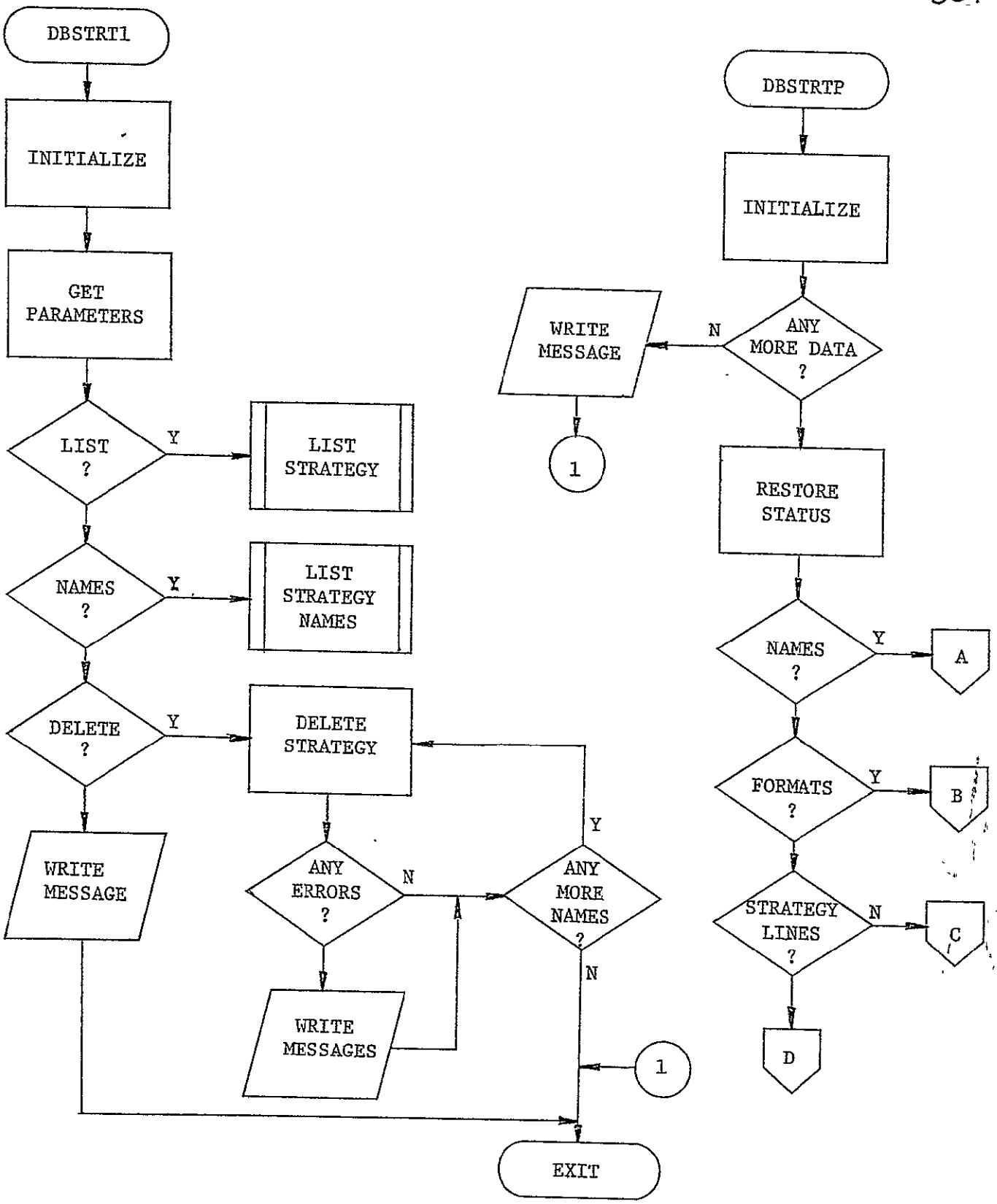


Figure 2B. Top Level Flowchart

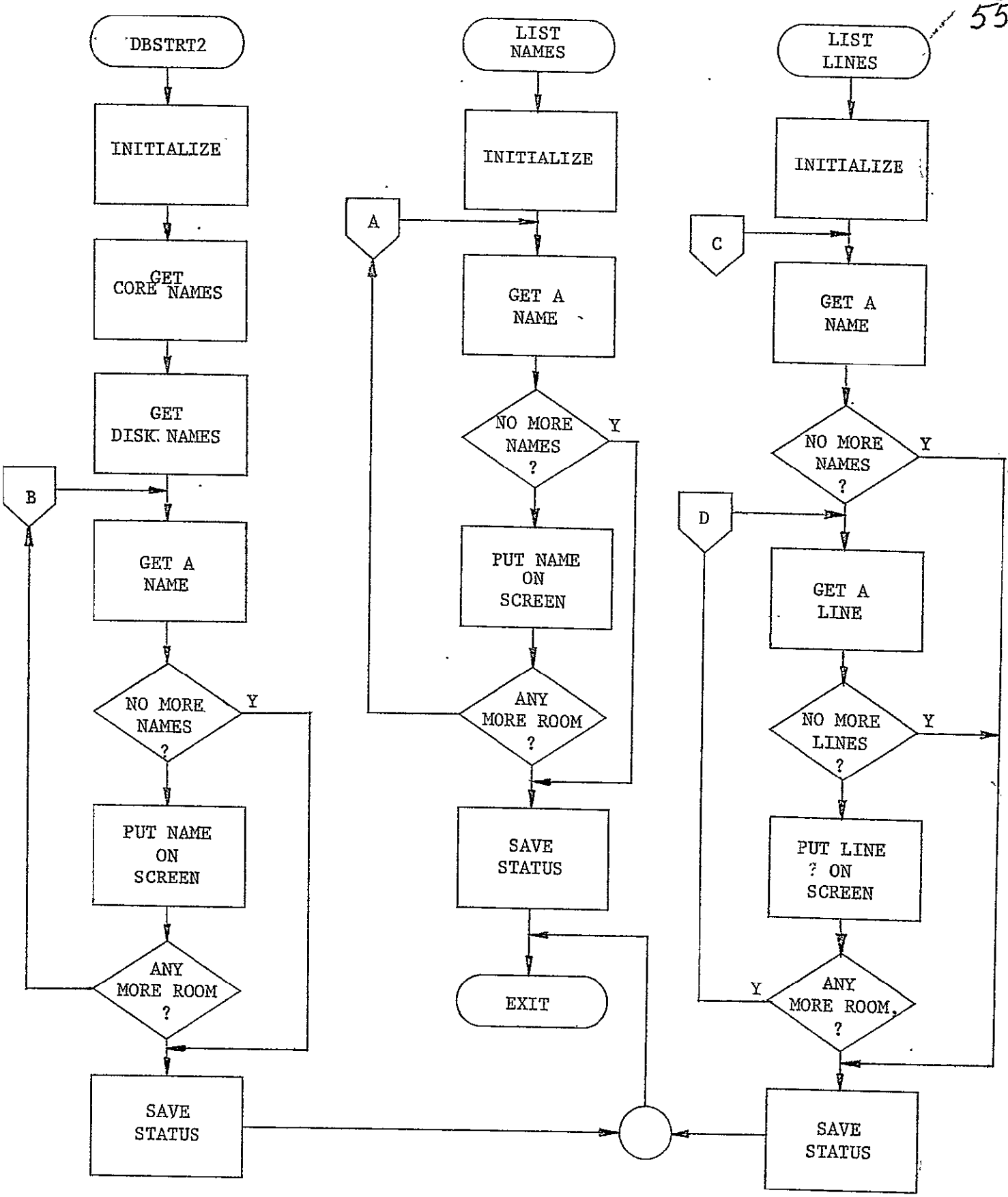


Figure 2C. Top Level Flowchart

TOPIC H.3 - STRATEGY ASSEMBLER ROUTINES

A. MODULE NAME

Program-ID - RTSSTRT
Module-ID - TSSTRT

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

These routines act as the assembler service routines for the strategy library. They permit the retrieval, modification and storing of the saved strategies and formats.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

C. Input Files

Strategy data set - is used for input for both stored strategies and stored formats.

IBALIB-member FORMATS is used for input for stored formats only.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Strategy Data Set-is used for output for both

stored strategies and stored formats.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

USERTAB-is used to obtain the NASIS-ID.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. ISDELRG

At this entry point the program initializes itself to delete a strategy or format region. It opens the strategy data set, if necessary, and extracts the region name passed by the caller. The program then proceeds to delete the region, one line at a time. If any errors are encountered, the region name is set to null. The program then returns to the caller.

b. TSGETRG

At this entry point the program initializes itself to get a line from a strategy or format region. It opens the strategy data set and member FORMATS of DBALIB(0) if necessary. It extracts the parameter passed by the user, and if a null line number is passed, sets up to read the first line of the region. If the high order bit of the line number is off, it sets up to read the line following that indicated by the line number. Otherwise, it positions the pile to the line number passed.

The program then attempts to read the line requested. If successful, it posts the line number, posts the data (with trailing blanks removed) and returns to the caller.

If an error occurs, the program sets the region name to null before returning. Likewise, if an end-of-region occurs, the line number is set to null before returning. If the region cannot be located in the strategy data set, the program checks the region name, and if it is a format request, tries to locate the region in member FORMATS of DBMLIB(0) and then processes the request as indicated above.

c. TSPUTRG

At this entry point the program initializes itself to put a line to a strategy or format region. It opens the strategy data set, if necessary, and extracts the region, live number and data parameters passed by the caller. If the line number is null, it sets up to add the line at the end of the region. In any case, it positions the file to the proper region and live within the region. The program then attempts to write out the new line from the data passed by the caller. If successful the program simply returns to the caller. If an error occurs, the program sets the region name to null before returning.

d. TSCHGRG

At this entry point the program initializes itself to change the name of a strategy or format region. It opens the strategy data set if necessary, and extracts the old and new region names passed by the caller.

The program firsts attempts to delete any existing region with the new region name. If an error occurs, other than region unknown, the program terminates and sets the old region, reads a line, positions itself to the new region and writes out the live. This process is repeated until all of the data lines have been copied. If any errors occur, the new region is deleted, the old region name is set to null and the program returns to the caller. If no errors have occurred,

the program deletes the old region and returns to the caller.

e. TSGETSN

At this entry point, the programs initializes itself to get a strategy region name. It opens the strategy data set, if necessary, and extracts the strategy name passed by the caller. If the name is null, the program sets up to get the first strategy name. Otherwise, it sets up to get the strategy name following that passed by the caller.

The program then attempts to read a line from the strategy data set. If successful, it extracts the region name and passes that back to the caller. If an error occurs, or if an end-of-file is sensed, the region name is set to null and the program returns to the caller.

f. TSGETFN

At this entry point, the program initializes itself to get a format region name. It opens the strategy data set and member FORMATS of DBALIB(0), if necessary, and extracts the region name passed by the caller. If the region name is null, the program sets up to get the first format name. Otherwise, it sets up to get the format name following that passed by the caller.

The program then attempts to read a line from both data sets. If an error occurs, or if both files indicate end-of-file, the region name is set to null and the program returns to the caller. Otherwise, the program compares the region names of the two lines. It posts the name, lowest in value, in the region name and returns to the caller.

F. CODING SPECIFICATION

1. Source Language

The module is written using the TSS 360 Assembler language

2. Suggestions and techniques

Any output operation to the strategy data set

results in the temporary closing of the data set, to ensure data set integrity in the event of a system crash.

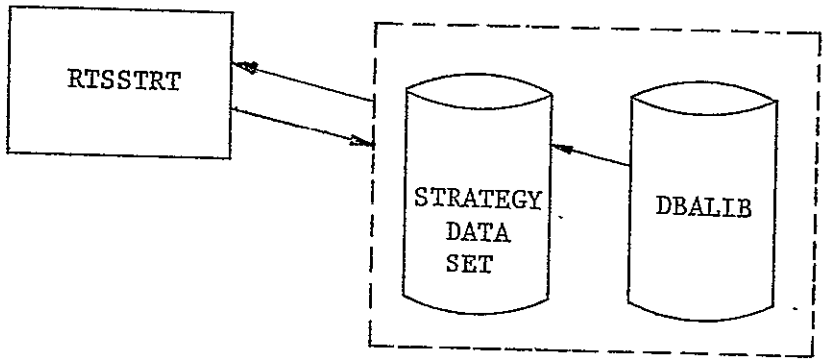


Figure 1. I/O Block Diagram

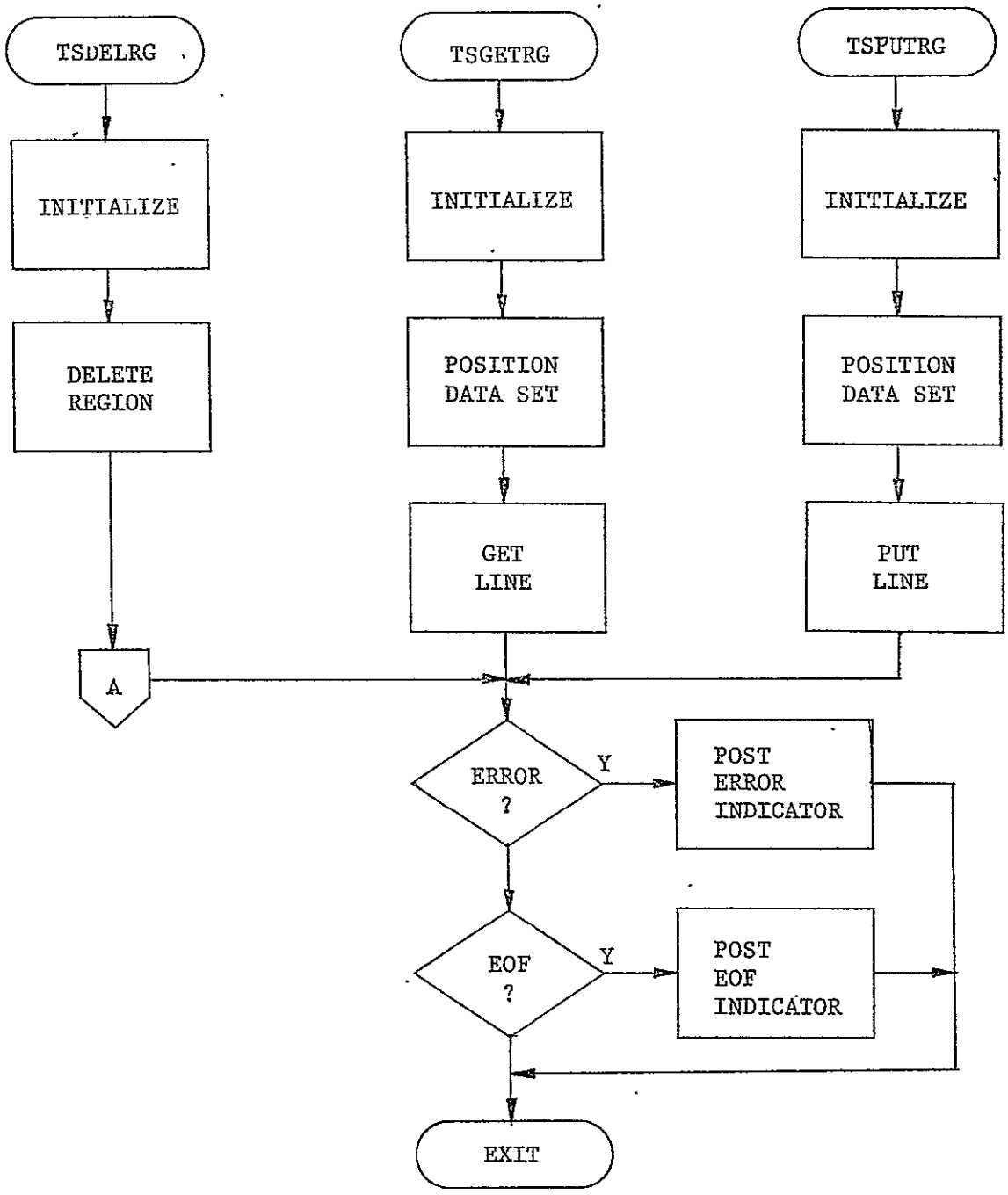


Figure 2A. Top Level Flowchart

13.2A

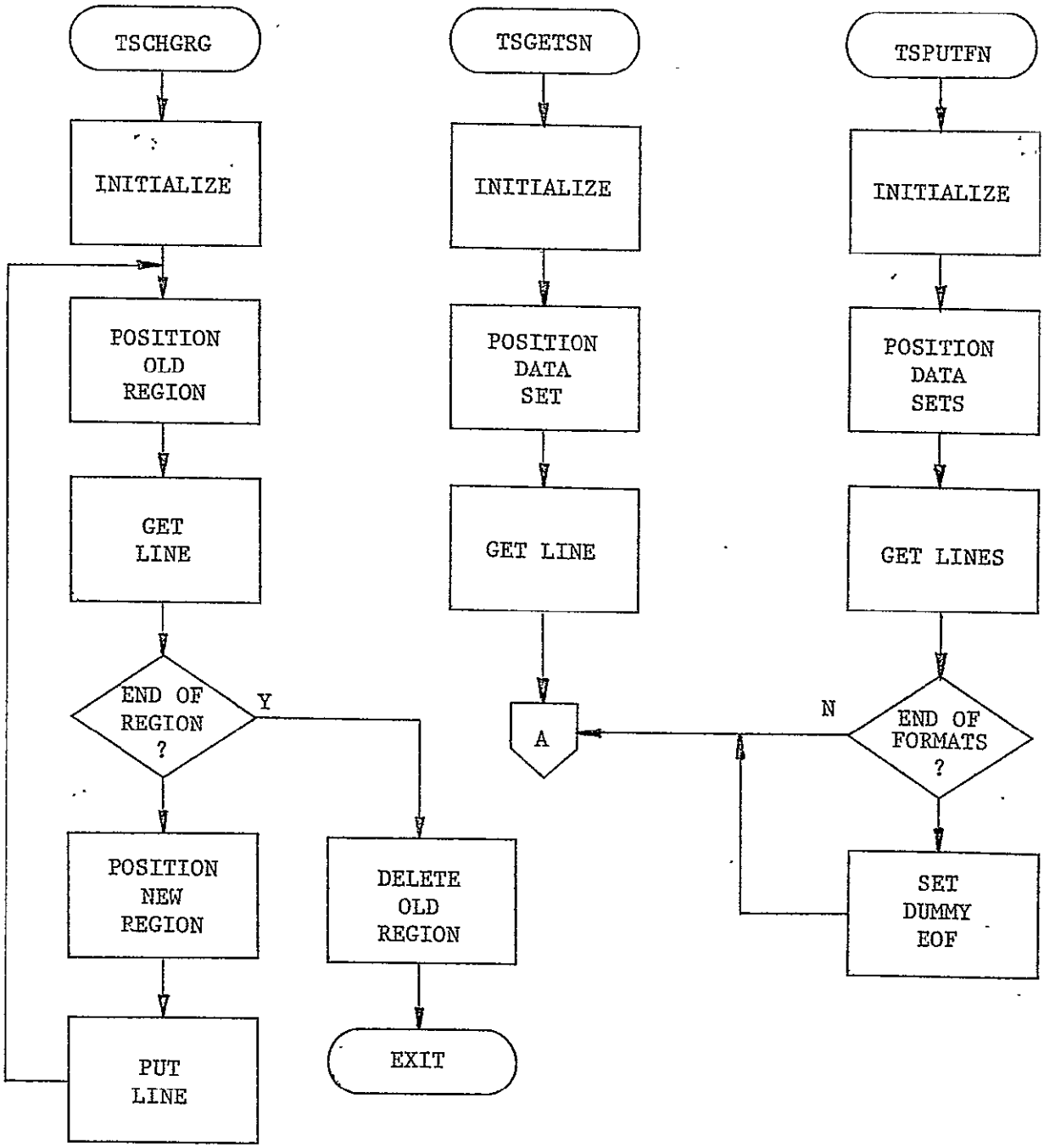


Figure 2B. Top Level Flowchart

IV 113.28

TOPIC H.4 - USER VERB TABLE

A. MODULE NAME

Program-ID - RDBUSER
Module-ID - DBUSER

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This routine uses the currently defined verb table to locate any user defined commands for that table. If any have been defined, they are appended to the list already existing in the table.

D. DATA REQUIREMENTS

1. I/O Block Diagram

Not Applicable

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

C. Input Files

Not Applicable

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

Not Applicable

c. Formatted Print Outs.

Not Applicable

d. Punched Card Output Files

4. Reference Tables

VERBTAB

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 1

2. Narrative

Upon entry, the program tests for the presence of a VERBTAB. If none is found, it exits immediately. Otherwise, the program extracts the default symbol from the table and gets the default value for that symbol.

The program then begins analyzing the data, until none remains, at which time it returns to the caller. The data is expected in command-name and entry point pairs. Each pair is extracted from the data, analyzed for valid construction and then posted to the next available slot in the table.

If there are any syntax errors, invalid names, or if the table is filled, the program will return to the caller, bypassing the remaining entries.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and Techniques

Not Applicable

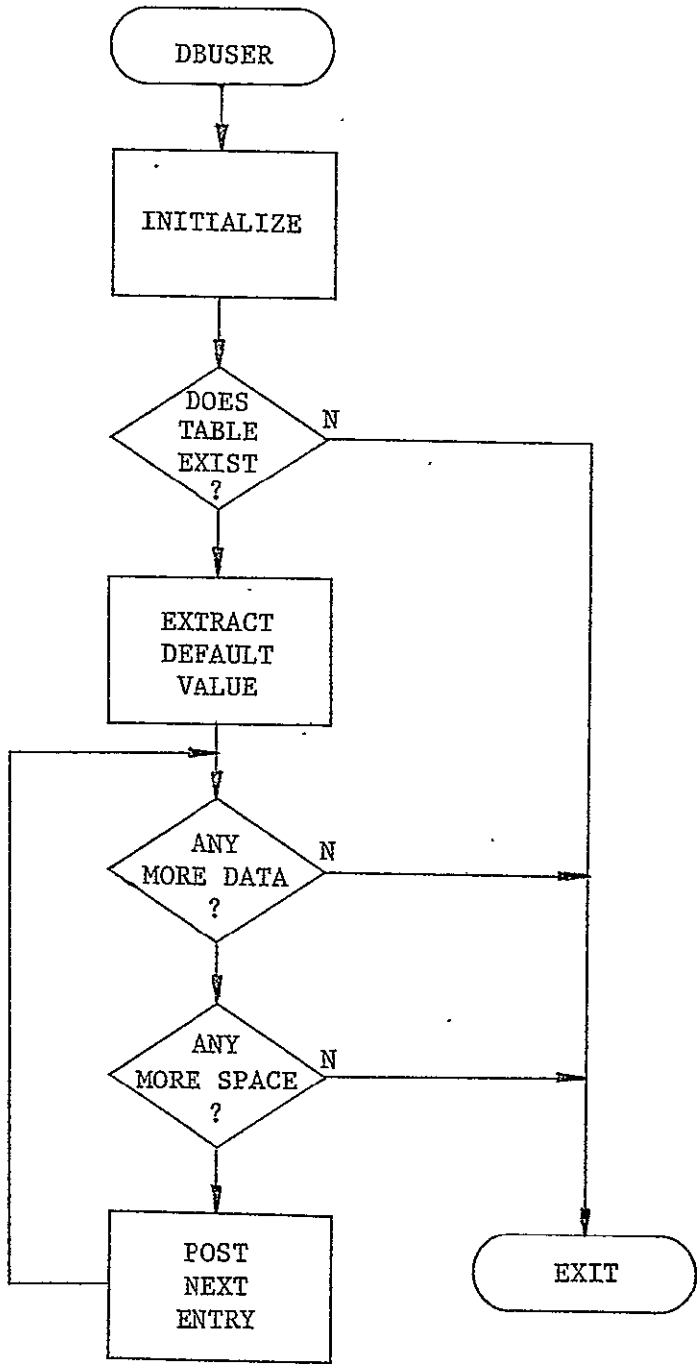


Figure 1. Top Level Flowchart - DBUSER

IV. H4.1

TOPIC H.5 - USER PROFILE ROUTINE

A. MODULE NAME

Program-ID - RDBPRC
Module-ID - DBPRO

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module performs the processing necessary for the implementation of the PROFILE, SYNONYM, DEFAULT, SYNONYM and DEFAULT commands.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

The program prompts the user for the parameters required by the various commands.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-line Terminal Displays

The display of the user's defaults and synonyms produce formatted terminal displays.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

USERTAB-the program extracts the user's NASIS-id from the user data table.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. DBPROF

At this entry point the program simply calls TSPROF to write out a copy of the user's current profile. If any errors are detected, an appropriate diagnostic message is written to the user the program then terminates.

b. DBDEF

At this entry point the program initializes itself to process defaults. It repetitively prompts for data and calls TSPDEF to process the request. If any errors are encountered, an appropriate diagnostic message is written to the user. The program then terminates.

c. DBSYN

At this entry point the program initializes itself to process synonyms. It repetitively prompts for data and calls TSPSYN to process the request. If any errors are detected, an appropriate diagnostic message is written to the user. The program then terminates.

d. DBDEFS

At this entry point the program initializes itself to display the data values corresponding to a set of default symbols. The program also initializes the screen and paging control data. The program then attempts to read in the list of symbols. If no data was entered, the program sets up to display all of the default values. Otherwise it saves the list of symbols entered.

The program then repetitively calls TSGDEF for each entry in the list, to obtain its default value. The values are formatted and posted to the screen. When the screen is filled, or when the list of names is exhausted, the program displays the screen to the user, posts the paging status data and terminates.

e. IBSYNS

At this entry point the program initializes itself to display the time values for a set of synonym terms. The program also initializes the screen and the paging control data. The program then attempts to read in the list of symbols.

If no data was entered, the program sets up to display all of the synonym values. Otherwise, it saves the list of symbols entered.

The program then repetitively calls TSGSYN for each entry in the list, to obtain its time value. The values are formatted and posted to the screen. When the screen is filled, or when the list of names is exhausted, the program displays the screen to the user, posts the paging status data and terminates.

f. DEPROPG

At this entry point the program re-initializes itself using the paging status data. If data remains, the program branches to the proper routine to produce the next screen image. Otherwise, the program writes a diagnostic message and terminates.

F. CODING SPECIFICATIONS

1. Source Language

   The program is written using the TSS 360 PL/I Language.

2. Suggestions and Techniques

   Not Applicable

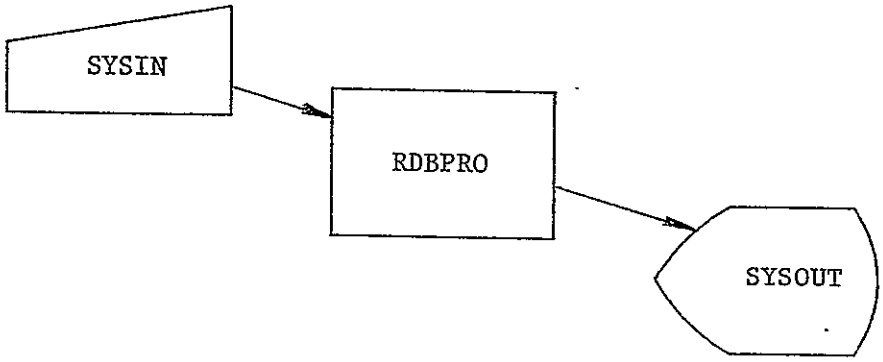


Figure 1. I/O Block Diagram

TV A.S.!

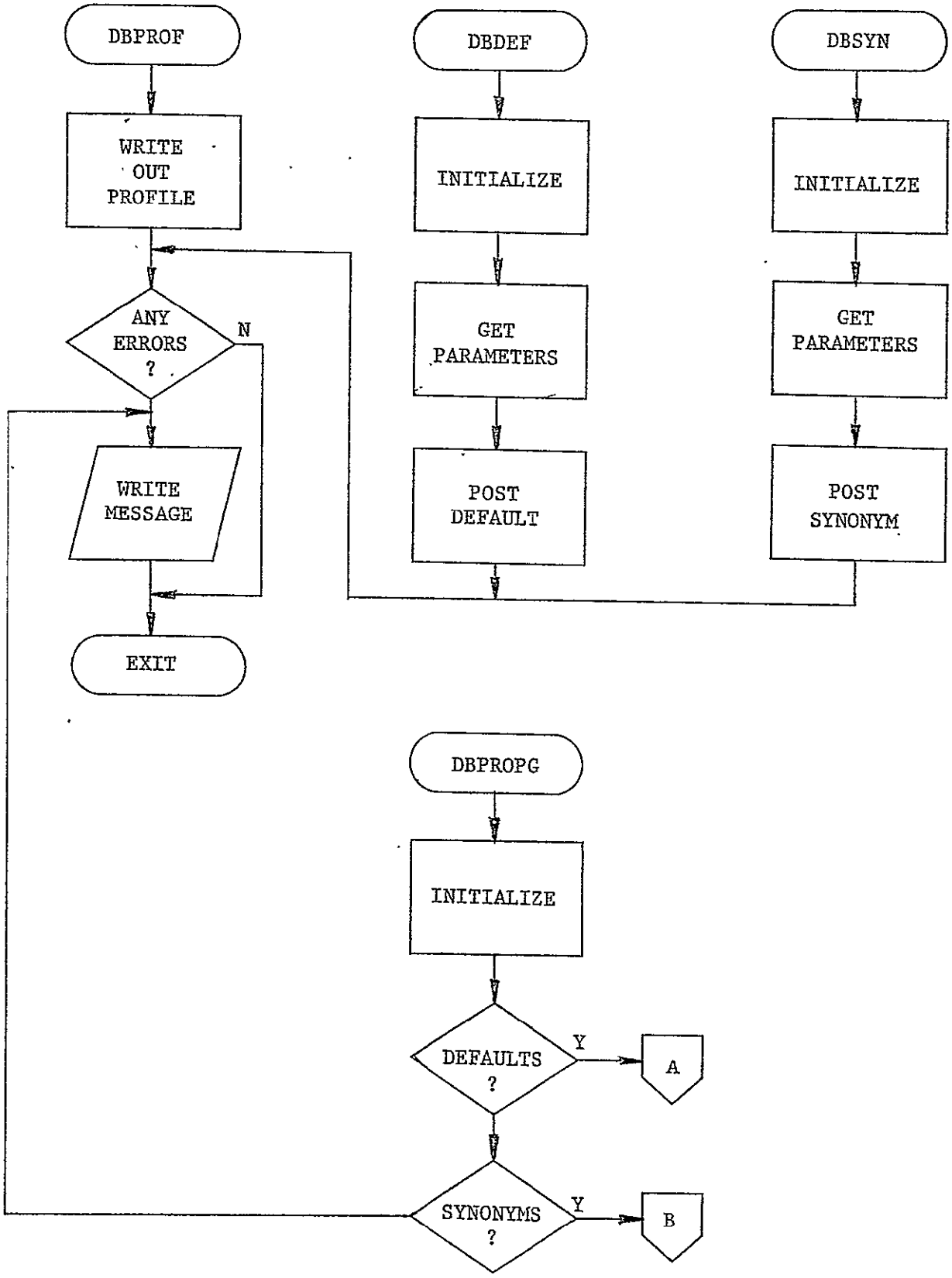


Figure 2A. Top Level Flowchart - DBPROF, DBDEF, DBSYN

IV A5.2A

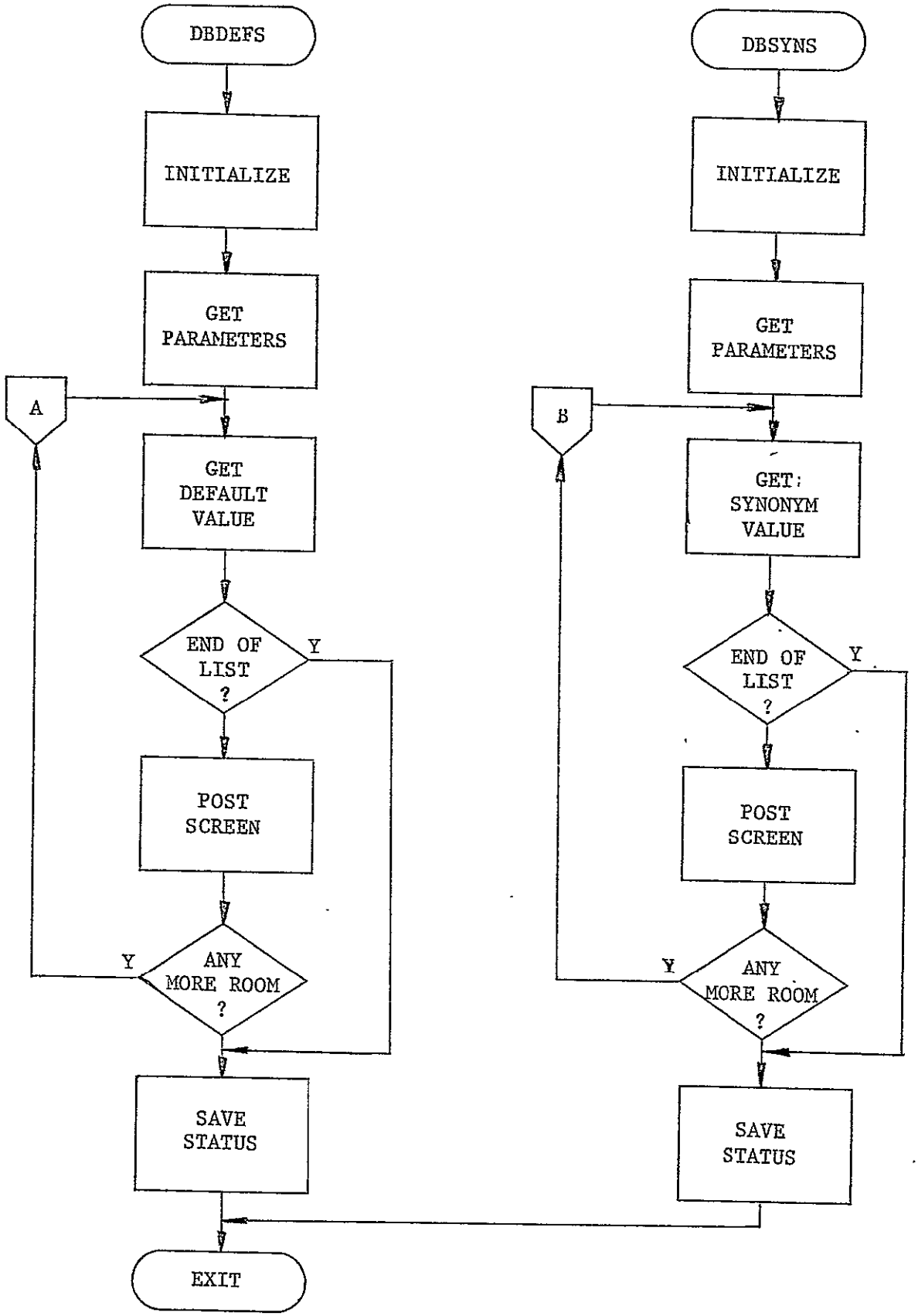


Figure 2B. Top Level Flowchart - DBDEFS, DBSYNS

TOPIC H.6 - USER PROFILE ASSEMBLER ROUTINES

A. MODULE NAME

Program-ID - RTSPRO
Module-ID - TSPRO

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

These routines act as the assembler service routines for the user's profile. They permit the retrieval, modification and storing of all synonym and default values.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

PROFILE LIBRARY or DBALIB(0)(NASISPRO) or LISRLIB(0)(NASISPRO) is used to initially obtain a profile for the user.

d. On-line Terminal Entries

Not Applicable

3. Output Data Sets

a. Output Files

PROFILE LIBRARY - the user's profile will be written out as a member of this library with the name of his NASIS-id.

b. On-line Terminal Displays

Not Applicable

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

e. Return Code

A return code will be posted with a value whose meaning is dependent upon the entry point called.

4. Reference Tables

USERTAB-the program extracts the user's NASIS-id from the user data table.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

a. TSPROF

At this entry point the program initializes itself to write out the current user's profile. It first allocates a new list and moves over all of the synonym entries not marked for deletion. It next moves over all of the default entries and re-orders the default data values. The program then attempts to locate an old profile for this user in the profile library. If one is found, it is deleted. The program then writes out the new profile and gives it the name of the user's NASIS-id. If any errors are encountered the error code is posted. The program then returns to the caller.

b. TSGSYN

At this entry point the program initializes itself to retrieve a synonym value. It first searches the synonym entries until it locates

the logical location for the symbol specified. If the entry is present and has not been deleted, or if the entry located is the symbol whose abbreviation was specified, the synonym value is extracted and passed back to the caller. If the entry located did not correspond to the symbol specified, a null value is returned to the caller.

c. TSGDEF

At this entry point the program initializes itself to retrieve a default value. It first searches the default entries until it locates the logical location for the symbol specified. If the entry is present, the data value offset is located and the data value is moved into the caller's area. The program then returns to the caller.

d. TSPSYN

At this entry point the program initializes itself to post a synonym value. It first checks to see if this is a delete request. If not, the program builds the new entry. It then searches the synonym entries until it locates the logical location for the symbol specified. If the symbol is to be deleted and it is not present, the program returns immediately. Otherwise, it performs the deletion by copying the entries prior to the deleted entry and those following the deleted entry, to a new profile similarly. Similarly, adds are processed by inserting the added entry between the two list segments. Modifications, if allowed, are performed in place. If a new profile was created, the old list is deleted. If the request was not for a deletion, the program computes the minimum abbreviation length. If it was a deletion, all synonyms for the entry deleted are flagged as deleted. The program then returns to the caller.

e. TSPDEF

At this entry point the program initializes itself to post a default value. It first checks to see if this is a delete request. If not, the program builds the new entry. It then searches the default entries until it locates the logical location for the symbol

specified. If the symbol is to be deleted and it is not present, the program returns immediately. Otherwise, it performs the deletion by copying the entries preceding the one to be deleted and those following it to a new profile. Similarly, adds are processed by inserting the added entry between the two list segments and appending the data value at the end of the profile. Modifications are performed in place, if possible, if not, the data value is simply added to the end of the profile. The program then returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 Assembler language.

2. Suggestions and Techniques

The entry searching routine should be coded as a binary search and the list moving routine should be coded as efficiently as possible.

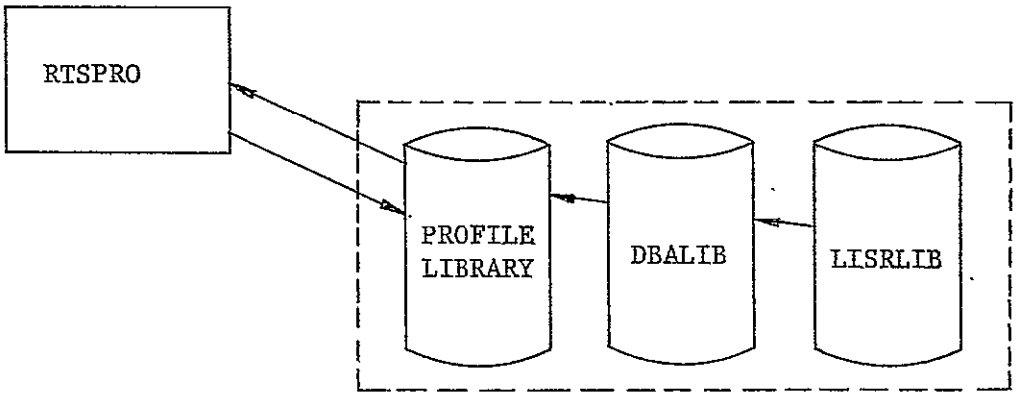


Figure 1. I/O Block Diagram

四开6.1

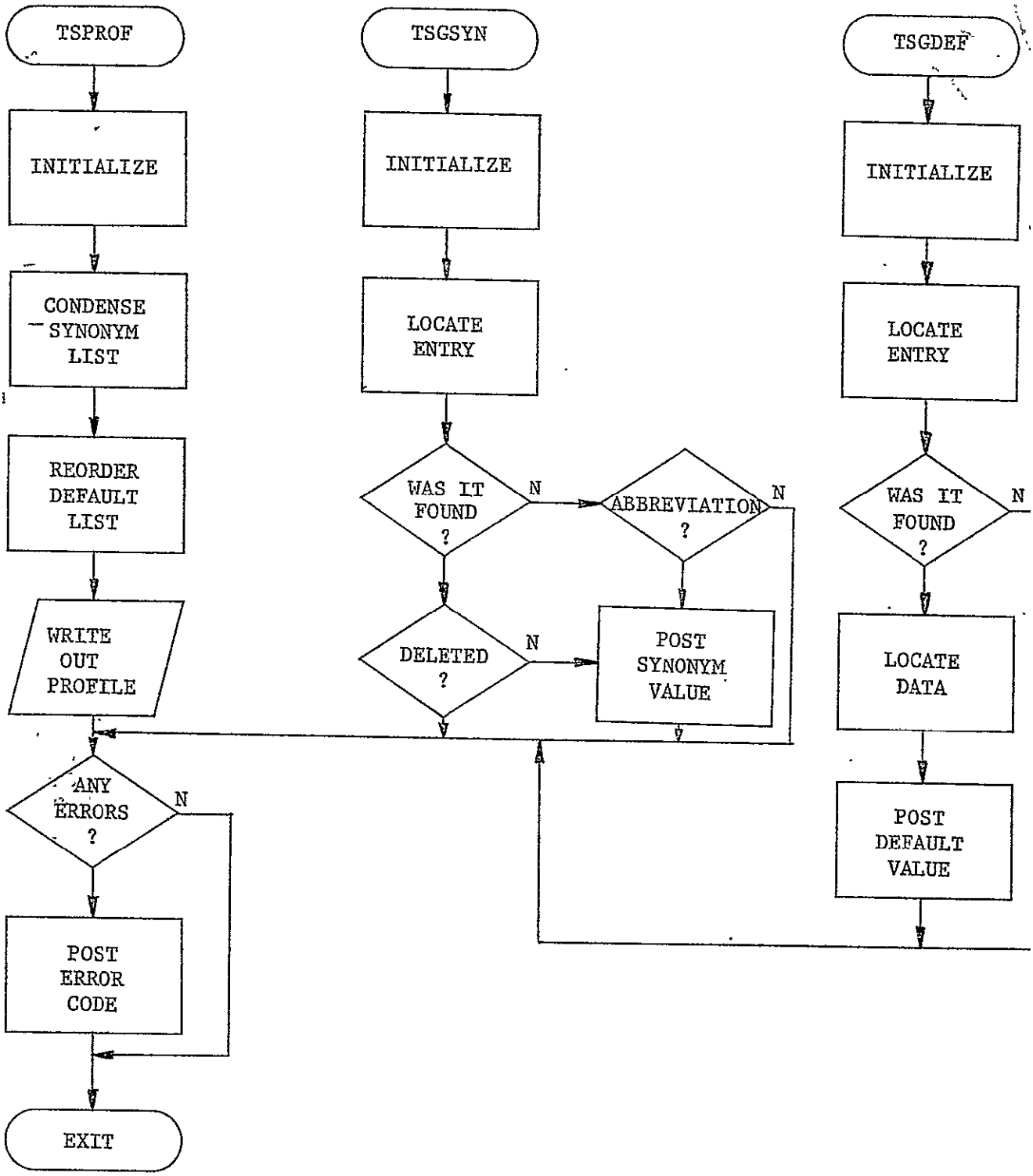


Figure 2A. Top Level Flowchart - TSPROF, TSGSYN, TSGDEF

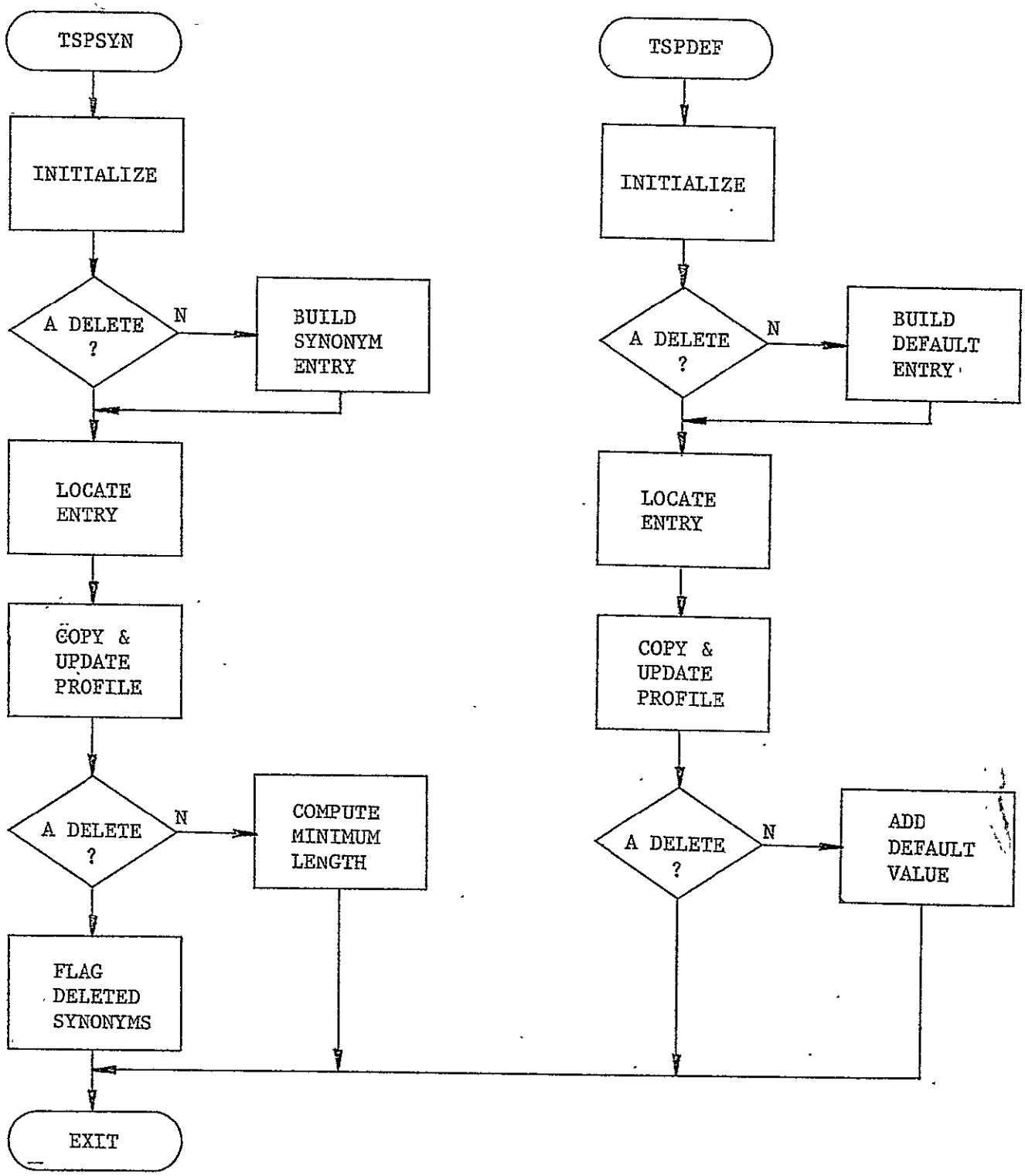


Figure 2B. Top Level Flowchart - TSPSYN, TSPDEF

17 116.253

TOPIC H.7 - TESTING FACILITY

A. MODULE NAME

Program-ID - RTSTEST
Module-ID - TSTEST

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This module provides a set of debugging services to be used in the testing and debugging of the TSS functions.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

The program can execute any of the TSS input functions.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line Terminal Displays

The program can execute any of the TSS output

functions.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

The program optionally allocates and initializes USERTAB.

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure 2

2. Narrative

Upon entry, the program initializes the variables that it uses, including the TC block. The program then calls TSTESTR to prompt the user for a debugging request. It verifies that the users input is one of the valid requests and that the associated parameters are also valid. If not, the program calls TSTESTW to issue a diagnostic message and then re-prompts the user.

If the request was END, the program simply terminates. If the request was TSS the program calls TSTESTP to allow the user to enter TSS command mode. If the request was PAD, the program moves into the output buffer the number of characters of prestored text specified by the user's parameter.

If the request was DO, the program compares the parameter to the list of valid TS2 functions and abbreviations and calls the one specified.

If the request was SET, the program passes the parameters passed into A=B pairs. The A component is compared to the list of valid data fields and abbreviations and the appropriate data field is assigned the value indicated by the B component.

If the request was EXP, the program displays a list of the abbreviations recognized and their

corresponding data field or function names.

If the request was DA, the program passes the parameters and displays the addresses of those data fields whose names or abbreviations were entered.

If the request was DIS, the program passes the parameters and displays the current values of the data fields whose names or abbreviations were entered.

If any of the requests are improperly specified or reference unknown data fields, a diagnostic message is issued to the user. Following this, or at the completion of the request, the user is prompted for his next request.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 PL/I language.

2. Suggestions and Techniques

Proper use of data field redefinition will simplify the processing of some of the requests.

A function will have to be written to return the string dope vectors as processable data in certain instances.

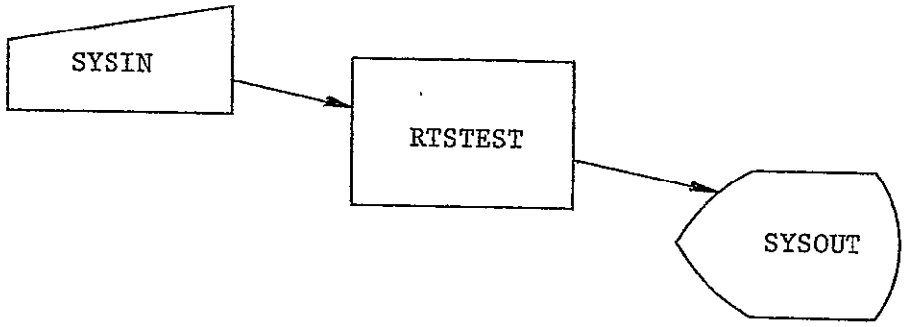


Figure 1. I/O Block Diagram

TV

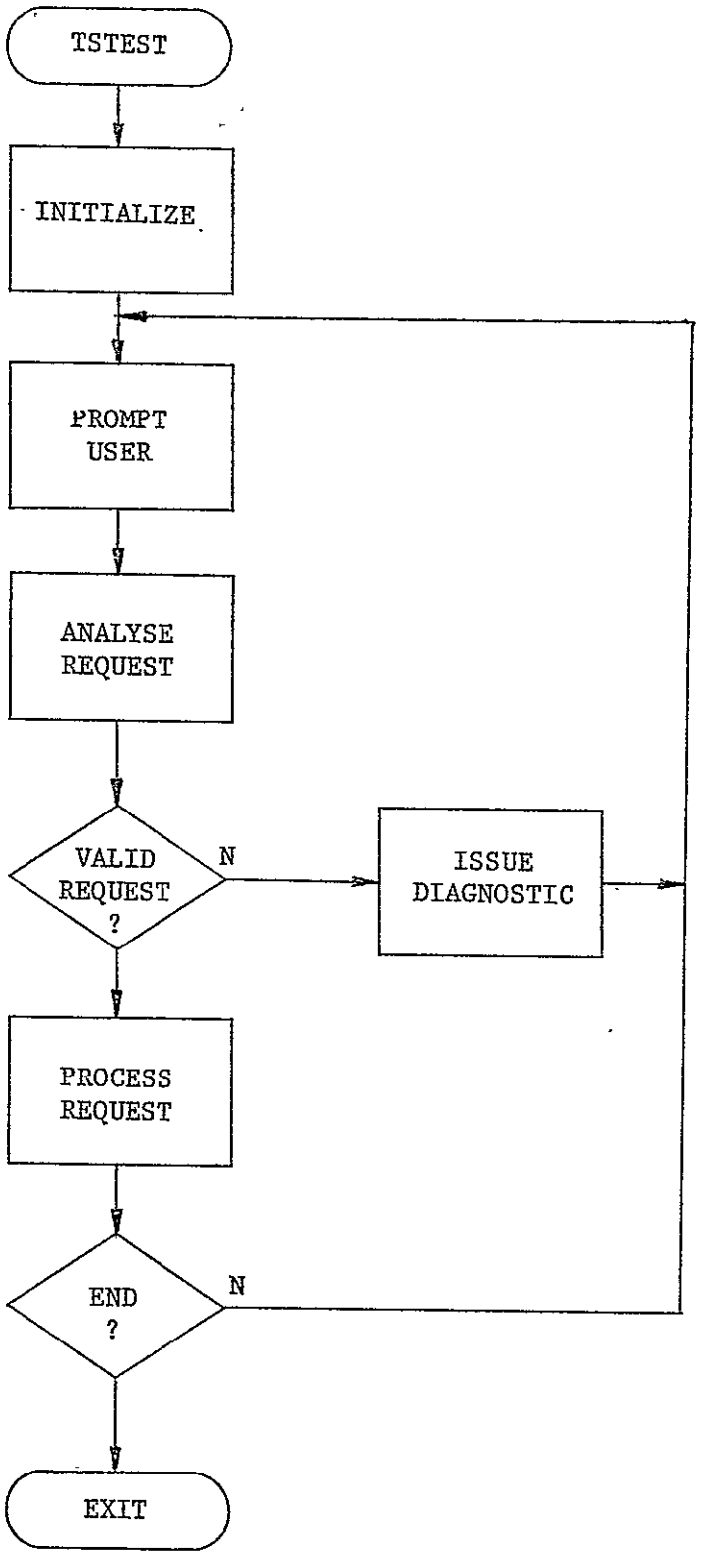


Figure 2 Top Level Flowchart - TSTEST

TOPIC H.8 - TESTING FACILITY I/O INTERFACE

A. MODULE NAME

Program-ID - RTSTESTX
Module-ID - TSTESTX

B. ANALYST

John A. Lozan
Neoterics, Inc.

C. MODULE FUNCTION

This program serves as the input/output interface between the terminal support test driver and the terminal.

D. DATA REQUIREMENTS

1. I/O Block Diagram

See Figure 1

2. Input Data Sets

a. Parameter Cards

Not Applicable

b. Punched Card Input Files

Not Applicable

c. Input Files

Not Applicable

d. On-Line Terminal Entries

At the read entry point the program accepts input from the terminal.

3. Output Data Sets

a. Output Files

Not Applicable

b. On-Line terminal displays

At the write entry point the program displays

the information passed by the caller.

c. Formatted Print Outs

Not Applicable

d. Punched Card Output Files

Not Applicable

4. Reference Tables

Nct Applicable

E. PROCESSING REQUIREMENTS

1. Top Level Flowchart

See Figure

2. Narrative

At each entry point the program initializes itself to perform these appropriate functions. The program then completes the entry linkage by calling the PL/I linkage module IHESADA.

If a read was requested, the program issues a GTWAR macro to read from the terminal. Any data that is entered is moved to the caller's parameter and its dope vector is adjusted to reflect the length of the data.

If a write was requested, the data contained in the caller's parameter is moved to the output area and written to the user by means of a GATNR macro.

If a pause was requested, the program issues a CLIC macro to place the task back into TSS command mode.

When the requested function has been completed and the parameter posted, if necessary, the program returns to the caller.

F. CODING SPECIFICATIONS

1. Source Language

The module is written using the TSS 360 Assembler language.

2. Suggestions and Techniques

Not Applicable

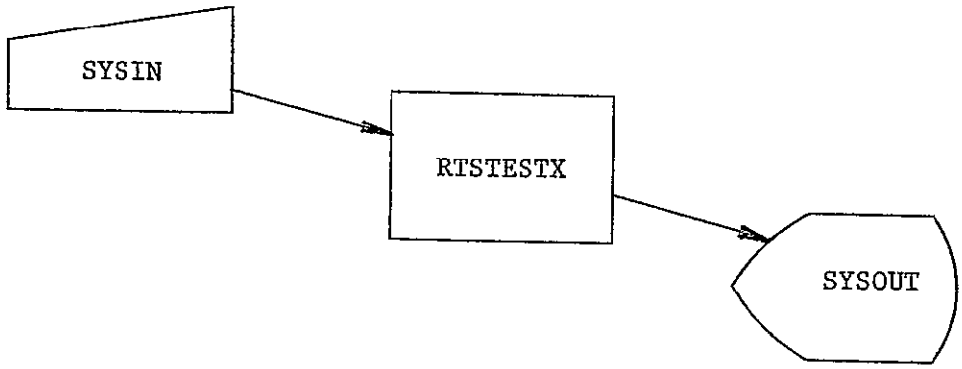
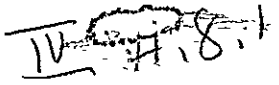


Figure 1. I/O Block Diagram



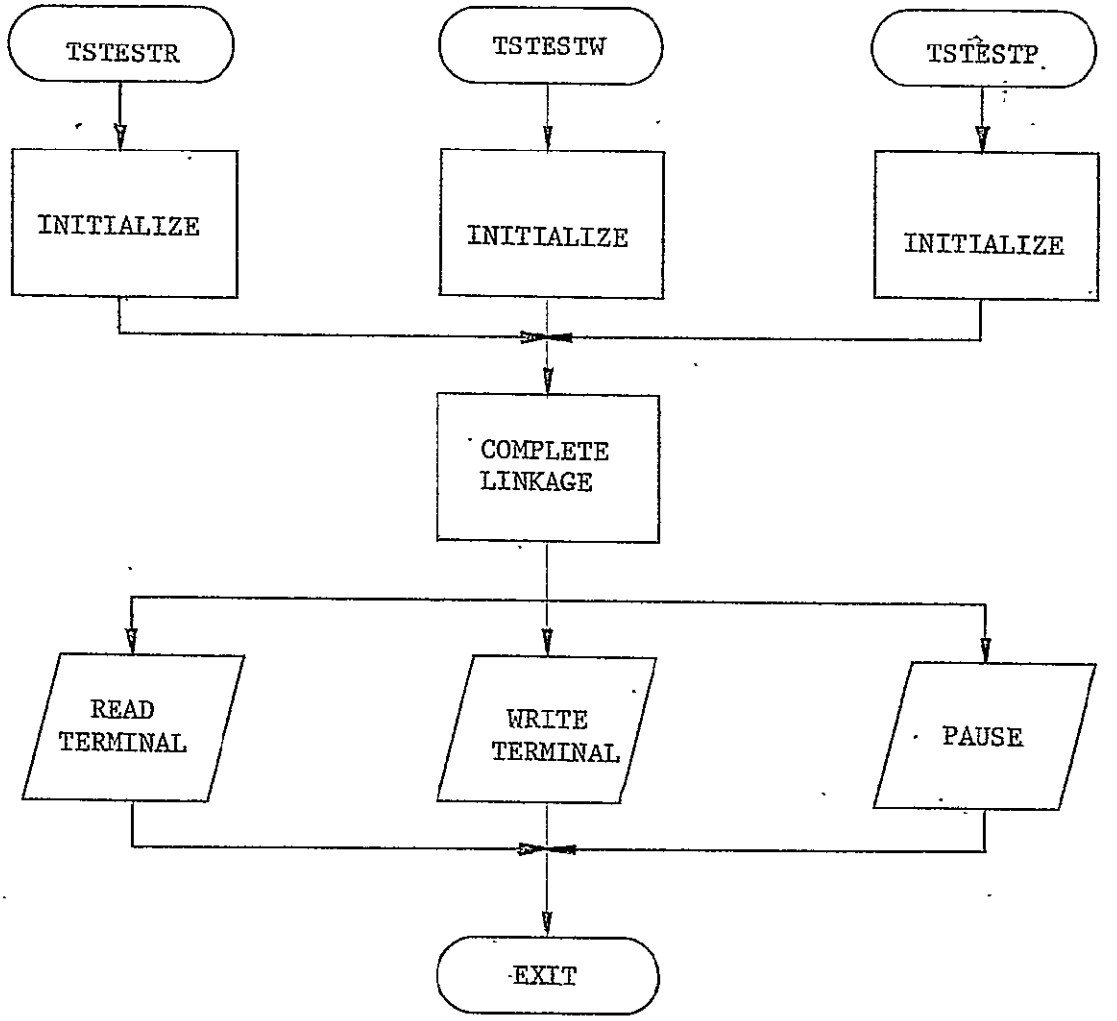


Figure 2. Top Level Flowchart - TSTESTR, TSTESTW, TSTESTP