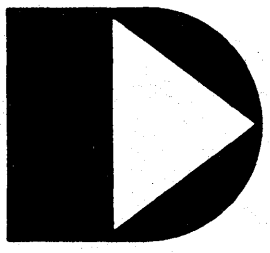
DATABUS MULTILINK 11
DBML11
User's Guide

Version 1

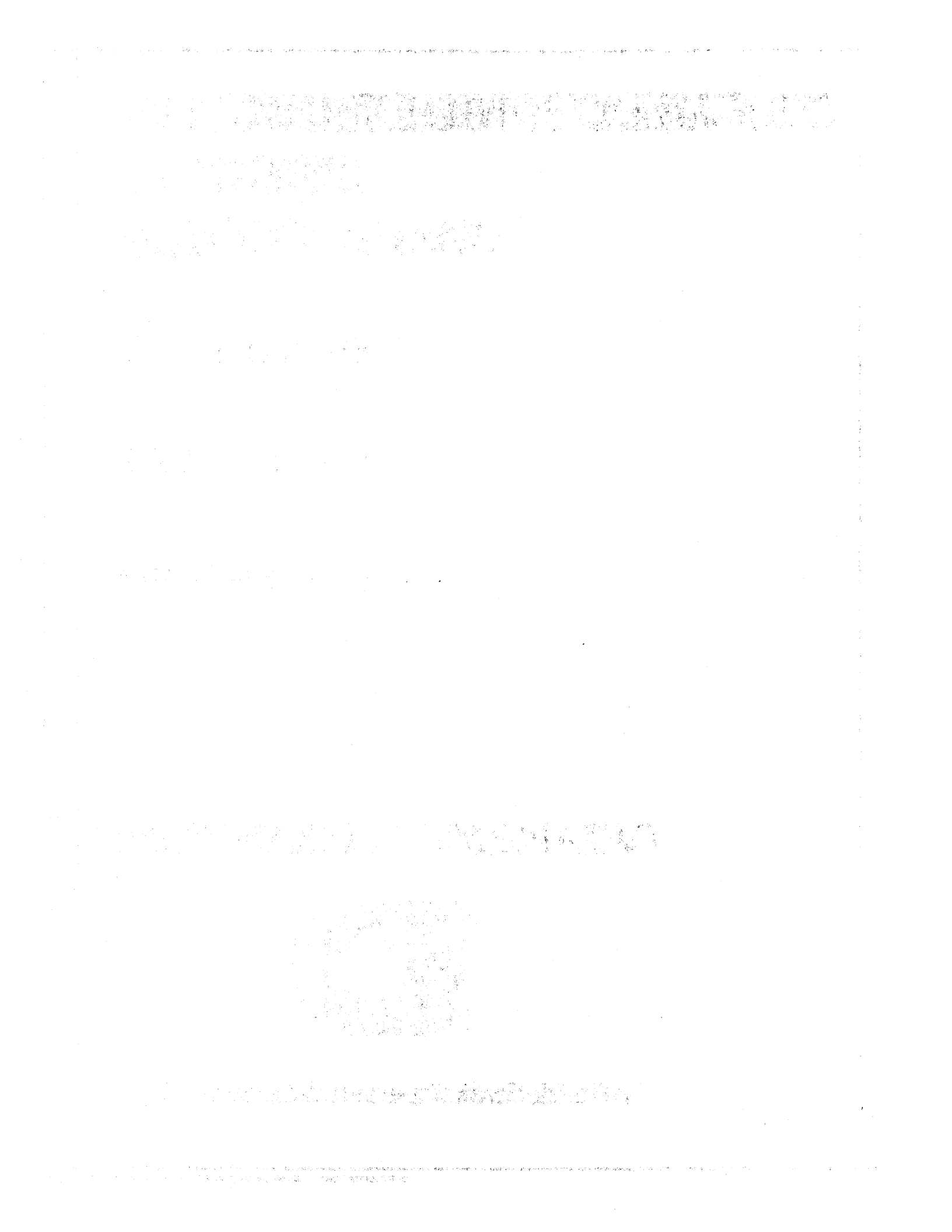
March, 1977

Model Code No. 50265

DATAPOINT CORPORATION



The leader in dispersed data processing ™



DATABUS MULTILINK 11
DBML11

User's Guide

Version 1

March, 1977

Model Code No. 50265

PREFACE

DATABUS is a high level business language system designed for use with the Datapoint Disk Operating Systems, (DOS "dot" Series). DATABUS MULTLINK 11 (DBML11) is a DATABUS language interpreter which supports data communications statements. This allows the executing DATABUS program to communicate with a remote (or host) processor. This Interpreter requires a compatible MULTILINK communications driver for execution. In addition DBML11 provides for executing a "background" utility DATABUS program concurrently with the primary program.

This manual describes the run-time characteristics of the DBML11 Interpreter. For a description of the DOS DATABUS language see the DATABUS compiler manual (DBCMP).

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

DATABUS MULTILINK 11 (DBML11) is similar to the Datapoint DATASHARE Multiple Terminal computer system. The primary difference is that DATASHARE supports multiple remote terminals whereas DATABUS MULTILINK 11 supports only the processor console as an operator input/output device. DATABUS MULTILINK also handles a high-speed line printer or servo printer and provides indexed-sequential as well as random and sequential file accessing, thus providing a powerful data entry and processing facility. DATABUS MULTILINK 11 also provides for data communications with a remote (or host) processor through the use of data communication statements and a compatible line driver. Additionally the primary and secondary programs may communicate with each other.

In addition, Datapoint DOS with its variety of utility and higher level language systems may be used in conjunction with DATABUS MULTILINK 11, enabling processing of tasks not appropriate to the DATABUS language.

Using virtual memory techniques, DATABUS MULTILINK 11 allows programs with a 32K byte area for executable statements. This, in combination with the ability of the compiler to accommodate over 3400 labels, enables the user to create and use large high level language programs. To provide rapid program execution, the data area of the executing program is manintained in main memory and not swapped.

DATABUS MULTILINK 11 provides for the simultaneous execution of two DATABUS programs. The primary program has access to the system keyboard and display, while the secondary program does not. The secondary program can be very useful for "background" tasks and as print spooling as well as communications.

Any of the Datapoint system printers may be connected to the DATABUS MULTILINK 11 configuration.

All program execution in DATABUS MULTILINK 11 occurs in the DATABUS language. Console command interpretation may be handled in a special DATABUS program, the MASTER program which enables the user to completely define his own console command and security system.

Program generation is performed under the Datapoint Disk

Operating System, using the general purpose DOS editor and DOS DATABUS Compiler, DBCMP.

CHAPTER 2. SYSTEM OPERATION

This chapter discusses loadiing the DATABUS MULTI-LINK 11 System on Diskete under DOS.C or on Cassette under DOS.C, .D, or .E and the use of the DATABUS MULTILINK 11 Interpreter. The use of the DOS DATABUS Compiler as discussed in the DBCMP Program User's Guide. DBML11 requires a Datapoint Diskette 1150 or a 5500 DOS for execution.

2.1 System Loading

The DATABUS MULTILINK 11 System is available on Diskette media. The DOS files furnished with DATABUS 11 are the Interpreter: DBML11/CMD, ML11CON/CMD, ROLLML11/SYS, and DBBACK/CMD. The first is the interpreter proper. The second and third are the Configurator and Rollout processors. The last is the rollout return processor.

2.1.1 Loading from Diskette

If the DATABUS MULTILINK 11 System is obtained on Diskette media, additional copies of the system should be generated for backup purposes using the DOS commands; DOSGEN, COPY, and/or BACKUP.

The DBML11 Interpreter system file (DBML11/CMD) may be renamed to any name desired. The ROLLOUT system file ROLLML11/SYS may not be renamed.

2.2 System Configuration

Before execution the DATABUS MULTILINK 11 Interpreter can be configured for use of external communications, rollout and the secondary program option and to allocate data area between the primary and secondary programs.

2.2.1 Configuration Execution

The DB MULTILINK 11 system may be confiigured to run with or without a utility program. The system is configured by typing:

ML11CON <file spec>

If the name of the configuration file <file spec> is omitted, DSML11/CFG will be assumed. If the configuration file already exists, the current configuration will be displayed followed by the message:

CHANGE OPTIONS?

Reply "Y" to change the configuration or "N" to exit from the configuration program.

Tne ML11CON program will ask the following sequence of questions:

SERVO PRINTER?

Reply "Y" if you want a servo printer configured for printing at the central system. Reply "N" if a local printer is to be used.

INHIBIT PRINT EJECT ON RELEASE?

Reply "Y" if you want to inhibit page ejects upon execution of a RELEASE instruction. Reply "N" if page ejects are desired.

EXTERNAL COMMUNICATIONS?

Reply "Y" if you wish the system to be configured with MULTILINK. Reply "N" if MULTILINK is not desired. Selection of EXTERNAL COMMUNICATIONS limits the baud rate available for port allocation to a maximum of 9600 baud.

ENABLE ACALL?

Reply "Y" if the file DBML11/ASM should be loaded and ACALL processing allowed. Reply "N" if this is not desired.

ENABLE ROLLOUT?

If a "Y" (YES) answer is given at this point the configurator will ask:

ENABLE ROLLBACK?

A "Y" (YES) entry here will cause the creation of ROLLFILE/SYS to be used for storage of the DBML11 variables during rollout. This will allow a rollback via the execution of DBBACK. ROLLFILE/SYS requires 114 disk sectors.

If a "N" (NO) entry was given for ROLLBACK the ROLLFILE/SYS will not be created and will be deleted if it exists. This will allow a rollout directly to DOS without saving the DBML11 variables.

DISABLE UTILITY PROGRAM?

Reply "Y" (YES) if no utility program will be used. Reply "N" (NO) if you wish to use a utility program.

If a secondary program was configured, the following message is displayed:

EQUAL DATA AREAS FOR BOTH PROGRAMS?

A "Y" response divides the available user data area into equal parts between the two programs. If equal division is not desired then a "N" should be entered and the following message will be displayed:

XXXX BYTES LEFT, PORT n DATA AREA SIZE:

where n=1 (for primary port) or 2 (for utility port)

The operator should enter the data area size for the appropriate port. Any size quantity may be assigned between 256 bytes and 4096 bytes, not exceeding the amount of space remaining (XXXX).

2.3 Program Execution

If the DATABUS MULTILINK 11 Interpreter is named DBML11/CMD then a DATABUS program (eg. "PROGA") can be executed as the Primary Program by entering:

DBML11 PROGA

In addition, the DATABUS Program "PROGB" can be executed as the Secondary Program by entering:

DBML11 PROGA,PROGB

It is not valid to execute a secondary program when the primary program is not executing. The DATABUS program compiled and filed under the name PROGA/DBC will begin execution. This program will continue executing until an irrecoverable or untrapped error is detected or until a STOP instruction is executed. At this time system control will return to the DOS. If no primary program is specified on the command line the program MASTER/DBC will be loaded and execution will begin. If a secondary program is configured but not specified on the command line, then the program UTILITY/DBC will be searched for and loaded and execution will begin.

The general form for the DATABUS MULTILINK 11 Interpreter command is:

DBML11 [<object>],[<object>][;<parameter>]

If DATABUS program is not specified, i.e., entering only:

DBML11

will cause the interpreter to assume a default program name of "MASTER", search for a program cataloged as MASTER/DBC and begin executing this program as the primary program. The very first program to begin execution under DBML11 is considered to be the "master" program, regardless of what name it has. This program will continue execution until a STOP is executed, at which time control will return to the DOS.

The "master" program can cause another DATABUS program to begin execution through the use of the CHAIN instruction. In this case, when this new program executes a STOP instruction or has an irrecoverable or untrapped error, control is transferred to the start of the master program instead of to DOS.

Two parameters are available for setting the Databus internal clock. These parameters are separated by commas and are entered on the command line. These are the clock set parameter and the date set parameter. The clock set parameter has the form:

Chhmm

where hhmm is the time used on a 24 hour clock. Hence, in order to set the time to 19:25, the parameter would be C1925. The date set parameter has the form:

Dddd/yy

where yy is the last two digits of the year and ddd is the Julian date. For example, March 3, 1977 would be specified as:

D062/77

If the Time or Date is entered incorrectly DBML11 will request that one or the other or both be re-entered. Parameters not specified in the command line will be assumed 0. The DATABUS MULTI-LINK 11 Interpreter will search for an overlay named "DBML11/COM" (where DBML11 is the name of the interpreter system files) if external communications has been configured. If the overlay exists and is loadable, then the external communications facility is enabled. If not, the following error message will be displayed before returning to DOS:

DBML11/COM DOES NOT EXIST OR IS NOT LOADABLE

The "ACALL" facility (See Section 4.4) is also enabled if an overlay named "DBML11/ASM" exists and is loadable. If an "ACALL" DATABUS instruction is encountered and the ACALL facility is not available then the DATABUS instruction is ignored.

CHAPTER 3. PHYSICAL SYSTEM CHARACTERISTICS

3.1 Virtual Memory

To achieve a reasonable amount of program space for many simultaneous programs, DATABUS employs a virtual memory technique. DATABUS code is very compact, with very few bytes of instructions being capable of invoking a large amount of processor activity. Therefore, the rate at which DATABUS program bytes are fetched is very low. Because of this low rate, the actual program code bytes can be kept in the randomly accessible disk or memory buffers with very little effect on program execution speed. Another characteristic of DATABUS code is that it is never modified. Because of this, program code need only be read in and never written back out to the disk.

A different story exists in the case of the program data, however. This data is accessed at a very high rate and must be in main memory to be effectively accessible by the DATABUS interpreter. For this reason the program data for all programs is kept resident in main memory. This fact will be shown later to have further advantages in the case of port I/O.

To implement an effective virtual memory accessing algorithm, the program code is kept on the disk as 250 byte pages. Because the code is paged in blocks, the DATABUS programmer can make his program run much more effeciently, in many cases, by forcing his code to cross as few page boundaries as possible. Each time a page boundary is crossed, a new page must be read in if it is not already in the buffer. The paging scheme used is purely demand with the least recently used page being destroyed to make space for the new page. Actually, in a lightly loaded system, a single program could get a number of pages all resident in disk or memory buffers at once and crossing a given page boundary would not cause a disk read, but any significant loading will cause this condition to cease. Therefore, the DATABUS programmer can assume that each time he crosses a page boundary, a new read will occur. This read will cause delay in the execution of the program. This time is time that cannot be used by any other program since the disk is busy. By causing an excessive number of page boundary crossings, the programmer can easily cause his program to execute very slowly.

However, an instruction called TABPAGE exists in DATASHARE to aid the programmer in making his execution speed as high as possible. This instruction causes the location counter in the compiler to be incremented until it is at the start of the next page (notning will be generated if the location counter is already at the start of a page). When this instruction is executed, it causes a GOTO to the start of the next page. By using this instruction, the programmer can cause logical parts of his program to contain as few page boundaries as possible. Another way to increase execution speed is to use in-line coding as much as possible, especially for short operations, instead of the subroutine calling feature if the subroutine is located in a page different from the calling location. This is economically feasible because of the large space available for each program (32K bytes).

3.2 Scheduling

To provide optimum response time, DATABUS MULTILINK 11 handles all port I/O using interrupt driven foreground routines, which means that data transfer between the terminal and the system can occur regardless of the computational task being handled by the background program at any given time. The foreground routines actually interpret the KEYIN and DISPLAY instructions, with the background interpretive code merely passing these instructions to the foreground through a circular buffer allocated for each port. Conventional systems use such a buffer to hold the actual characters transferred between the system and the terminal. However, DATASHARE uses this buffer to hold the interpretive code bytes, thus enabling many more bytes to be transferred than can actually be held in the buffer. For example, a DISPLAY statement may contain some quoted information and then a variable name. The variable name is represented by two bytes but the contents of the variable could be fifty bytes long, enabling two bytes of buffer space to invoke the transfer of fifty bytes to the terminal. This is made possible by the fact that all program data is resident in main memory which enables the foreground routine to be executing an I/O statement for a given port even though the background program for that port may not be swapped in at the time.

The foreground and background program for a given port always execute exclusively of each other to prevent conflicts over data values. When the background program executes a DISPLAY statement, the statement is stored in the buffer for the given port and then the background program is deactivated and the foreground program activated. When the foreground program has completely executed the I/O statement, it causes a high priority interrupt to the

background, which deactivates the current program and activates the one which was executing the DISPLAY statement which caused the interrupt. One important consideration which must be taken into account by the DATABUS programmer concerning port I/O is the fact that every time an I/O instruction is completed in the foreground, the background program is swapped in. If the programmer is not careful, he can cause the system to thrash (spend most of its time swapping background programs instead of doing useful work) by causing a high rate of I/O completion interrupts. An example would be using many separate DISPLAY statements instead of one long continued statement.

The above discussion concerns only port I/O. All disk I/O is performed under the DOS which is a background-only operation. This means that all DOS functions are non-interruptable and long directory searches (which can take up to several seconds with a multiple drive system) will cause the response to I/O completion interrupts to be delayed. Long DOS functions, however, occur infrequently and therefore can be ignored from an average response time calculation standpoint.

Printing under DATABUS MULTILINK II is performed in background and foreground. The background execution sets up a line image in a 132 position buffer and when vertical paper motion is necessary, this buffer is transferred to a 512 character circular buffer which is emptied by a simple foreground process. The background execution is suspended only if the 512 character buffer becomes full. Therefore, one can complete a number of print statements without being swapped out.

When the background program resumes execution due to the completion of a foreground I/O task, it is guaranteed a minimum amount of execution time. This prevents the system from spending all of its time swapping background tasks when the foreground I/O completion rate is high.

CHAPTER 4. PROGRAMMING CONSIDERATIONS

4.1 RELEASE

Execution of the RELEASE instruction will set the OVER condition flag if the printer is unavailable for use by the port executing the RELEASE.

4.2 DEBUG

The DEBUG instruction is treated as a NOP. That is, execution is continued with no "DEBUG" activity taken.

4.3 WRITE

When using the WRITE statement on sequential file, if the second parament,<nvar>, is equal -2 the sector will not be written but is merely flagged as pending a write to disk. The actual write to disk occurs under one of the following conditions:

a) The user program performs a "CHAIN", "ROLLOUT", "STOP", or otherwise terminates (eg. dueto error).

b) Immediately prior to a "PREPARE", "OPEN", or "CLOSE" instruction.

c) Immediately prior to any "TRAP", or "TRAPCLR" instruction referencing either an "IO" or "PARITY" trap.

d) The disk controller buffer containing the write pending sector becomes the least recently used buffer and becomes required for some other purpose.

The consequences of this are as follows:

a) The DATABUS interpreter may not discover that the disk has gone off-line until one of the above four conditions occurs.

b) The DATABUS interpreter may not discover that the disk secotr destined to receive the buffer has irrecoverable

bad parity until one of the above four conditions occurs.

c) Any data contained in writing-pending buffers will be lost if the processor is re-booted before the write is performed.

Thus a trap set for a parity or disk off-line condition during for example, a "WRITE" instruction may not actually get entered until many instructions later. Similarly, if the above conditions were not originally trapped, the error message generated may not have a program counter address pointing to the "WRITE" instruction responsible, but to an unrelated instruction much further along in the program. In general, unless an "IO" or "PARITY" trap is set before a given I/O instruction and one of the above four conditions occurs before the next I/O instruction involving a write to disk, the I/O instruction causing a disk off-line or parity trap may be indeterminate.

If it is necessary to re-boot and also critical that write-pending buffers be written, the DOS should be re-booted by executing the ROLLOUT instruction. For example: ROLLOUT "FREE".

4.4 ACALL Facility

DATABUS MULTILINK 11 supports the ACALL (Assembler Language CALL) facility. This facility allows an Assembler Language program to be invoked from a DATABUS program. The implementation consists of a user-written overlay which is loaded at the DATABUS MULTILINK 11 interpreter initialization time if the ACALL facility is enabled. This overlay must have the name DBML11/ASM where DBML11 is the name of the interpreter overlay. This user-written overlay processes the ACALL DATABUS instruction. If the ACALL facility is not enabled all ACALL instructions will simply be ignored. The ACALL processor should follow the guidelines below:

a) The ACALL processor must not modify the processor base register.

b) The ACALL processor should restrict the amount of time spent with interrupts disabled to the absolute minimum practical. Disabling interrupts for longer than 200 microseconds at a time may produce indeterminate results.

c) The ACALL processor when exiting should leave the processor stack as it was on entry.

The following sections describe the interface characteristics between the ACALL processor and the DATABUS MULTILINK 11 interpreter.

4.4.1 ACSTRT - Starting Location For ACALL Overlay

The address ACSTRT represents the first location of 255 bytes reserved for the overlay which processes the ACALL instruction.

4.4.2 GCMOP - Get The Next Operand Location

This routine is called to fetch the next operand from the list of operands specified in an ACALL DATABUS instruction. The routine is entered via a Assembler "CALL" instruction and returns FALSE CARRY if the end of the operand list has been encountered. If an operand is found, the routine returns TRUE CARRY with the address of the first byte of the operand in the HL register pair. The ACALL instruction interpreter overlay should call this routine until it returns FALSE CARRY before executing a RETURN instruction itself.

4.4.3 Condition Code Routines

The following routines exist in the DATABUS MULTILINK 11 interpreter to manipulate the DATABUS condition flags. They are used to return condition information to the DATABUS program. They are invoked by using the Assembler "CALL" instruction.

SETEQL - The routine sets the DATABUS EQUAL flag. May be called with any A-register value.

SETLSS - This routine sets the DATABUS LESS flag. May be called with any A-register value.

SETOVR - This routine sets the DATABUS OVER flag. May be called with any A-register value.

CLREQL - This routine clears the DATABUS EQUAL flag. Must be called with A-register = 0.

CLRLSS - This routine clears the DATABUS LESS flag. Must be called with A-register = 0.

CLROVR - This routine clears the DATABUS OVER flag. Must be called with A-register = 0.

4.5 EXTERNAL COMMUNICATIONS Facility

For users desiring to develop their own compatible communications line handler, the following sections describe the interface characteristics between the DATABUS MULTILINK 11 with MULTILINK facility and the compatible communication line handler.

All the interface routines are invoked by using the Assembler "CALL" instruction.

4.5.0.1 CRCCF - Change Calling Frequency

This routine is called with a value in the C-register specifying the number of milliseconds between calls to the external communications process. This number will be saved and used as the initial value of a down-counter used by the scheduler. The counter is decremented each millisecond. When the count reaches zero, it is reset to the initial value and the external communications process is called. The default value is 1, causing the process to be called each millisecond.

4.5.0.2 CRGUSR - Get port number of current USeR

When this routine returns, the C-register will contain the port number for the user "owning" the current comlst. If no comlst is currently active, then the port number for the last-used comlst is returned. The port number is given with a base of zero, that is, the value range from 0 to 1, corresponding to DATABUS "master" and "utility" ports respectively. This routine is most commonly used after a general poll to determine the owner of the comlst that was found.

4.5.1 CRSSAF - Set the Suspension Acknowledged Flag

This routine is called to set the suspension acknowledged flag in response to a denial of permission to enter a new control sequence. This routine should not be called until all "wrap-up" operations have been performed (which may take several interrupts to accomplish). The DATABUS system will normally do a rollout shortly after this routine is called.

4.5.2 CRRPGO - Request Permission to GO

This routine is called at a logical stopping point in the main communications loop. It returns TRUE ZERO if permission to continue is granted, or FALSE ZERO if permission is denied. Denial of permission indicates that the DATASHARE background is waiting to do a rollout (or some other operation which will make foreground activity impossible). If permission to go is denied, the external communications process need not terminate immediately, but it must eventually acknowledge that a suspension was requested. If it never acknowledges the suspension, the DATASHARE system will be hung in a background loop.

4.5.3 CRCS$ - Change State

Calling this routine effects a return to the scheduler and sets the entry point to the communications process code to the instruction following the call. The concept is identical to that used by the interrupt handler in the DOS. The entry point set by this routine is called the "state entry point".

4.5.4 CRCPS$ - Change Primary State

This routine works in the same manner as CRCS$. However, it sets a different entry point called the "primary entry point". The primary entry point is called by the scheduler when the process available flag is clear; the state entry point is called when the process available flag is set.

4.5.5 CRSSOK,CRSSCU - Set Status

CRSSOK - Set Status to OK
CRSSCU - Set Status Channel Unavailable

These routines are called to set a final status into the comlst, the list dequeues and release it from use by the communications process. There is no harm in calling these routines even when a comlst is not active.

4.5.6 CRRSB - Reset to Start of Block

This routine is called to reset various pointers to the values captured by the last call to CRNSB. It is used in a blocked message discipline when it is necessary to repeat the last message block. It is important to recognize that calling this routine does not result in a resetting of the formpointers or lenghtpointers of the variables which may have been affected by the message block in error. Hence, a repeated message block which is, for some strange reason, significantly shorter than the original block may cause some extraneous characters to remain in some of the variables.

4.5.7 CRNSB - Note Start of Block

This routine is called to capture the critical comlst pointers at their current values. These values can be restored by the CRRSB routine.

4.5.8 CRCAF,CRSAF - Clear-Set the Avialability Flag

CRCAF - Clear the Availability Flag
CRSAF - Set the Availability Flag

The availability flag is used to differentiate between "handshaking mode" and "active communication mode". When the availability flag is clear, the scheduler will enter the communication process at its primary entry point, and will also presume to dequeue all posted comlst with a status of channel unavailable. Application programs are thus notified when they attempt to post comlst when a condition of active communication is still problematical (i.e., handshaking may fail). Similarly, as long as the availability flag is clear, the communication process is free to do any required setup, waiting for ringing, handshake sequencing, etc., without the burden of disposing of comlst which it is presently unable to handle. When the active processing of message is possible, the process should set the availability flag and be prepared to enter itself at the state entry point on the next interrupt.

4.5.9 CRSUSR - Select USeR

This routine is called with a DATABUS port number in the C-register. The port number should be based at 0 (i.e., it should have a range of 0 to 1). The routine will return TRUE CARRY if such a port has not been configured into the DATABUS system. Otherwise, it will return FALSE CARRY with the hardware base register set to select the data area for the specified port. This routine may be used as a quick way to determine whether a "selectively addressed device" actually exists.

4.5.10 CRGENP - GENeral Poll

This routine performs the operation of "general poll" on all ports configured into the system. It scans each port in sequence for a posted sending comlst. It returns TRUE ZERO if no sending comlst exists anywhere. If a sending comlst is found, it returns FALSE ZERO after internally calling CRGSCL (i.e., the communications process need not perform any additional setup on the comlst; it is ready to deliver characters). This routine must be called at the "zero-th" level (i.e., the same stack level at which the process was entered by the scheduler) because it makes an intenal call CRCS$. By testing only one port on each interrupt, the consumption of excessive foreground time is minimized.

4.5.11 CRZTIM,CRGTIM - Zero, Get the TIMer

CRZTIM - Zero the TIMer
CRGTIM - Get the TIMe

These routines are used for timing intervals used for turn-around delays, time-out tests, etc. The time is counted in 1/4ths of a second, thus allowing a one byte counter to represent 250 milliseconds through 60 seconds. CRZTIM has no arguments or results; CRGTIM returns the time in the C-register.

4.5.12 CRGET - GET a character from the comlst

This is the basic routine used to get characters from a sending comlst. If a character is returned, it is delivered in the B-register. There are four possible return conditions for this routine, as follows:

FZ FS FC - Character in B-reg (=0203 if past length pointer).

FZ FS TC - end of physical variable encountered;
comlst toggled to next variable;
no character returned.
TZ FS TC - end of physical variable and end of comlst;
no character returned.
FZ TS TC - abnormal end of comlst (i.e., comlst is gone);
no character returned.

Characters are retrieved from a variable beginning with the first physical character and ending with the last physical character. For string variables, all characters past the lengthpointer are returned in the B-register as the character 0203. At the option of the communication process, this character may be ignored, converted to a blank, or used to trigger the generation of an end of field mark.

4.5.13 CRGDCL,CRGSCL - Get a Destination/Source COMLST

CRGDCL - Get a Destination Comlst
CRGSCL - Get a Source Comlst

These routines are called with the desired port number [0-1] in the C-register. If no comlst is found, the routines return TRUE ZERO. If the desired type of comlst is found for the specified port, the routines return FALSE ZERO with the comlst's status set to "in process" and the comlst still on the queue. If a comlst is found and it has a 'gone away' status, the routines return TRUE CARRY. All pointers are set up for the first get/put, and the start of the block is noted. Since comlsts are not automatically dequeued when exhausted or 'gone away', CRSSOK or CRSSCU must be called to dequeue one comlst before quequeing another.

4.5.14 CRTOGL - TOGgLe to the next variable

This routine may be called by the communication process to force a source comlst to move directly to the next variable for the next call to CRGET. It is appropiate to use this routine only when handling record formats that permit variable length fields. Three return conditions are possible, as follows:

FZ FS TC - comlst successfully toggled to next variable.
TZ FS TC - normal end of comlst encountered.
FZ TS TC - abnormal end of comlst encountered.

4.5.15 CRPUT - PUT a character into a comlst

This routine is called with a character given in the B-register. If the character is not an 0203, an attempt is made to place the character into the current variable in the comlst. If the character is an 0203, the lengthpointer for the current variable will be set and the comlst will be toggled to the next variable. There are four possible return conditions, as follows:

FZ FS FC - character was successfully put.
FZ FS TC - character not put; end of variable encountered; comlst toggled to next variable.
TZ FS TC - character not put; end of variable encountered; end of comlst encountered.
FZ TS TC - character not put; abnormal end of comlst.

The communication process should always put an 0203 following the last message character (unless the end of the comlst was already encountered) so that the lenghtpointer for the last variable will be properly set. It should be noted that all variables in a receiving comlst are cleared when the RECV verb is processed. In addition, each variable is cleared when it is initially toggled to. Thus, if a message block is received in error, some variables may be left with non-zero lengthpointers which will not be reset properly unless the correct message block has a text length no less than the error block minus one.

4.5.16 CRGRVA - Get the Route Variable Address

This routine is provided to allow the communication process to access the route variable, for such applications as might require additional input parameters, resulting status indicators, etc. It is vitally important that once the route variable address is retrieved, all interaction with the actual data in the variable should take place during the same interrupt so that the route variable does not have opportunity to disappear (e.g. as a consequence of a chain operation by the background). The routine returns TRUE SIGN if the comlst has "gone away", or FALSE SIGN with the route variable address in the BC-register pair, the formpointer in the D-register, and the lengthpointer in the E-register.

4.5.17 CRSTRT,CREND,COMPAG - External Definitions

These external definitions define addresses used by the communication process. CRSTRT should be used as the starting address of the code (e.g. ORIGIN CRSTRT). The highest address reached by the code should not reach or exceed CREND. COMPAG is used by 5500-only processes and must always be the value in the X-register when any interface routine is called. The X-register is initialized with the value of COMPAG by the scheduler before any call is made to the external communication process.

4.5.18 WHAT HAPPENS WHEN "THE COMLST GOES AWAY"

The expressions "abnormal end of comlst" and "the comlst went away" refer to a condition where the data area (hence the variables, hence the comlst) for a given port is no longer valid. This condition arises as a result of a chain operation, whether planned (as with the CHAIN and STOP verbs) or unplanned (as with a disconnecting port or the unintentional use of the INT key). When this happens to the port that "owns" the current comlst in use by the communications process, it becomes necessary to "make the comlst go away" with respect to the logical operations of the MULTILINK interface routines. Each comlst has its own status byte that contains a 'gone away' indicator. All interface routines that interact with a comlst or its related variables check the "comlst gone away flag" before proceeding, and, if the test is positive, produce harmless results. The communications process thus does not need to worry about performing an operation upon-existent data. However, it should not ignore the "abnormal end of comlst" return conditions unless infinite loops are desired. Once a 'comlst gone away' condition has been discovered, a call should be made to CRSSCU in order to dequeue the comlst.

4.5.19 LOGICAL RELATIONSHIPS

The logical flow relationship among several of the MULTILINK interface routines may be clarified by the following skeleton communications process:

```
        SET     CRSTRT
START   CALL    CRCPS$
        LC      3
        CALL    CRCCF
        shake hands, etc.
```

```
                CALL    CRSAF
        GO      CALL    CRCS$
                CALL    CRRPGO
                JTZ     OK
                CALL    CRSSAF
                JMP     GO
        OK      CALL    CRGSCL
        LOOP    CALL    CRCS$
                CALL    CRGET
                JTS     DONE
                JTZ     DONE
                JTC     LOOP
                CALL    SENDIT
                JMP     LOOP
        DONE    CALL    CRSSOK
                JMP     GO
        SENDIT  .
                .
                POP
                JMP     NOSEND
                .
                .
                RET
NOSEND  CALL    CRCAF
                JMP     START
                IFGE    $,CREND
                ERR     COMM PROCESS TOO LONG
                XIF
                END     START
```

APPENDIX A. INSTRUCTION SUMMARY

SYNTACTIC DEFINITIONS

<label>	is a letter, followed by any combination of up to seven (7) letters and digits.
<string>	is any sequence of characters with the exception of the forcing character (#) which itself will not become part of the character sequence. The character following the # will become part of the character sequence (eg. another # or ").
<svar>	A name assigned to a statement defining a character string variable.
<nvar>	A name assigned to a statment defining a numeric string variable.
<ssvar>	A name assigned to a statment defining a source character string variable. This variable is unchanged as a result of the instruction.
<dsvar>	A name assigned to a statement defining a destination character string variable. This variable is generally changed as a result of the instruction.
<snvar>	A name assigned to a statement defining a source numeric string variable. This variable is unchanged as a result of the instruction.
<dnvar>	A name assigned to a statement defining a destination numeric string variable. This variable is generally changed as a result of the instruction.
<slit>	is a literal of the form "<string>".
<nlit>	is a literal of the form "<string>" where <string> is a valid numeric string.

<char>	is any single character of the form "<string>" where string is of length one (1).
<occ>	is an octal control character (001 to 037 inclusive).
<list>	Any combination of <slit>, <occ>, <list controls> <nvar> and <svar>, separated by commas. The list may be continued on more then one line by placing a colon (:) after the last statement on the line to be continued.
<cmlist>	A name assigned to a COMLST data declaration.
<nlist>	A list of numeric variables each pair of which is separated by a comma (,). The list may be continued on more than one line by placing a colon (:) after the last variable on the line to be continued.
<slist>	A list of character string variables each pair of which is separated by a comma (,). The list may be continued on more than one line by placing a colon (:) after the last variable on the line to be continued.
<nslist>	Any combination of numeric and string variables separated by commas. The list may be continued on more than one line by placing a colon (:) after the last variable on the line to be continued.
<blist>	The name assigned to the first of a set of physically contiguous numeric string or character string variables.
<index>	A numeric variable used in connection with list accessing.
<dnum>	A decimal number between -128 and 127.
<dnum1>	A decimal number indicating the number of digits that should precede the decimal

	point.
<dnum2>	A decimal number indicating the number of digits that should follow the decimal point.
<dnum3>	A decimal number between 1 and 20 inclusive.
<dnum4>	A decimal number between 1 and 64 inclusive.
<flag>	One of the following flags: OVER, LESS, ZERO, or EOS (EQUAL and ZERO are two names for the same flag) that are used to indicate the result of some DATABUS operation.
<event>	The occurance of a program trap: PARITY, RANGE, FORMAT, CFAIL, or IO.
<skey>	A numeric or character string variable used with SEARCH.
<DOS file spec>	A DOS compatible file specification (see DOS user's guide).
<file>	A name assigned to a FILE declaration.
<ifile>	A name assigned to a IFILE declaration.
<rn>	A numeric variable which contains a positive record number (>=0) used to randomly READ or WRITE a file.
<seq>	A numeric variable which contains a negative number (<0) used to READ or WRITE a file sequentially.
<key>	A non-null string variable used as a key to indexed I/O accesses.
<null>	A null string variable used as a key to an indexed read.
<route>	a string variable used for routing purposes. See DATABUS COMPILER User's Guide for more complete description.

FOR THE FOLLOWING SUMMARY:

Items enclosed in brackets [ ] are optional.

Items separated by the ¦ symbol are mutually exclusive (one or the other but not both must be used).

COMPILER DIRECTIVES

```
<label>  EQU       <dnum>
<label>  EQUATE    <dnum>
         INC       <DOS file spec>
         INCLUDE   <DOS file spec>
```

FILE DECLARATIONS

```
<label>  FILE
<label>  IFILE
```

DATA DEFINITIONS

```
<label>  FORM      <dnum1>.<dnum2>
<label>  FORM      <dnum1>.
<label>  FORM      <dnum1>
<label>  FORM      <nlit>
<label>  DIM       <dnum>
<label>  INIT      <slit>
<label>  FORM      *<dnum1>.<dnum2>
<label>  FORM      *<dnum1>.
<label>  FORM      *<dnum1>
<label>  FORM      *<nlit>
<label>  DIM       *<dnum>
<label>  INIT      *<slit>
<label>  COMLST    <dnum4>
```

CONTROL

```
GOTO      <label>
GOTO      <label> IF <flag>
GOTO      <label> IF NOT <flag>
BRANCH    <index><prep><list>
CALL      <label>
CALL      <label> IF <flag>
CALL      <label> IF NOT <flag>
ACALL     <nslist>
RETURN
RETURN    IF <flag>
RETURN    IF NOT <flag>
STOP
STOP      IF <flag>
STOP      IF NOT <flag>
CHAIN     <svar>
CHAIN     <slit>
TRAP      <label> IF <event>
TRAPCLR   <event>
ROLLOUT   <svar>
ROLLOUT   <slit>
PI        <dnum3>
TABPAGE
```

CHARACTER STRING HANDLING

MATCH	<svar><prep><svar>
MATCH	<slit><prep><svar>
MOVE	<ssvar><prep><dsvar>
MOVE	<sslit><prep><dsvar>
MOVE	<ssvar><prep><dnvar>
MOVE	<sslit><prep><dnvar>
MOVE	<snvar><prep><dsvar>
MOVE	<snlit><prep><dsvar>
APPEND	<ssvar><prep><dsvar>
APPEND	<sslit><prep><dsvar>
APPEND	<snvar><prep><dsvar>
CMOVE	<ssvar><prep><dsvar>
CMOVE	<char><prep><dsvar>
CMOVE	<occ><prep><dsvar>
CMATCH	<svar><prep><dvar>
CMATCH	<char><prep><dvar>
CMATCH	<svar><prep><char>
CMATCH	<svar><prep><occ>
CMATCH	<occ><prep><dvar>
BUMP	<dsvar>
BUMP	<dsvar><prep><dnum>
RESET	<dsvar><prep><dnum>
RESET	<dsvar><prep><ssvar>
RESET	<dsvar><prep><snvar>
RESET	<dsvar>
ENDSET	<dsvar>
LENSET	<dsvar>
CLEAR	<dsvar>
EXTEND	<dsvar>
LOAD	<dsvar><prep><index><prep><slist>
STORE	<ssvar><prep><index><prep><slist>
STORE	<sslit><prep><index><prep><slist>
CLOCK	TIME<prep><dsvar>
CLOCK	DAY<prep><dsvar>
CLOCK	YEAR<prep><dsvar>
TYPE	<svar>
SEARCH	<skey><prep><blist><prep><nvar><prep><dsvar>
REPLACE	<ssvar><prep><dsvar>
REP	<ssvar><prep><dsvar>
REPLACE	<sslit><prep><dsvar>
REP	<sslit><prep><dsvar>

ARITHMETIC

ADD	<snvar><prep><dnvar>
ADD	<nlit><prep><dnvar>
SUB	<snvar><prep><dnvar>
SUB	<nlit><prep><dnvar>
SUBTRACT	<snvar><prep><dnvar>
SUBTRACT	<nlit><prep><dnvar>
MULT	<snvar><prep><dnvar>
MULT	<nlit><prep><dnvar>
MULTIPLY	<snvar><prep><dnvar>
MULTIPLY	<nlit><prep><dnvar>
DIV	<snvar><prep><dnvar>
DIV	<nlit><prep><dnvar>
DIVIDE	<snvar><prep><dnvar>
DIVIDE	<nlit><prep><dnvar>
MOVE	<snvar><prep><dnvar>
MOVE	<nlit><prep><dnvar>
COMPARE	<nvar><prep><nvar>
COMPARE	<nlit><prep><nvar>
LOAD	<dnvar><prep><index><prep><nlist>
STORE	<snvar><prep><index><prep><nlist>
STORE	<nlit><prep><index><prep><nlist>
CHECK11	<svar><prep><svar>
CK11	<svar><prep><slit>
CHECK10	<svar><prep><svar>
CK10	<svar><prep><slit>

INPUT/OUTPUT

KEYIN	<list>[;]
DISPLAY	<list>[;]
BEEP	
PRINT	<list>[;]
RELEASE	
PREPARE	<file>,<svar|slit>
PREP	<file>,<svar|slit>
OPEN	<file|ifile>,<svar|slit>
CLOSE	<file|ifile>
WRITE	<file>,<rn|seq>;<;|<list>[;]>
WRITE	<ifile>,<rn|seq|key>;<;|<list>[;]>
WRITAB	<file>,<rn|seq>;<;|<list>[;]
WEOF	<file|ifile>,<rn|seq>
UPDATE	<ifile>;<;|<list>[;]
READ	<file>,<rn|seq>;<;|<list>[;]>
READ	<ifile>,<rn|seq|key|null>;<;|<list>[;]>
READKS	<ifile>;<;|<list>[;]>
DELETE	<ifile>,<key>

```
INSERT    <ifile>,<key>
COMCLR    <cmlist>
COMTST    <cmlist>
COMWAIT
SEND      <cmlist>,<route>;<nslist>
RECV      <cmlist>,<route>;<slist>
DEBUG
```

APPENDIX B. INPUT/OUTPUT LIST CONTROLS

CONTROL	USED IN	FUNCTION
*P<m>:<n>	KD	Causes the cursor to be positioned horizontally and vertically to the column and line indicated by the numbers <m> (horizontal 1-80) and <n> (vertical 1-24). These numbers may either be literals or numeric variables.
*N	KDP	Causes the cursor or printer to be positioned in Column 1 of the next line.
*EL	KD	Causes the line to be erased from the current cursor position.
*EF	KD	Causes the screen to be erased from the current cursor position to the bottom of the display.
*ES	KD	Causes the cursor to be positioned at horizontal position 1 of the top row of the display and the entire display to be erased.
*EOFF	K	Prevents character echo to the display during Keyboard input operations.
*EON	K	Causes character echo to the display during Keyboard input operations.
*+	KDP	Turn on Keyin Continuous for KEYIN or suppression of spaces after the logical length for DISPLAY and PRINT.
*+	W	Turn on space compression during WRITE.
*-	KDP	Turn off Keyin Continuous (turned off at the end of the statement) or the suppression of spaces after the logical length.
*-	W	Turn off space compression during WRITE.

*<n>	P	Causes a horizontal tab on the printer to the column indicated by the number <n>.
*<n>	RW	Tab specification for READ or WRITAB operations.
*<nvar>		The logical file pointers are moved to that character position relative to the current physical record.
;	KDP	Suppress a new line function when occurring at the end of a list.
*F	P	Causes the printer to be positioned to the top of form.
*L	KDP	Causes a linefeed to be displayed or printed.
*C	KDP	Causes a carriage return to be displayed or printed.
*T	K	Time out after 2 seconds have elapsed between successively entered characters for KEYIN statement.
*W	KD	Pause for one second.
*JL	K	Left-justify numeric variable and zero-fill at right if there is no decimal point. Left justify string variable and blank fill to ETX (zero fill if *ZF option is given).
*JR	K	Right-justify string variable and blank-fill at left (zero fill if *ZF option is given).
*ZF	KDPW	zero-fill string or numeric variable.
*DE	K	Restrict string input to digits (0-9) only.
*IT	K	Turn-on Text Mode (invert alphabetic input).
*IN	K	Turn-off Text Mode.

*MP	W	Convert numeric variable to "Minus-overpunch" format.

APPENDIX C. FREEDOM PRINTER CONSIDERATIONS

Secondary tractor on the Datapoint 9232 FREEDOM printer may be selected by the following method:

1. Initialize a string variable to an octal five followed by two ASCII characters representing the left margin location in hexidecimal.

2. Use the initialized string variable as the 1st variable in each print statement.

For example:

```
P2          INIT   05,"3F"
.
            PRINT  P2,*F,DATA1,*20,DATA2
```

selects the secondary tractor and sets the left margin at print position 63 (hexidecimal 3F) before performing top of form and printing. Tabbing (*20) will be relative to the left margin.

APPENDIX D. ERROR CODES

If an event occurs and the trap corresponding to that event has not been set, the message:

```
* ERROR * LLLLL X *        or
* ERROR * LLLLL X * Q
```

appears on the console display. The first form appears for all traps except I/O traps. In the event of an I/O trap, a qualification letter is given where a "Q" is shown in the example (explained below under the "I/O" trap). The LLLLL is the current value of the program counter and the X is an error letter. In most cases, LLLLL points to the instruction following the one that caused the problem. However, in certain I/O errors, LLLLL will point after the list item where the problem occurred. The following error letters can appear:

A - interruptions already prevented
B - illegal operation code
C - chain failure
F - record format error
I - I/O error
L - invalid command from slave station
O - object code read failure
P - parity failure
R - record number out of range
U - call stack underflow or overflow

Note that A, B, L, O, and U errors cannot be trapped. The B error will only snow up if somehow an invalid object file is executed or if the system is failing. The U error will happen if the programmer forgets to perform a call or in some other fashion manages to execute a RETURN instruction without a corresponding CALL having been previously executed, or calls are nested more than eight levels deep. The A error will happen if a PI instruction is executed while interrupts are currently prevented. The O error will happen if the object file is positioned out of range, has a parity fault, or if its drive goes off line. The L error will generally indicate that the telephone circuit is bad.

The events that may be trapped are shown below. The capitalized name is the one used in the TRAP statement.

PARITY	- disk CRC error during READ or disk CRC error during write verification (the DOS retries an operation up to 5 times to get a good CRC before giving up and causing this event).
RANGE	- record number out of range (an access was made that was off the physical end of the file, a record was read which was never written, or a WRITAB was used on a record which was never written)
FORMAT	- data being read into a numeric variable was not all digits and decimal point and minus sign, or decimal point in input does not agree with the decimal point in FORM, or data from disk has a negative multi-punch but no room for a minus sign in FORM, or write specified multi-punch and the last item of the field is a decimal point. The operation stops with the item in error and the statement is aborted.
CFAIL	- the specified program was not in the DOS directory or a ROLLOUT was attempted with one of the necessary system files missing, or a program containing compile-time errors was loaded.
IO	- Error during I/O statement. Either a programming error or disk failure can cause this TRAP.

APPENDIX E. INTERPRETER I/O TRAP CODES

A - an access sequentially by key, with a null key or an UPDATE was attempted before any indexed sequential access was made using the logical file.
B - the READ mechanism ran off the end of a sector without encountering a physical end of record character (003).
C - an operation on a closed logical file was attempted.
D - a non-READ non-DELETE indexed sequential operation was attempted where the specified key already exists in the index.
E - an EOF mark without at least four zero's was encountered.
F - disk file space full.
I - the index file specified in an OPEN statement does not exist on the specified drive(s).
J - the index file found by the OPEN statement does not reside in the correct physical location on the disk (index files may never be moved, they must always be re-created).
K - a null key was supplied in an operation where the key may not be null.
M - the data file specified does not exist on the specified drives(s) or the specified drive is off-line.
N - the data file name specified in the OPEN or PREPARE statement was null.
O - the index file name specified in the OPEN statement was null.
P - the file specified in the PREPARE statement had some type of DOS protection (either write, delete, or both).
T - the tab value in the READ or WRITAB statement was off the end of the sector.
U - an EOF mark was encountered while a record was being deleted in the indexed sequential file.
W - an index file pointer sector could not be read.
X - an index file header sector could not be read.
Y - the R.I.B. of the data file pointed to by the index file could not be read. (VWXY errors can be caused by parity errors, the drive being switched off line, or the disk cartridge being swapped with another while an operation is taking place.)

APPENDIX F. USE OF COMMON TO RECOGNIZE ERRORS

Untrapped DATABUS errors may be recognized by the MASTER program through the use of Common Data Area. (The method for trapping DATABUS errors is described in the DBCMP manual.)

When an untrapped DATABUS error occurs the interpreter moves the following string into the first 14 bytes of the user's data area:

oct	oct	asc	asc	asc	asc	asc	asc	asc	asc	asc	asc	oct
013	001	n	n	n	n	n		t		*	q	203

where: nnnnn - is the location in the DATABUS program where the error occured.
t - is a letter representing the type of error.
q - is a letter used to qualify the type of error.

The string described above is moved into the user's data area after the error occured, but before the MASTER program is executed. This allows one of the following actions to occur:

a) If the first 14 bytes of the MASTER program's data area is <u>not</u> declared to be common, the string described above is overwritten when the MASTER program is loaded.

b) If the first 14 bytes of the MASTER program's data area is declared to be common, the string described above may be accessed by the MASTER program just like any other information passed through common.

Sample MASTER program:

```
ERROR    DIM       *11                 LEAVE 1ST 14 BYTES COMMON
...      ...       ...                 MORE DATA AREA GOES HERE
* BEGINNING OF EXECUTABLE CODE
.
         RESET     ERROR TO 9          POINT AT THE "*"
         CMATCH    "*" TO ERROR        IF NOT "*", THEN NO ERROR
         GOTO      OKAY IF NOT EQUAL
...      ...       ...                 ERROR PROCESSING GOES HERE
OKAY     ...       ...                 NORMAL PROCESSING HERE
```

APPENDIX G. PROGRAM EXAMPLES

The MASTER program merely requests the name of a program to be executed. A CHAIN is executed to the name given and if a chain failure occurs an indication is given that the name does not exist in the DOS directory and and another request for a program name is made. Note that the MASTER program is written without the use of cursor positioning in the KEYIN and DISPLAY statements to aid in a Teletype terminal compatibility. The entry of a "*" for the program name causes the system to hang up the phone which provides a normal termination using the DATABUS DSCNCT instruction.

Simple MASTER Program

```
.
. SIMPLE MASTER PROGRAM
.
PORTN    FORM     " 4"
FILNAM   DIM      8
.
         RELEASE
LOOP     KEYIN    *N,*EL,"PROGRAM NAME: ",FILNAM
         CMATCH    "*" TO FILNAM
         GO TO    DISCON IF EQUAL
         TRAP     NONAME IF CFAIL
         CHAIN    FILNAM
NONAME   DISPLAY  "*** NO SUCH PROGRAM ***"
         GOTO     LOOP
DISCON   DSCNCT
```

For a more complex example of ANSWER and MASTER programs see the DATABUS COMPILER User's Guide.

Manual Name_____

Manual Number_____

READER'S COMMENTS

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____

All comments and suggestions become the property of Datapoint.

Fold Here

Fold Here and Staple

First Class
Permit
5774
San Antonio
Texas

BUSINESS REPLY MAIL
No Postage Necessary if mailed in the United States

Postage will be paid by:

DATAPOINT CORPORATION
Product Marketing
8400 Datapoint Drive
San Antonio, Texas 78284