SYS.MCBF

MDBS - CP/M Interface Notes

CP/M with MICROSOFT Compiled BASIC/FORTRAN

```
mm       mm   ddddddd     bbbbbbb      ssssss
mmm     mmm   ddddddddd    bbbbbbbbb   ssssssss
mmmm   mmmm   dd      dd   bb      bb  ss      ss
mm mmmm mm    dd      dd   bb      bb  ss
mm  mm  mm    dd      dd   bbbbbbbbb   ssssssss
mm      mm    dd      dd   bbbbbbbbb    ssssssss
mm      mm    dd      dd   bb      bb         ss
mm      mm    dd      dd   bb      bb  ss      ss
mm      mm    ddddddddd    bbbbbbbbb   ssssssss
mm      mm    ddddddd      bbbbbbb      ssssss
```

Micro Data Base Systems, Inc.

P. O. Box 248

Lafayette, Indiana 47902

(317) 448-1616

(317) 742-7388

August 1980

CP/M is a registered trademark of the Digital Research Corporation.

I. INTRODUCTION

The MDBS.DDL and MDBS.DMS User's Manual was written in a non-machine-dependent manner. The purpose of the manual is to discuss the specifics of running the MDBS packages on CP/M systems with MICROSOFT's compiled BASIC-80 and FORTRAN-80 or related products.

The following files are on the two MDBS diskettes[1]:

(a) MDBS.DDL

1. DDL.COM - An absolute version of MDBS.DDL loading at 100 hex
2. INVNTRY - A sample data description file
3. RLC.COM - An absolute relocator
4. RELOC.IHF - A relocatable relocator loading at 100 hex.
5. DDL.IHF - A relocatable version of the MDBS.DDL system

(b) MDBS.DMS

1. DMS.REL - A relocatable version of the MDBS.DMS system
2. SAMPLE.BAS - Sample BASIC Program
3. SAMPLE.FOR - Sample FORTRAN Program
4. SAMPLDDL - DDL file for use with sample programs

MDBS.DDL is your personalized Data Definition Language Analyzer/Editor and MDBS.DMS your Data Management System. DDL loads at 100H and extends to approximately 3400H. It uses standard CP/M I/O entry points. If you are using nonstandard I/O entries or require other

[1]Depending on the size of the diskettes used, the files on the two diskettes may actually be organized differently.

patching, refer to Section IV for patching procedures.

MDBS.DMS is supplied in a relocatable format compatible with the object files produced by the MICROSOFT BASIC-80 and FORTRAN-80 compilers. The program area for the MDBS.DMS system is approximately 4400[1] (hex) bytes long. Additionally, memory is required for the various tables and buffers required by the system. This memory space is allocated by the calling program so that the user may easily change the amount of memory available to the MDBS.DMS system. Since increased memory available to MDBS.DMS will result in greater system throughput, maximization of available memory is encouraged.

In Section VI we show how to call the DMS routines from MICROSOFT BASIC-80 and FORTRAN-80.

Finally, in Section VII we provide some general notes for the CP/M user of the MDBS package.

You can run your version of DDL by following the procedure given in Figure 1. At the end of Step 2 you are in the DDL program. Refer to Section II.E of the MDBS user's manual for the DDL commands. However, to quickly see some action, continue with the procedure in Figure 1.

Steps 3-5 result in the reading of a sample data base description from the file INVNTRY. Step 6 results in the listing of the sample description. Step 8 returns control to the operating system.

[1] The size of the MDBS.DMS system will vary depending on the functions available and the operating system/host language selected.

Figure 1

Sample DDL execution

STEP		PROCEDURE/RESULTS
1.	you	DDL
2.	Computer	MDBS.DDL VER X.X
		(C) COPYRIGHT 1980, Micro Data Base Systems, Incorporated
		Reg # XXXXX
		your name and address
3.	you	R
4.	Computer	FILENAME
5.	you	INVNTRY
6.	you	L
7.	Computer	listing of sample data description
8.	you	BYE
9.	Computer	A>

(return to Monitor)

II. RELOCATING MDBS.DDL

To relocate and produce an executable form of MDBS.DDL and MDBS.DMS, we have provided an executable relocator and a relocatable form of the relocator. RLC.COM loads at 0100H. To use it merely enter the sequence of lines shown in Figure 2 at your terminal. After performing this sequence of steps you may want to save the file. If a relocator is needed at an address other than 0100H refer to Section III.

Figure 2

Producing an Executable Form of MBDS.DDL

Step	Procedure	Example
1. you	RLC	RLC
2. Computer	INPUT	
3. you	DDL.IHF	DDL.IHF
4. Computer	LOAD ADDRESS	
5. you	yyyy (,zzzz)	2D00
6. Computer	tttt	
(return to Monitor)	>	

Explanation:

This sequence of steps produces an executable form of MDBS.DDL in memory starting at memory location yyyyH + zzzzH and ORGed at yyyyH. The ",zzzz" is optional. The high memory address used by the routine is indicated by the "tttt" response from RLC. The user should insure that the relocator and the program locations are not in conflict with one another.

Following this procedure, the user may want to make patches to the executable program (as outlined in Section II.C.3 of the MDBS user's manual and in Sections IV and V of this manual) and then save the resulting executable program.

III. GENERATING A NEW RELOCATOR

To produce a relocator at an address other than 0100H follow the sequence of steps shown in Figure 3.

Figure 3

Producing an Executable Relocator

Step	Procedure	Example
1. you	RLC	RLC
2. Computer	INPUT	
3. you	RELOC.IHF	RELOC.IHF
4. Computer	LOAD ADDRESS	
5. you	yyyy (,zzzz)	6000
6. Computer	tttt	
(Return to Monitor)	>	

Explanation:

This sequence of steps produces an executable form of RELOC.IHF in memory starting at memory location yyyyH + zzzzH and ORGed at yyyyH (the "zzzzH" is optional). Executing RLC.COM results in its loading and subsequent execution at 0100H and the user must be careful not to have a memory conflict between RLC.COM and the newly formed RLC which will be placed at yyyyH + zzzzH.

The user may then want to save this program.

IV. MDBS.DDL PATCHING

MDBS.DDL consists of a program region and work area region. These are contiguous and the work area immediately follows the program area. The size of the work area can be increased or decreased through a patch to the system.

There are several addresses that the user should be aware of in MDBS.DDL. Figure 4 maps out these areas. A brief description of each item follows. All address patches should be made with normal 8080 conventions, i.e. the lower 8 bits of the address precede the higher 8 bits.

(a) Initial Entry Point

Upon entry at this point, all program variables and regions are either physically or logically re-initialized. Registers are not saved but the entering stack is preserved.

(b) BDOS Jump (Default 0005 hex)

This is the address of the disk management and peripheral processing routines.

(c) CP/M Warm Entry Jump (Default 0000 hex)

This address is the warm entry point to CP/M.

(d) FCB Location (Default 005C hex)

This points to the File Control Block area.

(e) BUFF Location (Default 0080 hex)

This points to a 128 byte buffer area.

(f) Reserved Regions

This area is reserved for internal use by the MDBS.DDL routines. It should not be altered.

(g) Echo Toggle (Default 01 hex)

This byte is checked to see if the user wants to have MDBS.DDL echo input to the output device. If the byte value is zero, echoing will take place. If it is the value one, no echoing will be performed.

(h) Size of BASIC Integer (Default 02 hex)

The value in this field gives the number of bytes required to store a numeric element in Microsoft BASIC. This value should be 2.

(i) Size of BASIC Single Precision Variable (Default 04 hex)

This field must contain the size of a Microsoft BASIC single precision variable. This value is 4.

(j) Last Word of Memory (Default OBFFF hex)

The address stored here gives the last available word of memory that the MDBS.DDL program may use. Note that MDBS.DDL uses all memory starting from its load address up to the value in this

field. Needless to say, the user should make sure that the last word of memory is physically beyond the end of the program.

(k) Screen Control Byte (Default OB hex)

This byte should have one of the following values:

Screen Width	Byte Value
less than or equal to 64 characters	11 (OB hex)
greater than or equal to 80 characters	15 (OF hex)
64 to 80 characters per line (call it N)	greatest integer less than or equal to: N/5 -1

(l) Re-entry Point

The user may re-enter MDBS.DDL while preserving all program variables and regions by issuing a jump to this address.

Figure 4

MDBS.DDL ADDRESSES

Address	Description	Default Value
0100 (hex)	Initial Entry Point	--
0103 (hex)	BDOS Jump	0005 (hex)
0106 (hex)	CP/M Warm Entry	0000 (hex)
0108 (hex)	FCB Location	005C (hex)
010A (hex)	BUFF Location	0080 (hex)
010C-0126	Reserved	--
0127 (hex)	Echo Toggle	01 (hex)
0128 (hex)	BASIC Integer Size	02 (hex)
0129 (hex)	BASIC Single Precision Size	04 (hex)
012A (hex)	Last Word of Memory	BFFF (hex)
012C (hex)	Screen Control Byte	11 (hex)
0139 (hex)	Re-Entry Point	--

V. INTERFACING MICROSOFT BASIC-80 AND FORTRAN-80

A. MDBS.DMS Entry Points

The MDBS.DMS system has 4 host language callable entry points. The entry points are:

DMS - The primary entry point for requesting DMS processing. All DMS commands described in the MDBS User's Manual (except DEFINE and EXTEND) are performed by calling this entry point.

DMSDEF - The entry point for the DEFINE and EXTEND commands.

SETPBF - An entry point which must be called prior to the first call to DMS or DMSDEF to allocate memory for use by the MDBS.DMS system.

ALTEOS - This entry is used by FORTRAN programs to permit the use of three branch (arithmetic) IF's for error checking (See below).

The same version of the DMBS.DMS system can be used with both BASIC and FORTRAN. A parameter specified on the call to SETPBF indicates in which language the calling program has been written (0 indicates BASIC, 1 FORTRAN).

B. BASIC-80 Interface

To call the MDBS.DMS system from BASIC-80 the following procedures must be used. First the user must define the working memory available for use by MDBS.DMS. This is done by the SETPBF entry described in Part D below. The calling sequence used is:


```
10 DIM B9%(2048)
15 N% = 4096
20 I% = 0
25 CALL SETPBF (B9%, N%, I%)
```

Note that each element of the integer array B9% occupies two bytes; thus, the call to SETPBF indicates that 2*2048, or 4096, bytes are available for use.

To define a data block, a call of the following form is used:

```
30 N% = 4
40 C$ = "DEFINE,OPENBLK"
50 CALL DMSDEF (E0%, C$, N%, O1$, O2$, O3$, O4$)
```

This defines a data block named "OPENBLK" with four string variables (O1$, O2$, O3$, O4$). E0% is an integer variable which returns the error status (equivalent to E0 in the MDBS User's Manual) and N% is a count of the number of variables passed.

Finally, a call is made to a standard DMS routine (OPEN in this case) by a call of the form:

```
60 O1$ = "SAMPLEDB.DB"
70 O2$ = "USER"
80 O3$ = "PASSWORD"
90 O4$ = "MOD"
100 C$ = "OPEN,OPENBLK"
110 CALL DMS (E0%, C$)
```


O1$ through O4$ are string variables which comprise the parameters for the OPEN call. The actual function to be performed is stored in string variable C$ and is passed in the call to DMS along with the error response variable, E0%.

C. FORTRAN-80 Interface

To call the MDBS.DMS system from FORTRAN-80 the following procedures must be used. First the user must define the working memory available for use by MDBS.DMS. This is done by the SETPBF entry described in Part D below. The calling sequence used is:

```
LOGICAL DMSBUF (4096)
CALL SETPBF (DMSBUF, 4096, 1)
```

To define a data block, a call of the following form is used:

```
REAL O1(3), O2(4), O3(3), O4
DATA O1 / 4HSAMP,4HLEDB,4H.DB  /, O2 / 4HUSER,3*4H     /,
*     O2 / 4HPASS,4HWORD,4H     /, O4 / 4HMOD  /
CALL DMSDEF (IER, 'DEFINE,OPENBLK.', 4, O1, O2, O3, O4)
```

This defines a data block named "OPENBLK" with four character variables (O1, O2, O3, O4). IER is an integer variable which returns the error status (equivalent to E0 in the MDBS User's Manual) and the 4 is a count of the number of variables passed.

Finally, a call is made to a standard DMS routine (OPEN in this case) by a call of the form:

```
CALL DMS (IER, 'OPEN,OPENBLK.')
```

Note that the command string ('OPEN,OPENBLK.', etc.) must always be terminated by a period in FORTRAN calls. This is due to the fact that FORTRAN, unlike BASIC, does not have a facility to pass the length of a literal string. The period provides the means for MDBS.DMS to determine the length of the string.

D. Use of SETPBF

As mentioned earlier, the SETPBF entry point passes to the MDBS.DMS system a contiguous section of memory. This memory segment is used to store tables and buffers for use by the MDBS.DMS system and MUST NOT BE ALTERED by the user's program. Since the MDBS.DMS system uses a dynamic buffering scheme, it is important that as much memory as possible be allocated for use by the MDBS system (we recommend that at least 4000 bytes be allocated). Since the amount of memory available to the MDBS.DMS system can radically affect the throughput of the system (more memory implies more disk buffers which implies fewer disk accesses), it is usually well worth your while to do a compilation and a LINK to determine the amount of memory used by the object program. The host language array can then be dimensioned to allocate most of the free memory for by MDBS.DMS. Note, however, that FORTRAN-80 allocates file buffers dynamically and sufficient memory must be left for this purpose.

The third parameter in the SETPBF call is a flag indicating whether the calling program is written in FORTRAN or BASIC. A value of 0 indicates BASIC; 1 indicates FORTRAN.

E. Use of ALTEOS

The error response (IER in our FORTRAN examples, E0 in the MDBS User's Manual) can in may cases be thought of as being logically divisible into the following three groups:

1. No problems (IER = 0)
2. End of Set Encountered, or Key Not Found (IER = 255)
3. Error Encountered (anything else)

Usually an error of the third type indicates a programming problem. An error of the second type can be expected from a number of commands (FMSK, SOM or FNM) and do not indicate a programming error, but rather the end of a set processing loop. Since in FORTRAN the three branch arithmetic IF exists, it is convenient to alter the 255 response from the DMS for use with such an IF statement. The entry ALTEOS has been defined for this purpose. After ALTEOS has been called, the normal 255 error response is returned as -1. This makes the task of examing the error response simpler for the FORTRAN programmer. Please refer to the sample program (SAMPLE.FOR) which is included with the MDBS system disks for an example of this feature. Subsequent calls to ALTEOS toggle this feature.

VI. LINKING THE HOST LANGUAGE PROGRAM AND MDBS.DMS

The MDBS.DMS system (on the DMS.REL file) may be linked with your compiled source program by using the LINK-80 linker. All that is necessary is to load the DMS and your program as shown in Figure 5. Steps 1 through 5 show how to load the programs. Step 7 demonstrates the method used to save the linked program as a COM file.

Figure 5

Linking the MDBS.DMS system

Step		Procedure	Example
1.	you	invoke linker	L80
2.	Computer	*	*
3.	you	load DMS	DMS
4.	Computer	*	*
5.	you	load program	BIN
6.	Computer	*	*
7.	you	save COM file	BIN/N
8.	Computer	*	*
9.	you	exit linker	/E

VII. GENERAL NOTES

1. Replicated items (see Section II.D.5 of the User's Manual) are given and returned to MDBS.DMS from the host language using arrays.

2. Binary and Integer Item Types are supplied from (and to) FORTRAN through INTEGER variables. CHARacter items in MDBS are represented in FORTRAN by appropriate use of LOGICAL, INTEGER or REAL arrays. The MDBS system views the character data as a sequence of bytes, so the representation selected in the FORTRAN program is irrelevant so long as a sufficient number of bytes have been allocated. LOGical Item Types correspond exactly to LOGICAL variables in FORTRAN. REAL Item Types are supplied through Single Precision and Double Precision variables. In the DDL Analyzer, a blank item size for a REAL variable results in a single precision default. Double Precision variables result when the appropriate item size (8) is specified.

3. Binary and Integer Item Types are supplied from (and to) BASIC through INTEGER variables. CHARacter items in MDBS correspond to string variables in BASIC. LOGical variables may be accessed from BASIC, but their use is undefined. REAL Item Types are supplied through Single Precision and Double Precision variables. In the DDL Analyzer, a blank item size for a REAL variable results in a single precision default. Double Precision variables result when the appropriate item size (8) is specified.

4. In BASIC, when retrieving character strings from a data base, the user should make sure that the destination string variable has been previously initialized to a length sufficiently large to hold the incoming string. If this is not done, the incoming string will be truncated. Further, the string should not be initialized by setting it equal to a program literal. This can cause the literal in the program to be modified with undesirable results. It is recommended that the string variable be initialized with the SPACE$ function. Alternatively, the FIELD command may be used to insure that adequate space has been allocated for a string.

5. Note that two byte integer variables run from -32767 to 32767 and not from -32768 to 32767.

6. In the DDL Analyzer, a new data base can be initialized on any drive. However, the initialized disk will have to reside on drive A during data base operations.

7. In the DDL Analyzer, when no drive is specified for a file, the last referenced drive is the default.

8. DEFINE and EXTEND calls are limited to 4 variables per call. Successive calls to EXTEND should be used to set up long data blocks.

Appendix: Using MDBS to implement an application system.

Figure A provides an overview of the four phases involved. The executable form of MDBS.DDL is used to define a data base. Application programs that access the data base must be compiled and linked to a relocatable form of MDBS.DMS (supplied on the DMS.REL file).

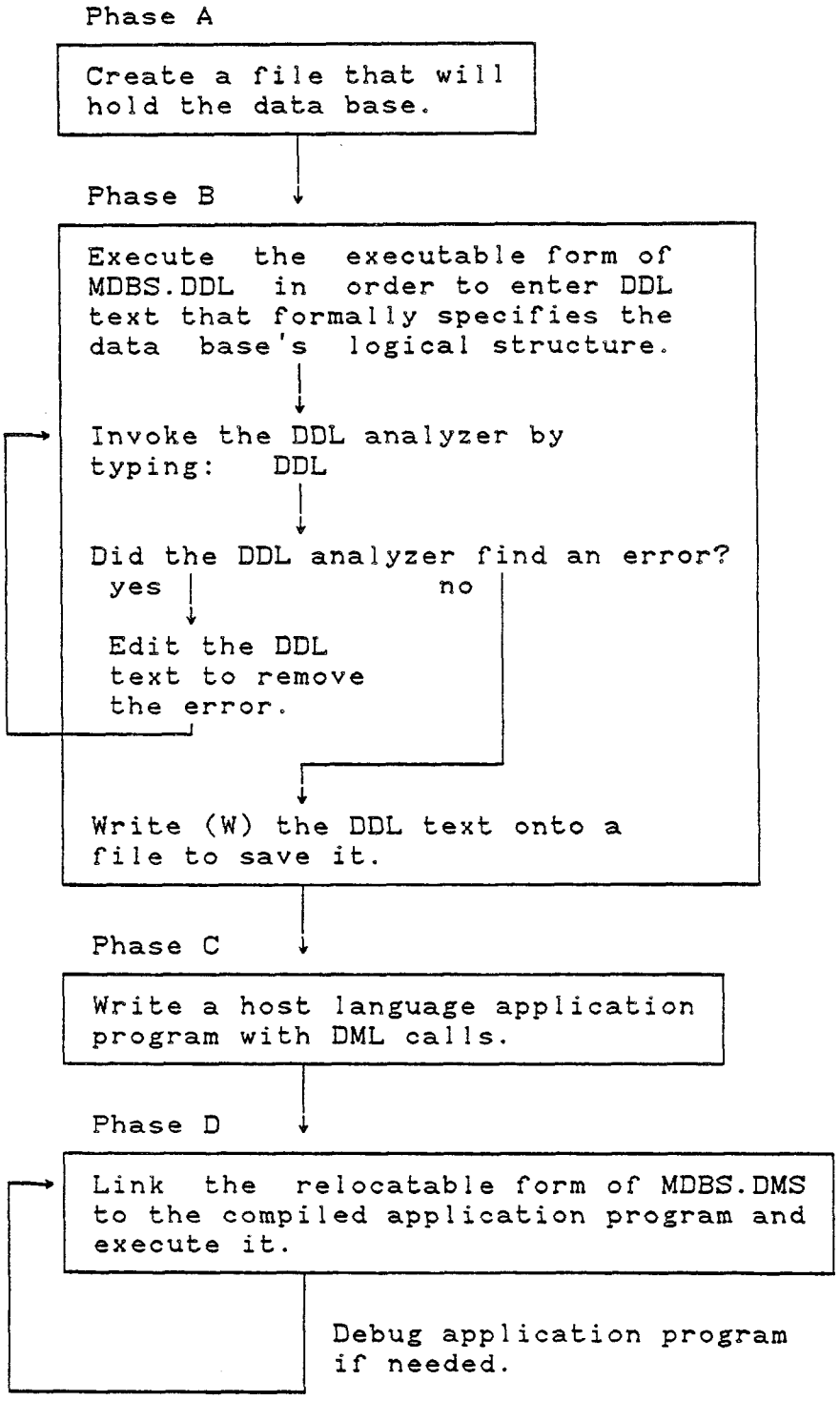

Figure A. Using MDBS to Implement an Application System

Phases A-D: Details

Phase A:

Create a file that will hold the data base. This file must have the same name as given on the FILE line in the DDL description. This file must be created on drive 1 (and on any other drives specified with the DRIVE card(s) in the DDL).

Phase B:

Execute the executable form of MDBS.DDL that was supplied (or which you have created in Figure 2). Now you are within the MDBS DDL system, so you can:

(a) Read a file of DDL source text (e.g.,the files INVNTRY and SAMPLDDL supplied on the diskettes you received).

(b) Generate a file of DDL source text for your own data base application (text entry aids are provided).

(c) Edit a file of DDL source text.

(d) Write (i.e., save on disk) a file of DDL source text.

(e) Submit a file of DDL source text to the DDL analyzer which checks your DDL text for correctness. Diagnostics are issued for errors; otherwise the message DDL PROCESSING COMPLETE is returned, indicating that the data base described in the DDL source text is now defined for the file created in Phase A. This data base file now contains a data base directory composed of all schema information.

Commands to accomplish the foregoing kinds of activities (along with descriptions of error messages returned from the DDL analyzer) are described in the MDBS User's Manual.

Phase C:

Write host language application programs (containing MDBS DML calls) for a data base defined in Phase B(e). These programs are used to add data to the data base, extract data from the data base, change data and their relationships within the data base, and/or delete data from the data base. (The SAMPLE file on the diskettes you received is an example application program for a data base whose source DDL text exists on the SAMPLDDL file.) The method for calling DML commands from a host language depends upon the nature of the host language. The necessary calling protocol for your host language is illustrated in the MDBS system specific manual.

The nature of each MDBS DML command that can be called from your host language is described in the MDBS User's Manual. Errors that can result from the incorrect DML usage are described for each command. It is suggested that you follow through the code of SAMPLE in order to become oriented to DML applicatins programming with your host language.

Phase D:

Link the compiled application program with the relocatable MDBS.DMS provided in the DMS.REL file (see Figure 5). Execute the result.

