Two MBASIC Programs That Write
Application Programs for Accessing a
Database

R. M. Smith
DSN Facility Operations Section

A method was desired to relieve the tedium of writing and testing application
programs. Two utility programs were developed to produce application
programs that perform relational operations on data. No coding is performed by
the user.

1. Introduction

The nature of the MBASIC Processor facilitates its use
by people who are not primarily programmers and who
may never have had any previous programming experi-
ence. Many managers, and other people needing to make
use of management data, fall into this category of user.
However, for many data users, programming is so much
less important than data usage that it would be beneficial
to reduce or eliminate programming in their data
activities. One approach to minimizing programming time
is to make use of a generalized application program
similar to the one described in a previous article (Ref. 1).
A generalized program, once written, allows the user to
concentrate upon data usage rather than program writing.
However, a generalized program requires that the user
spell out an access strategy each time the program is used.
If a specific application is required for repeated use, then
a specialized application program is most desirable. Two
different approaches to the use of specialized application
programs were described in Refs. 2 and 3. This article
describes the result of some preliminary efforts to design a
simple method of producing an MBASIC application
program while isolating the user from the task of writing
code.

Application programs that extract data from a database
may be data dependent (knowledge of data organization is
built into the application program, making the program
sensitive to changes in data organization) or may possess
varying degrees of data independence (immunity of an
application program to changes in data organization). The
greater the degree of data independence, the less the
effect of changes in the database. The programs described
in this article are preprocessors that accept relational
statements from a user, convert the statements to
MBASIC code, and store the statements as an application
program for later use.
II. Description of the Program 'WRITER'

'WRITER' produces specialized application programs with a moderate amount of data independence. User input to the program is fully prompted and makes use of relational operations (Refs. 4, 5) to specify database access. The relational operations may be invoked in any order repeatedly or used singly. Figure 1 illustrates a session invoking a restriction and a projection in sequence. Figure 2 illustrates the same process in relational notation. Figure 3 is a simplified flow diagram of the program showing the iterative nature of the main program in accepting user input and choosing a specified subprogram to assemble program statements.

In a typical sequence of events, the user

1. Enters the name of the new program to be written.
2. Enters the type of relational operation.
3. Enters the parameters that describe the selected relational operation.

'WRITER' then assembles a program by the following process:

1. Copies a set of generalized code lines to a temporary program file.
2. Appends code lines (to the temporary program file) that are created by 'WRITER' and are specific to the user's application.
3. Appends standardized subroutines (stored for this purpose) for each relational operation involved.
4. Copies the temporary program to a file named by the user.

Figure 4 is a copy of the program produced by 'WRITER' using the process depicted in Fig. 1. Line 100 and lines 902 through 918 are the generalized code lines mentioned previously. Lines 2000 through 2320 and 3000 through 3310 are standardized subroutines for a projection and restriction, respectively. (Lines 5000 through 5090 update a temporary directory relation that describes data files accessed by the application program.)

III. Description of the Program 'WRITPR'

'WRITPR' produces specialized application programs with no data independence. User input is fully prompted and makes use of relational operations (Refs. 4, 5) to specify database access. Figure 5 illustrates a session invoking a restriction and projection (see relational notation in Fig. 2). Figure 6 is a simplified flow diagram of the program. The user prompting sequence for 'WRITPR' is similar to that of 'WRITER' but 'WRITPR' differs in its approach to writing the application program (compare Figs. 3 and 6). 'WRITPR' assembles a program by writing code lines (on the program named by the user) that are specific to the user's application. Figure 7 is a copy of the program produced by 'WRITPR' using the process depicted in Fig. 5.

IV. Miscellaneous Information

To produce code that is specific to the user's application, both programs use "WRITE ON" statements that incorporate variables and "counters" into a completed statement for the application program. Examples of this process are shown in Figs. 8, 9, and 10 and are taken from the program 'WRITPR'. Figure 8 shows the lines of code that produce lines 110 and 120 of the program presented in Fig. 7. The code lines in Fig. 9 produce line 130 of Fig. 7, and the code lines in Fig. 10 produce line 150 of Fig. 7.

Each of the sessions (illustrated in Figs. 1 and 5) requires approximately 3 to 4 min of terminal time and produces programs that are fully functional, requiring no testing of the MBASIC code. Both provide the user with a uniform, extremely simple process for data access. Figure 11 illustrates the data output produced by the application programs written by 'WRITER' and 'WRITPR'.

Data files accessed by these programs must be in, at least, first normal form. The file used in this article (Fig. 12) is in third normal form (Ref. 4).
References


Fig. 1. Illustration of a session at a terminal using 'WRITER' to create an application program (User responses are to the right of each colon.)

```
RUN
ENTER NAME OF NEW PROGRAM: INEW

ENTER RELATIONAL OPERATION (R,P,J) OR "STOP": R
SOURCE RELATION: RMS*EQPT.ASSIGNMENT
DOMAIN NAME: OWNER
LOGICAL OPERATOR: =
DOMAIN VALUE: IC
PROJECT TO: =FILE

ENTER RELATIONAL OPERATION (R,P,J) OR "STOP": P
SOURCE RELATION: =FILE
QUANTITY OF DOMAINS: 3
TARGET RELATION: TERMINAL
DOMAIN NAMES: CON, OPSTAT, LOCATION

ENTER RELATIONAL OPERATION (R,P,J) OR "STOP": STOP
TEST INEW
```

Fig. 2. Relational notation describing the process illustrated in Figs. 1 and 5

RMS*EQPT.ASSIGNMENT*OWNER = 12
π TERMINAL (CON, OPSTAT, LOCATION)

Fig. 3. Simplified flow diagram of 'WRITER'

\[ \text{WRITER} \rightarrow \text{SELECT NEW PROGRAM NAME} \rightarrow \text{COPY "STANDARD" CODE TO NEW PROGRAM} \rightarrow \text{SELECT RELATIONAL OPERATIONS} \rightarrow \text{GOSUB} \rightarrow \text{STOP?} \rightarrow \text{STOP} \]

\[ \text{GOSUB} \rightarrow \text{SET VARIABLES FOR RELATIONAL OPERATION} \rightarrow \text{WRITE CODE ON NEW APPLICATION PROGRAM} \rightarrow \text{APPEND RELATIONAL SUBROUTINES} \rightarrow \text{RETURN} \]
Fig. 4. Data independent application program produced by using 'WRITER'
3000 OPEN "*STD1", OUTPUT, 3
3002 FLG0=0
3005 DIM FS(1), D(1), T(1), ID(1)
3010 OPEN DIR$, INPUT, 4
3020 AT END FILE(4) GO TO 3050
3030 INPUT FROM 4: FS(1), D(1), T(1), ID(1)
3035 STRING IDM(T(1))
3040 IF FS(1)=FLG$ THEN WRITE ON 3; D(1): '/' ; ID(1) ELSE GO TO 3050
3045 FLG0=FLG0+1
3050 IDENTIFIER = IDG(4) = ID(1)
3055 IF FS(1)=FLG$ THEN GO TO 3030
3060 N=D(1)
3070 IF T(1)=D(1) THEN GO TO 3090
3080 GO TO 3030
3090 CLOSE 3, 4
3100 DIM DI(1), DS(1)
3110 STRING AS(N)
3120 OPEN "*STD1", INPUT, 3
3130 AT END FILE(3) GO TO 3170
3140 INPUT FROM 3: DI(1), DS(1)
3150 IF DS(1)=ID$ THEN GO TO 3170
3160 GO TO 3140
3170 CLOSE 3
3180 =DI(1)
3190 IF LO$='=' THEN R=0 ELSE GO TO 3210
3200 GO TO 3240
3210 IF LO$='>' THEN R=1 ELSE GO TO 3230
3220 GO TO 3240
3230 IF LO$=CHAR(60) THEN R=-1
3240 OPEN "*HOLD1", OUTPUT, 5
3250 OPEN FLG$, INPUT, 4
3260 AT END FILE(4) GO TO 3300
3270 INPUT FROM 4: AS$ FOR I=1 TO N
3280 IF COMP(AS$(I), DV$)=R THEN WRITE ON 5 USING "<#>" ; AS$(I) ; CHAR(13)
3290 GO TO 3270
3300 CLOSE 5, 6
3310 RETURN

5000 OPEN "*TRANSF", OUTPUT, 1
5010 WRITE ON 4: RE$ ; I ; IDG ; IDM(I) FOR I=1 TO FLG0
5020 CLOSE 1
5030 APPEND "*TRANSF" TO DIR$
5040 RETURN

Fig. 4 (contd)
LOAD 'WRITPR'
RUN
ENTER NAME OF NEW PROGRAM: DNEW
SELECT A RELATIONAL OPERATION (REST, PROJ, JOIN) OR ENTER "STOP": REST
SOURCE RELATION: RMS*EOPT, ASSIGNMENT
DOMAIN NAME: OWNER
LOGICAL OPERATOR: =
DOMAIN VALUE: 12
PROJECT TO: TERM
PROJECT ALL DOMAINS? N
ENTER QUANTITY OF DOMAINS: 3
ENTER NAMES OF DOMAINS: CON, OPSTA, LOCAT
SELECT A RELATIONAL OPERATION (REST, PROJ, JOIN) OR ENTER "STOP": STOP
TEST 'DNEW'
END OF RUN

Fig. 5. Illustration of a session at a terminal using 'WRITPR' to create an application program
(User responses are to the right of each colon.)

Fig. 6. Simplified flow diagram of 'WRITPR'
Fig. 7. Data dependent application program produced by using 'WRITPR'

```
COPY 'DNEW' TO TERMINAL
100 STRING CON1,OWNER1,LOCAT1,OPSTA1,RECID1
110 OPEN 'RMS*EGP.T.ASSIGNMENT' FILE INPUT, 1
120 AT ENDFILE(1) 60 TO 160
130 INPUT FROM 1:CON1,OWNER1,LOCAT1,OPSTA1,RECID1
140 IF OWNER1='12' THEN PRINT CON1;OPSTA1;LOCAT1
150 60 TO 130
160 CLOSE I FOR I=1 TO 1
170 STOP CHAR(13)+'END OF RUN'
```

Fig. 8. Sample code lines from 'WRITPR' (lines 110 and 120 of Fig. 7)

```
3170 LINE=LINE+10
3180 WRITE ON 1:STR(LINE):' OPEN ':SP$(JJ):' INPUT ':JJ
3190 LINE=LINE+10
3200 WRITE ON 1:STR(LINE):' AT ENDFILE ':JJ:GO TO(':LINE+30+10*R
```

Fig. 9. Sample code lines from 'WRITPR' (line 130 of Fig. 7)

```
3210 LINE=LINE+10
3220 WRITE ON 1 USING '(#):STR(LINE):' INPUT FROM ':JJ:"
\DOM(I):STR(JJ):CHAR(44) FOR I=1 TO N-1\DOM(M):
STR(JJ):CHAR(13)
```

Fig. 10. Sample code lines from 'WRITPR' (line 150 of Fig. 7)
Fig. 11. Data printout resulting from running 'WRITPR' and 'WRITER'

Fig. 12. Structure and content of the relation (data file) used in this article