ELEMENTS OF DeltaBASIC

COMMANDS

AUTO
CLEAR
DEL
FILES
FREE
GO
HELP
LIST
LISTP
LOAD
OFF
RENAME
RUN
SAVE

STATEMENTS

CHAIN
CLOSE
DATA
DEF FN
DIM
END
EXIT
FOR/NEXT
FORM
GOSUB
GOTO
IF/THEN/ELSE
INPUT
LET
LINPUT
ON ATTN
ON (Error Condition)
ON GOSUB
ON GOTO
ON RESTORE
OPEN (disk file, data entry screen)
OPTION
PRINT (formatted, unformatted)
READ (internal data, disk file, data entry screen)
REM
RESET
RESTORE
RETURN
REWRITE (disk file, data entry screen)
STOP
WRITE (disk file, data entry screen)

FUNCTIONS

Numeric:

ABS
ATN
CMDKEY
ERR
EXP
INT
LEN
LINE
LOG
POS
RND
ROUND
SGN
SIN
SQR
STATUS
VAL

Character:

CHR$
CNVRT$
DATE$
STR$
TIME$

Substring function: (a:b) replaces string functions LEFT$, MID$ and RIGHT$ of other BASICs, allows insertion and deletion

SYSTEMS SUPPORTED*

Visual screen attributes, function keys, arrow keys, and diskette formats for the following are supported by DeltaBASIC.

Computers:

Kaypro
Epson QX-10
Radio Shack Model IV (CP/M Plus, [v. 3.0])
Xerox 820-II
Televideo (CP/M or TurboDOS)
A variety of disk formats, including IBM 3740 standard 8″ diskette (SS/SD), is available.

Terminals:

ADM-3A
Televideo 900 Series
Qume QVT-102
Wyse 100

*NOTE: New implementations are regularly added to this list. Please contact DeltaSoft, Inc., for further information on computer systems and terminals not listed.

About DeltaSoft, Inc.

DeltaSoft, Inc., was established by professional programmers to develop improved software for business applications. DeltaBASIC, based on a powerful subset of IBM System/34 BASIC, is designed to provide the programmer with the most effective utilization of the rapidly developing capabilitites of microcomputers. The present release is in 8080 Assembly Language and runs on CP/M, MP/M-80, and TurboDOS. An 8086/8088 version is under development.

Call or Write for More Information:

A DeltaSoft technical representative will be happy to discuss your specific applications. Please call us at (214) 581-1425.

DeltaSoft, Inc.
7524 S. Broadway
P. O. Box 7082
Tyler, TX 75711
(214) 581-1425

DeltaBASIC

Formatted Data Entry Screens

Keyed Data Files

Multi-User Support

Decimal Arithmetic

A Business-Oriented BASIC Language

DeltaBASIC

A Business-Oriented BASIC Language

Designed to fill the gap in business programming languages for microcomputers, DeltaBASIC provides the multi-user support and other programming features essential to the development of efficient, smoothly-functioning and well documented software.

Keyed files, decimal arithmetic, formatted data entry screens, and other amenities familiar to users of main-frame languages have been incorporated. With DeltaBASIC, the business programmer is now able to achieve maximum effective use of the rapidly developing capabilities of microcomputers.

FEATURES

Interactive Design of Formatted Data Entry Screens:

A menu-driven utility allows you to design screens interactively, using a full complement of visual and data entry attributes, including reverse video, blinking, and field protection. HELP messages may be incorporated to speed operator learning and data entry. Once designed, a screen may be accessed by simple READ/WRITE statements.

Keyed Data Files for Fast Indexed Access

DeltaBASIC provides fast access to files, using rapid sequential or random access by key in addition to access by record number.

For Single- or Multi-User Systems

DeltaBASIC's file- and record-locking capabilities allow programming for multi-user environments.

Decimal Precision

The precise calculations required by business math are achieved with DeltaBASIC's decimal arithmetic. The errors inherent in other BASIC's binary-to-decimal conversions have been eliminated. Precision is selectable to eight or sixteen digits.

Binary-Tree Data Management

DeltaBASIC uses internal binary-tree data management for faster program execution and file access.

Powerful Substring Function

One simple, versatile substring function (a:b) allows great flexibility in string handling. It not only replaces the MID$, LEFT$, and RIGHT$ in other BASICs, but also allows both insertion and deletion of characters within strings.

Powerful FORM Statement Provides Versatility in PRINT, READ, WRITE, and REWRITE

A comprehensive forms specification list allows precise positioning of decimals, dollar signs, leading or trailing pluses and minuses, and DR and CR codes; definition of character and numeric fields; and packed or zoned decimal fields where applicable.

Run-Time Module

Run-time modules are available from DeltaSoft as needed for the distribution of applications programs.

Plus Other Useful Features

Many other time-saving and convenient features are built into DeltaBASIC including program chaining, on-screen program editing, a HELP facility for identifying errors, long variable names, support of function and arrow keys, and an extensive error-trapping capability.

UTILITY PROGRAMS INCLUDED

SDU: Screen Design Utility for interactive design of formatted data entry screens.

KEYSORT: Sorts data files by key (predefined) to allow keyed sequential access.

GENMSG: A "message generator" which works with your text editor to create HELP messages to be used with data entry screens.

ORGANIZE: Provides conversion of ASCII text files to DeltaBASIC and the reverse; conversion of unkeyed files to keyed; or the removal of previously marked records (e.g., Inactive) from data files.

RENUM: Renumbers a DeltaBASIC program.

SPECIFICATIONS

Source Language: 8080 Assembly (8086/8088 under development)

Operating Systems Supported:
CP/M-80, MP/M-80, TurboDOS

Recommended Memory Size:
48K minimum

Essential Terminal Capabilities:
- Clear Screen
- Cursor Positioning

Supported Terminal Capabilities:
- Low Intensity
- Blink
- Non-Display
- Reverse Image
- Underline
- Function Keys
- Arrow Keys
- Field Color

Attributes Available to Formatted Data Entry Screens:
- Controlled Field Exit
- Adjust/Fill
- Mandatory Field Entry/Fill
- Field Type (Alphanumeric or Numeric)
- Auto Screen Entry
- Position Cursor
- Field Protection

Data File Access Methods:
- Sequential
- Relative by Record Number
- Sequential by Key
- Random by Key

Data Formats:
- Character
- Numeric
- Zoned or Packed Decimal

DeltaSoft, Inc.

RELEASE DOCUMENTATION (May 1, 1985)

DeltaBASIC Version 3.0 (T)

I. Configuration:

A. Operating System -- TurboDOS 1.3

B. MP/M Compatibility -- Full

C. Diskette Format -- IBM 8" SS/SD standard

D. Terminal -- Wyse 100

1. Keyboard Assignment

attention (ATTN) key	--	ESC (^[)
begin/end toggle	--	^B
cursor left	--	left arrow (^H)
cursor right	--	right arrow (^L)
cursor down	--	down arrow (^V)
cursor up	--	up arrow (^K)
delete right	--	^G
field exit	--	line feed (^J)
HELP key	--	^Z
insert toggle	--	^O
next line	--	^X
previous line	--	^E
rub out	--	RUB OUT
tab next field	--	TAB (^I)

Note: ^ means hold down CONTROL key and press letter key.

2. Function Keys

Function keys F1 through F8 are implemented.

3. Visual Attributes

low intensity	--	implemented
blink	--	implemented
non-display	--	implemented
reverse	--	implemented
underline	--	implemented

E. Printer -- system list device

Defaults:

auto form feed on page overflow	--	on
line length	--	132
page length	--	66
page overflow	--	64

II. Files included on diskette no. 1

A. DeltaBASIC system files

DBASIC.COM -- DeltaBASIC command file

DBERR.MSG -- Error message file for HELP facility

DBREF.MSG -- Command, statement,and function reference message file for HELP facility

SDU.COM -- Screen Design Utility command file

KEYSORT.COM -- Keysort Utility command file

ORGANIZE.COM -- Organize Utility command file

GENMSG.COM -- Help message generation command file

RENUM.COM -- Renumber Utility command file

B. DeltaBASIC demonstration programs

CUSMAN.BAS -- Customer Master Maintenance program

CUSMAN.FRM -- Format file containing data entry screens 'CMM-001' and 'CMM-002' used by CUSMAN.BAS

CUSMAS.DTA -- Customer master file (data) used by CUSMAN.BAS

CUSMAS.KEY -- Customer master file (key) used with CUSMAN.DTA

CUSMAS.DEF -- Customer master file definition

CMM-001.LIB -- Library (ASCII text) file of HELP messages for screen 'CMM-001'

CMM-001.MSG -- HELP message file for screen 'CMM-001' (Created from 'CMM-001.LIB' using the GENMSG Utility)

CMM-002.LIB -- Library (ASCII text) file of HELP messages for screen 'CMM-002'

CMM-002.MSG -- HELP message file for screen 'CMM-001' (Created from 'CMM-002.LIB' using the GENMSG Utility)

CUSREP.BAS -- Customer master file report program: reports menu

CUSREP.FRM -- Format file containing data entryscreen 'CMR-001', 'CMR-002', and 'CMR-003' used by CUSREP.BAS

CUSREP1.BAS -- Customer master file report program: sequential access

CUSREP2.BAS -- Customer master file report program: access by selected record number range

CUSREP3.BAS -- Customer master file report program: sequential access by key

CUSREP4.BAS -- Customer master file report program: sequential access by selected key range

DES.BAS -- Data entry screen demonstration program

DES.FRM -- Format file containing data entry screens 'DES-001', 'DES-002', and 'DES-003' used by DES.BAS

KED.BAS -- Keyed data file demonstration program

C. Miscellaneous Files

CUSDEL.DO -- TurboDOS 'DO' file used to remove records marked for deletion in CUSMAS.DTA

RENUM.DO -- TurboDOS 'DO' file used with RENUM utility to renumber DeltaBASIC programs

MENU.COM -- Menu utility provided with DeltaBASIC system

MENU01 -- Demonstration menu text file interpreted by MENU.COM

HELP.COM -- DeltaSoft interactive TurboDOS utility HELP facility

III. Special files on diskette no. 2 (if included)

Auxiliary Demonstration Data Files

K5000.DTA -- Data file used by KED.BAS above

K5000.KEY -- Key file used with K5000.DTA

IV. Features Highlighted by Demonstration Programs

A. CUSMAN.BAS -- Customer Master File Maintenance

1. Formatted Data Entry Screens with Help Messages

2. Random Access of a Keyed File

3. Multi-User Support (Shared File Access)

4. Substring Function

5. OFF statement with "Command line"

6. Error Trapping

7. Use of Function and Arrow Keys

B. CUSREP.BAS, CUSREP1.BAS, etc. -- Customer Master File Reports

1. Formatted Data Entry Screens

2. Sequential, Relative, and Keyed-Sequential File Access

3. FORM specifications including "PIC" for formatted output

4. Program chaining

5. Forms control utilizing internal page length and overflow

C. DES.BAS -- Data Entry Screen Demonstrator

1. Explicit demonstration of visual attributes

2. Explicit demonstration of data entry attributes

3. Use of substring function to build "INDIC" string

4. Use of Function keys

D. KED.BAS -- Keyed File Demonstrator (if included)

1. Access of relatively large (5000 records) data file by key

2. Random number generator

V. Sample Terminal/Computer Interface Module

The following listing may be helpful for basic reconfiguring of your version of DeltaBASIC. Either DDT.COM (CP/M or MP/M) or MONITOR.COM can be used to load, modify, and resave DBASIC.COM. Care must be execised to prevent changes in code other than those provided for below. DeltaSoft, Inc. assumes no responsibility for the modification of DBASIC.COM by the user. Please see the license agreement for more information.

NOTE: The configuration below is a sample only and does not correspond with your implementation of DeltaBASIC.

```
;************************************************************
;*                                                          *
;*                      DeltaBASIC                          *
;*                                                          *
```

```
                ;*            (C) COPYRIGHT 1984 BY DeltaSoft, Inc.        *
                ;*                                                         *
                ;*                    Version:   1.3                       *
                ;*                                                         *
                ;*                    Rev. Date: 23 May 1984               *
                ;*                                                         *
                ;*                    Rev. Time:  3:00 PM                  *
                ;*                                                         *
                ;*                    Terminal:  SAMPLE                    *
                ;*                                                         *
                ;***********************************************************
                ;
                ;       MISC. EQUATES
                ;
                ;       OFFH IS USED TO INDICATE A NULL CHARACTER STRING
                ;
00FF =          N       EQU     OFFH
                ;
                ;       THE HIGH BIT IS SET TO MARK THE END OF CHARACTER
                ;       STRINGS.  FOR CONVENIENCE, THE VALUE P CAN BE ADDED
                ;       TO THE LAST CHARACTER OF A STRING.
                ;
0080 =          P       EQU     128
                ;
                ;
0103                    ORG     103H
                ;
                ;       THE XFER SUBROUTINES ARE USER MODIFIABLE SO LONG AS
                ;       THE ADDRESSES IN THE TABLE BELOW ARE UPDATED.  THE
                ;       ADDRESS SPACE FOR THESE SUBROUTINES IS 03A2H TO 053FH.
                ;       TWO SUBROUTINES THAT COULD PROVE USEFUL ARE (1) DIRIO
                ;       (AT 3C30H WHICH OUTPUTS THE ASCII CHARACTER IN THE 'A'
                ;       REG TO THE TERMINAL AND (2) CURSO AT 01F6H WHICH
                ;       POSITIONS AT THE CURSOR AT THE ROW GIVEN IN THE H REG
                ;       AND COLUMN IN THE L REG.
                ;
                ;       THE GENERAL SPECS FOR THE XFER SUBROUTINES ARE GIVEN
                ;       BELOW.
                ;
                ;       NOTE:  NO ADDRESSES ARE SHOWN IN THE TABLE BECAUSE THEY
                ;       DIFFER DEPENDING ON THE SPECIFIC COMPUTER/TERMINAL
                ;       IMPLEMENTATION.
                ;
0103 C30000     TINIT:  JMP     .....           ;TERMINAL INITIALIZATION
0106 C30000     AWRSET: JMP     .....           ;VISUAL ATTRIBUTE WRITE SET
0109 C30000     AWRRST: JMP     .....           ;VISUAL ATTRIBUTE WRITE RESET
010C C30000     ARDSET: JMP     .....           ;VISUAL ATTRIBUTE READ SET
010F C30000     ARDRST: JMP     .....           ;VISUAL ATTRIBUTE READ RESET
0112 C30000     CLRATR: JMP     .....           ;CLEAR VISUAL ATTRIBUTES
0115 C30000     SETTMO: JMP     .....           ;GET TIME INTO [HL]
0118 C30000     SETDTO: JMP     .....           ;GET DATE INTO [HL]
                ;
                ;       CONFIGURATION DATA
                ;
                ;    1. OPERATING SYSTEM FLAG
```

```
                ;
                ;       0=CP/M 1=MP/M 2=TURBODOS
                ;
011B 00         OPSYS:  DB      0               ;CP/M
                ;
                ;    2. ANSI FLAG
                ;
                ;       0=NO 1=Y
                ;
011C 00         ANSI:   DB      0               ;NO
                ;
                ;    3. COMPATIBILITY FLAG (TURBODOS ONLY)
                ;
                ;       BIT 7 = PERMISSIVE FLAG
                ;           6 = SUSPEND FLAG
                ;           5 = GLOBAL-WRITE FLAG
                ;           4 = MIXED-MODE FLAG
                ;           3 = LOGICAL FLAG
                ;      (BITS 2-0 NOT DEFINED)
                ;
011D 00         COMPAT: DB      0               ;FULL MP/M
                ;
                ;    4. TERMINAL INITIALIZATION DATA
                ;
011E FF0000000INIT:     DB      N,0,0,0,0       ;N=NONE
0123 0000000000         DB      0,0,0,0,0       ;
0128 0000000000         DB      0,0,0,0,0       ;
012D 0000000000         DB      0,0,0,0,0       ;
                ;
                ;    5. MOVE CURSOR LEFT DATA
                ;
0132 880000000CURLFT:   DB      08H+P,0,0,0,0   ;^H
                ;
                ;    6. MOVE CURSOR RIGHT DATA
                ;
0137 8C000000CURRGT:    DB      0CH+P,0,0,0,0   ;^L
                ;
                ;    7. CURSOR POSITIONING DATA
                ;
                ;       SEE BELOW
                ;
013C 1BBD000000CURPS1:  DB      1BH,'='+P,0,0,0 ;CUR POS PREAMBLE:  'ESC='
0141 1F         CURPS2: DB      31              ;LINE NUMBER BIAS:  31
0142 FF         CURPS3: DB      N               ;SEPARATOR CHAR:  N=NONE
0143 1F         CURPS4: DB      31              ;COL NUMBER BIAS:  31
0144 FF0000000CURPS5:   DB      N,0,0,0,0       ;CUR POS TRAILER: N=NONE
                ;
                ;           8.  ROW/COL ORDER
                ;
                ;       00H = ROW/COL  0FFH = COL/ROW
                ;
0149 00         RCORD:  DB      00H             ;ROW/COL
                ;
                ;    9. ATTN CHARACTER DATA
                ;
```

```
014A 1B          ATNCHR: DB      1BH     ;ESC
                 ;
                 ;   10. KEY DATA TABLE
                 ;
014B FF0000      LICHR:  DB      N,0,0   ;LEAD-IN CHARACTER: N=NONE
014E 870000              DB      07H+P,0,0       ;DELETE RIGHT: ^G
0151 FF0000              DB      7FH+P,0,0       ;RUB OUT: RUB OUT
0154 880000              DB      08H+P,0,0       ;CURSOR LEFT: ^H
0157 8C0000              DB      0CH+P,0,0       ;CURSOR RIGHT: ^L
015A 8A0000              DB      0AH+P,0,0       ;CURSOR DOWN: ^J
015D 8B0000              DB      0BH+P,0,0       ;CURSOR UP: ^K
0160 8F0000              DB      0FH+P,0,0       ;INSERT TOGGLE: ^O
0163 820000              DB      02H+P,0,0       ;BEGIN/END TOGGLE: ^B
0166 8A0000              DB      0AH+P,0,0       ;FIELD EXIT: ^J
0169 890000              DB      09H+P,0,0       ;TAB NEXT FIELD: ^I
016C 850000              DB      05H+P,0,0       ;PREVIOUS LINE: ^E
016F 980000              DB      18H+P,0,0       ;NEXT LINE: ^X
0172 9A0000              DB      1AH+P,0,0       ;FIELD MESSAGE HELP: ^Z
                 ;
                 ;   11. FUNCTION KEY TABLE
                 ;
0175 810000      FICHR:  DB      01H+P,0,0       ;LEAD-IN CHARACTER: SOH (^A)
0178 B10000              DB      31H+P,0,0       ;FUNCTION KEY 1
017B B20000              DB      32H+P,0,0       ;FUNCTION KEY 2
017E B30000              DB      33H+P,0,0       ;FUNCTION KEY 3
0181 B40000              DB      34H+P,0,0       ;FUNCTION KEY 4
0184 B50000              DB      35H+P,0,0       ;FUNCTION KEY 5
0187 B60000              DB      36H+P,0,0       ;FUNCTION KEY 6
018A B70000              DB      37H+P,0,0       ;FUNCTION KEY 7
018D B80000              DB      38H+P,0,0       ;FUNCTION KEY 8
0190 800000              DB      N,0,0           ;NOT IMPLEMENTED
0193 800000              DB      N,0,0           ;NOT IMPLEMENTED
0196 800000              DB      N,0,0           ;NOT IMPLEMENTED
0199 800000              DB      N,0,0           ;NOT IMPLEMENTED
019C 800000              DB      N,0,0           ;NOT IMPLEMENTED
019F 800000              DB      N,0,0           ;NOT IMPLEMENTED
01A2 800000              DB      N,0,0           ;NOT IMPLEMENTED
01A5 800000              DB      N,0,0           ;NOT IMPLEMENTED
                 ;
                 ;   12. HOME CURSOR DATA
                 ;
01A8 8000000000HOME:     DB      N,0,0,0,0       ;NONE
                 ;
                 ;   13. CLEAR SCREEN DATA
                 ;
01AD 9A0000000CLR:       DB      01AH+P,0,0,0,0  ;^Z
                 ;
                 ;   14. PRINTER SKIP PERF
                 ;
                 ;       0=OFF 1=ON
                 ;
01B2 01          PINIT:  DB      1               ;ON
                 ;
                 ;   15. PRINTER LINE LENGTH
                 ;
```

```
01B3 50                 DB      80              ;80
                ;
                ;   16. PRINTER LINES PER PAGE
                ;
01B4 42                 DB      66              ;66
                ;
                ;   17. PRINTER OVERFLOW LINE
                ;
01B5 40                 DB      64              ;64
                ;
                ;   18. MISC TURBODOS PATCHES
                ;
                ;       SEE BELOW
                ;
                ;                                       TurboDOS Ver.   1.22    1.3
                ;
01B6 5000       TDOSA:  DB      50H,0   ;FUNCTION CALL ADDRESS    05H,0  50H,0
01B8 0D                 DB      13      ;SET COMPATIBILITY FLAG     120     13
01B9 12                 DB      18      ;SEND COMMAND LINE          108     18
01BA 1B                 DB      27      ;PRINT MODE SET              76     27
01BB 00000000           DB      0,0,0,0 ;LEAVE ROOM FOR MORE
                ;
                ;   19. ATTRIBUTE TABLE
                ;
                ;       ORDER: DIM,BLINK,BLANK,REV,UND
                ;
01BF 4002010408ATTBL:   DB      40H,02H,01H,04H,08H
                ;
                ;   20. TERMINAL SIGN-ON MESSAGE
                ;
                ;       MUST INCLUDE 32 POSITIONS
                ;
01C5 5465726D69ATRMSG:  DB      'Terminal: ADM-3A               ',' '+P
                ;                ....|....|....|....|....|....|.   .
                ;
                ;       DATA END
                ;
                ; ...
                ;
                ;*****************************************************************
                ;                                                                *
                ;               MISC. SUBROUTINES -- USER MODIFIABLE             *
                ;                                                                *
                ;*****************************************************************
                ;
                ;       XFER SUBROUTINES ARE USER MODIFIABLE.  THE ENTRY
                ;       POINT JUMP TABLE AT 0103H MUST BE MODIFIED TO
                ;       REFLECT ADDRESS CHANGES.  ALL REGISTERS CAN BE MODIFIED
                ;       BUT THE STACK MUST BE PRESERVED.
                ;
                ;       NOTE:  NO CODE IS GIVEN--ONLY SUBROUTINE DEFINTIONS.
                ;
                ;       TERMINAL INITIALIZE SUBROUTINE:  CALLED ONCE BEFORE
                ;       SIGN-ON.
                ;
```

```
;XFER0:
;
;
;       VISUAL ATTRIBUTE CONTROL
;
;
;       VISUAL ATTRIBUTE WRITE SET SUBROUTINE:  SETS SPECIFIED
;       VISUAL ATTRIBUTES AND POSITIONS CURSOR AT ROW/COL IN
;       HL BEFORE WRITE.
;
;       H=ROW  L=COL  B=FLD LEN  C=ATTR BYTE
;
;       ATTR BYTE - TRUE=1
;
;       (AS DEFINED ABOVE)
;
;       BIT   7     6     5     4     3     2     1     0
;             0   DIM     0     0   UND   REV  BLINK BLANK
;
;       NOTE: MUST RETURN WITH CURSOR AT POS(H,L)
;
;XFER1:
;
;       ATTRIBUTE WRITE RESET SUBROUTINE:  RESETS VISUAL
;       ATTRIBUTES AFTER WRITE IF NECESSARY.
;
;XFER2:
;
;       VISUAL ATTRIBUTE READ SET SUBROUTINE:  SETS SPECIFIED
;       VISUAL ATTRIBUTES AND POSITIONS CURSOR AT ROW/COL IN
;       HL BEFORE READ IF NECESSARY.
;
;       H=ROW  L=COL  B=FLD LEN  C=ATTR BYTE
;
;       ATTR BYTE - TRUE=1
;
;       BITS  7     6     5     4     3     2     1     0
;             0   DIM     0     0   UND   REV  BLINK BLANK
;
;
;       NOTE: MUST RETURN WITH CURSOR AT POS(H,L)
;
;XFER3:
;
;       ATTRIBUTE READ RESET SUBROUTINE:  RESETS VISUAL
;       ATTRIBUTES AFTER READ IF NECESSARY.
;
;XFER4:
;
;       CLEAR RESIDUAL ATTRIBUTES:  CLEARS VISUAL ATTRIBUTES
;       AFTER INTERRUPTION.
;
;XFER5:
;
;       GET TIME SUBROUTINE:  STORE HH:MM:SS AT ADDRESS
```

```
;       POINTED TO BY REG HL.  USE 99:99:99 IF TIME NOT
;       AVAILABLE.
;
;XFER6:
;
;       GET DATE SUBROUTINE:  STORE YYYYMMDD AT ADDRESS
;       POINTED TO BY REG HL.  USE 00000000 IF DATE NOT
;       AVAILABLE.
;
;XFER7:
;
```

DeltaBASIC License Agreement

IMPORTANT: Read this license agreement carefully **BEFORE** opening this package. If you do not agree with the terms and conditions contained herein, return the unopened package to DeltaSoft, Inc., 7524 S. Broadway Suite 119, Tyler, Texas 75703.

1. License

DeltaSoft, Inc. agrees to grant to the Licensee, upon the Licensee's acceptance of the terms and conditions of this agreement, a nontransferable and nonexclusive license to use the DeltaSoft, Inc. software enclosed in the accompanying package.

2. Restrictions on Use

The Licensee agrees to use the enclosed software only on the computer so designated on the license registration form. Neither this license nor the enclosed software may be assigned, sublicensed, or otherwise transferred by the Licensee without written permission of DeltaSoft, Inc.

3. Back-up Copies of Program

The Licensee may make up to two copies of the enclosed software provided it is for the Licensee's own use and provided all other restrictions on its use as set forth in this agreement are met.

4. Copies of Documentation

The Licensee may not copy in whole or in part the enclosed documentation or related materials. Addition copies of the documentation can be obtained from DeltaSoft, Inc., 7425 S. Broadway Suite 119, Tyler, Texas 75703.

5. Modification

The Licensee may modify the programs only to the extent necessary to adapt it to a terminal or printer. No other modifications may be made.

6. Copyright Notice

The software, documentation, and related materials contained herein are copyrighted by DeltaSoft, Inc. The Licensee agrees not to remove any copyright notices. Further, the Licensee agrees to reproduce such notices on any copies of the software made by the Licensee.

7. Nondisclosure

The Licensee agrees not to provide, disclose, or otherwise make available any DeltaSoft, Inc. software, documentation or related materials to any person other than employees of the Licensee and then only to the extent necessary to conduct business of the Licensee.

8. Term of License

The license shall remain in effect for such period as the Licensee complies with the terms and conditions of the license. The license is subject to cancellation by DeltaSoft, Inc. for failure of Licensee to comply with the terms and conditions of this license.

9. Enhancements and Updates

The version of software, documentation, and related materials was current at the time this package was manufactured. Enhancements and updates will be issued to the Licensee only if the Licensee subscribes to the DeltaSoft, Inc. Software Update Program.

10. Limited Warranty

DeltaSoft, Inc. makes no express or implied warranty of any kind with respect to the use of the software, documentation, and related materials enclosed in this package. DeltaSoft, Inc. expressly disclaims all implied warranties of merchantability or fitness for a particular purpose. DeltaSoft, Inc. shall not be liable for consequential damages or related expenses even if notified of the possibility of such damages. The Licensee must assume the entire risk of using this product.

DeltaBASIC Registration Form

This form must be filled out and returned to DeltaSoft, Inc., 7425 S. Broadway Suite 119, Tyler, Texas 75703 within 30 days after purchase to ensure proper registration.

Serial number: _____

Computer system on which the software will be used: _____

Serial number of computer system: _____

I, _____ (Licensee), agree to the terms and conditions of the DeltaSoft, Inc. DeltaBASIC License Agreement enclosed. If the software package was not purchased by an individual, I represent that I am the authorized agent of the company named below.

_____ _____ _____

signature of individual or agent representing company date

Address _____

City/State _____ Zip Code _____

Phone No. (___) _____

I am ___/am not ___ enclosing $75.00 for a one-year subscription to the DeltaSoft, Inc. Software Update Program. I understand that by subscribing, I will receive all software and publication updates for the above named software package for a period of one year.

COPYRIGHT NOTICE

TRADEMARK NOTICE

CP/M and MP/M are trademarks of Digital Research. TurboDOS is a trademark of Software 2000, Inc. IBM System/34 is a trademark of IBM Corporation.

DISCLAIMER OF WARRANTY

DeltaSoft, Inc. makes no express or implied warranty of any kind with respect to the use of the contents of this publication. DeltaSoft, Inc. expressly disclaims all implied warranties of merchantability or fitness for a particular purpose. DeltaSoft, Inc. shall under no circumstances be liable for consequential damages or related expenses even if notified of the possibility of such damages. The user must assume the entire risk of using the contents of this publication.

TABLE OF CONTENTS

Section 1. INTRODUCTION

DeltaBASIC was designed to meet your needs for a versatile but easy-to-use business programming language. Its most important features include:

1. Versatile disk file access for improved data management

2. Formatted data entry screens for better user interfacing and more complete utilization of computer system capability

3. Decimal arithmetic (8 or 16 digits) for improved accuracy.

To take full advantage of these and other features of DeltaBASIC, you should become familiar with the commands and statements described in this manual.

The reference manual is organized as follows:

Sections 2 and 3 define the terms and abbreviations used in this manual.

Section 4 describes the basic operating modes of DeltaBASIC.

Sections 5 through 7 describe the commands, statements, and functions available for use in DeltaBASIC.

Section 8 describes various utility programs that provide support functions to programs written in DeltaBASIC.

Appendix A lists error codes and their meanings.

Appendix B lists the reserved words that have restricted use in DeltaBASIC programs.

Appendix C describes DeltaBASIC's disk file I/O features.

Appendix D describes the powerful formatted data entry screen features of DeltaBASIC.

Appendix E provides information concerning multi-user capabilities.

Appendix F discusses DeltaBASIC's compatibility with other BASIC implementations.

Section 2. DEFINITIONS

Constants

Values that remain the same during program execution are called constants. Numeric constants are accepted within the range 1.0E-62 to 9.99...E+62 and have a selectable precision of 8 or 16 significant digits (see the OPTION statement). Character constants are formed by enclosing alphanumeric characters in quotation marks. Character constants can be from 0 to 255 characters in length.

Examples: 2.45
1.3E-20
"JAMES SMITH"

Variables

Values that can change during program execution are called variables. Variables take two possible forms: numeric or character. Numeric variables have the same size restraints as constants (above). Character variables can have any length from 1 to 255 characters, with the length assigned using the DIM statement. (If no length is assigned, it is automatically set at 18 characters.) Either numeric or character variables can be used in arrays.

Each variable must be given a name consisting of 1 to 8 alphanumeric characters (A-Z, 0-9). The first character of the name must be alphabetic. The name of a character variable must end with a dollar sign ($).

Examples: RATE
CUSTOMER$
CNST(10,5)
TABLE$(I)

Numeric and character variables cannot have the same name; ie., DAY and DAY$ cannot be used in the same program. Certain words are reserved for use by DeltaBASIC and may not be used as variable names. For a list of these reserved words, see Appendix B.

Numeric Expressions

Numeric expressions are composed of numeric variables and constants, intrinsic numeric functions, arithmetic operators (+,-,*,/,^), and parentheses. Algebraic rules are used to establish precedence for evaluating a numeric expression:

```
First:    1.  Expressions within parentheses
   |      2.  Exponentiation (^)
   |      3.  Negation (-)
   |      4.  Multiplication (*), Division (/)
Last:     5.  Addition (+), Subtraction (-)
```

When equal precedence is encountered, evaluation proceeds from left to right.

```
Examples: -35/Z
           2*SQRT(X^2+Y^2)+180
```

Character Expressions

Character expressions are composed of character variables and constants, concatenation operators (&), and intrinsic character functions.

```
Examples: A$&"  DB"
          STR$(65)
```

Relational Expressions

Relational expressions are composed of two numeric expressions or two character expressions separated by a relational operator. The valid relational operators are <,>,=, or a combination of two of these. The combination <> means "not equal to". Relational expressions are evaluated by the program as either true or false.

```
Examples: HOURS<=12
          FIRST$>LAST$&"A"
```

Logical Expressions

Logical expressions are composed of relational expressions, logical operators (NOT, AND, and OR), and parentheses. Logical expressions are evaluated by the program as either true or false. Evaluation precedence is as follows:

```
First:    1.  Expressions within parentheses
   |      2.  NOT
   |      3.  AND
Last:     4.  OR
```

When equal precedence is encountered, evaluation proceeds from left to right.

```
Examples: NOT(A<B)
          (C$<>TEMP$) AND A<1
```

Section 3. ABBREVIATIONS

The following abbreviations are used to simplify the commands, statements, and functions described in this manual.

num-cnst	numeric constant
char-cnst	character constant
cnst	constant--either numeric or character
cnst-list	constant list (comma used as separator)
num-var	numeric variable or numeric array element
char-var	character variable or character array element
var	variable--either numeric, character, or array element
var-list	variable list (comma used as separator)
num-expr	numeric expression
char-expr	character expression
expr-list	expression list (comma used as separator)
num-fnc	numeric function
char-fnc	character function
rel-expr	relational expression
rel-optr	relational operator
log-expr	logical expression
log-optr	logical operator
char-str	character string--not enclosed in quotes
line-num	line number (1-65534)
file-ref	file reference (1-254)
drv:name.typ	disk file drive, name, and type.
RETURN	the RETURN key
(return key)	press the RETURN key
ATTN	the attention key (usually the ESCAPE key)
I/O	input/output

Throughout the remainder of the manual, braces ({}) indicate optional parameters. The braces themselves are not part of the format.

Section 4. BASIC OPERATION

As with most BASIC interpreters, DeltaBASIC has two operational modes: the command/program edit mode and the program execution mode.

To bring up the command/program edit mode from the operating system, enter

DBASIC (return key)

Then, after sign-on, you can either load an existing program from disk or enter the lines of a new program. The command/program mode allows you to enter commands, add lines to the current program, or change or remove existing lines.

Certain special keys that can be used when in the command/program edit mode are listed below. The caret (^) means the control key is held down while the key is pressed.

^H	Moves the cursor to the left one character.
^L	Moves the cursor to the right one character.
^J	Moves the cursor up one line (multiline statements).
^K	Moves the cursor down one line (multiline statements).
^B	Moves the cursor to the beginning of the line.
^O	Turns the insert capability on/off.
^G	Deletes one character to the right.
RUB OUT	Deletes one character to the left.
RETURN	Enters the command or line.

These key choices may vary, depending on computer system and terminal used.

To enter the program execution mode from the command/program edit mode, enter

RUN (return key)

Execution will begin with the first line of the current program and continue until a STOP, END, or OFF statement is executed or an untrapped error occurs. The program can also be interrupted by pressing the ATTN key.

To enter the program execution mode directly from the operating system, enter

DBASIC {drv:}name{.BAS} (return key)

where the file name refers to a DeltaBASIC program saved in non-source form. After DeltaBASIC loads, the program will load and begin execution.

Section 5. COMMANDS

The following DeltaBASIC commands are used to control such tasks as clearing a program from memory or loading a program from disk. Commands are not usually found in programs but are executed directly from the keyboard while DeltaBASIC is in the command/program edit mode. To enter a command, type the command name and any appropriate parameters, then press the RETURN key.

AUTO [line-num,num-cnst]

The AUTO command automatically generates line numbers, freeing you from the task of typing a line number before entering each step of your program. Numbering begins with line-num and is incremented by num-cnst. If line-num and num-cnst are not specified, numbering will begin with 10 and increment by 5. If a duplicate line number occurs, the old line is displayed instead of a blank line. To end the AUTO mode and return to the command/program edit mode, press the ATTN key.

Example:

```
AUTO (return key)
```

A blank line with line number 10 appears at the bottom of the screen with the cursor positioned for editing. When the line is completed, pressing the RETURN key enters it. The line number is then incremented by 5 and a blank line 15 is displayed ready for editing.

```
AUTO 100,10 (return key)
```

Same as above, except the first line number is 100 and the increment will be 10.

CLEAR

The CLEAR command closes any open files, then clears the program and data areas of memory.

Be careful--this command permanently erases all the program and data information currently in memory. It is not possible to recover this information.

Example:

```
CLEAR (return key)
```

Open files are closed and the current program and data areas are cleared from memory. DeltaBASIC then returns to the command/program edit mode.

DEL line-num 1 {,line-num 2}

The DEL command removes line-num 1 or, when a second line number is entered, all lines from line-num 1 through line-num 2. DEL does close all open files.

Examples:

DEL 35 (return key)

Line 35 is removed from the program.

DEL 100,200 (return key)

Lines 100 through 200 inclusive are removed.

FILES {drv:}

The FILES command displays a directory of the files on the specified drive or, if no drive is specified, on the default drive.

Examples:

FILES (return key)

A directory of files on the default disk drive is displayed.

FILES C: (return key)

The directory for disk drive C is displayed.

FREE {drv:}name.typ

The FREE command erases the named file from the disk. The file name and file type are both required. If the disk drive is not specified, the default disk drive is assumed.

FREE can also be used as a statement in a program.

Examples:

FREE PAYMAS.BAS (return key)

The PAYMAS.BAS file is removed from the default drive.

FREE C:PAYTRN.DTA (return key)

The PAYTRN.DTA file is removed from disk drive C.

GO {line-num}
{END}

The GO command starts the program running again after a non-error interruption. (Non-error interruptions include pressing the ATTN key and execution of a STOP statement.) If lines are not added or removed during the interruption, variables are preserved and disk files are left open.

The primary use of GO is in debugging a program. STOP lines are placed at various points in the program so that program and variable status can be checked. GO then permits execution to resume at the specified line number or--if no line number is specified--at the next line. If lines are added, removed, or changed, execution will start at the first line of the program.

If after interrupting a program, it is decided that resumption of the program is not needed, the GO END command closes all open files to prevent loss of data and ends the program.

Example:

```
20 A=16*B+C
21 STOP
30 IF A<30 THEN 40
35 B=A+C
40 A=A+1
 ...
```

Line 21 has been inserted so that the values of A, B, and C can be printed and checked. After this has been done, entering GO causes execution to resume at line 30.

If the ATTN key is pressed just just as line 35 is to be executed, GO will continue execution at line 35. GO 40 would cause program execution to start at line 40.

HELP [parameter]

The HELP command causes the computer to display an explanatory message. The message displayed depends on the parameter specified, as follows:

HELP

Displays a message to explain the last error. To conserve memory, error codes are abbreviated. Because you may have trouble interpreting these codes at first, the HELP command provides the additional information needed to understand them. As you learn the meaning of the abbreviated error codes, the HELP facility will become less important.

HELP num-cnst

Displays a message to explain the error whose number is given by the numeric constant. Error numbers are given in Appendix A.

HELP command name
statement name
function name

Displays a brief explanation of the command, statement, or function.

Examples:

```
20 IF COUNT>3 THAN A=10
```

Line 20 contains a syntax error: THAN should be THEN. When line 20 is executed, an error is detected and this message displayed:

```
(1) SYNTAX Error in Line 20
```

While the meaning of this error is probably clear enough, entering HELP will provide this additional information:

```
SYNTAX ERROR: an unrecognizable command/statement was encountered.
```

Suppose you correct the THEN error and try again to execute. This time a different error message appears.

```
(2) SYN EOS Error in Line 20
```

Although the abbreviation SYN clearly points to a syntax error, the EOS may be a little mysterious. Entering HELP gives this additional information:

```
SYNTAX ERROR AT END OF STATEMENT: colon or line end expected.
```

EOS is short for "END OF STATEMENT". Looking at the end of line 20, you see that the letter O was used instead of the number 0. Changing this makes line 20 correct.

```
HELP 10 (return key)
```

The HELP message for error number 10 is displayed.

LIST {line-num}

The LIST command lists the program currently in memory. If the optional line number is not specified, the entire program is listed. If the optional line number is used, then the program line with a line number greater than or equal to the one specified will be the last line listed. In this case, the last line will be positioned near the bottom of the screen with the cursor located at the first character of the line. At this point, you can change the line and place it back in memory by pressing the RETURN key. In addition to the command/program edit mode cursor control keys (see BASIC OPERATION, Section 4), the following special keys are activated at this time:

^E	Move to the previous line in the program.
^X	Move to the next line in the program.

Example:

```
LIST 120 (return key)
```

The program is listed with line 120 near the bottom of the screen. The cursor appears at the beginning of line 120 so that the line can be changed. Pressing the RETURN key replaces the line in the program.

LISTP [line-num 1,line-num 2]

The LISTP command lists the current program to the printer (the list device) either in its entirety or over the optional line number range (line-num 1 through line-num 2).

Examples:

LISTP (return key)

The entire program is listed to the printer.

LISTP 100,300 (return key)

Program lines 100 through 300 are listed to the printer.

LOAD {drv:}name{.typ}{,S}

The LOAD command loads the named DeltaBASIC program from disk. The file name can include a disk drive reference and file type. The S option is used to load a source (ASCII) file. The defaults file types are BAS for non-source files and SRC for source files.

Open files are closed before loading begins. Loading a non-source file clears the data area of memory, thus destroying all previously defined variables.

Loading a source program also clears the data area, but the program area is left intact, with the new program merged into the current program. In case of duplicate line numbers, the old line is replaced by the new line.

Examples:

LOAD PAYMAIN (return key)

The program PAYMAIN.BAS (BAS is the default file type) is loaded into memory.

LOAD C:PAYMAIN,S (return key)

The source (ASCII) program PAYMAIN.SRC is loaded from drive C, merging with the current program.

OFF [char-expr]

The OFF command returns control to the operating system after first closing any open files. For computer systems using TurboDOS and CP/M 3.0, the optional character expression specifies a command line to be executed upon return to the operating system.

OFF can be used as program statement to terminate program execution. Unlike the END statement, which returns to the command/program edit mode, OFF returns to the operating system.

Example:

```
OFF (return key)
```

Operation returns to the operating system after closing any open files.

```
999 OFF "KEYSORT CUSMAS"
```

In CP/M 2.2 systems, the program terminates and returns control to the operating system. In TurboDOS and CP/M 3.0 systems, the program terminates, returns control to the operating system, and then begins execution of the specified command line.

RENAME {drv:}name 1{.typ},name 2{.typ}

The RENAME command changes the name of the file specified by name 1 to the new name specified by name 2. The file to be renamed must already exist and the new name must not be the same as that of an existing file.

Example:

```
RENAME PAYMAS.BAS,PAYMAS.BAK (return key)
```

The file PAYMAS.BAS is renamed PAYMAS.BAK.

RUN

The RUN command executes the DeltaBASIC program currently in memory. Before execution begins, the data area of memory is cleared and any open files are closed.

Example:

RUN (return key)

Program execution begins at the first line of the current program.

SAVE {drv:}name{.typ}{,S}

The SAVE command saves the DeltaBASIC program currently in memory storing it on the disk in the specified drive and in a file of the specified name and type. Certain words are reserved for use by DeltaBASIC and may not be used as disk file names. For a list of these reserved words, see Appendix B. With the S option, the program is saved in source (ASCII) form. The defaults file types are BAS for non-source files and SRC for source files.

Examples:

```
SAVE D:PAYMAS (return key)
```

The program in memory is saved using the file name PAYMAS and type BAS on the disk in D drive. Non-source form is used.

```
SAVE HOURS,S (return key)
```

The program in memory is saved on the default drive using the file name HOURS and type SRC. Source (ASCII) form is used.

Section 6. STATEMENTS

DeltaBASIC statements are the instructions used in a program to accomplish such tasks as reading data, performing computations, or printing results. In the command/program edit mode, a statement preceded by a line number becomes part of the current program. Most statements can be executed directly by entering the statement without a line number and pressing the RETURN key.

You can put more than one statement on a line, but such multiple statements must be separated by a colon. The maximum line length is 240 characters.

CHAIN {drv:}name{.BAS} {,FILES}{,var-list}

The CHAIN statement provides a way to pass execution from the program in memory to a program on disk. It replaces the current program with the program specified, then continues execution. The chained program must be in non-source form (type BAS).

Files will be closed unless the FILES option is taken. All variables except those in the variable list will be removed from the data area. When an array name appears in the variable list, the entire array remains after chaining to the new program.

Examples:

```
100 CHAIN ACTPAY.BAS,PAYDATE,CLIENT
```

The current program is replaced with the program ACTPAY.BAS from the default drive. All open files are closed, and the variables PAYDATE and CLIENT are preserved.

```
10 CHAIN E:ACTPAY,FILES
```

The current program is replaced with the program ACTPAY.BAS from drive E. Open files are left open, but all variables are cleared.

CLOSE # file-ref: {IOERR line-num}

or {EXIT line-num}

The CLOSE statement closes the file previously opened under the file reference given. IOERR or EXIT provides a branch line number should an error occur.

It is very important to close files that are no longer in use. The CLOSE statement provides this capability for individual files during program execution. To close all open files at program termination, use the END or OFF statement. GO END will take care of closing all open files after an interruption. Remember, STOP does not close files.

Example:

```
20 CLOSE #1: IOERR 100
 ...
100 REM I/O ERROR PROCESSING
 ...
```

The file with file reference number 1 is closed. Should an I/O error occur, execution will continue at line 100.

DATA cnst-list

The DATA statement is used to create a data list within a program. This internal data list is then used in conjunction with the READ statement to assign data values to program variables.

After entering RUN or executing RESTORE, an internal data pointer is set to the first constant of the first DATA statement. As READs are executed, the data pointer will move through the constants in that line and to other DATA statements as necessary, always keeping track of the next constant to be read.

The internal data pointer can be positioned at any DATA statement in the program using the ON...RESTORE statement. (See ON...RESTORE, below.)

A DATA statement must be the first statement on a line and cannot be followed by any executable statement on the same line. Remarks are permitted at the end of a DATA line. (See REM, below.)

A null data entry leaves the corresponding variable unchanged. (See the example below.)

Example:

```
10 READ NAME$,RATE,TIME
 ...
20 READ NAME$,RATE,TIME
 ...
30 DATA "SMITH,JIM",4.50,6.25,"DOE,JANE",,12.00
```

When line 10 is executed, variable assignments are made as follows:

```
NAME$:   SMITH,JIM
RATE :   4.50
TIME :   6.25
```

The data pointer will then point to DOE,JANE. When line 20 is executed, these additional assignments will be made:

```
NAME$:   DOE,JANE
TIME :   12.00
```

RATE remains 4.50 since a null entry appears in the DATA statement.

DEF FNname{$}{(var-list)}=expr

The DEF statement defines a user function that can be referenced by name in the program, as follows:

FNname{$}{(expr-list)}

During execution, each expression in the optional expression list is evaluated and its value passed to the corresponding variable in the DEF statement. The expression in the DEF statement is then evaluated and the result is passed back.

Expression list items must agree in number and type (numeric or character) with the items in the DEF variable list. Functions can be either numeric or character, but character functions must be indicated by the dollar sign ($) in the function name. The function name contains from 1 to 8 alphanumeric characters, the first character of which must be alphabetic. Certain words are reserved for use by DeltaBASIC and may not be used as function names. For a list of these reserved words, see Appendix B. Variables used in a function are local to that function; i.e., are used only within that function.

A DEF statement must be executed before the user function is referenced in the program. Once it is defined, it cannot be redefined by second DEF statement. A function statement cannot include a reference to itself or refer to a function that includes a reference to itself.

Examples:

```
10 DEF FND(S,D)=S*D
 ...
30 DISTANCE=2*FND(SPEED,DURATION)
```

Assume SPEED is 30 and DURATION is 10. When line 30 is executed, the following assignments are made:

S: 30
D: 10

FND is evaluated as 300 and passed back to the expression of line 30. Evaluation is completed and the assignment made:

DISTANCE: 600

```
10 DEF FNCLASS$(P$,N$)=P$&" "&N$
 ...
30 CLASS$=FNCLASS$(PREFIX$,NUMBER$)
```

Assume PREFIX$ is ART and NUMBER$ is 124A. When line 30 is executed, the following assignments are made:

P$: ART
N$: 124A

Using these values, the user-defined function of line 10 is evaluated to ART 124A. This result is passed back to the character expression of line 30,

```
DIM var(m[,n,...])        [,var(m[,n,...])]        [,...]
    var$(m[,n,...])[*L]   [,var$(m[,n,...])[*L]]
    var$*L                [,var$*L]
```

The DIM statement is used to assign space for numeric and character arrays in the data area of memory, and to fix the maximum length of character variables.

The constants m,n,... are maximum dimension sizes and L is the maximum character string length for a character array element.

Array subscripts begin with 1. Array variables can be dimensioned only once in a program.

Example:

```
10 DIM SALES(10,10),SALEM$(10)*30,TEMP$*10
```

When executed, the following allocations are made:

```
SALES :  two-dimensional numeric array of size 10 by 10
SALEM$:  one-dimensional character array of size 10 by 1;
         each character element with length 30
TEMP$ :  single character variable with length 10
```

END

The END statement is used to terminate a DeltaBASIC program. It closes all files and returns DeltaBASIC to the command/program edit mode.

Program exit should be through the END statement if you want to remain in DeltaBASIC command/program edit mode afterwards. Terminating a program with END (or OFF) is especially important to ensure that files otherwise left open are properly closed.

Example:

```
9999 END
```

When program termination is required, statement 9999 is executed. All open files are closed and DeltaBASIC enters the command/program edit mode.

EXIT {IOERR line-num} {,IOERR line-num} {,...}
{DUPKEY line-num} {,DUPKEY line-num}
{NOKEY line-num} {,NOKEY line-num}
{EOF line-num} {,EOF line-num}

The EXIT statement can be used with OPEN, READ, WRITE, REWRITE, and CLOSE statements to provide branching when the specified errors occur. (See the appropriate statements for more information concerning these errors.) The EXIT statement can be accessed only by way of one of these I/O statements; when otherwise encountered during execution, EXIT is ignored much as a REM statement would be.

EXIT is especially useful when a number of I/O operations have the same error processing lines. Each I/O statement would contain an EXIT line number referring to the EXIT statement, which would in turn contain the line numbers for the various error traps.

Examples:

```
10 READ #1,USING 11: A,B  EOF 100 IOERR 130
11 FORM ...
 ...
100 REM END-OF-FILE ERROR PROCESSING
105 ...
 ...
130 REM I/O ERROR PROCESSING
135 ...
```

EXIT is not used in this program segment. If an end-of-file error occurs while file #1 is being read, the program branches to line 100 for recovery processing. If an I/O error occurs, the program branches to line 130.

Now consider an equivalent method using the EXIT statement.

```
10 READ #1,USING 11: A,B EXIT 20
11 FORM ...
20 EXIT EOF 100,IOERR 130
 ...
100 REM END-OF-FILE ERROR PROCESSING
105 ...
 ...
130 REM I/O ERROR PROCESSING
135 ...
```

An error in the READ statement of line 10 causes execution to divert to the EXIT statement of line 20. Here the various error conditions are handled. Besides making the READ statement simpler, the EXIT statement can be used by other I/O statements for error branching to line 20.

FOR var=num-expr 1 TO num-expr 2 {STEP num-expr 3}

```
 ...
included DeltaBASIC statements
 ...
NEXT var
```

The FOR/NEXT statements set up a loop that performs the included DeltaBASIC statements one or more times, depending on the beginning value of the index variable (num-expr 1), ending value (num-expr 2), and the step size (num-expr 3). If not specified, the step size is 1.

The included DeltaBASIC statements will be executed at least one time before termination conditions are checked. The "exit" value of the index variable will be the first value that meets or exceeds num-expr2.

FOR/NEXT loops can be nested to a level of 8 deep.

Exiting a FOR/NEXT loop before normal termination is allowed but not recommended. Repeated exits of this type will cause the maximum FOR/NEXT level to be exceeded.

Examples:

```
10 FOR INDEX=1 to 20 STEP 2
20 PRINT INDEX
30 NEXT INDEX
```

These FOR/NEXT statements cause line 20 to be executed 10 times, printing the values from 1 to 19 in steps of 2. The "exit" value of INDEX is 21.

FORM specification list

The FORM statement gives the format specification for PRINT, READ, WRITE, and REWRITE statements. The specification list can consist of any combination of the following specifications, separated by commas:

"char-cnst"

Character constant: the character string between the quotation marks is output at the current position. Allowed in PRINT and WRITE only.

C w

Character field: the matching character value is treated as a character string that is left-justified in a blank field of width w.

CUR(m,n)

Cursor: the next value will be output at line m and position n. Allowed in PRINT only.

N w{.d}

Numeric field: the matching numeric value is treated as a character string of width w, including d decimal positions. For output, the width must include both a sign and decimal point position. For input, an explicit decimal overrides the implicit decimal position d.

PD w.d

Packed decimal: the matching numeric value is given in packed decimal format. For output, the value is rounded to d decimal places, then packed two binary coded decimal digits per position and written right-justified into a field of width w. Only the low nibble of the leftmost position contains a digit. The high nibble is 0000 binary for a positive value and 1000 binary for a negative. As with zone decimal, no explicit decimal is written. For input, the value is read from a field w wide and then given d decimal positions. Packed decimal is not allowed in the PRINT statement.

PIC(char-str)

Picture: the matching numeric value is written into a field with width and composition given by the character string. Valid characters are described below:

#

Number digit: always replaced by a digit.

Z

Leading zero: any leading zero replaced by a blank.

$

Leading dollar sign: dollar sign placed to the left of the first nonzero digit.

+

Leading plus sign: plus sign placed to the left of the first nonzero digit for a positive value, or a minus sign placed to the left of the first nonzero digit for a negative value.

-

Leading negative sign: blank placed to the left of the first nonzero digit for a positive value, or minus sign placed to the left of the first nonzero digit for a negative value.

Blank (ASCII space)

A blank is placed in the field.

,

Comma: a comma is inserted in the field if a nonzero digit precedes it; otherwise, the comma is replaced with a blank.

.

Decimal point: a decimal point is placed in the field. Only one decimal point can appear in the character string.

+

Trailing plus sign: a plus sign is placed to the right of the last digit for a positive value, or a minus sign is placed to the right of the last digit for a negative value.

-

Trailing minus sign: a blank is placed to the right of the last digit for a positive value, or a minus sign is placed to the right of the last digit for a negative value.

CR or DB

Credit or debit code: CR or DB is placed to the right of the last nonzero digit for a negative value, or two blanks are placed to the right of the last digit for a positive value.

POS n

Position: the next value will be input or output at position n.

SKIP n

Skip lines: the next n lines are skipped. Allowed in PRINT only.

X n

Skip positions: the next n positions are skipped.

ZD w.d

Zone decimal: the matching numeric value is given in zoned decimal format. For output, the value is rounded to d decimal positions and written in ASCII without a decimal point, right-justified into a field of width w. For a negative output value, the high bit of the leftmost position is set to 1. For input, the value is read from a field w wide and then given d decimal positions. Zone decimal is not allowed in the PRINT statement.

Note: Specifications N, C, ZD, and PD can be preceded by a constant replication factor and an asterisk (*); for example, 2*N 6.2 will set up two contiguous numeric fields of width 6 with 2 decimal positions.

A FORM statement must be the first statement on a line and cannot be followed by any executable statement on the same line. Remarks are permitted at the end of a FORM line.

A dollar sign ($), leading plus (+), or leading minus (-) character cannot follow a #.

Examples:

```
20 Print #255,USING 21: A,B$
21 FORM "A=",N 9.2,X 2,C 4
```

Assume A is 12.258 and B$ is BAL. When line 20 is executed, the printed line would appear as follows:

```
A=    12.26  BAL
....|....1....|....2
          0         0
```

Now consider the effect of using a PIC specifier in place of N 9.2 in the above example:

PIC specification	Value of A	Printed Result
######.##	12.258	000012.26
ZZZZZ#.##	12.258	12.26
ZZZZZ#.##	-12.258	- 12.26
+++++#.##	12.258	+12.26
+++++#.##	-12.258	-12.26
-----#.##	12.258	12.26
-----#.##	-12.258	-12.26
$$$$$#.##CR	12.258	$12.26
$$$$$#.##CR	-12.258	$12.26CR

Here is an example using zoned decimal format:

```
30 WRITE #3,USING 31: A,B$
31 FORM ZD 5.2,POS 9,C 10
```

Assume A is 2.528 and B$ is ABCD. When line 30 is executed, the written record would appear in hexadecimal as follows:

```
VALUE +0  0  2  5  3           A  B  C  D

  HEX 30 30 32 35 33 20 20 20 41 42 43 44 20 20 20 20 20 20...
  POS -- -- -- --  5 -- -- -- -- 10 -- -- -- -- 15 -- -- --

 SPEC |   ZD 5.2   | POS 9 -->|            C 10           |
```

To see how packed decimal format works, consider the following data record and program segment:

```
VALUE  +0 02 58 -0 02 58 +0  0  2  5  8 -0  0  2  5  8

  HEX  00 02 58 80 02 58 30 30 32 35 38 B0 30 32 35 38
  POS  -- -- -- --  5 -- -- -- -- 10 -- -- -- -- 15 -- -- --

 SPEC  |PD 3.1| |PD 3.1| |   ZD 5.1   | |    ZD 5.1  |
```

```
40 READ #6,USING 41: A,B,C,D
41 FORM 2*PD 3.1,2*ZD 5.1
```

When line 40 is executed, the values assigned are as follows:

```
A:   25.8
B:  -25.8
C:   25.8
D:  -25.8
```

GOSUB line-num

The GOSUB statement branches to a subroutine at the specified line number. A RETURN in the subroutine branches back to the statement immediately following the GOSUB.

GOSUBs can be nested up to 20 deep.

Exiting a subroutine without using RETURN is allowed but is not considered good practice since it leaves the nested level incorrect. Repeated exits of this type will cause the maximum GOSUB level to be exceeded.

Example:

```
10 GOSUB 50
20 ...
 ...
50 REM SUBROUTINE
 ...
60 RETURN
```

When line 10 is executed, the program branches to the subroutine at line 50. The RETURN at line 60 branches back to line 20.

GOTO line-num

The GOTO statement branches to the specified line number.

Example:

```
10 GOTO 50
 ...
50 ...
```

When statement 10 is executed, the program branches to line 50.

IF log-expr THEN line-num or statement {ELSE line-num or statement}

The IF statement branches or executes a statement depending on whether the logical expression evaluates to true or false. If true, execution continues at the line number or statement following THEN. If false, execution continues at the next statement in the program or, if the ELSE option is used, at the line number or statement following the ELSE.

Examples:

```
10 IF A>=B+C THEN B=A
20 ..
```

When line 10 is executed, A>=B+C is evaluated. If A is greater than or equal to the sum of B and C (making the the expression true), then B is assigned the value of A. If A is not greater than or equal to the sum of B and C (making the expression false), then the value of B is not changed. In both cases, execution continues at line 20.

```
10 IF NAME$<>TEMP$ AND A<B THEN A=B-C ELSE 100
20 ...
 ...
100 ...
```

If NAME$ is not equal to TEMP$ and A is less than B, then the program assigns the value of expression B-C to A and continues at line 20; otherwise, it branches to line 100.

INPUT {"char-str":} var-list

The INPUT statement allows the operator to enter data values from the keyboard during program execution. When the INPUT statement is executed, a question mark (?) is displayed as a prompt unless the optional character string (message prompt) is included. The user then enters the data values (separated by commas) that are to be assigned to each variable in the variable list, and presses the RETURN key to continue program execution.

If the correct number of data values is not entered, a question mark (?) is displayed and INPUT continues. If a variable/data type mismatch occurs, a REDO message is displayed and INPUT must be redone from start.

To enter leading blanks with a character string, enclose them within the quotation marks.

Example:

```
10 INPUT "ENTER NAME, AMOUNT: ": NAME$,AMOUNT
```

When line 10 is executed, the message is displayed and execution pauses while the name and amount are entered, separated by a comma:

```
ENTER NAME, AMOUNT: JOE SMITH,1200 (return key)
```

The following assigments are made:

```
NAME$  :  JOE SMITH
AMOUNT :  1200
```

{LET} num-var=num-expr
{LET} char-var {(num-expr 1:num-expr 2)}=char-expr

The LET statement assigns the value of an arithmetic or character expression to the specified variable.

For the character LET, the optional substring specification can be used to delete, replace, or insert characters in the character variable, depending on the optional parameters, as follows:

num-expr 1 <= num-expr 2

Character positions from num-expr 1 through num-expr 2 are replaced by the character expression. The number of replacement characters can be less than, equal to, or greater than the number being replaced.

num-expr1 > num-expr 2

The character expression is inserted before the character position given by num-expr 1. In this case, num-expr 2 is ignored.

Examples:

```
10 A=2*B
20 C$=ADRS$&" "&ZIP$
```

Assume B is 6, ADRS$ is "100 MAIN ST", and ZIP$ is "29012". When lines 10 and 20 are executed, the following assignments are made:

```
A :  12
C$:  100 MAIN ST 29012
```

To see how LET can delete characters, assume A$ in the following statements is ABCD.

```
10 A$(2:2)=""
```

When executed, line 10 will delete the letter B (the second character position) from A$, so that A$ becomes ACD.

Characters can be substituted in a similar manner:

```
10 A$(2:3)="123"
```

This statement replaces BC with 123, so that A$ becomes A123D.

Now consider a substring assignment to insert characters.

```
10 A$(2:1)="XY"
```

Since num-expr 2 is less than num-expr 1, num-expr 2 is ignored and the characters XY are inserted before B. A$ becomes AXYBCD.

LINPUT {"char-str":} char-var

The LINPUT statement assigns an entire line entered from the keyboard to the specified character variable. Unlike the regular INPUT statement, LINPUT reads a comma as part of the data entered rather than as a data separator. A question mark (?) is displayed as a prompt unless the optional character string (message prompt) is included.

Example:

```
10 LINPUT NAME$
```

This statement assigns keyboard-entered characters to NAME$ until the RETURN key is pressed. If the line entered is

```
SMITH,JOE Q. (return key)
```

then the following assignment is made:

```
NAME$:  SMITH,JOE Q.
```

ON ATTN GOTO line-num
IGNORE

The ON ATTN statement selects alternate ways of handling program interruptions caused by pressing the ATTN key. Normally the program is terminated and the command/program edit mode is entered. Using ON ATTN GOTO will instead cause branching to the specified line number should the ATTN key be pressed. The ON ATTN IGNORE form will result in the ATTN key being ignored altogether.

The ON ATTN selection can be turned off by using:

```
ON ATTN GOTO 0
```

Example:

```
10 ON ATTN GOTO 100
20 ...
 ...
100 | ATTN RECOVERY PROCESSING
```

If the operator presses the ATTN key at any time after line 10 is executed, the program braches to line 100 for recovery processing.

```
10 ON ATTN IGNORE
 ...
```

Following execution of line 10, pressing the ATTN key will have no effect on program operation.

ON Error Condition GOTO line-num

where Error Condition can be CONV, ERROR, OFLOW, UFLOW, SOFLOW, or ZDIV.

The ON Error Condition statement sets the line number for an error trap. If that error occurs, the program branches to the specified line number for recovery processing. Error conditions result from a variety of causes. See Appendix A for more information.

The error trap can be removed by using:

```
ON error condition GOTO 0
```

Example:

```
10 ON ZDIV GOTO 100
 ...
30 B=12/A
35 A=A+1
 ...
100 B=1E+60
105 GOTO 35
```

Assume A is 0. When line 10 is executed, the division-by-zero error trap line is set to 100. When a zero division error occurs in line 30, the program branches to line 100, where a recovery procedure is executed.

ON num-expr GOSUB line-num 1{,line-num 2,...}

The ON...GOSUB statement causes a branch to one of several lines, depending on the value of the numeric expression. The value is first rounded to an integer, then if it is 1, the program branches to line-num 1; if 2, to line-num 2; etc. An invalid expression value causes execution to continue at the statement following the ON...GOSUB.

After branching to a subroutine, subsequent execution of a RETURN branches back to the first statement following the ON...GOSUB. As with a standard GOSUB, leaving the subroutine other than by a normal RETURN statement should be avoided.

Example:

```
10 ON 2*A-1 GOSUB 20,30,40
15 A=A+1
 ...
20 B$=HI$
25 RETURN
30 B$=MED$
35 RETURN
40 B$=LOW$
45 RETURN
```

Assume A is 2. When line 10 is executed, the numeric expression 2*A-1 evaluates to 3 and the program branches to line 40. At line 45, the program branches back to line 15.

ON num-expr GOTO line-num 1[,line-num 2,...]

The ON...GOTO statement causes a branch to one of several lines, depending on the value of the numeric expression. The value is first rounded to an integer, then if it is 1, the program branches to line-num 1; if 2, to line-num 2; etc. An invalid expression value causes execution to continue at the statement following the ON...GOTO.

Example:

```
10 ON 3*(A+1)/5 GOTO 20,30,40
15 A=A+1
 ...
20 B$=HI$
25 GOTO 50
30 B$=MED$
35 GOTO 50
40 B$=LOW$
50 ...
```

Assume A is 2. When line 10 is executed, the numeric expression 3*(A+1)/5 evaluates to 1.8, which is rounded to 2. The program therefore branches to line 30.

ON num-expr RESTORE line-num 1{,line-num 2,...}

The ON...RESTORE statement sets the internal data pointer to one of several lines, depending on the value of the numeric expression. The value is first rounded to an integer, then if it is 1, the internal data pointer is restored to line-num 1; if 2, to line-num 2; etc. An invalid expression value results in no RESTORE being executed.

Example:

```
10 ON 3*(A+1)/5 RESTORE 20,30,40
15 A=A+1
 ...
20 DATA 7,8,9,0
30 DATA 4,5,6,0
40 DATA 1,2,3,0
```

Assume A is 2. When line 10 is executed, the numeric expression 3*(A+1)/5 evaluates to 1.8, which is rounded to 2. The data pointer is set to the first data value on line 30.

OPEN #file-ref: char-expr,file attributes {IOERR line-num}

or {EXIT line-num}

(Open a disk file)

The OPEN statement opens a disk file for input, output, or update access. The file reference number can be any integer from 1 through 254. (File #0 is reserved for regular display terminal operation and #255 is reserved for printer operation. These special file references do not require OPEN statements.)

The disk file ID is specified by char-expr and includes the following parameters (separated by commas):

(i) NAME={drv:}name{.typ} -- required data file name.

(ii) NEW -- used only if a new file is being created.

(iii) RECL=record length (1-2048 bytes) -- required if NEW specified.

(iv) KEYL=key length (1-32) -- required if NEW specified and the new file is to be keyed.

(v) KEYP=key position -- required if NEW specified and the new file is to be keyed.

(vi) RANDOM -- used if a keyed file is to be accessed randomly by key or if records are to be added to a keyed file.

(vii) SHR -- required if the file is to have a shared status in a multi-user environment.

The file attributes include the following parameters (each preceded by a comma):

(i) KEYED if the file is to be accessed by key or RELATIVE if the file is to be accessed by record number. If neither KEYED nor RELATIVE is specified, then sequential access is selected by default.

(ii) INPUT if the file mode is input (reads only allowed) or OUTPUT if the file mode is output (writes only allowed). If neither INPUT nor OUTPUT is specified, then update mode is selected (both reads and writes allowed).

(iii) BEGIN if sequential/output access is to begin at record one rather than at the end of the file.

If an I/O error occurs and IOERR line-num is specified, then execution continues at the line number given. If EXIT line-num is given instead, the EXIT statement at the specified line is used. If neither IOERR nor EXIT line-num is specified, an error interruption results.

For more information about disk file I/O, see Appendix C.

Examples of disk file openings:

```
10 OPEN #1: "NAME=PAYTRN.DAT,NEW,RECL=200",OUTPUT IOERR 200
```

Line 10 creates the new file PAYTRN.DAT with a record length of 200. Since neither KEYED nor RELATIVE access is specified, records will be arranged sequentially, in the order they are written. The first record written will be record 1. The OUTPUT attribute prescribes that file reads are prohibited. If an I/O error occurs, execution will continue at line 200.

```
10 OPEN #1: "NAME=PAYTRN.DAT",OUTPUT
```

Since neither KEYED nor RELATIVE is specified, access will be sequential. With the OUTPUT attribute, file reads are prohibited. Since BEGIN is not specified, new records will be added to the end of the file. This file opening would be used to add records to an existing file.

```
10 OPEN #1: "NAME=PAYTRN.DAT" EXIT 100
```

Since neither KEYED nor RELATIVE is specified, access will be sequential. Without INPUT or OUTPUT specified, both file reads and rewrites are permitted (update mode). The first read will access record 1. Subsequent reads will access the remainder of the file in a sequential manner (record 2, record 3, etc., to the end-of-file). If an error occurs, the EXIT statement at line 100 is referenced for error recovery processing. This file opening would be used to update records in the file PAYTRN.DAT.

```
10 OPEN #1: "NAME=PAYTRN.DAT",RELATIVE,INPUT
```

Record file PAYTRN.DAT is opened for read-only access by relative record number. Requesting a record beyond the current end of file results in an error.

```
10 OPEN #1: "NAME=PAYMAS.DAT,NEW,RECL=300,KEYL=10,KEYP=1,RANDOM",KEYED,OUTPUT
```

This file opening would be used to build a new keyed file. File PAYMAS.DAT is created with record length 300. Key length is 10 and the key begins in position 1. Since OUTPUT and RANDOM are specified, only new records can be added to the file.

```
10 OPEN #1: "NAME=PAYMAS.DAT,RANDOM",KEYED,INPUT
```

Keyed file PAYMAS.DAT is opened for random access by key. Since INPUT is specified, only reads are permitted. This file opening would be used to obtain data randomly by key.

```
10 OPEN #1: "NAME=PAYMAS.DTA",KEYED
```

Keyed file PAYMAS.DAT is opened for sequential access by key. Since neither INPUT nor OUTPUT is specified, both reads and rewrites are permitted (update mode). This file opening could be used to update records in order by key.

OPEN #file-ref: char-expr {IOERR line-num}

or {EXIT line-num}

(Open a work station file)

The OPEN statement opens a work station file for update access. The file reference number can be any integer from 1 through 254. (File #0 is reserved for regular display terminal operation and #255 is reserved for printer operation. These special file references do not require OPEN statements.)

The file ID is specified by char-expr and includes the following parameters (separated by commas):

(i) WS -- required.

(ii) NAME={drv:}name.FRM -- required screen format file name.

(iii) RECL=constant -- required.

The file specified must be of type FRM and created by the screen design utility program SDU.COM. The record length must be greater than or equal to the length of the longest screen input or output buffer in the screen format file.

If an I/O error occurs and IOERR line-num is specified, then execution continues at the line number given. If EXIT line-num is given instead, the EXIT statement at the specified line is used. If neither IOERR nor EXIT line-num is specified, an error interruption results.

When a work station file is opened, the display file (#0) automatically closes. The INPUT and PRINT statements cannot be used with the display terminal until the work station file is closed.

For more information about work station files, see Appendix D.

Examples:

```
10 OPEN #1: "WS,NAME=PAY.FRM,RECL=200"
```

This statement closes the display station file (#0) and opens work station file #1. PAY.FRM contains the screen formats to be used. The maximum input or output buffer length is 200, which requires the record length to be set to 200. Screen output and keyboard input are performed using READ #1.../WRITE #1... statements. If work station file #1 is later closed, the display station file #0 will automatically reopen.

OPTION {PRTZO n}, {,SPREC or LPREC}, {,PTRSET s,1,n,o}, {,PTRSET s,char-expr 1, char-expr 2}

The OPTION statement provides the user the choice of certain options:

(i) PRTZO n: print zone width of n (1 to 80) characters (default width is 16)

(ii) SPREC: short precision in calculation--8 digits (default is SPREC)

LPREC: long precision in calculation--16 digits.

(iii) PRTSET s,1,n,o: Printer forms control, where:

s=ON: automatic form feed on page overflow
s=OFF: automatic form feed suppressed

1=line length: 0 indicates no change from the previous setting (default value is 132)

n=lines per page: 0 indicates no change (default value is 66)

o=page overflow line: 0 indicates no change (default value is 64).

(iv) PRTSET s,char-expr 1,char-expr 2: Printer mode control for TurboDOS operating system only, where:

s=D ... printing is direct
s=S ... printing is to spooler
s=C ... printing is to console
s=U ... unchanged

char-expr 1=Printer or Spooler assignment (A-P,U)

char-expr 2=Spool drive (A-P,U)

Options do not take effect until the statement is executed.

Example:

```
10 OPTION PRTZO 20,LPREC,PTRSET ON,0,58,52,PRTSET S,"A","E"
```

When line 10 is executed, the following options are selected:

Print zone width is set to 20.
Long precision arithmetic is selected.
Printer will automatically form feed on page overflow, line length is unchanged, lines per page is set to 58, and page overflow point is set at line 52.
Output to printer is routed to spooler A on drive E (TurboDOS only).

PRINT {#255:} list {; or ,}

(Unformatted PRINT)

The unformatted PRINT statement outputs the specified list to the display terminal or, if #255 is used, to the printer. The list can consist of any combination of the following data items:

```
arithmetic expressions
character expressions
TAB({1,}p) -- moves to position p or, optionally, to line 1 position p
NEWPAGE    -- sends clear screen command to display or new page command
              to printer
```

The items must be separated by a comma to left-justify data items into columns or zones (see OPTION, above) or by a semicolon if data items are to be adjacent on the line. Ending the list with a comma or semicolon inhibits a final carriage and line feed.

Examples:

```
10 PRINT "TODAY ";B$;" IS";A;" YEARS OLD."
```

Assume A is 10 and B$ is ROBERT. When line 10 is executed, the following line is output to the display terminal:

```
 TODAY ROBERT IS 10 YEARS OLD.
 ....|....1....|....2....|....3
         0         0         0
```

Note that spaces between words must be supplied in the character expressions.

```
10 FOR I=1 TO 4
20 PRINT #255: I,I*I
30 NEXT I
```

This short program outputs the following multiplication table (arranged in 16-position columns) to the printer:

```
  1               1
  2               4
  3               9
  4               16
 ....|....1....|....2
         0         0
```

PRINT {#255,}USING line-num: expr-list

(Formatted PRINT)

The formatted PRINT statement outputs the expression list to the display terminal, or if #255 is specified, to the printer, using format specifications given by the FORM statement at the USING line number. (See the explanation of FORM statements above.) The expression list consists of numeric or character expressions separated by commas.

Example:

```
10 PRINT USING 11: B$,2*A
11 FORM "SERIAL NO. ",C 10,X 2,"WEIGHT: ",N 8.2
```

Assume A is 12.347 and B$ is A10-1897. When line 10 is executed, the following is output to the display terminal:

```
 SERIAL NO. A10-1897    WEIGHT:    24.69
 ....|....1....|....2....|....3....|....4
          0         0         0         0
```

Modifying line 10 as shown below will direct output to the printer instead.

```
10 PRINT #255,USING 11: B$,2*A
```

READ var-list

(Read internal data)

This READ statement assigns the named variable a data value from the current position of the internal data pointer. The data pointer is then advanced to the next data value. (See the DATA statement above.)

Example:

```
10 READ NAME$,AMOUNT
 ...
50 DATA JOHN JONES,10.50,JANE SMITH,8.40
```

Assume the data pointer is set to JOHN JONES. Line 10 makes the following assignments:

```
NAME$:  JOHN JONES
AMOUNT: 10.50
```

The data pointer then advances to JANE SMITH.

READ #file-ref,USING line-num {,KEY=char-expr} : var-list {IOERR line-num}
{,KEY>=char-expr} {NOKEY line-num}
{,REC=num-expr} {EOF line-num}

or {EXIT line-num}

(Read a disk file)

This READ statement reads data from the disk file specified by the file reference number. Variables separated by commas in the variable list are assigned values from the current record of the file, using the specifications in the FORM statement given by the USING line number.

KEY=char-expr specifies that the record with the given key be read. If the record is not found and NOKEY is specified, execution continues at the NOKEY line number; otherwise, an error is reported. The file must be open for keyed-random/input or keyed-random/update access.

KEY>=char-expr specifies that the first record with a key greater than or equal to the given key be read. The file must be open for keyed-sequential/input or keyed-sequential/update access.

REC=num-expr specifies that the record with the given record number be read. The file must be open for relative/input or relative/update access.

Reading a non-keyed file sequentially results in the next record being read until an end of file indication is encountered. If a keyed file is read sequentially, the records are taken in order by key until all sorted keys are read (see KEYSORT utility, Section 8). In either case, if an EOF line number is given, the program will then branch to that line; otherwise, an error results. IOERR, NOKEY, and EOF errors can be trapped at the specified line numbers, or an EXIT statement can be used.

Examples:

```
10 READ #1,USING 15,KEY="PENCIL": COST  NOKEY 100 IOERR 200
15 FORM POS 10,ZD 7.2
 ...
100 REM NOKEY ERROR PROCESSING
 ...
200 REM I/O ERROR PROCESSING
```

Assume file #1 is open for keyed/random access. When line 10 is executed, file #1 is read using the key PENCIL. If the record is found, the stored value is assigned to COST. If no key is found, the program branches to line 100. Should an I/O error occur, execution will continue at line 200.

```
10 READ #15,USING 15: KIND$,SIZE  EOF 100
15 FORM X 2,C 10,X 6,PD 4
 ...
100 REM END-OF-FILE ERROR PROCESSING
```

Assume file #15 is open for non-keyed sequential access. When line 10 is executed, the next record of file #15 is read and the stored values are assigned to variables KIND$ and SIZE. If the end-of-file is encountered, execution continues at line 100.

READ #file-ref,USING line-num: var-list

(Read a work station file)

This READ statement reads data from a work station file. In this case, the only data record is the screen input buffer. The USING line number gives the FORM statement that describes how the variables are read. When a work station read is executed, the user enters data into the fields displayed on the screen. Pressing the return key or a special function key enters the screen data into the appropriate variables, and execution continues at the next statement.

Whenever a work station file is open, INPUT and PRINT cannot be used with the display terminal.

The following special keys are activated while the work station file is open:

^L	Moves the cursor one character to the right.
^H	Moves the cursor one character to the left.
^G	Deletes one character to the right.
DEL	Deletes one character to the left.
^I	Moves the cursor to the next field.
^Z	Display field help message if defined. See Note 1 below.
LINE FEED	Moves the cursor to the next field with blank or zero fill.
RETURN	Enters the screen data.
F1,F2,...	(Function keys) Enters the screen data. See Note 2 below.

Note 1: The field message facility requires that a message file of the same name as the screen and type of MSG be present on the default disk drive. Pressing the ^Z key displays the corresponding field message in the message file. Pressing ^Z a second time displays the global message for the same file. At either point, pressing the space bar returns to the data entry screen. The field message file is created with the GENMSG utility. (See UTILITY PROGRAMS, Section 8.)

Note 2: After returning to DeltaBASIC, the numeric function CMDKEY can be used to determine the last key pressed. (See INTRINSIC FUNCTIONS, CMDKEY, in Section 7.)

Examples:

```
30 READ #3,USING 31: NAME$,ADRS$,CITY$,ZIP$,BALANCE
31 FORM 3*C 30,C 5,N 8.2
```

Assume file #3 is open as a work station file. After line 30 is executed, the user makes changes to the data items on the screen. The work station input buffer is updated to reflect this data entry. After the RETURN key is pressed, the work station input buffer is read and assignments are made to the four variables according to the FORM specification in line 31.

REM

REM is a non-executing statement that is used to include remarks or other documentation in a program. The length of a remark is limited by the total length of the line on which it is found. The REM statement must be the last statement on a line.

The special character | can also be used for including comments in a program. It is not actually treated as a DeltaBASIC statement but rather as an end-of-line blanking character--anything entered beyond the | will be ignored by the statement processor. While | can appear at the beginning of a line, it cannot appear following a colon used as a multi-statement separater.

Example:

```
10 REM THIS IS A COMMENT
 ...
20 | THIS IS ALSO A COMMENT
 ...
30 COUNT=COUNT+1 | A COMMENT CAN BE HERE, TOO.
```

RESET

The RESET statement performs a disk reset to allow diskettes to be changed. Open files are not closed.

Example:

```
10 RESET
```

When line 10 is executed, the disk system is reset.

RESTORE

The RESTORE statement moves the internal data pointer to the first data value of the first DATA statement in the program.

Example:

```
10 RESTORE
 ...
20 READ A,B
 ...
100 DATA 12,14
200 DATA 14,16
```

Line 10 sets the data pointer to 12 in line 100. The READ statement of line 20 will then make the following assignments:

```
A : 12
B : 14
```

RETURN

The RETURN statement branches back from a subroutine to the main program. Execution continues at the next statement after the GOSUB that called the subroutine.

Example:

```
10 GOSUB 50
20 ...
 ...
50 REM SUBROUTINE
 ...
60 RETURN
```

When line 10 is executed, the program branches to the subroutine at line 50. The RETURN at line 60 branches back to line 20.

REWRITE #file-ref,USING line-num: expr-list {IOERR line-num}

or {EXIT line-num}

(Rewrite a disk file)

This REWRITE statement rewrites a disk record that has previously been read. Expressions are evaluated then rewritten into the disk record using the format given by the FORM statement at the specified USING line number. IOERR or EXIT provides a branch line number should an error occur.

REWRITE is useful for updating records in a disk file. All or part of the last record read can be rewritten.

The file must be open in update mode; otherwise an error occurs. In addition, the last operation to the file must have been a read.

When rewriting a keyed file, an error occurs if the key field is changed.

Example:

```
30 READ #3,USING 35: A,B,C
35 FORM 3*PD 4.1
40 C=A+B+C
50 REWRITE #3,USING 55: C  EXIT 100
55 FORM POS 9,PD 4.1
 ...
100 EXIT IOERR 200
 ...
200 REM I/O ERROR PROCESSING
```

Assume disk file #3 is open for keyed-sequential/update access. When line 30 is executed, data values A, B, and C are read from the record of the next key in the disk file. C is then re-computed (line 40) and rewritten to the original record (line 50). Fields for values A and B on the record are not changed. The EXIT line number (100) will be used if an error occurs. If an I/O error has occured, the program will then branch to line 200.

REWRITE #file-ref,USING line-num{,INDIC char-expr}: expr-list {IOERR line-num}

or {EXIT line-num}

(Rewrite for a work station file)

This REWRITE statement rewrites data and attributes to a work station record that has previously been read. It is useful for updating a screen when only certain fields need changing. Expressions are evaluated then rewritten into the work station output buffer using the format given by the FORM statement at the specified USING line number.

The optional INDIC parameter is used to control visual and data entry indicators specified when the work station file is created. Characters in the INDIC character expression correspond positionally to indicators 01 to 99. The first character of the expression controls indicator 01. A character 1 sets indicator 01 to on while a character 0 sets it to off. The second character controls indicator 02 and so forth to 99. Indicators are initialized to off when the work station file is opened and remain off unless set on with INDIC. When INDIC is not used, indicators remained unchanged. For REWRITE, only fields with the output attribute conditioned by an indicator will be rewritten and then only if that particular indicator is turned on. Also, no screen clear is issued before the screen is rewritten.

IOERR or EXIT provides a branch line number should an error occur. An error occurs if the last operation to the work station file was not a READ.

Example:

```
20 WRITE #1,USING 25,FORMAT "CMM-001",INDIC "10": CNAME$,CZIP$,STAT$
25 FORM C 30,C 5,C 1
30 READ #1,USING 35: CNAME$,ZIP$,STAT$
35 FORM C 30,C 5,C 1
40 IF STAT$="A" OR STAT$="I" OR STAT$="D" THEN 70
50 REWRITE #1,USING 55,INDIC "11": " "
55 FORM POS 36,C 1
60 GOTO 30
70 ...
```

Assume work station file #1 is open and screen CMM-001 consists of three update fields: CNAME$, CZIP$, AND STAT$. Further, assume only the following attributes are conditioned by indicators:

FIELD	ATTRIBUTE	INDICATOR
STAT$	Output	01
STAT$	Position	02

When line 20 is executed, the screen is cleared and the screen format CMM-001 is displayed. Since attribute 01 is set on, the STAT$ field will be output. When line 30 is executed, screen input is accepted. Note that since indicator 02 is off, the cursor will appear at field CNAME$, the first field on the screen. When data entry is complete, the operator presses the RETURN key and execution continues at line 40. If STAT$ is a valid character (A, I, or D), the program continues at line 70. Otherwise, line 50 causes a rewrite and

sets indicators 01 and 02 on. Since only the STAT$ output field is conditioned by an indicator, it is rewritten while all other fields will remain unchanged. Because indicator 02 is turned on, the cursor will be positioned on the STAT$ field for the next read.

STOP

The STOP statement stops the program. All files remain open and a stop message is printed. To continue the program, a GO command must be entered from the keyboard. (See the explanation of the GO command in Section 5.)

Example:

```
100 STOP
```

When line 100 is executed, program execution is interrupted until a GO command is entered. Files open at this point are left open.

WRITE #fil-ref,USING line-num {,REC=num-expr}: expr-list {IOERR line-num}
{DUPKEY line-num}

(Write to a disk file)

or {EXIT line-num}

The WRITE statement writes data to a disk file specified by file reference number. Expressions are evaluated then written to the current record of the disk file using the specifications given in the FORM statement at the specified USING line number.

REC=num-expr specifies the record with the given record number be written. The file must be open for RELATIVE access.

Keyed files must be open for keyed-random access when writing new records. The key for the new record is taken from the key position and length as defined for the file. Before the record is added to the file, the key is checked against the key list. If a duplicate is found and DUPKEY is specified, then execution continues at the DUPKEY line number; otherwise, an error results.

When a non-keyed sequential file is written, new records are added to the end of file unless the BEGIN parameter was specified when opening the file. See the OPEN statement above for more information.

An I/O error can be trapped at the specified line by IOERR or an EXIT statement can be used.

Examples:

```
10 WRITE #1,USING 15: PART$,QTY  DUPKEY 100 IOERR 200
15 FORM C 7,X 30,ZD 6
 ...
100 REM DUPKEY ERROR PROCESSING
 ...
200 REM I/O ERROR PROCESSING
```

Assume file #1 is open for keyed-random access, and that the key position of the record begins in position 1 and has length of 7. Further, assume PART$ is 10-1054 and QTY is 100. When line 10 is executed, 10-1054 (PART$) is checked against both the sorted and unsorted key lists. If a duplicate is found, execution continues at line 100; otherwise, a new record is written to file #1 and 10-1054 is added to the unsorted key list. Should an I/O error occur, execution will continue at line 200.

```
10 WRITE #15,USING 15: A+B,C  EXIT 100
15 FORM POS 30,ZD 7.1,X 2,PD 4
 ...
100 EXIT IOERR 200
 ...
200 REM I/O ERROR PROCESSING
```

Assume file #15 is open for non-keyed sequential access. Since BEGIN is not specified, the new record containing the values of A+B and C is added to the end of the file. If an error occurs during the write, execution continues at line 100.

WRITE #fil-ref,USING line-num,FORMAT char-expr {,INDIC char-expr}: expr-list

(Write to a work station file)

The file reference must refer to a file opened as a work station. The screen is cleared, then the expression list values are displayed on the screen and written into the work station output buffer using the format given by the FORM statement at the specified USING line number.

FORMAT specifies which screen in the format file is to be used.

The optional INDIC parameter is used to control visual and data entry attribute indicators specified when the work station file is created. Characters in the INDIC character expression correspond positionally to indicators 01 to 99. The first character of the expression controls indicator 01. A character 1 sets indicator 01 to on while a zero sets it to off. The second character controls indicator 02 and so forth to 99. Indicators are initialized to off when the work station file is opened and remain off unless set on with INDIC. When INDIC is not used, indicators remained unchanged.

Example:

```
30 WRITE #3,USING 31,FORMAT "CMM-001",INDIC "1010": NAME$,ADRS$,CITY$
31 FORM 3*C 30
```

Assume file #3 is open as a work station file. When line 30 is executed, format CMM-001 is written to the display terminal along with the character values NAME$, ADRS$, and CITY$. Fields and attributes conditioned by indicators 01 and 04 are turned on; attributes conditioned by indicators 02 and 03 are turned off. This data is also written to the work station input buffer so that it can be changed during subsequent READ operations.

Section 7. INTRINSIC FUNCTIONS

DeltaBASIC provides a number of intrinsic functions that can be used in numeric and character expressions. Those that produce numerical results are classified as numerical functions, while those that produce character results are classified as character functions. The first list below contains numerical functions and the second, character functions.

INTRINSIC NUMERICAL FUNCTIONS

ABS(num-expr)

Gives the absolute value of the numeric expression.

ATN(num-expr)

Gives the angle (in radians) whose tangent is given by the numeric expression.

CMDKEY

Gives one of the following values, depending on which key was pressed to leave a work station screen read:

Key pressed	Value
RETURN	0
function key F1	1
function key F2	2
...	

ERR

Gives the error number of the last error. See Appendix A for a list of error numbers and messages.

EXP(num-expr)

Gives the value of constant e raised to the power of the numeric expression.

INT(num-expr)

Gives the largest integer less than or equal to the numeric expression.

LEN(char-expr)

Gives the number of characters in the string currently assigned to the character expression.

LINE

Gives the line number of the last error.

LOG(num-expr)

Gives the natural logarithm (base e) of the numeric expression.

POS(char-expr 1,char-expr 2 {,num-expr})

Gives the position of the first occurrence of char-expr 2 within char-expr 1. The search begins with position 1 unless num-expr is specified in which case the search begins with the position given by num-expr.

RND(num-expr)

Gives a pseudo-random number between 0 and 1, with the following variations:

num-expr < 0 then the pseudo-random number generator is reseeded.
num-expr = 0 then previous pseudo-random number is given.
num-expr > 0 then a new pseudo-random number is generated.

ROUND(num-expr 1,num-expr 2)

Rounds the value given by num-expr 1 to the number of decimal places given by the value of num-expr 2. If num-expr 2 is zero, num-expr 1 will be rounded to a whole number. A negative value for num-expr 2 will cause rounding to that many positions left of the decimal point.

SGN(num-expr)

Gives the value:

1 if num-expr > 0.
0 if num-expr = 0.
-1 if num-expr < 0.

SIN(num-expr)

Gives the sine of the angle (in radians) specified in the expression.

SQR(num-expr)

Gives the square root of the numeric expression.

STATUS(num-cnst 1 [,num-cnst 2])

Gives the status of certain system parameters, depending on num-cnst 1:

num-cnst 1	STATUS
0	Last keyboard key pressed
1-254	File information--see below
255	Current printer line
256	Number of bytes of unused memory
257	Current user number

If num-cnst 1 is a file reference number of an open file, then num-cnst 2 determines which additional file parameters are given.

num-cnst 2	STATUS
0	Record length--all DeltaBASIC files
1	End-of-file record number--all DeltaBASIC files
2	Number of sorted keys--keyed files only
3	Number of unsorted keys--keyed files only

VAL(char-expr)

Gives the numeric value of the character expression.

INTRINSIC CHARACTER FUNCTIONS

CHR$(num-expr 1[,num-expr 2])

Gives the ASCII character with value of num-expr 1. When num-expr 2 is specified, the ASCII character is repeated the number of times given by num-expr 2.

CNVRT$(char-expr,num-expr)

Converts the value of the numeric expression to a character string, using the format specified by the character expression. The N, ZD, PD, and PIC format specifications can be used (see the FORM statement in Section 6 above).

DATE$

Gives the date in character string representation YYYYMMDD. For example, the value of DATE$ for March 16, 1945 is 19450316.

STR$(num-expr)

Converts the value of the numeric expression to a character string.

TIME$

Gives the time in character string representation (HH:MM:SS).

Character substring function

(num-expr 1:num-expr 2)

Creates a substring beginning with the position given by num-expr 1 and ending with the position given by num-expr 2. If the substring specifications are beyond the last character of the string, a null character string is returned. The substring function can follow character variables and character functions.

Examples:

If A$ is ABCD, then A$(2:3) will be BC.

If TIME$ is 02:30:15, then TIME$(4:5) will be 30.

Section 8. UTILITY PROGRAMS

In addition to DeltaBASIC itself there are several utility programs provided by Deltasoft, Inc. to assist you in program development and file maintenance. The utility programs discussed in this section are:

PROGRAM	DESCRIPTION
GENMSG	Used to create field message files
KEYSORT	Used to keysort a DeltaBASIC keyed data file
ORGANIZE	Used to perform certain file maintenance operations
RENUM	Used to renumber DeltaBASIC programs
SDU	Used to design formatted data entry screens

GENMSG

GENMSG is a utility for generating field message (HELP) files used with formatted data entry screens. GENMSG is invoked from the operating system by using:

```
GENMSG {drv:}filename (return key)
```

The filename refers to an existing ASCII text file that contains a list of the messages intended to accompany a formatted data entry screen. The filename must be the same as the screen name and be of file type LIB. Each field message in the LIB file consists of a comment line beginning with a pound sign (#) followed by one or more message lines, the last of which is terminated with a vertical bar symbol (|).

The LIB file can be created with any standard text editor. Each line should be terminated by a carriage return and line feed. The order of the message corresponds to the order of the fields on the formatted data entry screen, except that the first message block is reserved as a global message for the entire screen. The second message is associated with the data entry field nearest the upper left corner of the screen. The remaining messages are associated with the remaining data entry fields, proceding from left to right and top to bottom down the screen.

GENMSG uses the LIB file to create a second file with the same name but of type MSG. This file, not the LIB file, is accessed by DeltaBASIC to display field messages.

To illustrate the use of screen messages, consider the simple formatted data entry screen below:

```
HELP MESSAGE EXAMPLE SCREEN

Data Output Field    00000000

Data Input Field 1:  ********

Data Input Field 2:  ********
```

Assume the screen name is HME-001 and is contained in format file HELPEX.FRM. The screen consists of four constant or label fields, one output only field (designated by 00000000), and two data entry fields (designated by ********).

The first step is to create a HELP message (LIB) file with the name HME-001 using a standard text editor. The result appears below:

```
# The following is a global HELP message for screen HME-001:


Screen HME-001 is used to illustrate HELP messages for DeltaBASIC.|

# The following is a HELP message for data entry field 1:


This is the HELP message for data entry field 1.|

# The following is a HELP message for data entry field 2:


This is the HELP message for data entry field 2.|

(end-of-file mark: 1A hex)
```

Next the LIB message file must be converted to a MSG file for use by DeltaBASIC. This is accomplished by using the GENMSG utility as follows:

GENMSG HME-001 (return key)

When the GENMSG utility finishes, the file HME-001.MSG will exist on the default drive.

The screen HELP messages are now ready to access from DeltaBASIC. Assume the HELPEX.FRM is open as a work station file and screen HME-001 has been displayed with a WRITE statement. After executing a work station read, the screen looks like this:

```
                    HELP MESSAGE EXAMPLE SCREEN

                     Data Output Field    HELPTEST

                     Data Input Field 1:  #

                     Data Input Field 2:
```

The string "HELPTEST" occupies the output field and blanks occupy the two data entry fields. The pound symbol (#) indicates the location of the cursor. Pressing the HELP key (^Z) displays the HELP message for the first data entry field. (The second HELP message is selected--the first is the screen global message.) The screen then looks like this:

```
This is a help message for data entry field 1.
```

Pressing the HELP key another time displays the global HELP message:

```
Screen HME-001 is used to illustrate HELP messages for DeltaBASIC.
```

Pressing any other key returns the original work station screen with everything exactly as it was before.

Had the cursor been positioned at the second data entry field, pressing the HELP key would have displayed the third HELP message:

```
This is a help message for data entry field 2.
```

Otherwise, operation would be exactly as in the case above.

KEYSORT

The KEYSORT utility reorders the key file of a DeltaBASIC keyed data file.

The key file (type KEY) is actually composed of two lists. The first list contains sorted keys with pointers to records in the data file. The second list contains unsorted keys and pointers associated with data that has been added to the file since the last KEYSORT was performed. When the file is accessed randomly, the sorted and unsorted lists are both checked using a binary tree search algorithm. If the requested key is found, the pointer is used to gain access to the corresponding data record. Keyed sequential access is managed differently. Pointers from the sorted key portion of the key file are accessed sequentially, resulting in key-order processing of the corresponding data records. In this case, keys in the unsorted part of the key file are not accessed.

The KEYSORT utility sorts the unsorted keys and merges them into the sorted list ensuring that all data records will be accessed during keyed sequential processing. For more information, see Appendix C, Disk File I/O.

The KEYSORT utility is invoked from the operating system using the following command:

```
KEYSORT {drv:}name (return key)
```

The filename must refer to a keyed data file. No file type is needed.

ORGANIZE

The ORGANIZE utility is used to perform certain data file management operations. These include:

(i) Convert an ASCII text file to a DeltaBASIC data file.
(ii) Convert a DeltaBASIC data file to an ASCII text file.
(iii) Remove marked records from a DeltaBASIC data file.
(iv) Convert a DeltaBASIC data file to a keyed file.

ORGANIZE is an interactive program that is invoked from the operating system by entering:

ORGANIZE (return key)

You are then presented with a series of questions that, when answered, determine the specific operation that is to take place. The remainder of this section describes the dialog for each of the four operations listed above.

(i) Converting an ASCII text file to a DeltaBASIC data file.

Occasionally it is necessary to convert data from an incompatible format for use with DeltaBASIC. The first step involves building an ASCII text file with the data written on each line exactly in the position it will occupy in the DeltaBASIC data file. Each text line should be terminated by a carriage return and line feed. ORGANIZE is then used to copy each line of ASCII text into a separate record in the DeltaBASIC data file, preserving the data's positional relationship. If the text line is shorter than the DeltaBASIC data file record, the remainder of the DeltaBASIC data record will be filled with blanks. If the text line is longer than the DeltaBASIC data file record, the text beyond the end of the record will be ignored. The interactive steps for this operation are as follows:

ORGANIZE Question	Your Reply	Explanation
INPUT FILE		
File name?	{drv:}name{.typ}	Enter the name of the ASCII text file
File type?	A	Enter A for ASCII
OUTPUT FILE		
File name?	{drv:}name{.typ}	Enter the name of the new DeltaBASIC data file
Record length?	num-cnst	Enter the record length of the DeltaBASIC data file
Continue?	Y or N	Enter Y to complete the operation or N to restart

(ii) Converting a DeltaBASIC data file into an ASCII text file

You can use ORGANIZE to prepare an ASCII text file that can be used by another program such as a text editor. This is in effect the reverse of operation (i) above. The resulting ASCII text file will have line lengths exactly equal to the record length of the DeltaBASIC data file. Follow these steps to complete this operation:

ORGANIZE Question	Your Reply	Explanation
INPUT FILE		
File name?	{drv:}name{.typ}	Enter the name of the DeltaBASIC data file
File type?	D	Enter D for data
Key position?	(return key)	Press the RETURN key
Delete position?	(return key)	Press the RETURN key
OUTPUT FILE		
File name?	{drv:}name{.typ}	Enter the name of the ASCII text file
File type?	A	Enter A for ASCII
Continue?	Y or N	Enter Y to complete the operation or N to restart

(iii) Removing marked records from a DeltaBASIC data file

Deleting a record from a DeltaBASIC data file (keyed or not) is done by marking the record in some way, then copying all unmarked records to a new file. In defining the fields of a data file, it is useful to include a one character status field for this purpose. To eliminate marked data records using ORGANIZE, follow these steps:

ORGANIZE Question	Your Reply	Explanation
INPUT FILE		
File name?	{drv:}name{.typ}	Enter the name of the DeltaBASIC data file
File type?	D	Enter D for data
Key position?	(return key)	Press the RETURN key
Delete position?	num-cnst	Enter the position of the delete field
Delete character?	num-cnst or "character"	Enter the numeric value of the ASCII delete character or the character itself enclosed in quotes.
OUTPUT FILE		
File name?	{drv:}name{.typ}	Enter the name of the new DeltaBASIC data file
File type?	D	Enter D for data
Record length?	num-cnst	Enter the record length of the new DeltaBASIC data file
Continue?	Y or N	Enter Y to complete the operation or N to restart

Note: If the new data file is to be keyed, use the procedure of (iv) below to rebuild the keyfile.

(iv) Converting a DeltaBASIC data file to a keyed file

Several situations may arise that call for the conversion of a DeltaBASIC data file into a keyed file. The list below contains a few examples:

1. A new DeltaBASIC data file was built from an ASCII text file or by deleting records from an old data file (see above) and it needs to be converted to a keyed file.

2. A DeltaBASIC data file was built or extended using sequential processing to save time and now must be converted to keyed format for further work.

3. An existing DeltaBASIC keyed data file needs rekeying, possibly with a new key position and/or length.

Follow these steps to convert a DeltaBASIC data file to a keyed file:

ORGANIZE Question	Your Reply	Explanation
INPUT FILE		
File name?	{drv:}name{.typ}	Enter the name of the DeltaBASIC data file
File type?	D	Enter D for data
Key position?	num-cnst	Enter the position of the key
Key length?	num-cnst	Enter the length of the key
OUTPUT FILE		
File name?	{drv:}name.KEY	Enter the name of the DeltaBASIC key file
Continue?	Y or N	Enter Y to complete the operation or N to restart

RENUM

The RENUM utility program is used to renumber lines of a non-source DeltaBASIC program. RENUM is invoked from the operating system using the following command:

RENUM {drv:}name.BAS {;num-cnst 1,num-cnst 2,num-cnst 3,num-cnst 4} (return key)

The parameters are defined as follows:

{drv:}name.BAS	The non-source DeltaBASIC program to be renumbered.
num-cnst 1	New beginning line number (default 10)
num-cnst 2	Increment lines by this value (default 5)
num-cnst 3	Begin renumber at this line number (default 1)
num-cnst 4	End renumber at this line number (default 65534)

After successful completion, the newly renumbered version will have the same name as the original but with type REN. The original program is unchanged.

SDU

SDU (Screen Design Utility) is used to create and modify formatted data entry screens in a screen format file. SDU is a menu driven, interactive program with full screen editing capabilities. For a given formatted data entry screen, constant and data fields can be visually arranged in any way desired so long as a blank space precedes and follows each field. Data entry and visual attributes can be selected for each field.

The Screen Design Utility is invoked by entering:

SDU {{drv:}name} (return key)

where name refers to a screen format file of type FRM.

SDU then displays a menu of eight options along with the name of the currently selected format file and a directory of associated screens. The menu options are listed below with a detailed description of each. To select an option, simply press the key shown in parenthesis.

(A)dd a new screen to a format file.

The add facility will first prompt you to specify a screen name of eight characters or less. After the screen name has been entered you will be given the opportunity to enter the new screen from the (K)eyboard or retrieve it from the (D)isk. If you press the D key you will be prompted for the format file and name of the screen to be added, after which SDU will automatically enter the (U)pdate option of the menu for the new screen. If you press the K or RETURN key (K is the default), the display will be cleared, after which you may place constant and data fields on the screen as they will be viewed. SDU requires that the first and last column of the display not be used and that there be a minimum of two spaces horizontally between any two fields.

The text editing capablities of SDU can be reviewed by pressing the help key (^]) or by referring to the table below. Note that some of the editing features are not allowed when adding a screen.

After all fields have been placed satisfactorily, a specification character should be placed before each field and a terminator character after each field.

Specification characters determine the default data entry and visual attributes for a field. The basic specification characters are:

c -- low intensity, constant field

d -- low intensity, input/output data field

C -- normal intensity, constant field

D -- normal intensity, input/output data field

In addition, four other specification characters are user definable with similar default values:

a -- low intensity, constant field

b -- low intensity, input/output data field

A -- normal intensity, constant field

B -- normal intensity, input/output data field

These defaults can be changed by entering ^X while adding or updating a screen.

A terminator character (|) must be placed to mark the end of each field. When all fields, specification characters, and terminator characters have been properly placed, "enter" the screen by pressing the ATTN key. Control is automatically passed to the (U)pdate option of the menu so that changes can be made to the attributes of each field. If no changes are necessary, press the ATTN key again to return to the menu; otherwise, refer to the (U)pdate option for detailed instructions (ignore the part requesting a screen name to update).

(U)pdate an existing screen in a format file.

The update facility will ask you to specify the name of a screen in the currently selected format file. After the screen name has been entered, the previously defined constant and data fields will be displayed and you can then add, change, or remove fields as desired. SDU requires that the first and last column of the display not be used and that there be a minimum of two spaces horizontally between any two fields.

The text editing capablities of SDU can be reviewed by pressing the help key (^]) or by referring to the table below. Note that some of the editing features are not allowed when updating a screen.

To add a field, enter the field on the screen with specification and terminator characters as described in the (A)dd a Screen menu option above. Next move the cursor to the beginning of the field (adjacent to the specification character) and press ^A. The field will then be added and the attribute screen displayed to give you the option to modify the defaults given by the specification character. To change the attributes of a previously created field, move the cursor to the beginning of the field and press ^C after which the list of attributes will be displayed for modification. To remove a field, move the cursor to the beginning of the field and press ^R. Note that, once removed, a field cannot be recovered.

When finished, press the ATTN key to "enter" the screen. For convenience, you will be offered the option of viewing or printing the row and column positions, lengths, buffer positions, and attributes of the screen fields. This information is very useful when writing the FORM specification using the screen in DeltaBASIC.

(D)elete a screen from a format file.

The delete facility will prompt you to specify the name of a screen in the currently selected format file. The screen will be permanently deleted (erased) from the format file. Once deleted a screen cannot be recovered.

(C)hange the name of an existing screen.

The change facility will prompt you to specify the name of a screen in the currently selected format file. If the screen exists you can optionally change the name or press the RETURN key to leave it unchanged.

(V)iew screen attributes.

The view facility will prompt you to specify the name of a screen in the currently selected format file. After the screen name has been entered you can choose to list on the (S)creen or (P)rinter the row and column positions, lengths, buffer positions, and attributes of all fields in the specified screen.

(R)emove a format file.

The remove facility will remove (erase) the currently selected format file from memory and the disk. Note that removing a format file is final--once removed it cannot be recovered.

(S)elect a new format file.

The select facility will prompt you to specify the name of a format file. After the name has been entered, the format file, if found, will become the currently selected format file and its associated screens will be displayed under the menu.

(E)xit to System.

This option will ensure that all changes to the currently selected format file and associated screens are placed on the disk and return control to the operating system.

Editing Key Table

When you are adding and updating screens, a versatile set of editing keys is available for your use. In the table below, the keys are arranged into four groups: cursor movement, insert/delete, field, and miscellaneous. To use an editing key, hold the CONTROL key down and press the specified key. Note the exceptions for commonly used keys such as RETURN or ATTN. A list of the editing keys and their definitions follow:

	Editing Key	Function
(Cursor Movement)	^K	Cursor up
	^J <LINE FEED>	Cursor down
	^H	Cursor left
	^L	Cursor right
	^^ <HOME>	Cursor to home position
	^B	Cursor to bottom line
	^T	Cursor to top line
	^W	Cursor word left
	^I <TAB>	Cursor word right
	^M <RETURN>	Cursor to new line
	^S	Cursor to screen left
	^D	Cursor to screen right
(Insert/Delete)	^F	Insert mode ON/OFF toggle
	^V	Same as above
	^N	Insert blank line
	^_ <DEL>	Delete character left
	^G	Delete character right
	^Y	Delete line
	^U	Delete line left
	^P	Delete line right
(Field)	^A	Add a field
	^C	Change a field
	^R	Remove a field
	^X	Change default attributes
(Miscellaneous)	^O	Abort add or update
	^Q	Reprint screen
	^Z	Clear screen
	^[<ESCAPE>	Enter screen
	^] <HELP>	Display help screen

Note: When in the (A)dd mode, the Field keys may not be used. When in the (U)pdate mode, the Insertion/Deletion and Clear Screen keys may not be used.

SDU Errors

Errors can occasionally occur when using SDU. A list of SDU error messages with explanations follow:

Directory full:	The disk directory is full. See your operating system manual for explanation and corrective measures.
Field overlap:	An attempt has been made to force two fields to use overlapping character positions on the screen. The second field has been removed before the screen was saved (this error occurs when ATTN key is pressed and the screen is entered).
File not found:	An attempt has been made to select a format file that was not present on the disk.
Format file full:	An attempt has been made to place more than the maximum number of twelve screens in a format file.
Read:	Disk read error. See your operating system manual for explanation and corrective measures.
Screen duplication:	An attempt has been made to add a screen that has the same name as a previously entered screen.
Screen not in format file:	An attempt has been made to update, delete, or view a screen that does not exist in a format file.

Appendix A. ERROR MESSAGES

When an error occurs and no error trap is in effect, a short error message is displayed on the terminal. The message has the following form:

(error number) error code Error in Line line number

The *HELP* command can be used to call up an explanation of the last error identified. The table below lists the error codes with corresponding error numbers and explanations. If the error can be trapped, the trap condition to use is given. (See ON...ERROR in Section 6.)

CODE	NUMBER	TRAP	EXPLANATION
SYNTAX	01		SYNTAX ERROR: an unrecognizable command or statement was encountered.
SYN EOS	02		SYNTAX ERROR AT END OF STATEMENT: colon or line end expected.
CHV EXP	03		CHARACTER VARIABLE EXPECTED: type mismatch.
NUM EXP	04		NUMERIC VARIABLE EXPECTED: type mismatch.
ARY EXP	05		ARRAY VARIABLE EXPECTED: type mismatch.
INV VAR	06		INVALID VARIABLE: an array variable was expected.
UDF VAR	07		UNDEFINED VARIABLE: the variable referenced does not exist.
PAR MIS	08		PARENTHESIS MISSING: an open or close parenthesis is missing.
REL EXP	09		RELATIONAL OPERATOR EXPECTED: a relational operator (< > =) was expected.
INV CFN	10		INVALID CHARACTER FUNCTION: a character function was expected.
CFN NAM	11		CHARACTER FUNCTION NAME: the $ is missing from the function name.
INV NFN	12		INVALID NUMERIC FUNCTION: a numeric function was expected.
VAR EXP	13		VARIABLE EXPECTED: a variable was expected.

CODE	NUMBER	TRAP	EXPLANATION
SYN EOL	14		SYNTAX ERROR AT END OF LINE: line end expected.
DUP VAR	15		DUPLICATE VARIABLE: variable already defined.
DIM SZE	16		DIMENSION SIZE: array dimension exceeds maximum.
ON RNGE	17	ERROR	ON RANGE: range of ON value exceeds maximum.
NUM DTA	18		NUMERIC DATA: numeric data was expected.
FOR LEV	19		FOR LEVEL: exceeds maximum allowed nesting (8 max).
FOR RNG	20		FOR RANGE: limits and step incompatible.
OUT DTA	21	ERROR	OUT OF DATA: attempt was made to READ past last data value.
NUM RNG	22		NUMERIC RANGE: integer value maximum exceeded (255 max).
OUT MEM	23		OUT OF MEMORY: program needs more memory to execute.
INV STM	24		INVALID STATEMENT: statement not recognizable.
STK OVF	25		STACK OVERFLOW: expression too complex.
STK UNF	26		STACK UNDERFLOW: (System error)
ATTN	27		
LNE RNG	28		LINE RANGE: beginning line number exceeds ending line number.
UNK LNE	29		UNKNOWN LINE: line referenced does not exist.
SUB STR	30	ERROR	SUBSTRING: cannot be formed as specified.
INV ARG	31		INVALID ARGUMENT: type mismatch or value excessive.
DIR STM	32		DIRECT STATEMENT: not allowed in direct execution.
INV UFN	33		INVALID USER-DEFINED FUNCTION: not a valid user defined function.
SUB LVL	34		SUBROUTINE LEVEL: exceedes maximum (20 max).
INV RET	35		INVALID RETURN: encountered without a corresponding GOSUB.

CODE	NUMBER	TRAP	EXPLANATION
MIS FOR	36		MISSING FOR: a NEXT was encountered without a corresponding FOR.
TAB	37	CONV	TAB: argument is invalid.
INV IFN	38		INVALID INTRINSIC FUNCTION: not a valid intrinsic function.
UNK UFN	39		UNKNOWN USER DEFINED FUNCTION: was encountered.
	40		Not used.
NUM OVR	41	OFLOW	NUMERIC OVERFLOW: floating point value exceeds maximum.
NUM UND	42	UFLOW	NUMERIC UNDERFLOW: floating point value smaller than minimum.
DIV ZER	43	ZDIV	DIVISION BY ZERO: expression contains a division by zero.
STR LEN	44	SOFLOW	STRING LENGTH: character string length exceeds maximum for variable.
DUP KEY	45	DUPKEY	DUPLICATE KEY: key file already contains this key.
NO KEY	46	NOKEY	NO KEY: key specified not in key file.
EOF	47	EOF	END OF FILE: cannot READ/INPUT past end of file.
	48		Not used.
	49		Not used.
	50		Not used
PER I/O	51	IOERR	PERMANENT I/O: error encountered in disk input/output operation.
CHG KEY	52	IOERR	CHANGE KEY: attempt was made to change the key portion of the record.
INC KEY	53	IOERR	INCORRECT KEY:
INC DTA	54	IOERR	INCORRECT DATA:
NO DTA	55	IOERR	NO DATA:

CODE	NUMBER	TRAP	EXPLANATION
DSK FUL	56	IOERR	DISK FULL: the disk is full.
FLE EXS	57	IOERR	FILE EXISTS: attempt was made to create a file that already exists.
NO FILE	58	IOERR	NO FILE: attempt was made to access a file that does not exist.
KEY L/P	59	IOERR	KEY LENGTH/POSITION: specified incorrectly.
DIR FUL	60	IOERR	DIRECTORY FULL: disk directory is full.
NO PRGM	61	IOERR	NO PROGRAM: program file specified does not exist.
FLE TYP	62	IOERR	FILE TYPE: mismatch.
PRGM LD	63	IOERR	PROGRAM LOAD: an error was detected during program load.
REC LEN	64	IOERR	RECORD LENGTH: is 0 or exceeds 2048.
	65		Not used.
	66		Not used.
FRM BUF	67	CONV	FORMAT BUFFER: workstation format buffer length exceeds record length specified in OPEN statement.
INV PRT	68	CONV	INVALID PRINT: operation CUR, TAB, or SKIP not allowed in disk or workstation files.
CHR I/O	69	CONV	CHARACTER VARIABLE INPUT/OUTPUT: character variable expected in I/O operation.
NUM I/O	70	CONV	NUMERIC VARIABLE INPUT/OUTPUT: numeric variable expected in I/O operation.
FLD LEN	71	CONV	FIELD LENGTH: attempt was made to write data into a field that is too small.
FRM PRT	72	CONV	FORM PRINT: wrong FORM specification for PRINT statement.
FRM SPC	73	CONV	FORM SPECIFICATION: incorrect FORM specification.
INV FRM	74	CONV	INVALID FORMAT: invalid workstation format.

CODE	NUMBER	TRAP	EXPLANATION
FRM MDE	75	CONV	FORMAT MODE: format mode error.
	76		Not used.
FLE REN	77	IOERR	FILE RENAME: file must exist and not be in use by another process.
FLE NAM	78		FILE NAME: must be 8 characters or less plus optional type.
DRV SEL	79		DRIVE SELECT: cannot select specified disk drive.
UNKNOWN	80		UNKNOWN: system error.
HLP FLE	81		HELP FILE: DBXERR.MSG and DBXREF.MSG must be on default disk drive.
FLE OPN	82		FILE OPEN: file specified already open.
FLE MOD	83		FILE MODE: open mode (or parameter) incorrect for access attempted.
INV FLE	84		INVALID FILE: file specified not open.
FLE NAV	86		FILE NOT AVAILABLE: in use by a NOSHR process.
FLE CLS	87	IOERR	FILE CLOSE: cannot close file.
INV REC	88		INVALID RECORD NUMBER: zero not allowed.
PR WRT	89		PREVIOUS WRITE: last access must have been a write.
MIS REC	90		MISSING RECORD NUMBER: record number must be specified.
MIS USI	91		MISSING USING: USING statement missing.
PR READ	92		PREVIOUS READ: last access must have been a read.
INV DEV	93		INVALID DEVICE: not a valid device.
PR LIN	94		PREVIOUS LINE: previous line error.
	95-100		Not used.

Appendix B. RESERVED WORDS

The following words are reserved for use by DeltaBASIC and cannot be used as variable names, function names, or disk file names.

ABS	AND	ASC	ATN	ATTN
AUTO	BEGIN	CHAIN	CHR	CLEAR
CLOSE	CMDKEY	CNVRT	CONV	CUR
DATA	DATE	DEF	DEL	DIM
DUPKEY	ELSE	END	EOF	ERR
ERROR	EXIT	EXP	FILES	FN
FOR	FORM	FORMAT	FREE	GO
GOSUB	GOTO	HELP	IF	IGNORE
INDIC	INPUT	INT	IOERR	KEY
KEYED	KEYL	KEYP	LEN	LINE
LINPUT	LIST	LISTP	LOAD	LOG
LPREC	NAME	NEWPAGE	NEXT	NOKEY
NOT	OFF	OFLOW	ON	OPEN
OPTION	OR	OUTIN	OUTPUT	PIC
POS	PRINT	PRTZO	RANDOM	READ
REC	RECL	RELATIVE	REM	RENAME
RESET	RESTORE	RETURN	REWRITE	RND
ROUND	SAVE	SGN	SIN	SKIP
SOFLOW	SPREC	SQR	STATUS	STEP
STR	TAB	THEN	TIME	TO
UFLOW	USING	VAL	WRITE	WS
ZDIV				

Appendix C. Disk File I/O

DeltaBASIC provides a number of different techniques for data file management. Each involves a different combination of file access method (sequential, relative, keyed-random, and keyed-sequential) and mode (output, input, and update). To help you compare these techniques, the descriptions that follow include examples based on the sample data file given below:

File Name: INVMAS.DAT Record Length: 40

Field Description	Variable Name	FORM Spec	FORM Size	Start Pos	End Pos
Part number	PART$	C	5	1	5
Description	DESC$	C	30	6	35
Quantity on hand	QUAN	N	4.0	36	39
Status (D=delete)	STAT$	C	1	40	40

```
             1         2         3         4
Rec#  ....|....0....|....0....|....0....|....0

 1    81201SOCKET WRENCH                   10
 2    10202BALL PEEN HAMMER                 5
 3    64394PHILLIPS SCREW DRIVER           15
 4    40121HAND SAW                         8
 5    (end-of-file record)
```

Sequential Access

With sequential access, records in the file are processed in order by record number. Sequential access is selected by default when neither the RELATIVE nor the KEYED attribute is used in the OPEN statement.

Sequential Access in Output Mode

For sequential/output access, records are added at the current end-of-file record or--if the BEGIN parameter is used--beginning at record 1. File reads are not allowed. Sequential/output access is selected in the OPEN statement when OUTPUT is used without either RELATIVE or KEYED as a file attribute.

If a new file is created, records are added beginning with record 1, then record 2, and so forth. For example, the sample data file above could be created using the program below:

```
 1 |
 2 | A PROGRAM ILLUSTRATING SEQUENTIAL/OUTPUT ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT,NEW,RECL=40",OUTPUT
20 STAT$=" "
25 READ N | THE NUMBER OF DATA SETS
30 FOR I=1 TO N
35 READ PART$,DESC$,QUAN
40 WRITE #1,USING 45: PART$,DESC$,QUAN,STAT$
45 FORM C 5,C 30,N 4,C 1
50 NEXT I
55 CLOSE #1:
60 STOP
65 DATA 4
70 | DATA SETS
75 DATA 81201,SOCKET WRENCH,10
80 DATA 10202,BALL PEEN HAMMER,5
85 DATA 64394,PHILLIPS SCREW DRIVER,15
90 DATA 40121,HAND SAW,8
```

The program creates a new file, INVMAS.DAT, with record length 40 (line 15). The access method is sequential and the mode is output. The number of data records to be added is read from internal data (line 25), then a FOR/NEXT loop is used to read the data sets and write new records to the data file (lines 30-50). Note that the records are added to the file in order, beginning with record 1 and continuing through record 4. The data order within the file is determined by the order in which the records are written.

If a file already exists when sequential/output is selected, the absence or presence of the BEGIN attribute determines where the new records are added. These situations are illustrated by two program examples below:

In the first program, the BEGIN attribute is absent resulting in new data being added at the current end-of-file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING SEQUENTIAL/OUTPUT ACCESS WITHOUT BEGIN
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",OUTPUT
20 STAT$=" "
25 READ N | THE NUMBER OF DATA SETS TO BE ADDED
30 FOR I=1 TO N
35 READ PART$,DESC$,QUAN
40 WRITE #1,USING 45: PART$,DESC$,QUAN,STAT$
45 FORM C 5,C 30,N 4,C 1
50 NEXT I
55 CLOSE #1:
60 STOP
65 | DATA SETS
70 DATA 2
75 DATA 91324,WRENCH,5
80 DATA 68923,PLIERS,12
```

Line 15 opens file INVMAS.DAT for sequential/output access. Since the BEGIN attribute is absent, new data records are added at the current end-of-file. The number of data sets to add is read from internal data (line 25), and a FOR/NEXT loop appends the new records to the file (lines 30-50). The program arranges the sample data file as shown below:

```
                1         2         3         4
Rec#  ....|....0....|....0....|....0....|....0

 1    81201SOCKET WRENCH                    10
 2    10202BALL PEEN HAMMER                  5
 3    64394PHILLIPS SCREW DRIVER            15
 4    40121HAND SAW                          8
 5    91324WRENCH                            5
 6    68923PLIERS                           12
 7    (end-of-file record)
```

The first four records are the same as before. The new data has been added in records 5 and 6, moving the end-of-file to record 7.

In the second program, the BEGIN attribute is present resulting in new data being written beginning at record 1 of the file. Data currently in the file is lost.

```
 1 |
 2 | A PROGRAM ILLUSTRATING SEQUENTIAL/OUTPUT ACCESS WITH BEGIN
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",OUTPUT,BEGIN
20 STAT$=" "
25 READ N | THE NUMBER OF DATA SETS TO BE ADDED
30 FOR I=1 TO N
35 READ PART$,DESC$,QUAN
40 WRITE #1,USING 45: PART$,DESC$,QUAN,STAT$
45 FORM C 5,C 30,N 4,C 1
50 NEXT I
```

```
55 CLOSE #1:
60 STOP
65 | DATA SETS
70 DATA 2
75 DATA 91324,WRENCH,5
80 DATA 68923,PLIERS,12
```

The program is identical to one immediately above except for the addition of the BEGIN attribute (line 15). After the program is executed, the sample data file looks like this:

```
                1         2         3         4
Rec#  ....|....0....|....0....|....0....|....0

 1    91324WRENCH                           5
 2    68923PLIERS                          12
 3    (end-of-file record)
```

Note that all the original data in the file is gone and only the new data remains.

Sequential Access in Input Mode

For sequential/input access, records are read from the file in order by record number beginning with record 1. No file writes are allowed. Sequential/input access is selected in the OPEN statement when INPUT is used without either RELATIVE or KEYED as a file attribute.

The program below uses this processing technique with the original sample data file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING SEQUENTIAL/INPUT ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",INPUT
20 READ #1,USING 25: PART$,DESC$,QUAN,STAT$  EOF 40
25 FORM C 5,C 30,N 4,C 1
30 PRINT PART$,DESC$,QUAN,STAT$
35 GOTO 20
40 CLOSE #1:
45 STOP
```

Line 15 opens the file INVMAS.DAT for sequential/input access. The program then reads data records in record number order beginning with record 1, and prints them on successive lines (lines 20-30). When the end-of-file record is reached (line 20), the file is closed and execution ends (lines 40-45). The resulting printout appears below:

```
81201          SOCKET WRENCH                      10
10202          BALL PEEN HAMMER                    5
64394          PHILLIPS SCREW DRIVER              15
40121          HAND SAW                            8
```

Note that the file data is accessed and printed in exactly the order it appears in the file.

Sequential Access in Update Mode

For sequential/update access, records are read and ,if desired, rewritten (with the REWRITE statement) in record number order beginning with record 1. Existing records can be changed but new records cannot be added (the WRITE statement is not allowed). Sequential/update access is selected in the OPEN statement when no file attributes are used.

The program below illustrates how to update the original sample data file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING SEQUENTIAL/UPDATE ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT"
20 READ #1,USING 25: PART$,DESC$,QUAN,STAT$  EOF 50
25 FORM C 5,C 30,N 4,C 1
30 IF PART$>"50000" THEN 20
35 REWRITE #1,USING 40: "D"
40 FORM POS 40,C 1
45 GOTO 20
50 CLOSE #1:
55 STOP
```

The program opens file INVMAS.DAT for sequential/update access (line 15). Records are then read in order as they appear in the file (line 20), and those with part numbers less than or equal to 50000 are rewritten with the status field changed to D (lines 30-45). When the end-of-file is reached (line 20), the file is closed and execution ends (lines 50-55). The sample data file now looks like this:

```
                1         2         3         4
Rec#  ....|....0....|....0....|....0....|....0

 1    81201SOCKET WRENCH                   10
 2    10202BALL PEEN HAMMER                 5D
 3    64394PHILLIPS SCREW DRIVER           15
 4    40121HAND SAW                         8D
 5    (end-of-file record)
```

Records 2 and 4 are marked with a D. No other file changes are made.

Relative Access

When RELATIVE is used in the file ID of the OPEN statement, records in the file are accessed by specifying a record number in the REC parameter of READ and WRITE statements. Relative access differs from sequential in that records can be accessed in any order simply by specifying their record numbers.

Relative Access in Output Mode

For relative/output access, records are replaced or added to the file by specifying the record number using the REC parameter in a WRITE statement. File reads are not allowed. Relative/output access is selected in the OPEN statement when both RELATIVE and OUTPUT are used as file attributes.

The program below illustrates how to replace a record in the original sample data file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING RELATIVE/OUTPUT ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",RELATIVE,OUTPUT
20 READ R,PART$,DESC$,QUAN
25 WRITE #1,USING 30,REC=R: PART$,DESC$,QUAN," "
30 FORM C 5,C 30,N 4,C 1
35 CLOSE #1:
40 STOP
45 DATA 2,20013,CARRIAGE BOLT - 1/2 INCH,50
```

The program opens the file INVMAS.DAT for relative/output access (line 15). New data is then written to record 2 (lines 20-30), replacing the existing record. The sample data file now appears as appears as shown below:

```
                1         2         3         4
Rec#  ....|....0....|....0....|....0....|....0

 1    81201SOCKET WRENCH                    10
 2    20013CARRIAGE BOLT - 1/2 INCH         50
 3    64394PHILLIPS SCREW DRIVER            15
 4    40121HAND SAW                          8
 5    (end-of-file record)
```

Record 2, which originally contained the BALL PEEN HAMMER record, now contains the CARRIAGE BOLT record.

Relative Access in Input Mode

For relative/input access, records are read from the file by specifying the record number in the REC parameter of the READ statement. File writes are not allowed. Relative/input access is selected in the OPEN statement when both RELATIVE and INPUT are used as file attributes.

The program below illustrates how to read a record in the original sample data file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING RELATIVE/INPUT ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",RELATIVE,INPUT
20 READ #1,USING 25,REC=3: PART$,DESC$,QUAN
25 FORM C 5,C 30,N 4
30 PRINT PART$,DESC$,QUAN
35 CLOSE #1:
40 STOP
```

The program opens file INVMAS.DAT for relative access in the input mode (line 15). Record 3 is specified in the REC parameter of the READ statement, indicating that record is to be read (line 20). The following line is then printed (line 30):

```
64394          PHILLIPS SCREW DRIVER                    15
```

Relative Access in Update Mode

For relative/update access, a record is read by specifying the record number using the REC parameter in a READ statement and then, if desired, all or part of the record can rewritten (with the REWRITE statement). Existing records can be changed and new records can be added using the WRITE statement. Relative/update access is selected in the OPEN statement when RELATIVE is used without either INPUT or OUTPUT as a file attribute.

The program below illustrates this type of access with the original sample data file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING RELATIVE/UPDATE ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",RELATIVE
20 R=1
25 READ #1,USING 30,REC=R: PART$  EOF 55
30 FORM C 5
35 IF PART$<"50000" THEN 50
40 REWRITE #1,USING 45: "D"
45 FORM POS 40,C 1
50 R=R+1:GOTO 25
55 CLOSE #1:
60 STOP
```

File INVMAS.DAT is opened for relative/update access in line 15, then records are read in the order specified by the value of variable R in line 25. R is initially 1 and then is incremented by 1 to access all records in the file (line 50). Records with part numbers less than or equal to 50000 are rewritten with the status field changed to D (lines 35-45). The end-of-file error address in the READ statement ends execution after the last record is

read. The sample data file would then appear as follows:

```
                1         2         3         4
Rec#  ....|....0....|....0....|....0....|....0

 1    81201SOCKET WRENCH                    10D
 2    10202BALL PEEN HAMMER                  5
 3    64394PHILLIPS SCREW DRIVER            15D
 4    40121HAND SAW                          8
 5    (end-of-file record)
```

Records 1 and 3 are marked with a D. No other file changes are made.

Keyed Access

A "key" is part of a record--a word, a number, or some other label in character string form--used to identify that record for later retrieval through keyed access. Keyed access is specified by using KEYED as a file attribute in the OPEN statement. The position of the key within the record and its length are specified at the time the keyed file is created. Each time a new record is added to a keyed data file, key information is extracted and added with the corresponding record number to a separate KEY file. The KEY file consists keys and pointers organized into two distinct lists. The first list contains keys that have been sorted into alphanumeric order using the KEYSORT utility. The second list contains keys added since the last use of the KEYSORT utility in the order which they were added to the file.

The choice of the key field within the data record is application dependent. If the sample data file above is to be keyed, it is reasonable to use the part number as the key. In this case, the OPEN statement creating the file would include (in the file ID) a key position (KEYP) of 1 and a key length (KEYL) of 5. For example, the sample data file above could be created as a keyed file using the program below:

```
 1 |
 2 | A PROGRAM ILLUSTRATING KEYED-RANDOM/OUTPUT ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT,NEW,RANDOM,KEYP=1,KEYL=5,RECL=40",KEYED,OUTPUT
20 STAT$=" "
25 READ N | THE NUMBER OF DATA SETS
30 FOR I=1 TO N
35 READ PART$,DESC$,QUAN
40 WRITE #1,USING 45: PART$,DESC$,QUAN,STAT$
45 FORM C 5,C 30,N 4,C 1
50 NEXT I
55 CLOSE #1:
60 STOP
65 DATA 4
70 | DATA SETS
75 DATA 81201,SOCKET WRENCH,10
80 DATA 10202,BALL PEEN HAMMER,5
85 DATA 64394,PHILLIPS SCREW DRIVER,15
90 DATA 40121,HAND SAW,8
```

The example is identical to the program presented earlier to create a sequential file with the exception of the OPEN statement (line 15).

The sample data file (INVMAS.DAT) and its companion KEY file (INVMAS.KEY)-- after keysorting-- appear as given below:

```
                1         2         3         4
Rec#  ....|....0....|....0....|....0....|....0
```

Rec#		Key	Record Pointer
1	81201SOCKET WRENCH 10	10202	2 (sorted)
2	10202BALL PEEN HAMMER 5	40121	4
3	64394PHILLIPS SCREW DRIVER 15	64394	3
4	40121HAND SAW 8	81201	1
5	(end-of-file record)		
			(unsorted)

Note that the data file is in the order it was written; only the key entries have been sorted.

With keyed files, two basic access methods are available: random by key and sequential by key.

The first, random by key--activated by specifying the RANDOM parameter in the file ID--lets you add a new record to the data file and key to the KEY file or search for a existing data record using the KEY file. With this access method, new keys are added to the unsorted list in the KEY file. It should be noted that random key searches check both the sorted and unsorted lists in the KEY file, thus making the newly added key immediately available.

The second, sequential by key--activated by not specifying RANDOM in the file ID--lets you process records from the data file in the order of the keys in the sorted KEY file list. The keys in the unsorted list and their corresponding data records are not accessed. New records cannot be added to the file using keyed-sequential access.

These two access methods in combination with various modes are discussed below.

Random by Key Access in Output Mode

For keyed-random/output access, new records are added to the data file using the WRITE statement with new keys and record pointers automatically added to the unsorted list in the KEY file. File reads are not allowed. Keyed-random/output access is selected in the OPEN statement when RANDOM is used in the file ID and both KEYED and OUTPUT are used as file attributes.

The program below adds two records to the original sample data file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING KEYED-RANDOM/OUTPUT ACCESS
 3 |
10 DIM DESC$*10
15 OPEN #1: "NAME=INVMAS.DAT,RANDOM",KEYED,OUTPUT
20 STAT$=" "
25 READ N | THE NUMBER OF DATA SETS TO BE ADDED
30 FOR I=1 TO N
35 READ PART$,DESC$,QUAN
40 WRITE #1,USING 45: PART$,DESC$,QUAN,STAT$
45 FORM C 5,C 30,N 4,C 1
50 NEXT I
```

```
55 CLOSE #1:
60 STOP
65 | DATA SETS
70 DATA 2
75 DATA 91324,WRENCH,5
80 DATA 68923,PLIERS,12
```

In line 15, the file INVMAS.DAT is opened for keyed-random/output access. The number of data sets to be added is then read from internal data (line 25). The new records are added to INVMAS.DAT, and the part number keys and record pointers are automatically added to the unsorted list in INVMAS.KEY (lines 30-50). Following execution of the program, the sample data file and corresponding KEY file look like this:

```
                 1         2         3         4      |    Key   Record
Rec#  ....|....0....|....0....|....0....|....0       |          Pointer
                                                      |
  1    81201SOCKET WRENCH                    10       |   10202     2 (sorted)
  2    10202BALL PEEN HAMMER                  5       |   40121     4
  3    64394PHILLIPS SCREW DRIVER            15       |   64394     3
  4    40121HAND SAW                          8       |   81201     1
  5    91324WRENCH                            5       |
  6    68923PLIERS                           12       |   91324     5 (unsorted)
  7    (end-of-file record)                           |   68923     6
```

Note that the KEY file now has two distinct key lists, the sorted list and the unsorted list. As has been mentioned before, random accesses by key will check both lists for the presence of a certain key. Sequential accesses by key will not access the keys in the unsorted list. To ensure access to all data, use the KEYSORT utility to sort the unsorted list and merge it into the sorted list. After keysorting, the sample data file and KEY file would look like this:

```
                 1         2         3         4      |    Key   Record
Rec#  ....|....0....|....0....|....0....|....0       |          Pointer
                                                      |
  1    81201SOCKET WRENCH                    10       |   10202     2 (sorted)
  2    10202BALL PEEN HAMMER                  5       |   40121     4
  3    64394PHILLIPS SCREW DRIVER            15       |   64394     3
  4    40121HAND SAW                          8       |   68923     6
  5    91324WRENCH                            5       |   81201     1
  6    68923PLIERS                           12       |   91324     5
  7    (end-of-file record)                           |
                                                      |             (unsorted)
```

Note that the two originally unsorted keys now occupy their proper place in the sorted list.

Random by Key Access in Input Mode

For keyed-random/input access, a record in the data file is read by giving the desired key in the KEY parameter of the READ statement. File writes are not allowed. Keyed-random/input access is selected in the OPEN statement when RANDOM is used in the file ID and both KEYED and INPUT are used as file attributes.

The program below accesses and prints record 6 (the PLIERS) by specifying the corresponding part number as key.

```
 1 |
 2 | A PROGRAM ILLUSTRATING KEYED-RANDOM/INPUT ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT,RANDOM",KEYED,INPUT
20 READ #1,USING 25,KEY="68923": PART$,DESC$,QUAN,STAT$  NOKEY 50
25 FORM C 5,C 30,N 4,C 1
30 PRINT PART$,DESC$,QUAN,STAT$
35 CLOSE #1:
40 STOP
50 PRINT "KEY NOT FOUND"
55 CLOSE #1:
60 STOP
```

The file INVMAS.DAT is opened for keyed-random/input access in line 15. Lines 20-30 then read and print the record with the specified key.

```
68923          PLIERS                    12
```

If the key is not found, the program branches to line 50 for error processing.

Random by Key Access in Update Mode

For keyed-random/update access, a record in the keyed data file can be accessed by giving the desired key in the KEY parameter of the READ statement and, if desired, the REWRITE statement then used to update the record. The key portion of the data record must not be changed during updating, or an error will occur. New records can be added to the file using the WRITE statement. Keyed-random/update access is selected in the OPEN statement when RANDOM is used in the file ID and KEYED is used without either INPUT or OUTPUT as a file attribute.

In the example program below, record 6 (the PLIERS) is read and rewritten with the status changed.

```
 1 |
 2 | A PROGRAM ILLUSTRATING KEYED-RANDOM/UPDATE ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT,RANDOM",KEYED
20 READ #1,USING 25,KEY="68923": PART$,DESC$,QUAN,STAT$  NOKEY 50
25 FORM C 5,C 30,N 4,C 1
30 REWRITE #1,USING 40: "D"
35 FORM POS 40,C 1
40 CLOSE #1:
45 STOP
50 PRINT "KEY NOT FOUND"
55 CLOSED #1:
60 STOP
```

The program opens file INVMAS.DAT for keyed-random/update access (line 15), then reads the record with the specified key and rewrites with the status field changed to D (lines 20-40). After the program is executed, the sample data file would appear as given below:

```
                1         2         3         4          |    Key   Record
Rec#  ....|....0....|....0....|....0....|....0          |          Pointer
                                                         |
 1    81201SOCKET WRENCH                        10       |   10202     2 (sorted)
 2    10202BALL PEEN HAMMER                      5       |   40121     4
 3    64394PHILLIPS SCREW DRIVER                15       |   64394     3
 4    40121HAND SAW                              8       |   68923     6
 5    91324WRENCH                                5       |   81201     1
 6    68923PLIERS                               12D      |   91324     5
 7    (end-of-file record)                               |
                                                         |                (unsorted)
```

Record 6 has been marked with a D.

Sequential Access by Key in Input Mode

For keyed-sequential/input access, records are read in the order of the keys in the sorted list of the KEY file. File writes are not allowed. Processing begins with the first key in the KEY file unless the KEY>= parameter appears in the READ statement. In that case the first record processed is that of the first key alphanumerically greater than or equal to the KEY>= parameter. Keyed-sequential/input access is selected in the OPEN statement when RANDOM is not used in the file ID and both KEYED and INPUT are used as file attributes.

The program below prints a list of all parts in the original sample data file.

```
 1 |
 2 | A PROGRAM ILLUSTRATING KEYED-SEQUENTIAL/INPUT ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",KEYED,INPUT
20 READ #1,USING 25: PART$,DESC$,QUAN,STAT$  EOF 40
25 FORM C 5,C 30,N 4,C 1
30 PRINT PART$,DESC$,QUAN,STAT
35 GOTO 20
40 CLOSE #1:
50 STOP
```

Suppose the expanded sample data file has not been keysorted and appears as given below:

```
                1         2         3         4     |     Key   Record
Rec#  ....|....0....|....0....|....0....|....0      |           Pointer

 1    81201SOCKET WRENCH                    10      |     10202     2 (sorted)
 2    10202BALL PEEN HAMMER                  5      |     40121     4
 3    64394PHILLIPS SCREW DRIVER            15      |     64394     3
 4    40121HAND SAW                          8      |     81201     1
 5    91324WRENCH                            5      |
 6    68923PLIERS                           12      |     91324     5 (unsorted)
 7    (end-of-file record)                          |     68923     6
```

Executing the program would produce this printout.

```
10202          BALL PEEN HAMMER          10
40121          HAND SAW                  8
64394          PHILLIPS SCREW DRIVER     15
81201          SOCKET WRENCH             10
```

Note in particular two things: first, the data is accessed in the order of the keys, not in the order the data appears in the file. Second, the unsorted data is not accessed at all. To ensure that all data is read, the file must be keysorted using the KEYSORT utility.

Shown below is the same program, changed slightly by including the KEY>= parameter in the original READ statement and adding a second READ without KEY>=. The technique allows keyed-sequential/input access to begin with a key other than the first.

```
 1 |
 2 | A PROGRAM ILLUSTRATING KEYED-SEQUENTIAL/INPUT ACCESS WITH KEY>=
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",KEYED,INPUT
20 READ #1,USING 25,KEY>="50000": PART$,DESC$,QUAN,STAT$  EOF 45
25 FORM C 5,C 30,N 4,C 1
30 PRINT PART$,DESC$,QUAN,STAT
35 READ #1,USING 25: PART$,DESC$,QUAN,STAT$  EOF 45
40 GOTO 30
45 CLOSE #1:
55 STOP
```

After program execution, the printout would appear as shown below:

```
64394          PHILLIPS SCREW DRIVER     15
81201          SOCKET WRENCH             10
```

Processing began with the first key greater than 50000. Thus, only records with keys greater than 50000 are printed. Note that the READ with KEY>= is only used once. As in the previous case, the unsorted keys are not accessed.

Sequential Access by Key in Update Mode

For keyed-sequential/update access, records are read in the order of the keys in the sorted list of the KEY file and, if desired, the REWRITE statement then

used to update a record. (The WRITE statement cannot be used, which means that new records cannot be added to the file.) Processing begins with the first key in the KEY file unless the KEY>= parameter appears in the READ statement. In that case, the first record processed is that of the first key alphanumerically greater than or equal to the KEY>= parameter. Keyed-sequential/update access is selected in the OPEN statement when RANDOM is not used in the file ID and KEYED is used without either INPUT or OUTPUT as a file attribute.

The program below illustrates this access technique with the expanded sample data file (after being keysorted).

```
 1 |
 2 | A PROGRAM ILLUSTRATING KEYED-SEQUENTIAL/UPDATE ACCESS
 3 |
10 DIM DESC$*30
15 OPEN #1: "NAME=INVMAS.DAT",KEYED
20 READ #1,USING 25: PART$,DESC$,QUAN,STAT$  EOF 50
25 FORM C 5,C 30,N 4,C 1
30 IF QUAN>10 THEN 20
35 REWRITE #1,USING 40: "I"
40 FORM POS 40,C 1
45 GOTO 20
50 CLOSE #1:
55 STOP
```

The program opens the file INVMAS.DAT for sequential access by key, in update mode (line 15). The records are then processed in key order, with those records having a quantity less than or equal to 10 changed to the I status (lines 20-45). After the program is executed, the sample data file appears as shown below:

```
                1         2         3         4     |     Key   Record
Rec#  ....|....0....|....0....|....0....|....0      |           Pointer
                                                    |
 1    81201SOCKET WRENCH                    10      |     10202     2 (sorted)
 2    10202BALL PEEN HAMMER                  5I     |     40121     4
 3    64394PHILLIPS SCREW DRIVER            15      |     64394     3
 4    40121HAND SAW                          8I     |     68923     6
 5    91324WRENCH                            5I     |     81201     1
 6    68923PLIERS                           12      |     91324     5
 7    (end-of-file record)                          |
                                                    |                (unsorted)
```

Records with quantities less than 10 reflect an I status.

This completes the description of the various file processing techniques available to the DeltaBASIC user. A word of caution is in order. While the data file is identical for all file access techniques, a problem can arise if you add records to a keyed file not opened as a keyed file. In this case, the record is added to the data file but the key and pointer are not added to the KEY file. The resulting discrepancy causes this record data to become inaccessable later when the file is used again as a keyed file. To correct this situation, you can use the ORGANIZE utility to rebuild the KEY file.

Appendix D. FORMATTED DATA ENTRY SCREENS

DeltaBASIC uses formatted data entry screens created by the utility program SDU. To access a formatted data entry screen, first a work station (WS) file must be opened specifying the format file that contains the screen; then the screen can be displayed and data entered using WRITE and READ statements.

Data is passed between the keyboard/screen and DeltaBASIC using an input buffer and an output buffer. All screen fields that accept input are associated with contiguous data fields in the input buffer. Similarly, all non-constant output fields are associated with contiguous data fields in the output buffer. The arrangement of the buffer data fields is in the order left-to-right, top-to-bottom on the screen. Thus, the screen field nearest the top left corner of the screen would appear as the first data field in the appropriate buffer. Since output-only fields do not appear in the input buffer, screen fields do not necessarily occupy the same position in both buffers. SDU provides a detailed listing of the field positions in both buffers which can be used to develop the FORM specification needed with the READ or WRITE statements.

DeltaBASIC supports a number of data entry and visual attributes. Attributes for a field can be selected using the attribute screen of SDU. Most attributes can be set on by choosing the Y option or off by choosing the N option. In some cases, other options are provided. For instance, some attributes can be conditionally set from DeltaBASIC using indicators. There are 99 indicators, each of which may be set on or off using the INDIC option in the WRITE or REWRITE statments. An attribute is set on only if its associated indicator is on.

The available attributes are listed below. The indicator option is denoted by xx.

(Data Entry Attributes)

Controlled Field Exit {Y,N} — When set on forces the user to press a key such as RETURN or FIELD EXIT to exit an input/output field. When set off the field is exited when the field length is exceeded by the operator.

Adjust/Fill {N,B,Z} — When set off no special action is taken. When set to B, the data in the field is right justified and filled with blanks to the left upon exit. When set to Z, the data in the field is right justified and filled with zeros to the left upon exit.

Mandatory Entry {Y,N} — When set on forces the operator to enter at least one non-blank character before exiting the field.

Mandatory Fill {Y,N} — When set on forces the operator to enter non-blank characters throughout the field.

Field Type {N,A} — When set to N, any data entered into the field must be numeric; i.e., 0 through 9, +, -, E, or the decimal point. When set to A, all alphanumeric characters are allowed into the field.

Input Allowed {Y,N} — When set on allows input from the field. When set off

input is NOT allowed--setting this attribute off creates an OUTPUT ONLY field.

Constant Output {Y,N} -- When set on causes a specified constant to be output to the screen as the screen is initially written.

Auto RETURN {Y,N} -- When set on indicates that the entire screen is to be "entered" when the field is exited; i.e., the contents of all the input fields are returned to the program. When set off the cursor will procede to the next field unless the RETURN or a function key is pressed.

Position Cursor {Y,N,xx} -- When set on, unconditionally or with a specified indicator, causes the cursor to be positioned to this field when the screen is initially written. When more than one field on the screen has this attribute set on, the cursor will move to the first such field. If no field is found with the position indicator set on, the cursor defaults to the first data entry field.

Protect {Y,N,xx} -- When set on, unconditionally or with a specified indicator, causes input to be disallowed from the field. When set off input is allowed from the field.

(Visual Attributes)

Low Intensity {Y,N,xx} -- When set on, unconditionally or with a specified indicator, causes the data in the field to be displayed in low intensity (dim).

Blink {Y,N,xx} -- When set on, unconditionally or with a specified indicator, causes the data in the field to be displayed blinking.

Non-Display {Y,N,xx} -- When set on, unconditionally or with a specified indicator, causes the data in the field to NOT be displayed.

Reverse Image {Y,N,xx} -- When set on, unconditionally or with a specified indicator, causes the data in the field to be displayed in reverse video.

Underline {Y,N,xx} -- When set on, unconditionally or with a specified indicator, causes the data in the field to be displayed underlined.

Output {Y,xx} -- When set on, unconditionally or with a specified indicator, allows output to the field. This attribute is the equivalent of the Protect Attribute for output. Note that this may not be set unconditionally off.

NOTE: The implementation of visual attributes can vary depending on the capabilites of the computer system and/or terminal on which they are being used.

Appendix E. MULTI-USER INFORMATION

DeltaBasic supports multi-user file and record locking in both the MP/M and TurboDOS operating environments. If a file is to be shared by several programs at the same time, then the SHR parameter must appear in the OPEN statement of each program. To gain exclusive use of a file, it is necessary to OPEN it without the SHR parameter at a time when the file is not in use by any other program. Once exclusive use of the file is obtained, no other program can gain access to the file.

Record locking occurs whenever a program has accessed a record and the file is open in either the output or update mode. The program retains exclusive use of the record during the write operation (output mode) or between the read and rewrite operations (update mode). In standard implementations of MP/M and TurboDOS, a second program attempting to gain access to a locked record is suspended until the other program releases the record. It is important to note that all physical disk records that contain the data record accessed are locked. The possibility therefore exists that more than one data record can be locked when only one record is accessed. Record locking does not occur when a program accesses the record in the input mode. Thus, two or more programs can access the same record at the same time if all have the file open in the input mode.

Appendix F. LANGUAGE COMPATIBILITY

DeltaBASIC is based on a subset of IBM System/34 BASIC but is not directly compatible with this or any other BASIC language implementation. At the same time, most of the commands, statements, and functions are similar to those generally considered traditional to the BASIC language. The greatest differences will be encountered in these areas:

1) Formatted READ and WRITE
2) Disk I/O and file handling
3) Substring operations

Appendix G. INDEX