VICTOR

Applications Programmer's Tool Kit II

Volume I

Applications Programmer's Tool Kit II

Volume I

Second VICTOR printing April, 1984.

ISBN 0-88182-115-2 Printed in U.S.A.

Errata Sheet for the Applications Programmer's Tool Kit II

This page contains corrections and additions to the *Applications Programmer's Tool Kit II* manual.

All PLINK and PLIB files have been updated.

Keyboard files have been added and existing files have been updated.

FABS and AUTOSORT have been removed. Any references to these programs in the manual should be ignored.

Building an operating system with SYSELECT produces a VICTOR MS-DOS 2.11, BIOS 2.93 operating system. This operating system will provide all the functions contained in earlier versions of MS-DOS 2.11 plus these enhancements:

- You can now reboot your computer from the keyboard. To do so, press and hold down the Alt key, the Caps Lock key, and the Decimal key on the numeric keypad. You can press these three keys in any order, but if you release one of the keys or press any other key before the three-key sequence is complete, the reboot command is canceled.
- A reboot interrupt has been added at interrupt 69h. You can reboot the computer from within an application by issuing an interrupt 69h. No parameters are required for this interrupt.
- A hot key interrupt has been added at interrupt 68h. When the keyboard scan codes are available from the hardware they are passed in AX to interrupt 68h; interrupt 68h is called before any processing of the scan code occurs. An application that is monitoring this interrupt can do one of the following:
  - —Return without changing any of the registers.
  - —Tell the BIOS to ignore the current key by returning zero in AX.
  - —Map the current key to another key by returning a different scan code in AX.

Possibly for a single volume Application Programmer's Toolkit.
If so, this page probably is superseded by the two volume Application Programmer's Tool Kit II vols I and II April 1984.

CONTENTS

1. FABS/86M

2. AUTOSORT/86M

3. EFONT

4. KEYGEN

5. SYSGEN

6. MODCON

OVERVIEW

The *Applications Programmer's Tool Kit II—Volume I* is conceptually divided into two Sections: Data Base Support, and System Configuration.

Data Base Support consists of:

- ▶ Fabs/86M — A Fast Access B-tree System used to organize data for minimum retrieval time; designed to be called from high-level languages.
- ▶ AUTOSORT/86M — A comprehensive sort utility that can be used stand-alone or called from application programs.

System Configuration consists of:

- ▶ EFONT — A font editor used to define or modify the characteristics of individual keys on the keyboard.
- ▶ KEYGEN — A keyboard generator used to define the characteristics of individual keys on the keyboard.
- ▶ SYSGEN — A system generation program that lets you generate a custom MS-DOS operating system.

MODCON — A console modification utility that allows you to set and save keyboard tables and character sets.

FABS/86M

VICTOR Publications
380 El Pueblo Road
Scotts Valley, California 95066
(408) 438-6680

First VICTOR printing January, 1983.
Second VICTOR printing November, 1983.
ISBN 0-88182-107-1 Printed in U.S.A.

CONTENTS

1. An Overview of FABS/86
   - 1.1 General Information 1-1
   - 1.2 Data Storage and Retrieval 1-1
     - 1.2.1 The FABS/86 Difference 1-3
     - 1.2.2 Using FABS/86 to Retrieve a Record 1-4
     - 1.2.3 Keys 1-5

2. FABS/86 Commands
   - 2.1 Search-Next-After 2-2
   - 2.2 Build 2-3
   - 2.3 Close 2-4
   - 2.4 Create 2-4
   - 2.5 Delete 2-5
   - 2.6 Get Maximum Key Length 2-6
   - 2.7 Get Next Record Number 2-7
   - 2.8 Get Number of Deletes 2-8
   - 2.9 Get Number of Primary Keys 2-8
   - 2.10 Get Number of Records 2-9
   - 2.11 Insert 2-10
   - 2.12 Open 2-11
   - 2.13 Replace 2-11
   - 2.14 Search 2-12
   - 2.15 Search First 2-13
   - 2.16 Search Generic 2-14
   - 2.17 Search Last 2-15
   - 2.18 Search Next 2-16
   - 2.19 Search Previous 2-17
   - 2.20 Write Page Map 2-17
   - 2.21 Obsolete Node Buffers 2-18

3. Using FABS/86 with Programming Languages
3.1 Interfacing the MS-BASIC Interpreter to FABS/86 3-1
3.1.1 Loading the FABS86M.COM Module 3-1
3.1.2 Linking FABS/86 to Compiled Languages 3-2
3.1.3 Calling FABS/86 from MS-BASIC 3-2
3.1.4 Test Programs 3-3
3.2 FABS/86 and the MS-BASIC Compiler 3-5
3.2.1 Calling FABS/86 from the MS-BASIC Compiler .. 3-5
3.2.2 Test Programs for the MS-BASIC Compiler 3-6
3.3 Using FABS/86 with MS-Pascal 3-8
3.4 FABS/86 and MS-FORTRAN 3-9
3.5 FABS/86 and MS-COBOL 3-10

4. FABS/86 Error and Warning Codes 4-1

Appendix A: FABS/86 PUBLIC Interfaces and Absolute Offset Entry Points A-1

TABLES

1-1: Maximum Keys 1-7
4-1: FABS/86 Error and Warning Codes 4-1
A-1: FABS/86 PUBLIC Entry and Access A-1
A-2: FABS/86 Absolute Offsets A-5

CHAPTERS

1. An Overview of FABS/86 1

2. FABS/86 Commands 2

3. Using FABS/86 with Programming Languages 3

4. FABS/86 Error and Warning Codes 4

Appendix A: FABS/86 PUBLIC Interfaces and Absolute... A
Offset Entry Points

1

AN OVERVIEW OF FABS/86

GENERAL INFORMATION 1.1

When you compile data with a computer, a major problem is how to retrieve it. How can you locate the information you want in the least amount of time using the simplest search procedure? The FABS/86 (Fast Access Btree Structure) program can help you solve this problem.

FABS/86 is an assembly language program module that uses key files for fast data retrieval with large data files. FABS/86 assigns input data to key files, and arranges those files in balanced trees (Btrees) to speed data retrieval. When you need to access a record in your data file, FABS/86 searches down the tree to locate that record. If necessary, FABS/86 rearranges the key files on that tree for easier access to your data base.

You should have at least a basic knowledge of programming in a high-level language (such as Pascal or BASIC) before you start using FABS/86. You should also have a rudimentary knowledge of how files are maintained in a computer system. If you are a beginning programmer, you should find FABS/86 easy to understand once you have mastered the data file concepts in your programming language.

Note: FABS/86M may be incorporated into any application program distributed on and for the VICTOR/SIRIUS machine without payment of royalty.

DATA STORAGE AND RETRIEVAL 1.2

Whether you are using a computer or a file cabinet, the first step in the process of storing data is about the same. You give each piece of data a key name (an account number, subject, and so on) that distinguishes it from other pieces of data in the same filing system. The difference

between computers and file cabinets starts when you actually put the new piece of data into the system. With a file cabinet, you shift all of the other records in the cabinet to make room for the new data. When you're storing data on a diskette or hard disk, however, it takes too long for the computer to move all the other records. This problem is generally overcome with one of the following techniques:

1. The new data is kept separate from the existing records until the insertion process is complete. After you've entered all of the new data, it is sorted into the existing data file.
2. An additional file (a key file) is created. This key file contains a field (key) into which the data is sequenced. It also contains an associated record pointer (the record number) of the data record that contains the key. Using this system, only the smaller key file has to be shuffled around each time you enter new data. Any data you enter is easily retrievable: Once the key is found, the data record pointer is used to access the data you want.

Both of these techniques have their problems, however. If you're using the first method, for example, your data is normally retrieved by using a "binary search." A binary search divides the data file in half, and determines whether the desired field (key) is in the upper or lower half of the file. Then, the search determines if the key is in the upper or lower half of the half chosen in the first step, and so on, until the key is found or the file is exhausted. With large data files, a binary search can take a long time.

A binary search also causes problems during the insertion process, because you need to make sure that none of the inserted records are already in your file. In this case, all unsorted data inserts are searched to ensure that they do not already exist in your file.

Another method maintains keys in a sorted key file and uses an overflow file for inserted keys not yet sorted into the key file. The binary search technique is used to find the key and the associated data record pointer. If the key is not found in the sorted key file, the overflow file is searched. As the overflow file gets bigger, retrieval gets slower. Periodically, you must re-sort the key file to incorporate the keys in the overflow file.

FABS/86 also maintains keys in a sequential key file. Instead of storing data in a linear manner, however, the FABS/86 key file is a multi-path balanced tree. This design makes FABS/86 well suited for the maintenance and manipulation of very large data files (the most difficult).

With FABS/86, the data file and key file space is dynamically allocated—that is, the files grow as needed. Although FABS/86 does not actually read or write to your data file, it does provide you with the record number for all of your data file reads and writes.

Suppose you have a data file that you want to access by the name field. To insert a new record (with a given name) into the file, you must call FABS/86 with the Insert command and the key. In about two seconds, FABS/86 returns the number of the data record where you must write the data associated with that key. After another quarter-second or so, your key file and data file are in perfect order. (These times assume that you are using a floppy disk system. A hard disk is faster.)

If you have enough room on your diskette or hard disk (and you are not limited by your programming language), you can insert 50 thousand keys or more without much effect on the access time. Typically, this time is one second or less to search for the key and read the data record. Repeated accesses normally take about a quarter of a second each.

You should note that there are no overflow files associated with FABS/86. The estimated insertion times include reorganizing the key file (if necessary). Any portions of the key file that are changed during insertion are saved on diskette (or hard disk) so that the key file is always current. Sorting is not required at any time on either the data file or the key file.

1.2.2 USING FABS/86 TO RETRIEVE A RECORD

Since FABS/86 knows which data record has been attached to what key, finding a record is easy—just tell FABS/86 to search for a particular key. FABS/86 gives you the random data record number that you must read to get the desired record. Each time you delete a record, FABS/86 retains the record numbers for automatic re-use when you insert more records.

FABS/86 also lets you look for groups of records, rather than just a specific record. This is called a "generic" search. For example, you could do a generic search for every record with a name field that starts with the letter "M". After you give the proper instructions, FABS/86 finds the record number of the first occurrence of a name field starting with "M". A Search Next command (see Chapter 2.18) continues the search sequentially through the name field.

A generic search is also useful in accessing data that is sequenced in several levels. For example, the data file you're searching might be in alphabetical order by state, then by city under each state, and by zip code under each city. First, you concatenate the state, city, and zip code fields to form the insert key. Then you use the Search Generic command along with the Search Next command to access only those records associated with a particular state, or a particular city and state.

With FABS/86, you can retrieve data sequenced on more than one key by using a single key file. When the key file is created, you specify the number of primary keys that you want for the data file. When inserting or deleting, you must specify all the key values for the data record. When searching your file, you specify which primary key number you want.

FABS/86 lets you assign the same value to more than one key ("duplicate" keys). When you want to access a record assigned to a duplicate key, FABS/86 supplies the number of the first occurrence of that key in the index file. Use the Search Next command to find the next occurrence of the key. To access a series of duplicate keys, test the key of each data record you read against the original key to see if you are still in the block of duplicates. This procedure is also used when you need to read through a block to determine which keys are to be deleted.

Multiple Keys

FABS/86 also supports multiple primary keys—that is, an area of the data file can be accessed in ascending or descending order by more than one key. The number of primary keys you can use is limited only by the length of the Insert and Delete command strings (255 bytes). Of course, the more keys you use, the longer it takes to insert or delete them. The search time should not be seriously affected, however.

When you create an index file, you must specify the number of primary keys. When inserting and deleting keys, all of the primary keys must be specified (except for the Replace command).

FABS/86 also lets you use duplicate multiple primary keys. You should be aware, though, that using this type of key presents some problems. If you want to delete a duplicate multiple key, you must decide which of the records is to be deleted. Then, you have to extract all of the primary keys from that record and use them to form the Delete command.

ASCII Keys

Keys are normally maintained in the key file as ASCII characters. (This mode is specified by entering an "A" for the KEYTYPE when you create the key file.) Any key having less than the maximum length is padded with zeros to bring it up to maximum length. When you specify more than one primary key, the maximum length applies to all keys. Each key occupies the maximum space.

Integer Keys

When creating a key file, you can specify the KEYTYPE as I (for integer). With an integer file, FABS/86 converts the keys to a 2-byte integer numeric value. Some limitations must be observed when you use integer keys: Any keys specified in the command string must be ASCII strings with a range of 0 to 65535. They cannot contain nulls, signs, or other characters. The maximum key length will be forced to 2 bytes. When a key file is specified as integer, all the primary keys must be integer. Also, generic searches are not permitted with integer keys.

Maximum Number of Keys

With a balanced tree, it is impossible to predict the maximum number of keys you can use in a data file. This number depends on the length of the key, and on the sequence in which your keys are inserted. However, you can establish a "worst case" and a "best case" for each length; the average tells you approximately how many keys of a particular length you can probably use.

FABS/86 has a constant node length of 512 bytes and can have up to 5 levels. The root node can have anywhere from one key to the maximum number of keys. All other nodes will be between half full and full depending upon the insertion sequence.

The following table gives you an idea of how many keys can be used with various key lengths. (N is the minimum number of keys per node—512 bytes.)

Table 1-1: Maximum Keys

KEY LENGTH	N	WORST CASE	BEST CASE	AVERAGE
2	36	107,136	214,272	160,704
4	28	83,328	166,656	124,992
6	23	68,448	136,896	102,672
8	19	56,544	113,088	84,816
10	16	47,616	95,232	71,424
14	13	38,688	77,376	58,032
20	10	29,760	59,520	44,640
30	7	20,832	41,664	31,248
40	5	14,250	29,760	22,005
50	4	5,620	23,808	14,714

2

FABS/86 COMMANDS

The following commands can be executed with FABS/86:

A	Search-next-after
B	Build key file
K	Close key file
C	Create key file
D	Delete key(s)
M	Get maximum key length
Q	Get next record number
U	Get number of open deletes
T	Get number of records
I	Insert key
O	Open key file
R	Replace key
S	Search for key
F	Search for first key
G	Search for generic key
L	Search for last key
N	Search for next key
P	Search for previous key
W	Write page map
Z	Obsolete node buffers

This chapter discusses the functions of FABS/86 command strings. Each string is described in the following manner:

- First, you are given the format to use when you enter the command. Each element of the string is explained.
- Next, you are told the operation the command performs and when the string should be used. When appropriate, you are referred to other command strings that can be used with the one being explained.

- Last, any parameters returned by the string are explained. This tells you the meaning of each parameter that FABS/86 returns when you use the command string.
- FABS/86 allows you to change the delimiter in the command line by putting the desired delimiter as the first character in the command line. Any character with an ASCII code of 2F Hex or less will work, periods excluded.

2.1 SEARCH-NEXT-AFTER (A)

The command string is:

```
CMND$ = "A\PKN\RN\FN\" + KEY$
```

where:

PKN is the primary key number.

RN is the record number.

FN is the file number.

KEY$ is the key to search after.

This command returns the next key in the key file with a value greater than KEY$. In the case of duplicates, the record number of the next duplicate with the next highest record number will be returned. This command can be used after the key is deleted to place the pointer after the key deleted, so a command like "NEXT" or "PREVIOUS" will start at the correct place.

The command string is:

CMND$** = "**B\FN**" + **PK1$ + "\" + ... + "\" + **PKn$**

where:

B is the command.

FN is the file number (1 to 6).

PKn$ is the value of the nth primary key.

This command is identical to the Insert (I) command, except that Build does not write the map file to your diskette. All other FABS/86 commands update the diskette before returning to the calling program.

Because the Build command does not write to the map file (as the Insert command does), you save time if you use Build instead of Insert when building key files for large data files.

The Write Map File command must be executed after a series of Build commands to ensure that the correct map data has been entered on diskette. No other FABS/86 command should be executed during this procedure.

When in doubt use the Insert command.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The date record number

ADRKEY = No significance

2.3 CLOSE (K)

The command string is:

```
CMND$ = "K\FN"
```

where:

K is the command.

FN is the file number (1 to 6).

The Close command closes a key file when you reach the end of the host language program. (This is the only time that you need to close a key file.) The index file on your diskette is updated after each FABS/86 operation (unless you used the Build command).

Parameters returned:

ERRF% = Error/Warning code

RECNO = No significance

ADRKEY = No significance

2.4 CREATE (C)

The command string is:

```
CMND$ = "C\[d:]filename[.ext]\MAXKL\NPK\KT\FN"
```

where:

C is the command.

MAXKL is the maximum key length (100 bytes max). The usual key length is 8 to 10 bytes.

NPK is the number of primary keys for this file.

KT is the key type (I = Integer, A = ASCII).

FN is the file number (1 to 6).

Use Create to create a key file (filename.ext) and a map file (filename.MAP) with the attributes specified in the command string. The key file is opened for access under the file number contained in the command string. If a file with the same name already exists, that file is deleted.

Parameters returned:

ERRF% = Error/Warning code

RECNO = No significance

ADRKEY = No significance

DELETE (D) 2.5

The command string is:

CMND$** = "**D\RN\SBDFL\FN**" + **PK1$ + "\" + ... + "\" + **PKn$**

where:

D is the command.

RN is the record number of the data record containing the keys.

SBDFL is the prompt "Search Before Delete Flag (Y/N)". If you answer yes, all primary keys are searched to ensure their presence before any are deleted. This process protects your key file against faulty programs.

FN is the file number (1 to 6).

PKn$ is the value of the nth primary key.

Use Delete to delete the specified keys from the key file and return the associated data record number in the data file. You should put "deleted" into some field of the data record if the key file is destroyed. Doing so lets you rebuild the key file, excluding the deleted data records.

FABS/86 maintains pointers to all deleted records and reclaims these records on future inserts on a last-in, first-out basis.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The deleted data record number

ADRKEY = No significance

2.6 GET MAXIMUM KEY LENGTH (M)

The command string is:

CMND$ = "M\FN"

where:

M is the command.

FN is the file number (1 to 6).

This command causes FABS/86 to return the maximum key length permitted in the key file. (The maximum key length was specified by a Create command.) Any attempt to insert a longer key causes a syntax error.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The maximum key length

ADRKEY = No significance

GET NEXT RECORD NUMBER (Q) 2.7

The command string is:

```
CMND$ = "Q\FN"
```

where:

Q is the command.

FN is the file number (1 to 6).

When you use the Get Next Record Number command, FABS/86 returns the record number to be assigned the next time you use the Insert command. (This assumes that no Delete command is used prior to the Insert command.) If there are no unreclaimed deleted data records, this command returns the next available data record in the file. If there are unreclaimed deletes, the record number of the last delete is returned.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The next data record number

ADRKEY = No significance

2.8 GET NUMBER OF DELETES (U)

The command string is:

```
CMND$ = "U\FN"
```

where:

2

U is the command.

FN is the file number (1 to 6).

This command returns the number of unreclaimed deleted data records. This number tells you how many records can be inserted before your data file expands.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The number of deleted records

ADRKEY = No significance

2.9 GET NUMBER OF PRIMARY KEYS (H)

The command string is:

```
CMND$ = "H\FN"
```

where:

H is the command.

FN is the file number (1 to 6).

This command returns the number of primary keys in the key file. (This number is specified in the Create command.)

Parameters returned:

ERRF% = Error/Warning code

RECNO = The number of primary keys

ADRKEY = No significance

GET NUMBER OF RECORDS (T) 2.10

The command string is:

CMND$ = "T\FN"

where:

T is the command.

FN is the file number (1 to 6).

Get Number of Records returns the total number of records in your data file, including the unreclaimed deleted records. To determine the number of active data records, subtract the value returned by Get Number of Deletes (U).

Parameters returned:

ERRF% = Error/Warning code

RECNO = The number of records

ADRKEY = No significance

2.11 INSERT (I)

The command string is:

CMND$** = "**I\FN**\" + **PK1$ + "\" + ... + "\" + **PKn$**

where:

2

I is the command.

FN is the file number (1 to 6).

PKn$ is the nth primary key value.

Use this command to insert keys into the key file. The number of primary keys included in the command must equal the number you specified in the Create command. Duplicate keys are permitted. Variable length keys are also permitted, but are padded with zeros (not spaces) to bring them up to the maximum key length. If you want the keys padded with spaces, you must enter the spaces yourself. Each key in your key file has the maximum length specified in the Create command.

When control is returned to the calling program, you should write the entire data record to the data file at the record number specified by RECNO.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The data record number

ADRKEY = No significance

OPEN (O) 2.12

The command string is:

```
CMND$ = "O\[d:]filename[.ext]\FN"
```

where:

O is the command.

FN is the file number (1 to 6).

Use this command to open an existing key and map file for access. You can open up to six key files at a time.

Parameters returned:

ERRF% = Error/Warning code

RECNO = No significance

ADRKEY = No significance

REPLACE (R) 2.13

The command string is:

```
CMND$ = "R\PKN\RN\FN\" + OLDKEY$ + "\" + NEWKEY$
```

where:

R is the command.

PKN is the primary key number.

RN is the record number of OLDKEY$.

FN is the file number (1 to 6).

OLDKEY$ is the value of the key to be replaced.

NEWKEY$ is the new key value.

Use this command to replace a single key with another key having the same record number and primary key number. The returned record number is the same as the specified record number.

Parameters returned:

ERRF% = Error/Warning code

RECNO = Same as specified record number

ADRKEY = No significance

2.14 SEARCH (S)

The command string is:

```
CMND$ = "S\PKN\FN\" + KEY$
```

where:

S is the command.

PKN is the primary key number.

FN is the file number (1 to 6).

KEY$ is the value of the key.

This command returns the record number of the specified key string (KEY$). The record number associated with the first duplicate is returned if there are duplicate keys with the value KEY$. You can use the Search Next (N) command to access the others. (You should test each time to see if the key value is equal to KEY$.)

Parameters returned:

ERRF% = Error/Warning code.

- If ERRF% = 0, KEY$ was found.
- If ERRF% = 12, KEY$ was not found and the value of KEY$ is between the first key and the last key.
- If ERRF% = 13, KEY$ was not found and the value of KEY$ is less than the value of all existing keys. The record number of the first key is returned.
- If ERRF% = 15, KEY$ was not found and the value of KEY$ is greater than the value of all existing keys. The record number of the last key is returned.
- If ERRF% = 16, there are no keys in the key file.

RECNO = The appropriate record number

ADRKEY = The FABS/86 memory address where the key can be found

SEARCH FIRST (F) 2.15

The command string is:

```
CMND$ = "F\PKN\FN"
```

where:

F is the command.

PKN is the primary key number.

FN is the file number (1 to 6).

Search First returns the number of the data record containing the smallest key value for the specified primary key.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The record number

ADRKEY = The FABS/86 memory address where the key can be found

2.16 SEARCH GENERIC (G)

The command string is:

CMND$** = "**G\PKN\FN**" + **KEY$

where:

G is the command.

PKN is the primary key number.

FN is the file number (1 to 6).

KEY$ is a left-justified partial key.

Search Generic returns the number of the first occurrence of the left-justified partial key. This number helps you find the start of a category of keys. The Search Next (N) command can then be used to access the remainder of the category of keys. Remember to test each time to see if the key in the data file is the same as KEY$.

The Search Generic command cannot be used with integer keys.

See the discussion of the Search (S) command for the values of the error/warning code (ERRF%) returned if KEY$ is not found in the key file.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The record number

ADRKEY = The memory address (in FABS/86 segment) where the key can be found

SEARCH LAST (L) 2.17

The command string is:

CMND$ = "L\PKN\FN"

where:

L is the command.

PKN is the primary key number.

FN is the file number (1 to 6).

Search Last returns the number of the data record that contains the largest key value for the specified primary key.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The record number

ADRKEY = The FABS/86 memory address where the key can be found

2.18 SEARCH NEXT (N)

The command string is:

CMND$ = "N\FN"

where:

2

N is the command.

FN is the file number (1 to 6).

Search Next returns the number of the data record containing the next key in sequence. This command is reliable only if the last command for the same file number was one of the Search commands. The Search Next command does not cross over primary key boundaries. Error code 15 appears when Search Next reaches the end of the group of primary keys.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The record number

ADRKEY = The FABS/86 memory address where the key can be found

SEARCH PREVIOUS (P) 2.19

The command string is:

```
CMND$ = "P\FN"
```

where:

P is the command.

FN is the file number (1 to 6).

Search Previous returns the number of the data record containing the previous key in sequence. This command is reliable only if the last command for the same file number was one of the Search commands. The Search Previous command does not cross over primary key boundaries. Error code 13 appears when the bottom of the group of primary keys is reached.

Parameters returned:

ERRF% = Error/Warning code

RECNO = The record number

ADRKEY = The memory address where the key can be found

WRITE PAGE MAP (W) 2.20

The command string is:

```
CMND$ = "W\FN"
```

where:

W is the command.

FN is the file number (1 to 6).

Use this command to write the page map to your diskette (or hard disk) after a series of Build (B) commands. The Write Page Map command should be executed immediately after the Build commands; no other FABS/86 command should be used between the Build commands.

Parameters returned:

ERRF% = Error/Warning code

RECNO = No significance

ADRKEY = No significance

2.21 OBSOLETE NODE BUFFERS (Z)

The command string is:

CMND$ = "Z\FN"

where:

Z is the command.

FN is the file number.

Any command following the Z command will reload the node buffers and the map from the disk. This provides one user with the ability to ensure updated key data after another user has changed the key file.

3

USING FABS/86 WITH PROGRAMMING LANGUAGES

INTERFACING THE MS-BASIC INTERPRETER TO FABS/86 3.1

LOADING THE FABS/86M.COM MODULE 3.1.1

The FABS/86M.COM module is loaded and fixed in memory (until the next restart) by executing it as you would any transient program. Type:

```
fabs86m
```

This command loads the module and displays the sign-on message along with the "FSEG = &HXXXX" statement showing the segment where the module was loaded. This segment varies for different system configurations.

When you use this procedure, FABS86M.COM should always be the first program you load when your computer is powered up or restarted. By doing this, you ensure that the FABS86M.COM module is loaded in the same place.

WARNING: Don't load FABS86M.COM more than once between restarts. It will load higher each time you load it.

3.1.2 LINKING FABS/86 TO COMPILED LANGUAGES

The FABS86M.OBJ file is a relocatable module used to link FABS/86 to compiled (.OBJ) files generated by the MS-BASIC Compiler, MS-COBOL, MS-Pascal, and other compiled languages. Public declarations in the FABS86M.OBJ module provide linkages to the calling programs.

3.1.3 CALLING FABS/86 FROM MS-BASIC

Before you can begin the FABS/86 calling subroutine, your MS-BASIC program must have the FSEG statement declared. This statement is displayed when the FABS86M.COM module is loaded:

```
FSEG      &Hxxxx
```

where xxxx is the load segment.

Your MS-BASIC program must contain the following FABS/86 calling subroutine:

```
DEF SEG = FSEG
FABS86M = &H5
CALL FABS86M(CMND$,ERRF%,RECNO%,ADRKEY%)
DEF SEG
RECNO = RECNO%
IF RECNO<0 THEN RECNO=RECNO+65536
RETURN
```

(Chapter 2 defines the CMND$, ERRF%, RECNO%, and ADRKEY% parameters for each FABS/86 command string.)

The value of the key can be returned after a First, Last, Next, or Previous command if you use the following subroutine:

```
RKEY$= ""
ADRKEY = ADRKEY%
IF ADRKEY<0 THEN ADRKEY=ADRKEY+65536!
FOR I = ADRKEY TO ADRKEY+MAXKLEN-1
DEF SEG = FSEG
RCHAR = PEEK(I)
DEF SEG
RKEY$ = RKEY$ = CHR$(RCHAR)
NEXT I
```


where:

MAXKLEN is the maximum key length (specified in the Create command).

RKEY$ is the key value.

To execute FABS/86, you simply define a command string (CMND$) for the particular command you desire and then GOSUB to the calling subroutine which actually calls FABS/86.

TEST PROGRAMS 3.1.4

The following test programs are included on the FABS/86 program diskette to show the capabilities of FABS86M:

- FABS86M.COM: The FABS/86 module.
- FABSBLD.BAS: Builds test key and data files.
- FPRINT.BAS: Displays the data file.
- FABSTEST.BAS: Demonstrates the execution of FABS/86 commands.

The FABSBLD.BAS program constructs the FTEST.DAT, FTEST.KEY, and FTEST.MAP files needed by the FABSTEST.BAS

program. If they are not already present on your diskette, run FABSBLD.BAS to create them.

Follow these steps to run the test programs:

1. Copy the test programs from the FABS/86 program diskette to a diskette that contains the MS-DOS operating system. Save the original for backup.
2. Reboot your computer with the MS-DOS diskette in drive A. Load FABS/86 by typing **FABS86M** after the A > prompt.
3. Ensure that the FSEG statement displayed when FABS/86 is loaded is the same as the one at the beginning of the FABSBLD.BAS, FPRINT.BAS, and FABSTEST.BAS programs. If the FSEG statement is not the same, change the others to reflect the segment where FABS86M.COM was loaded.
4. Transfer your MSBASIC.COM (hereafter called MS-BASIC) to the MS-DOS diskette.
5. Enter:

   msbasic fabsbld

   to build the test files. It takes about 20 minutes to insert the 1000 keys (500 records of two primary keys each). About half of this time is used to generate the keys randomly. The keys have a maximum length of 10 bytes. The data file contains the two primary keys and a 12-byte string.
6. Enter:

   msbasic fprint

   to run the FPRINT.BAS program. When prompted, select key sequence, primary key 1, and ascending order. The data record number is displayed on the right.

7. Run the FABSTEST.BAS program by entering:

```
msbasic fabstest
```

The list of FABS/86 commands is displayed.

When prompted, select Generic search, primary key 1, and "R" for the key value. The data record for the first key beginning with "R" is returned. Use the Previous and Next commands to verify the key.

You can insert and delete records at will. Remember that deleted records are reused on a last-deleted, first-reused basis.

FABS/86 AND THE MS-BASIC COMPILER 3.2

CALLING FABS/86 FROM THE MS-BASIC COMPILER 3.2.1

The FABS86M.OBJ object file lets you link FABS/86 with compiled programs using the LINK.EXE program. The compiled MS-BASIC program must contain the following FABS/86 calling subroutine:

```
CALL FABSMB(CMND$,ERRF%,RECNO%,ADRKEY%)
RECNO = RECNO%
IF RECNO<0 THEN RECNO = RECNO + 65536!
```

See Chapter 2 for an explanation of CMND$, ERRF%, RECNO%, and ADRKEY% for each command string.

The actual key value can be returned after any Search command by using the following subroutine:

```
      ADRKEY = ADRKEY%
      IF ADRKEY<0 THEN ADRKEY = ADRKEY+65536!
      CALL GFSEG(FSEG%)
      FSEG = FSEG%
      IF FSEG<0 THEN FSEG = FSEG + 65536!
      RKEY$ = ""
      FOR I = ADRKEY TO ADRKEY+MAXKLEN-1
         DEF SEG = FSEG
         RCHAR = PEEK(I)
         DEF SEG
         IF RCHAR = 0 THEN YYY
         RKEY$ = RKEY$ = CHR$(RCHAR)
      NEXT I
YYY REM   RKEY$ = ACTUAL KEY VALUE
```


3.2.2 TEST PROGRAMS FOR THE MS-BASIC COMPILER

The following example programs are included on the distribution diskette:

- FABS86M.OBJ: The FABS/86 relocatable object module
- MCBUILD.BAS: The test build program
- MCBUILD.OBJ: The object file
- MCTEST.BAS: The FABS/86 test program
- MCTEXT.OBJ: The object file
- MCPRINT.BAS: Prints the files
- MCPRINT.OBJ: The object file
- FTEST.KEY: The test key file
- FTEST.MAP: The test map file
- FTEST.DAT: The test data file

Follow these steps to run a test program:

1. Copy the test programs from the distribution diskette to another diskette that contains the MS-DOS operating system. Save the original for backup.

2. Using LINK.EXE, link the following .OBJ files to form the indicated RUN (.EXE) files. (See the *Systems Programmer's Tool Kit II, Volume I*, for instructions on using LINK.EXE.)

RUN FILES	OBJECT FILES
MCPRINT.EXE	MCPRINT.OBJ + FABS86M.OBJ
MCTEST.EXE	MCTEST.OBJ + FABS86M.OBJ
MCBUILD.EXE	MCBUILD.OBJ + FABS86M.OBJ

3. Run the MCPRINT.EXE program by entering:

```
mcprint
```

   Answer the prompts that follow by selecting key sequence, primary key 1, and ascending order. Observe the speed with which FABS/86 displays the data file in key sequential order. The data record is displayed on the right.

4. Run the MCTEST.EXE program by entering:

```
mctest
```

   The key and data files are opened and the list of FABS/86 commands is displayed. When prompted, select Generic search, primary key 1, and "R" for the key value. The data record for the first key beginning with R is returned. Use the Previous and Next commands to verify the key. You can insert and delete records at will. Remember that deleted records are reused on a last-deleted, first-reused basis.

5. If you want to build a larger set of test files, change the FOR-NEXT loop in the MCBUILD.BAS program to reflect the number of records you desire. Then, use BASCOM.COM to compile it and obtain the MCBUILD.OBJ file for linking with the FABS86M.OBJ file to form MCBUILD.EXE. Then, run MCBUILD.EXE to build the new set of test programs.

3.3 USING FABS/86 WITH MS-PASCAL

The following is a typical calling program from MS-Pascal:

```
program pastest;
type
  cmdstring = lstring(255);
var
  effr:          word;
  recno:         word;
  cmnd:          cmdstring;

function fbspas (vars cmd: lstring;
           vars err: word) : word; external;

begin
cmnd:= "C\FTEST.KEY\10\2\A\3";
recno:= fbspas (cmnd,errf);
end.
```

This program creates the FTEST.KEY and FTEST.MAP files. The key file is set for two primary keys with a maximum key length of 10 bytes. Both keys are ASCII keys; the key file is opened as file number 3.

The returned parameter (recno) has no significance with a Create command. The error code (errf) should be tested for zero to ensure that there was no error.

An additional entry point (FBPAS1) is provided. It returns the key address (in the FABS segment) in addition to the preceding parameters. The function call is:

```
recno:=fbpas1 (cmnd,errf,keyadr);
```

To get the KEYADR segment, enter:

fseg: = gfseg1

You can access the key if you know the segment and the offset of the key.

Link FABS86M.OBJ to your MS-Pascal object file using LINK.EXE.

FABS/86 AND MS-FORTRAN 3.4

To call FABS/86 from MS-FORTRAN, you must define the command string (CMND) and execute an external function as follows:

```
C     TEST PROGRAM FOR CALLING FABS/86
      PROGRAM FBSTST
      CHARACTER *80 CMND
      INTEGER *2 ERRF, RECNO, KEYADR, FBSFOR
      CMND= 'C\TEST.KEY\10\2\A\3'
      RECNO= FBSFOR(CMND,ERRF,KEYADR)
        WRITE(*,200) 'ERCODE =' ERRF
200   FORMAT(1X,A8,I6)
      STOP
      END
```

This program creates empty FABS/86 key and map files (TEST.KEY and TEST.MAP) which are ready for keys to be inserted. The key file has two primary ASCII keys of 10 bytes each; it is opened for access as file number 3.

With MS-FORTRAN, the command string must be terminated with an up-arrow (^) because no length byte is passed with the string. The returned parameter RECNO (normally record number) is returned in the AX register as the returned function. FABS/86 treats RECNO as a 16-bit positive number with a range of 0 to 65535. MS-FORTRAN may return a negative number if greater than 32767. In this case, you would add 65536 to RECNO to place it in the range of 0 to 66535.

The error code is returned in ERRF. This will be zero if there is no error or warning.

The pointer to the actual key in memory after any Search command is returned in KEYADR. This is the temporary address of the key in the FABS segment. You can get the FABS segment by using the following external function:

```
FSEG=GFSEG1
```

where FSEG is the FABS segment.

The FABS86M.OBJ module is linked to your FORTRAN object file using LINK.EXE.

3.5 FABS/86 AND MS-COBOL

To call FABS/86 from MS-COBOL, you must define the command string (CMND) and call an external procedure as shown by the following sample program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBTST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  CMND PIC X(80) VALUE IS 'C\TEST.KEY\10\2\A\1'
77  ERRF   PIC 99999 COMP-0 VALUE 0.
77  RECNO  PIC 99999 COMP-0 VALUE 0.
77  KEYADR PIC 99999 COMP-0 VALUE 0.
77  ERC    PIC 99999.
77  RN     PIC 99999.
77  KA     PIC 99999.
PROCEDURE DIVISION.
MAIN.
CALL 'FBSCOB' USING CMND, ERRF, RECNO, KEYADR
MOVE ERRF TO ERC.
MOVE RECNO TO RN.
MOVE KEYADR TO KA.
DISPLAY 'ERRF =' ERC.
DISPLAY 'RECNO =' RN.
DISPLAY 'KEYADR =' KA.
STOP RUN
```

This program creates a TEST.KEY and TEST.MAP file on the disk and opens the key file for access as file number 1. The key file has two ASCII primary keys of 10 bytes each.

The command string (CMND) must be terminated with an up-arrow (^) since MS-COBOL does not provide a length byte.

The KEYADR pointer gives the offset of the actual key in the FABS segment. It is important for all Search commands. The FABS segment is returned by the following external procedure:

```
CALL 'GFSEG' USING FSEG
```


where FSEG is the FABS segment.

The FABS86M.OBJ module is linked to your MS-COBOL object file using LINK.EXE.

4

FABS/86 ERROR AND WARNING CODES

Table 4-1 describes the error and warning codes displayed by FABS/86. Except for 0, any error code numbered 12 or less is normally a fatal error. The same is true of error codes numbered 16 or higher. All error and warning codes are returned from the calling subroutine in the variable ERRF%.

Table 4-1: FABS/86 Error and Warning Codes

WARNING NUMBER	DESCRIPTION
0	The FABS/86 operation was successful.
4	Improper key for integer key file.
5	Attempted Generic search on integer key.
6	Key not found on Delete.
7	Incorrect number of primary keys.
8	Syntax error in command string.
9	No more key space.
10	Input key larger than maximum.
11	Tried to access unopened file.
12	Key specified was not found, but smaller and larger keys were found.
13	Key not found. Key smaller than all keys.
15	Key not found. Key larger than all keys.
16	Key not found. No keys in key file.
22	File not present when opened.
23	Out of directory space.
24	Diskette full.
25	Write error.
26	File not present when closed.
27	Read error. End of file.
29	Cannot close current extent (CP/M-86).
30	Random record out of range (CP/M-86).

A

FABS/86 PUBLIC INTERFACES AND ABSOLUTE ENTRY OFFSET POINTS

PUBLIC INTERFACES A.1

The following are the FABS/86 PUBLIC entry and access locations.

Table A-1: FABS/86 PUBLIC Entry and Access

PUBLIC SYMBOL	NORMALLY USED BY
FABSMB	MS-BASIC Compiler
FBSPAS	MS-Pascal (doesn't return key address)
FBPAS1	MS-Pascal (returns key address)
FBSFOR	MS-FORTRAN
GFSEG	General
GFSEG1	General
KEYADR	General

Each location is discussed in the following sections.

FABSMB Entry Point

On entering at FABSMB, nonsegmented pointers representing the addresses of CMND, ERROR CODE, RECNO, and KEYADR (in that order) must have been pushed on the stack.

All pointers are assumed to be offsets to the Data Segment register (DS) at entry. CMND must be four bytes: the first two bytes contain the length of the command string; the next two bytes contain the offset of the command string relative to the DS register at entry.

At exit from FABS/86, the four pointers (8 bytes) are removed from the stack and the returned parameters are placed in the specified addresses. The segment associated with the key address can be obtained by calling either GFSEG or GFSEG1.

FBSPAS Entry Point

On entering at FBSPAS, segmented pointers (8 bytes) representing the addresses of CMND and ERROR CODE (in that order) must have been pushed on the stack. The first byte of CMND must contain the length followed by the actual string of characters.

At exit from FABS/86, the two pointers (8 bytes) are removed from the stack and the error code is placed in the specified offset and segment. The returned parameter (RECNO) is placed in the AX register as the returned function.

If the key address is needed, use the FBPAS1 entry point.

FBPAS1 Entry Point

On entering at FBPAS1, segmented pointers representing the addresses of CMND, ERROR CODE, and KEYADR (in that order) must have been pushed on the stack.

The first byte of CMND must contain the length of the command string followed by the actual string of characters.

At exit from FABS/86, the three pointers (12 bytes) are removed from the stack, and the error code and key address are placed in the specified segment and offset. You must use either GFSEG or GFSEG1 to get the FABS segment (same as the KEYADR segment) if you want to access the key. The returned parameter (RECNO) is returned in the AX register.

FBSFOR Entry Point

On entering at FBSFOR, segmented pointers representing the addresses of CMND, ERROR and KEYADR (in that order) must have been pushed on the stack. The CMND pointer must point to the first actual byte of the command string. The command string must be terminated with an up-arrow (^), so that FABS/86 can determine the string length.

At exit, the three segmented pointers (12 bytes) are removed from the stack, and ERROR CODE and KEYADR are placed in the specified addresses. You must use either GFSEG or GFSEG1 to get the KEYADR segment (same as the FABS segment) to access the key value. The returned parameter (RECNO) is returned in the AX register.

FBSCOB Entry Point

On entering at FBSCOB, nonsegmented pointers representing the addresses of CMND, ERROR CODE, RECNO, and KEYADR (in that order) must have been pushed on the stack. All pointers are assumed to be offsets to the Data Segment register (DS) at entry. The command string (CMND) must be terminated with an up-arrow (^) so that FABS/86 can determine the string's length.

At exit the parameters ERROR CODE, RECNO, and KEYADR are placed in the specified addresses. You must use either GFSEG or GFSEG1 to get the KEYADR segment (same as the FABS segment) to access the actual value of the key.

GFSEG Entry Point

This procedure returns the FABS segment in the specified variable as follows:

```
CALL    GFSEG(FSEG)
```

On entering at GFSEG, a nonsegmented pointer must have been pushed on the stack. The FABS segment is placed in the specified address relative to the Data Segment register (DS) at entry.

At exit, the pointer (2 bytes) is removed from the stack.

GFSEG1 Entry Point

This is used as an external function to return the FABS segment as follows:

```
FSEG = GFSEG1
```

When this function is called, FABS/86 returns the FABS segment in the AX register.

KEYADR Public Variable

The FABS/86 internal variable KEYADR contains the temporary address of the actual key value (in the FABS segment) after any normal Search command.

KEYADR resides at offset 28 (hex) in the FABS segment. It consists of a 2-byte offset pointer followed by a 2-byte variable containing the KEYADR segment (same as the FABS segment).

A.2 ABSOLUTE OFFSET ENTRY POINTS

Absolute entry points let compiled languages call an absolute offset in the FABS/86 segment, instead of a PUBLIC entry point. The FABS.COM module can then be loaded and fixed in memory, as is normally done with the MS-BASIC Interpreter. By loading this module, you eliminate the need to link the FABS.REL file into all programs that use large amounts of disk space.

The FABS.COM file is loaded by typing:

```
fabs86m
```

following the A > prompt.

The FABS segment is displayed when the FABS module is loaded. The absolute entry points are offsets in the FABS segment. Table A-2 shows the absolute offsets to call for the various languages.

Table A-2: FABS/86 Absolute Offsets:

LANGUAGE	EQUIVALENT PUBLIC SYMBOL	OFFSET (DECIMAL)
MS-BASIC Interpreter	None	5
MS-BASIC Compiler	FABSMB	8
MS-Pascal	FBSPAS	11
Any	GFSEG	14
MS-FORTRAN	FBSFOR	17
MS-COBOL	FBSCOB	20
MS-Pascal	FBPAS1	23
Any	GFSEG	126

If you are using the MS-BASIC Compiler, for example, you load the FABS module and note the load segment:

```
FSEG = &Hxxxx
```

where xxxx is the start segment of the beginning of the FABS module.

The following statement must be included in your MS-BASIC program just prior to calling FABS/86:

```
DEF SEG=FSEG
```

Then execute an absolute call with an offset of 8. You return to the MS-BASIC segment DEF SEG.

INDEX

Absolute offset entry points, A-1, A-4 to A-5

Binary search, 1-2
Build (B), 2-3

Close (K), 2-4
Codes
  error, 4-1
  warning, 4-1
Command strings, 2-1 to 2-2
Commands, 2-1
  Build (B), 2-3
  Close (K), 2-4
  Create (C), 2-4 to 2-5
  Delete (D), 2-5 to 2-6
  Get maximum key length (M), 2-6
  Get next record number (Q), 2-7
  Get number of deletes (U), 2-8
  Get number of primary keys (H), 2-8 to 2-9
  Get number of records (T), 2-9
  Insert (I), 2-10
  Obsolete node buffers (Z), 2-18
  Open (O), 2-11
  Replace (R), 2-11 to 2-12
  Search (S), 2-12
  Search first (F), 2-13 to 2-14
  Search generic (G), 2-14 to 2-15
  Search last (L), 2-15
  Search next (N), 2-16
  Search-next-after (A), 2-2
  Search previous (P), 2-17
  Write page map (W), 2-17
Create (C), 2-4 to 2-5

Data
  accessing, 1-3
  entering, 1-2
  retrieval, 1-2, 1-4
  storage, 1-1, 1-3
Delete (D), 2-5 to 2-6

Entering data, 1-2
Entry point
  FABSMB, A-1 to A-2
  FBPAS1, A-2
  FBSCOB, A-3
  FBSFOR, A-3
  FBSPAS, A-2
  GFSEG, A-3 to A-4
  GFSEG1, A-4
  KEYADR PUBLIC, A-4
Error codes, 4-1

FABS86M.COM, 3-1
Files
  data, 1-3
  key, 1-3, 2-5
  overflow, 1-3

Generic search, 1-4
Get maximum key length (M), 2-6
Get next record number (Q), 2-7
Get number of deletes (U), 2-8
Get number of primary keys (H), 2-8 to 2-9
Get number of records (T), 2-9

Insert (I), 2-10
Interfacing, 3-1

Keys, 1-5 to 1-7
ASCII, 1-5
duplicate, 1-5
integer, 1-6
length, 1-6 to 1-7, 2-6
maximum, 1-7
maximum number of, 1-6
multiple, 1-5

Linking, 3-2

MS-BASIC Compiler, 3-5 to 3-6, 3-7
MS-BASIC Interpreter, 3-1
MS-COBOL, 3-10 to 3-11
MS-FORTRAN, 3-9 to 3-10
MS-Pascal, 3-8

Obsolete node buffers (Z), 2-18
Open (O), 2-11
Overview, 1-1

Predicting, 1-6
Programming languages, 3-1
Public interfaces, A-1

Replace (R), 2-11 to 2-12

Search (S), 2-12 to 2-13
Search first (F), 2-13 to 2-14
Search generic (G), 2-14 to 2-15
Search last (L), 2-15
Search next (N), 2-16
Search-next-after (A), 2-2
Search previous (P), 2-17

Test programs, 3-3 to 3-5, 3-6 to 3-7

Warning codes, 4-1
Write page map (W), 2-17 to 2-18

AUTOSORT/86M

VICTOR Publications
380 El Pueblo Road
Scotts Valley, California 95066
(408) 438-6680

First VICTOR printing February, 1983.
Second VICTOR printing December, 1983.

ISBN 0-88182-109-8 Printed in U.S.A.

CONTENTS

1. General Information
   - 1.1 AUTOSORT/86 Features 1-1
   - 1.2 Getting Started 1-3

2. AUTOSORT/86 Command Strings
   - 2.1 AUTOSORT/86 Modes 2-2
     - 2.1.1 Mode Parameters 2-2
     - 2.1.2 Mode Description 2-3
     - 2.1.3 Parameter String Description 2-10
   - 2.2 Sample Command String 2-12

3. Sort Parameters
   - 3.1 Parameter File Overview 3-1
   - 3.2 Sort Parameter Definition 3-1

4. Record Select Features 4-1

5. Using AUTOSORT/86 with Programming Languages
   - 5.1 AUTOSORT/86 and MS-BASIC 5-1
   - 5.2 AUTOSORT/86 and the MS-BASIC Compiler 5-3
   - 5.3 AUTOSORT/86 and MS-Pascal 5-6
   - 5.4 AUTOSORT/86 and MS-FORTRAN 5-9
   - 5.5 AUTOSORT/86 and MS-COBOL 5-10

6. Error Indications 6-1

APPENDIXES

A. AUTOSORT/86 PUBLIC Interfaces A-1
B. Stand-Alone Sorting B-1

TABLES

2-1: Order of Parameters in Parameter String 2-11
2-2: Sample Command String Parameter Description 2-13
A-1: AUTOSORT/86 PUBLIC Entry and Access Locations A-1
B-1: TEST.DAT Field Description B-2

CHAPTERS

1. General Information 1

2. AUTOSORT/86 Command Strings 2

3. Sort Parameters 3

4. Record Select Features 4

5. Using AUTOSORT/86 with Programming Languages .. 5

6. Error Indications 6

Appendix A: AUTOSORT/86 PUBLIC Interfaces ..... A

Appendix B: Stand-Alone Sorting ..... B

1

GENERAL INFORMATION

AUTOSORT/86 is a sort/merge/select utility designed for use with very large files that have fixed-length fields within fixed-length records. It is compatible with the MS-DOS operating system, but it does not support path names. It supports string fields and the MS-BASIC integer, single-precision, and double-precision fields.

Note: AUTOSORT/86M may be incorporated in any application and distributed for use on VICTOR/SIRIUS machines without payment of royalties.

AUTOSORT/86 FEATURES 1.1

Nine modes of sort/merge/select are available. The mode is specified in the command string (CMND$).

- Mode 0: Full record sort/select using an existing parameter file.
- Mode 1: Merges two sorted files into one sorted file using an existing parameter file to specify the sort keys.
- Mode 2: Full record sort/select using the parameters specified in the command string; does not write the parameter file to the disk.
- Mode 3: Full record sort/select using the parameters specified in the command string; writes the parameter file to the disk.
- Mode 4: Sort/select using a parameter file. The output file contains only the Data Record pointer and sort keys.
- Mode 5: Sort/select using a command string. The output file contains only the Data Record pointer and sort keys.

- Mode 6: Sort/select using a parameter file. The output file contains only the 2-byte Data Record pointer.
- Mode 7: Sort/select using a command string. The output file contains only the 2-byte Data Record pointer.
- Mode 8: Creates a parameter file on the disk from the command string. No sort/select is done.

AUTOSORT/86 can be used as a stand-alone sort routine, or it can be called from the MS-BASIC Interpreter, the MS-BASIC Compiler, MS-Pascal, MS-FORTRAN, or MS-COBOL. However, MS-COBOL ISAM files are not supported by AUTOSORT/86. Multiple users can sort simultaneously as long as each user's program specifies a different user number in the command string. AUTOSORT/86 creates unique temporary files for each user, eliminating the possibility of conflicting temporary filenames.

Record lengths can be 5000 bytes or more if the specified sort buffer is at least 40K. File size is determined by your operating system and the available disk work file space. The logical record counter overflows at 65536.

In a 128K system, a typical default buffer size is about 60K if the sort module is loaded low enough to permit a 60K buffer. AUTOSORT/86 creates up to 30 work files; each file is a little smaller than the sort buffer size. When you still have more to sort after these 30 work files are filled, AUTOSORT/86 temporarily merges the work files into a single file and creates up to 29 more work files. This process continues until either the input file or work space is depleted. The worst-case requirement for work space is about twice the size of the input file, if temporary merges are required. The temporary merge occurs when there are less than 30 work files and the specified buffer size is too small to allow 30 work files to be merged. A maximum of 10 sort keys is permitted, either ascending or descending (independently) on each key.

AUTOSORT/86 deletes or retains records by comparing up to four independent select keys with any fields in the record (fixed, variable, string, or numeric). AUTOSORT/86 checks whether the select key is "less than", "equal to", or "greater than" the selected fields. A select OR function (if activated) lets records be retained if any one of several select keys matches the fields.

An alpha option translates all lowercase alpha characters to uppercase for sorting. This causes "a" and "A" to sort as the same character.

The disk change capability is directed by the sort parameters. This capability lets you change the work file disk or the output disk during the sort process. (A screen prompt appears when you must change a diskette.) If the output file is given the same name as the input file, the input file is deleted after the work files are created. This saves disk space; however, a power loss or malfunction during the final merge can cause you to lose the data file. For this reason, you should always back up your data files.

GETTING STARTED 1.2

Before using AUTOSORT/86, you should read the manual and then run the stand-alone test programs in Appendix B. Then, Chapter 5 explains how to call the sort as a subroutine.

2

AUTOSORT/86 COMMAND STRINGS

Mode and sort parameters are passed to the AUTOSORT/86 module in a single command string (CMND$) with each parameter separated by a backslash (\). AUTOSORT/86 allows the user to change the delimiter in the command by putting the desired delimiter as the first character in the command line. Any character with an ASCII code of 2F Hex or less will work, periods excluded.

For simplicity, the command string (CMND$) is shown as two strings separated by a backslash. The "\" indicates a backslash is necessary between the adjacent parameters when the two strings are concatenated.

The AUTOSORT/86 command string format is:

CMND$ = AMODE$ + "\" + PARM$

where:

AMODE$ contains parameters that must be defined during run time, and are associated with the sort mode.

PARM$ represents the sort parameters defined during run time or specified as parameters in a parameter file that has been previously stored on disk.

The sort parameter string (PARM$) begins with the input filename. String elements are in the same order as they were entered into the sort parameter file created by the parameter file generator program, PFG86M.COM. (See Chapter 2.1.3, "Parameter String Description," for a description of these parameters.) To call AUTOSORT/86, you create the command string (CMND$) for the desired mode. Then, call the AUTOSORT/86 module according to the procedure specified for your higher-level language.

2.1 AUTOSORT/86 MODES

2.1.1 MODE PARAMETERS

The first parameter in the mode parameter string (AMODE$) defines the mode of operation of the sort module. (See Chapter 2.1.2 for an explanation of the various modes.)

The second parameter is the user number—a single character inserted into all temporary files to make them unique. Any normal filename character is permitted. The user number lets several users sort simultaneously on the same disk using different user numbers.

The third parameter sets the drive where the sort buffer memory area is saved during the sort process. Drives A through Z can be specified. (Use "0" for the default drive.)

The fourth parameter sets the sort buffer size (in bytes). You should make this size as large as possible; a zero defaults to the maximum size (approximately 60K for a 128K system). The sort buffer is above the sort module in memory. Its contents are written to disk before the sort and restored to disk after the sort. If you have limited disk space, you might want to set the sort buffer size to some smaller value. If you have very large records, be sure to leave enough room for the parallel merge. Work files, if needed, require additional disk space.

Some modes may have additional parameters in the mode parameter portion (AMODE$) of the command string (CMND$).

Sort parameters can be defined dynamically during run time as a sort parameter string (PARM$); or they can be defined external to the program by using the PFG86M.COM program. Also, certain modes let you define the sort parameters dynamically and write them to the disk as a sort parameter file for later use.

The parameter file generator (PFG86M.COM) creates the parameter files for modes 0, 1, 4, and 6. It is also useful when creating the parameter string (PARM$) for modes 2, 3, 5, 7, and 8. PFG86M.COM requests the identical parameters (starting with the input filename) in the same order as are needed to form the sort parameter string (PARM$). Remember that the parameters in PARM$ must be separated by backslashes.

To create a parameter file using PFG86M.COM, enter:


```
pfg86m
```

and then answer the questions. (See Chapter 3, "Sort Parameters," for a detailed description of PFG86M.)

MODE DESCRIPTION 2.1.2

Mode 0

The command string format is:

```
AMODE$ = "0\Un\Dm\Bufsize"
PARM$ = "D:PFNAME"
CMND$ = AMODE$ + "\" + PARM$
```

where:

0 is the mode.

Un is the user number. For a single user, enter 1.

Dm is the temporary memory storage drive. Drives A through Z can be specified. (Use zero for default drive.)

Bufsize is the number of bytes in the buffer. 0 defaults to maximum.

D is the drive containing PFNAME (optional).

PFNAME is the name of the parameter file. The extension must be .SRT.

In mode 0, a full record sort/select is performed using sort parameters from a parameter file previously created by PFG86M.COM or by using modes 3 or 8.

See Chapter 2.1.3, "Parameter String Description."

Mode 1

The command string format is:

```
AMODE$ = "1\Un\Dm\Bufsize"
PARM$ = "D:PFNAME\D:INPUT1\D:INPUT2\D:OUTPUT"
CMND$ = AMODE$ + "\" PARM$
```

where:

1 is the mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to the maximum.

D is the appropriate drives (optional).

PFNAME is the parameter filename with extension .SRT.

INPUT1 is the name of the first input file.

INPUT2 is the name of the second input file.

OUTPUT is the name of the output file.

This mode merges two sorted files into one sorted file. A previously created parameter file must exist on disk to provide the sort/select key information. If the parameter file is set to skip records, the records in the first input file (INPUT1) are skipped.

The input and output filenames specified in the parameter file on the disk are ignored since they are already specified in the command string. For a correctly ordered merge, the sort keys must specify the same order as the sorted input files; otherwise, the sorted results are unpredictable.

If select keys are specified in the parameter file, the appropriate records are selected.

Mode 2

The command string format is:

```
AMODE$ = "2\Un\Dm\Bufsize"
PARM$ = The sort parameters
CMND$ = AMODE$ + "\" + PARM$
```

where:

2 is the sort/select mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to maximum.

PARM$ is the sort parameter string beginning with the input filename.

This mode does a full record sort/select using the parameters specified in the command string. No parameter file is written to disk.

See Chapter 2.1.3.

Mode 3

The command string format is:

```
AMODE$ = "3\Un\Dm\Bufsize\D:PFNAME"
PARM$ = The sort parameters
CMND$ = AMODE$ + "\" + PARM$
```


where:

3 is the mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to maximum.

D is the drive on which to put the parameter file.

PFNAME is the name to give the parameter file.

PARM$ is the sort parameter string beginning with the input filename.

This mode writes a parameter file to the disk for later use and then does a full record sort using the parameters specified. The name you give the parameter file is inserted as parameter number 5 in AMODE$.

See Chapter 2.1.3.

Mode 4

The command string format is:

```
AMODE$ = "4\Un\Dm\Bufsize"
PARM$ = "D:PFNAME"
CMND$ = AMODE$ + "\" + PARM$
```

where:

4 is the mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to the maximum.

D is the drive containing the sort parameter file to use.

PFNAME is the sort parameter filename. (The extension is assumed to be .SRT.)

This mode uses a previously created sort parameter file. It creates an output file with records containing only the data record number (2 bytes) and the sorted fields in the order specified:

```
output record = [rec. no.][field #1]..[field #n]
```

The output record length equals 2 plus the sum of the sort key lengths.

Mode 5

The command string format is:

```
AMODE$ = "5\Un\Dm\Bufsize"
PARM$ = The sort parameters
CMND$ = AMODE$ + "\" + PARM$
```

where:

5 is the mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to the maximum.

PARM$ is the sort parameter string beginning with the input filename.

This mode uses a sort parameter string (PARM$). Mode 5 creates an output file which contains only the data record pointer (2 bytes) and the sort keys in the order specified:

```
output record = [rec. no.][field #1]..[field #n]
```

Mode 6

The command string format is:

```
output record = [rec. no. (2 bytes)]
AMODE$ = "6\Un\Dm\Bufsize"
PARM$ = "D:PFNAME"
CMND$ = AMODE$ + "\" + PARM$
```

where:

6 is the mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to the maximum.

D is the drive containing the sort parameter file to use.

PFNAME is the sort parameter filename. (The file extension is assumed to be .SRT.)

This mode uses a sort parameter file to do the sort/select. Then, it produces an output file consisting of only the input data record pointers (2 bytes per data record).

Mode 7

The command string format is:

```
output record = [rec. no.(2 bytes)]
AMODE$ = "7\Un\Dm\Bufsize"
PARM$ = The sort parameters
CMND$ = AMODE$ + "\" + PARM$
```


where:

7 is the mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to the maximum.

PARM$ is the sort parameters beginning with the input filename.

This mode uses a sort parameter string (PARM$) to do the sort/select, and produces an output file containing only the input data record pointers (two bytes per record).

Mode 8

The command string format is:

```
AMODE$ = "8\Un\Dm\Bufsize\D:PFNAME"
PARM$ = The sort parameters
CMND$ = AMODE$ + "\" + PARM$
```

where:

8 is the mode.

Un is the user number. Use 1 for a single user.

Dm is the temporary memory storage drive. (0 is the default.)

Bufsize is the buffer size. 0 defaults to the maximum.

D is the drive on which to put the sort parameter file.

PFNAME is the name to give the parameter file. (The extension is assumed to be .SRT.)

PARM$ is the sort parameter string beginning with the input filename.

Note: Although Un, Dm and Bufsize are not used, each must still be set to an acceptable value.

This mode only creates a sort parameter file on a disk. No sort/select is done.

2.1.3 PARAMETER STRING DESCRIPTION

Table 2-1 shows the order of the parameters in the parameter string (PARM$) for modes 2, 3, 5, 7 and 8. This is the same order in which the parameters are requested when building a parameter file on disk.

Table 2-1: Order of Parameters in Parameter String

PARAMETER	INPUT	MAXIMUM CHARACTERS
INPUT FILE	d:filename.ext	14
OUTPUT FILE	d:filename.ext	14
NUMBER RECS TO SKIP	nnn	3
LOGICAL RECORD LENGTH	nnnn	4
CHANGE WORK FILE DISK	Y/N	1
CHANGE OUTPUT FILE DISK	Y/N	1
WORK FILE DRIVE	A–Z or 0	1
Sort Key 1		
FIELD STARTING POSITION	0 or nnnn (0 to stop key input)	4
FIELD LENGTH	nnn	3
ASCEND OR DESCEND FLAG	A or D	1
ALPHA/HEX/INTEGER/SGL/DBL	A/H/I/S/D (Repeats the last 4 for next key. Enter 0 to stop key input, except when all 10 keys are used, then go to the next input.)	1
ACTIVATE SEL "OR" FUNCTION	Y/N	1
Select Key 1		
DELETE OR RETAIN	0 or D or R (0 to stop key input)	1
FIELD STARTING POSITION	nnnn	4
FIELD LENGTH	nnn	3
ALPHA/HEX/INTEGER/SGL/DBL	A/H/I/S/D	1
LESS-EQUAL-GREATER	L or E or G	
DELETE-RETAIN KEY (Sum of all keys must not exceed 135.)	(Repeats the last 6 for next key. Enter 0 to stop key input if all 4 keys are not used; otherwise input is complete.)	135

2.2 SAMPLE COMMAND STRING

The following command string (CMND$) sorts the input file TEST.DAT. TEST.DAT contains 128 byte records in ascending order on the field starting at byte 4; each byte record has a length of 8 bytes in ascending order. The command string deletes all records that have "DELETED" at byte position 4. Mode 2 is used for a full record sort, so the parameter file is not written to the disk. The output file is named SORTED.

See Chapter 3, "Sort Parameters," for the definition of each parameter.

```
AMODE$ = "2\1\A\0"        (mode parameter string)

PARM1$ = "TEST.DAT\SORTED\0\128\N\N\B"
PARM2$ = "4\8\A\A\0"       (sort keys)

PARM3$ = "N\D\4\7\A\E\DELETED\0"        (sel keys)

PARM$ = PARM1$ + "\" + PARM2$ + "\" + PARM3$
   (sort parameter string)

CMND$ = AMODE$ + "\" + PARM$        (final command string)
```

Table 2-2 explains the parameters contained in the sample command string.

Table 2-2: Sample Command String Parameter Description

PARAMETERS	DESCRIPTION
(AMODE$)	
2	Mode
1	User number
A	Memory storage drive is "A"
0	Default to maximum buffer size
(PARM1$)	
TEST.DAT	Input file (default drive)
SORTED	Output file (default drive)
0	Skip 0 header records
128	Record length = 128 bytes
N	Do not change work diskette
N	Do not change output diskette
B	Work file drive is "B"
(PARM2$) SORT KEYS	
4	Sort key starts at byte 4
8	Sort key is 8 bytes long
A	Ascending order
A	Alpha (make upper/lowercase same)
0	End of sort keys (repeat the last 4 if more sort keys)
(PARM3$)	
N	Do not use select "OR" function
SELECT KEYS	
D	Delete option (0 if no select keys)
4	Select key starts at byte 4
7	Select key is 7 bytes long
A	Alpha (upper/lowercase same)
E	Select on "EQUAL"
DELETE	Select key value = "DELETED"
0	No more select keys (repeat last 6 if more select keys)

3

SORT PARAMETERS

PARAMETER FILE OVERVIEW 3.1

The sort parameter files contain three types of parameters: file, key, and control. The file parameters specify the drive, filename, and file type of the output and input files. The allowable drives are A through Z.

The sort and select key parameters are specified by the byte position in the record and the number of bytes in the key. Up to ten ascending or descending keys are allowed. The keys are sorted on the hex value of each byte if the Hex (H) control character is selected. If the Alpha (A) control character is selected, upper- and lowercase characters sort as the same value. The Integer (I), Single-precision (S), and Double-precision (D) control characters expect to find fields that are compatible with MS-BASIC integer, single-precision, and double-precision fields. Negative numbers are considered smaller than positive; a larger negative number sorts smaller than a lesser negative number.

If the disk change options are used, a screen prompt asks you to insert either the work file diskette or the output diskette, and tells you the drive in which to place it.

SORT PARAMETER DEFINITION 3.2

Generally, parameter files are created dynamically (if needed) during run time; however, they can also be created by the PFG86M.COM program.

When the command string (CMND$) is used to specify the sort parameters, the actual sort parameters begin with the input filename

contained in the command string. The first four parameters specify the mode, the user number, the memory storage drive, the sort buffer size, and possibly additional parameters.

The parameter file generator program is executed by entering "PFG86M" after the system prompt. Then the following prompts appear. (The action you need to take is described following each prompt.)

ENTER PARAMETER FILE TO CREATE (D:FILENAME), DO NOT ENTER FILE EXTENSION (ASSUMED .SRT).

Enter the desired name.

ENTER "INPUT" FILE (D:FILENAME.EXT)

Enter the input file.

ENTER "OUTPUT" FILE (D:FILENAME.EXT)

Enter the output file.

ENTER NUMBER OF RECORDS TO SKIP

Enter the number of header records to skip.

ENTER LOGICAL RECORD LENGTH

Enter the record length.

CHANGE WORK FILE DISKETTE?

Enter Y or N.

CHANGE OUTPUT FILE DISKETTE?

Enter Y or N.

ENTER WORK FILE DRIVE

Enter drive names from A to Z. Enter 0 to use default drive.

ENTER KEY #1 STARTING POSITION (0 TO STOP)

Enter starting byte (0 if no more keys). Enter 0 for the first key if there are no select keys.

ENTER KEY #1 LENGTH

Enter the length (1 to 255).

ENTER ASCEND/DESCEND FLAG

Enter A or D.

ENTER ALPHA/HEX/INTEGER/SGL/DBL FLAG

Enter A to sort upper/lowercase as the same value. Enter H to sort on the hex value. Enter I to sort integer fields. Enter S to sort single-precision fields. Enter D to sort double-precision fields.

The sort keys repeat until a 0 is entered for the FIELD #/STARTING POSITION, or until 10 keys are used.

ACTIVATE THE SELECT KEY "OR" FUNCTION?

Enter N to use multiple select keys as an AND function and Y to use multiple keys as an OR function. For single select keys, use N. If there are no select keys, use either.

DELETE OR RETAIN RECORD?

Enter D to delete or R to retain the records whose fields match the select key. Enter 0 if no more select keys. Entering 0 for the first select key results in no select function being applied.

ENTER SELECT FIELD STARTING POSITION

Enter the starting byte number.

ENTER SELECT FIELD LENGTH

Enter the length of the select field in the data record.

ALPHA/HEX/INTEGER/SGL/DBL FLAG

Enter A to treat upper/lowercase the same. Enter H to use actual hex value. Enter I to sort integer fields. Enter S to sort single-precision fields. Enter D to sort double-precision fields.

DELETE ON "LT, EQ, GT" THE SELECT KEY?

Enter L for less than; enter E for equal; enter G for greater than.

ENTER D/R (DELETE/RETAIN) KEY D/R KEY:

Enter the actual select key value.

The select keys repeat until a 0 is entered for the DELETE or RETAIN parameter, or until four select keys are used. The parameter file is created on the specified drive.

4

RECORD SELECT FEATURES

AUTOSORT/86 can create an output file that contains only selected records from the input file. The select can occur during the sort process, or it may occur independent of a sort (if no sort keys are specified).

Four independent select keys can be specified. This lets you delete or retain records if the specified fields are less than, equal to, or greater than the DELETE/RETAIN select keys.

If Y is entered for the select key OR function and more than one select key is used, records that match any of the keys can be deleted or retained.

If N is entered for the select key OR function and more than one select key is used, all fields must match for the record to be retained (AND function).

The select keys are totally independent of the sort keys. Selects can be performed on the same keys or on different keys than the sorts.

For additional flexibility, the characters $<$, $=$, $>$ can be inserted at any place in the select key, with the following results:

- When a $<$ is encountered in the select key, the select key character is considered "less than" the corresponding character in the data field.
- When a $>$ is encountered in the select key, the select key character is considered "greater than" the corresponding character in the data field.
- When an $=$ is encountered in the select key, the select key character is considered "equal to" the corresponding character in the data field.

If the select key is longer than the data field, it is truncated to the length of the data field. If the select key is shorter than the data field, the data field is compared only for the length of the select key. The select key is considered a "match" if all characters in the select key match the data field for the length of the select key.

The select characters can be used only with string fields. They are not permitted with integer, single-precision, or double-precision fields.

5

USING AUTOSORT/86 WITH PROGRAMMING LANGUAGES

Read Appendix B and run the stand-alone tests before continuing with this chapter.

AUTOSORT/86 AND MS-BASIC 5.1

To call AUTOSORT/86 from an MS-BASIC program, the AS86M.COM module must be resident at a known segment in memory. To load the AS86M.COM module, type:

as86m

after the A> prompt. The MS-BASIC Interpreter and programs are loaded above the AS86M.COM module by MS-DOS. When the AS86M.COM module is loaded, the load segment is displayed as:

ASSEG = &Hnnnn

where nnnn is the segment where the AS86M.COM module is loaded.

This statement must be placed at the beginning of all MS-BASIC programs that call the AUTOSORT/86 program (AS86M.COM). Also, you need the following subroutine in your MS-BASIC program:

```
DEF SEG = ASSEG
ASORT = &H3
CALL ASORT(CMND$,SORTERR%,RECCNT%)
DEF SEG
RECCNT = RECCNT%
IF RECCNT<0 THEN RECCNT=RECCNT+65536!
RETURN
```

To call a sort, simply declare a command string (CMND$) and GOSUB to the preceding subroutine. After a Return, you should test the returned variable SORTERR%. If it is not zero, there was an error. The variable RECCNT is equal to the number of logical records in the output file.

Follow these steps to run the test programs:

1. Copy the following programs from your AUTOSORT/86 disk to a new, formatted disk:
   - AS86M.COM
   - TEST.DAT
   - TEST.BAS
   - PRINT.BAS
2. Copy your MSBASIC.COM file to the new diskette.
3. Place the new diskette in drive A and type **AS86M** so that the AS86M.COM module is loaded and fixed in memory.
4. Record the ASSEG statement displayed when AS86M.COM was loaded.
5. Edit the TEST.BAS and PRINT.BAS programs so that the ASSEG statement is replaced with the one displayed for your system configuration.
6. Run the test program by entering:

   msbasic test

   after the A> prompt. A mode 2 (full record) sort is done on 500 25-byte records. See Appendix B for a description of the parameters used in this sort.
7. Run the PRINT.BAS program to display the TEST.DAT (input) file or the SORTED (output) file.
8. List the TEST.BAS program and use it as an example of how this particular command (CMND$) was formed.

You can change the mode from 2 to 5 or 7 to do different kinds of sorts. If the mode is changed to 5, the output records consist of a 2-byte integer field (the data record pointer) and an 8-byte string field (10-byte record length). If you set the mode to 7, the output records consist of a 2-byte integer field for a total record length of 2 bytes. The PRINT.BAS program must be modified to print the output file for mode 5 or 7 because it expects a 25-byte record and the same fields as in the input file.

5.2 AUTOSORT/86 AND THE MS-BASIC COMPILER

Follow these steps to combine AUTOSORT/86 with a compiled MS-BASIC program:

1. Put this subroutine in the MS-BASIC program:

```
CALL SORTMC(CMND$,SORTERR%,RECCNT%)
RECCNT = RECCNT%
IF RECCNT<0 THEN RECCNT=RECCNT+65536!
RETURN
```

2. To call a sort, you simply define a command string (CMND$) and GOSUB to the preceding subroutine. The command string provides the sort attributes.
3. Compile your MS-BASIC program using the BASCOM program. (See the *MS-BASIC Compiler Reference Manual.*) You now have an .OBJ file for your program.
4. Using LINK.EXE, link your .OBJ file with the AS86M.OBJ file to form the executable program. (See MS-LINK in the *Systems Programmer's Tool Kit II, Volume I.*)

   Note: When linking the AS86M.OBJ module to your programs, the AS86M.OBJ module **must** be specified for linking after the .OBJ file you have created with the MS-BASIC Compiler.
5. Execute your program.

Follow these steps to run the MS-BASIC Compiler test program:

1. Copy the following programs from your AUTOSORT/86 disk to a new, formatted disk:
   - ▶ AS86M.OBJ: Sort module
   - ▶ TEST.DAT: Test data file
   - ▶ TESTBC.BAS: MS-BASIC test program
   - ▶ TESTBC.OBJ: Object file for test program
   - ▶ PRINT.BAS: Program for printing files
   - ▶ PRINT.OBJ: Object file for PRINT.BAS
2. Copy your BASRUN.EXE file to the new diskette.
3. Link the following object files to form the indicated executable files.

EXECUTABLE FILES	OBJECT FILES
TESTBC.EXE	TESTBC.OBJ + AS86M.OBJ
PRINT.EXE	PRINT.OBJ

4. Run the test program by entering:

   testbc

   after the system prompt.

   A mode 2 (full record) sort is done on 500 25-byte records. The sort field starts at byte 4 and is 8 bytes long. See Appendix B for a description of the parameters used in this sort.

5. Run the PRINT.EXE program to display the TEST.DAT (input) file or SORTED (output) file.
6. List the TESTBC.BAS program and use it as an example of how this particular command CMND$) was formed.

You can change the mode to 5 or 7 for different kinds of sorts. If the mode is 5, the output records consist of a 2-byte integer field (the data record pointer) and an 8-byte string field (10 byte record length). If the mode is 7, the output records consist only of a 2-byte integer field for a total record length of 2 bytes. The PRINT.BAS program must be modified to print the output file for mode 5 or 7 because the program expects a 25-byte record and the same fields as in the input file.

In order to call AUTOSORT/86 using an absolute call from the MS-BASIC Compiler, AS86M.COM must be resident at a known memory location. The AUTOEXEC.BAT file delivered on the distribution disk will automatically load AS86M.COM. When AS86M.COM is loaded, the segment is displayed as:

ASSEG = &Hnnnn

where nnnn is the segment where it is loaded (in hexadecimal).

This statement must appear in your MS-BASIC program to set the variable "ASSEG" equal to the segment of AUTOSORT/86. Another method of setting the value of ASSEG is by reading the value of the file ASSEG with:

```
OPEN "D:ASSEG" FOR INPUT AS #1
INPUT #1 ASSEG
CLOSE #1
```

The file ASSEG was written to disk when AS86M.COM was loaded.

In addition, you will need the following subroutines in your MS-BASIC Compiler program:

```
DEF SEG = ASSEG
ASORT% = &H9
CALL ABSOLUTE (CMND$,SORTERR%,RECCNT%,ASORT%)
RECCNT = RECCNT%
IF RECCNT<0 THEN
    RECCNT = RECCNT +65536!
RETURN
```

To call a sort, you simply declare a command string (CMND$) and GOSUB to the above subroutine.

5.3 AUTOSORT/86 AND MS-PASCAL

The following sample Pascal program does a mode 0 (zero) sort (full record sort using a parameter file from the disk), saves the buffer area on the default drive, and uses the maximum buffer space.

```
program pasprog(output);
 type
  cmdstring = lstring(255);
 var
   ercode:     word;
   reccnt:     word;
   command:    cmdstring;
 function sortps(vars cmd: lstring;
          vars err: word) : word; external;

 PROCEDURE errtn(code:word);
 BEGIN
  writeln(output,'Fatal error number ',code);
 END;

 PROCEDURE rcnt(nrecs:word);
 BEGIN
  writeln(output,'Number of records is ',nrecs);
 END;

 begin
  command:= '0\1\0\20000\TEST';
  reccnt:= sortps(command,ercode);
  rcnt(reccnt);
  if ercode <> 0 then errtn(ercode);
 end.
```

To call a sort, simply define a command string and execute the external procedure (sortps). Remember to use an lstring for the command string and segmented variables (vars) where indicated.

To link AUTOSORT/86 to the Pascal program, you must first compile your Pascal program using the directions given in the *MS-Pascal Reference Manual.* This will produce an object version of your program (YOURPROG.OBJ).

Then, you must link your program with the AS86M.OBJ module using the instructions given for linking modules using LINK.EXE. (See MS-LINK in the *Systems Programmer's Tool Kit II, Volume I.*)

Note: The AS86M.OBJ module **must not** be the first module specified for linking.

When requested by the linker, enter the object modules as shown:

```
Object Modules: YOURPROG,AS86M
```


AUTOSORT/86 assigns buffer space above its location in memory. The contents of this buffer area are written to the specified diskette during the sort procedure and are replaced before returning to the Pascal program. This lets you sort large files (several megabytes) from within the Pascal program in a reasonable time without the overhead of dedicated buffer space.

A sample Pascal program (PASTEST.PAS and PASTEST.OBJ) is provided on the distribution diskette. (The sample Pascal program on the disk is not the same as that in the last example.) Follow these steps to run the sample program:

1. Copy the following programs to a new, formatted diskette:
   - ▶ PASTEST.PAS: The sample source file
   - ▶ PASTEST.OBJ: The sample object file
   - ▶ TEST.DAT: The test data file
   - ▶ AS86M.OBJ: The AUTOSORT/86 module
   - ▶ PRINT.BAS: An MS-BASIC file used to display the TEST.DAT and SORTED files.

2. Link PASTEST.OBJ and AS86.OBJ to create PASTEXT.EXE. (See the *MS-Pascal Reference Manual.*)

3. Run the PASTEST.EXE program.

   A mode 2 (full record) sort is done on 500 25-byte records. The sort field starts at byte 4 and is 8 bytes long. See Appendix B for a description of the parameters used in this sort.

4. If you have the MS-BASIC Interpreter, run the PRINT.BAS program to display the TEST.DAT (input) file or the SORTED (output) file. If you do not have the Interpreter, use the PRINT.BAS program as a guide to constructing a Pascal program that displays the files.

5. List the PASTEST.PAS program and use it as a guide to incorporate AUTOSORT/86 into your Pascal program.

You can change the mode (the first parameter) in the command string to produce different types of output files. If the mode is 5, the output records consist of a 2-byte integer field, the data record pointer, and an 8-byte string field (for 10 byte total record length). If the mode is 7, the output records consist only of a 2-byte integer field (for a total record length of 2 bytes). The PRINT.BAS program must be modified to print the output file (SORTED) for mode 5 or 7 because the program expects a 25-byte record and the same fields as the input file (TEST.DAT).

The following sample FORTRAN program does a mode 0 sort (full record sort using a parameter file from the disk), saves the buffer area on the default drive, and uses the maximum buffer space.

```
C     TEST PROGRAM FOR AUTOSORT AND FORTRAN

PROGRAM FORTEST
CHARACTER *80 CMND
INTEGER *2 RECCNT, ERCODE, SORTFO
CMND = '0\1\0\0TEST'
RECCNT = SORTFO(CMND, ERCODE)

WRITE (*,100) 'ERCODE =', ERCODE
WRITE (*,200) 'RECCNT =', RECCNT

100   FORMAT (1X,A8,I6)
200   FORMAT (1X,A8,I6)
      STOP
      END
```


The CMND character string must be terminated with an up-arrow. This is because no length byte is passed with a FORTRAN string. See the description of mode 0 in Chapter 2 for an explanation of this command string.

To link AUTOSORT/86 with your FORTRAN program, first compile your program to produce the object file. Using LINK.EXE, link your program with AS86M.COM when requested by the linker:

```
Object Modules: YOURPROG+AS86M
```

Do **not** load the AS86M.OBJ module first.

AUTOSORT/86 assigns buffer space above its location in memory. The contents of this buffer area are written to the specified diskette during the sort procedure and are replaced before returning to the FORTRAN program. This lets you sort large files (several megabytes) from within the FORTRAN program in a reasonable time without the overhead of dedicated buffer space.

5.5 AUTOSORT/86 AND MS-COBOL

The following sample COBOL program does a mode 0 sort (full record sort using a parameter file from the disk), saves the buffer area on the default drive, and uses the maximum buffer space.


```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBTEST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 CMND PIC X(80) VALUE IS '0\1\0\0\TEST'
77 ERCODE PIC 99999 COMP-0 VALUE 0.
77 RECCNT PIC 99999 COMP-0 VALUE 0.
77 ERC    PIC 99999.
77 RCNT   PIC 99999.
PROCEDURE DIVISION.
MAIN.
  CALL 'SORTCO' USING CMND, ERCODE, RECCNT
  MOVE ERCODE TO ERC.
  MOVE RECCNT TO RCNT.
  DISPLAY 'ERCODE =' ERC.
  DISPLAY 'RECCNT =' RCNT.
  STOP RUN
```

The CMND character string must be terminated with an up-arrow so that AUTOSORT/86 can find the length of the string. See the discussion of mode 0 in Chapter 2 for an explanation of this command string.

To link AUTOSORT/86 with your COBOL program, first compile your program to produce the object file. Using LINK.EXE, link your program with AS86M.OBJ when requested by the linker:

```
Object Modules: YOURPROG+AS86M
```

Do **not** load the AS86M.OBJ module first.

AUTOSORT/86 assigns buffer space above its location in memory. The contents of this buffer area are written to the specified diskette during the sort procedure, and are replaced before returning to the COBOL program. This lets you sort large files (several megabytes) from within the COBOL program in a reasonable time without the overhead of dedicated buffer space.

6

ERROR INDICATIONS

When an error is detected, the error number is returned to the calling program. If the returned error number is 0, no error occurred. These error numbers are generated by AUTOSORT/86. Each is followed by an explanation and suggestions for correcting the error condition.

1 READ PAST END OF FILE

This error should not occur. It probably indicates a system or diskette malfunction.

2 READ ERROR OR BAD FILE

Same as error 1.

3 FILE NOT PRESENT WHEN OPENED

A file was not present on a specified drive. Check the directory.

4 OUT OF DIRECTORY SPACE

Too many directory entries. Delete unnecessary files.

5 NOT USED

6 NOT USED

7 FILE NOT PRESENT WHEN CLOSED

Same as error 1.

8 INSUFFICIENT DISK SPACE

Delete unnecessary files.

9 SORT BUFFER SPACE TOO SMALL

Set larger buffer size in the command string.

11 NOT USED

13 SYNTAX ERROR IN THE COMMAND STRING

Check the command string carefully.

14 SELECT KEY LENGTH IS ZERO

Select key length cannot be zero.

15 SYNTAX ERROR IN THE COMMAND STRING

Check the command string carefully.

A

AUTOSORT/86 PUBLIC INTERFACES

Table A-1 shows the PUBLIC entry and access locations used by AUTOSORT/86.

Table A-1: AUTOSORT/86 PUBLIC Entry and Access Locations

PUBLIC SYMBOL	NORMALLY USED BY
SORTMC	MS-BASIC Compiler
SORTPS	MS-Pascal
SORTFO	MS-FORTRAN
SORTCO	MS-COBOL

SORTMC ENTRY POINT A.1

On entering at SORTMC, three nonsegmented pointers must be pushed on the stack. These represent the addresses of the command string descriptor, sort error code, and record count, in that order. All pointers are assumed to be offsets to the Data Segment register (DS) at entry.

The command string descriptor must be 4 bytes. The first 2 bytes contain the length of the command string; the next 2 bytes contain the offset of the command string relative to the DS register at entry.

At exit from AUTOSORT/86, the three pointers (6 bytes) are removed from the stack and the returned parameters are placed in the specified addresses.

A.2 SORTPS ENTRY POINT

On entering at SORTPS, two segmented pointers (8 bytes) must be pushed on the stack. These represent the addresses of the command string descriptor and error code, in that order. The segment is pushed on the stack first, followed by the offset. The first byte of the command descriptor must contain the length, followed by the actual string of characters.

At exit from AUTOSORT/86, the two pointers (8 bytes) are removed from the stack and the error code is placed in the specified offset and segment. The returned parameter (RECCNT) is placed in the AX register as the returned function.

A.3 SORTFO ENTRY POINT

On entering at SORTFO, two segmented pointers must have been pushed on the stack. These represent the addresses of the command string descriptor and error code, in that order.

The command string segmented pointer must point to the first actual byte of the command string. The command string must be terminated with an up-arrow so that AUTOSORT/86 can determine the length.

At exit, the two segmented pointers (8 bytes) are removed from the stack and the error code is placed in the specified address. The returned parameter (RECCNT) is returned in the AX register.

SORTCO ENTRY POINT A.4

On entry at SORTCO, three nonsegmented pointers must be pushed on the stack. These represent the addresses of the command string descriptor, error code, and RECCNT, in that order.

All pointers are assumed to be offsets to the Data Segment register (DS) at entry. The command string must be terminated with an up-arrow, so that AUTOSORT/86 can determine the length.

At exit, the parameters ERROR CODE and RECCNT are placed in the specified addresses.

B

STAND-ALONE SORTING

To do a stand-alone sort/select, the following files must be available on one of the drives:

- SORTM.COM: This program requests some information and then loads and calls the sort module.
- AS86M.COM: The sort module.
- A previously created sort parameter file.

Follow these steps to run a sample stand-alone sort:

1. Copy SORTM.COM, AS86M.COM, TEST.DAT, and TEST.SRT from the distribution disk to a newly formatted disk in drive A.
2. Execute the sort caller by entering "SORTM" after the system prompt. A series of prompts appears.
3. When the sort program driver is requested, enter:

   a:

4. When the sort mode is requested, enter:

   0

5. When the user number is requested, enter:

   1

6. When the parameter filename is requested, enter:

   test

   The sort begins and will complete in 10 to 15 seconds.

The TEST.DAT file consists of 500 records of 25 bytes each. Table B-1 shows the fields contained in TEST.DAT.

Table B-1: TEST.DAT Field Description

START	LENGTH	DESCRIPTION
1	1	Constant "R"
2	2	2-character string
4	8	8-character string
12	2	Integer
14	4	Single-precision
18	8	Double-precision

The parameter file (TEST.SRT) specifies a sort on the string field, starting at byte position 4. The field is 8 bytes long. Mode 0 is selected for a full record sort. The output file is named SORTED.

Important Note: Since the record is less than 128 bytes long and the file was created using MS-BASIC, you may have enough null bytes at the end of the file for them to appear as additional null records. In an ascending sort, these null records migrate to the beginning of the file. To avoid this problem, a single byte field with a constant "R" is placed at the beginning of each record. Then, a select key that retains only those fields having an "R" at position 1 is defined in the parameter file.

If you have MS-BASIC, run the PRINT.BAS program to display the TEST.DAT and SORTED files. Here is how the parameter file TEST.SRT is created using PFG86M.COM.

First, "PFG86M" is entered after the system prompt. Then, the following answers are given in response to prompts:

PARAMETER FILE NAME:	**TEST**
INPUT FILE NAME:	**TEST.DAT**
OUTPUT FILE NAME:	**SORTED**
NO. OF RECS TO SKIP:	**0**
LOGICAL REC LENGTH:	**25**
CHANGE WORK DISK?	**N**
CHANGE OUTPUT DISK?	**N**
WORK FILE DRIVE:	**0**
KEY #1 START POSITION:	**4**
KEY #1 LENGTH:	**8**
KEY #1 ASCEND/DESCEND:	**A**
KEY #1 ALPHA/HEX/INTEGER/SGL/DBL:	**A**
KEY #2 START POSITION:	**0**
SEL KEY "OR" FUNCTION:	**N**
SELECT KEY #1	
DELETE OR RETAIN:	**R**
SEL FIELD START:	**1**
SEL FIELD LENGTH:	**1**
ALPHA/HEX/INTEGER/SGL/DBL:	**A**
LT,GT,EQUAL:	**E**
ACTUAL SELECT KEY:	**R**
SELECT KEY #2	
DELETE OR RETAIN:	**0**

When all of these prompts are answered, AUTOSORT/86 creates parameter file TEST.SRT.

INDEX

Alpha option, 1-3
AUTOSORT/86
  features, 1-1 to 1-3
  and programming languages, Chapter 5

Command strings, Chapter 2
  format, 2-1
  sample, 2-12 to 2-13
Control characters
  Alpha, 3-1
  Double-precision, 3-1
  Hex, 3-1
  Integer, 3-1
  Single-precision, 3-1

Disk change option, 1-3, 3-1

Error indications, 6-1 to 6-2

Mode parameters, 2-2 to 2-3
Modes, 1-1 to 1-2
  descriptions, 2-4 to 2-13
MS-BASIC Compiler, 5-3 to 5-5
MS-BASIC Interpreter, 5-1 to 5-3
MS-COBOL, 5-10 to 5-11
MS-FORTRAN, 5-9 to 5-10
MS-Pascal, 5-6 to 5-8

Parameter files, 2-3
  generator, 2-3
  overview, 3-1
  prompts, 3-2 to 3-4
Parameter string description, 2-10 to 2-11
Programming languages and AUTOSORT/86, Chapter 5

PUBLIC entry and access locations, A-1 to A-3

Record select features, Chapter 4

Select keys, 1-3, 4-1
Sort buffer size, 2-2 to 2-3
Sort parameters, 2-3, Chapter 3
  definition, 3-1
SORTCO, A-3
SORTFO, A-2
SORTMC, A-1
SORTPS, A-2
Stand-alone sort/select, Appendix B

Work files, 1-2 to 1-3

ALTERING CHARACTER SETS WITH EFONT

From Michael Wishnietsky's PUB Bulletin Board

#290 (of 290) JAN EWING, on 31-MAR-86 03:16
Subject: ALTERING CHARACTER SETS

Re: NIGHT OWL's message of March 9, 1986

Any of the 4200 byte character sets you asked about can be loaded with the extended characters from any other set by this process:

1. If you're using the 2.11 version of EFONT and you're not comfortable with the provisions for moving about in subdirectories, make sure the character sets you want to manipulate are both on the same directory (or disk) as EFONT.

2. Load EFONT, select the DISK OPTION (#5) from the menu, then load the special character set you're interested in (we'll call it SPECIAL.CHR).

3. Loading the character set will return you to the MAIN MENU. As soon as the set is loaded, select DISK (#5) again.

4. From DISK, select the HEADER OPTION (#4).

5. You'll note that one of the lines on the HEADER listing says "Number of Records (4 chars/record)" and next to that, highlighted in reverse video will be a number. It's likely that this number will be "0032." This number controls the number of characters you can load into a set. If it says "0032" then use the cursor keys to move the cursor down and type in "0064". Then return to the MAIN MENU (#1).

6. Once back at the main menu, hit the "+" key once or twice, until you have a blank space in the character set area. (Whether you hit it once or twice depends on whether the extended characters are already installed. If the number in the

header was 0032 when you began working, you'll only have to hit it once. This all assumes the number was 0032.

7. Again select DISK (#5) and load the second character set, the one from which you wish to take the control characters. I suggest you use AMERWP.CHR, although it probably wouldn't matter since the extended characters are usually the same. This isn't ALWAYS true, however, since the characters are easily manipulated and may have been at one time or another, sooooooooooo, to be on the safe side, use AMERWP.CHR. Taking the characters from it won't change it any.

8. You have now loaded AMERWP.CHR into your working area on the two blank pages after SPECIAL.CHR. BOTH sets are loaded. Your task, should you choose to accept it (a snap), is to move the second page of AMERWP.CHR into the second page position of SPECIAL.CHR.

9. In the upper right hand corner of the screen you'll see a line that says "CHARACTER SET NUMBER". This number will be "0" when you first load SPECIAL.CHR. It will increase each time you hit the "+" key.

10. Use the "-" key to go back to CHARACTER SET NUMBER 1 (this will be the 1st page of AMERWP.CHR). Use the cursor keys to go to the lower right hand corner of the working area.

11. Select the COPY OPTION (#3). The command line will change and you'll see an option called RANGE and an option called COPY. Press RANGE. Then use the "right-arrow" cursor key to move the cursor. It will take you to the next page. Continue pressing that key and move the cursor through the entire page (this will be CHARACTER SET NUMBER 2 now, the second page of AMERWP.CHR). Pressing RANGE selects the first character to be put into the copy buffer. As you move the cursor, each character it passes over will be added to the buffer.

12. Move the cursor until you again have a blank page. Then press RANGE again, and the buffer will be closed. The characters are inside just Waaaaaaaiting to be copied out into a new position.

13. Use the "+" or "-" key to go to CHARACTER SET NUMBER 0. Position the cursor at the lower right hand corner of the area. Press COPY. Voila!! The second page of AMERWP.CHR will appear magically over it's first page. This is SUPPOSED to happen.

14. You now have two pages of characters. CHARACTER SET NUMBER 0 is the SPECIAL.CHR you wanted to use, and CHARACTER SET NUMBER 1 is the page of control characters (the ones from 128 up) you wanted from AMERWP.CHR.

15. Beware, however, it must now be saved. Make SURE (by using the "+" and "-" keys) that CHARACTER SET NUMBER 0 is showing.

16. AGAIN, select DISK (#5). When the command line changes select SAVE. The screen should tell you that the name of the character set is the same as the SPECIAL.CHR set. If you want to use the same name, select the option that says NAME OK. If not, type in a new name. The program automatically adds ".CHR."

That's it. Exit EFONT and your new set is ready to use. It's about a tenth as complicated to do it as it appears to be from all these directions. It takes no time at all. The method will be obvious once you've done it once, and the command line makes it almost foolproof. Nothing is changed until you save your set, so you can play around without fouling anything up.

EFONT

Second VICTOR printing November, 1983.

ISBN 0-88182-101-2 Printed in U.S.A.

CONTENTS

1. Introduction 1-1

2. The Work Screen and the Main Menu 2-1

3. Functions Accessed Through the Main Menu
   - 3.1 Exit 3-1
   - 3.2 Cell Mode 3-1
   - 3.3 Copy 3-2
   - 3.4 Line Mode 3-3
   - 3.5 Disk Mode 3-4
   - 3.6 Width Mode 3-7

4. Editing Keys
   - 4.1 Cursor Keys 4-1
   - 4.2 Dot Edit Modes 4-2

5. Sample EFONT Editing Session 5-1

CHAPTERS

1. Introduction .....

2. The Work Screen and the Main Menu .....

3. Functions Accessed Through the Main Menu .....

4. Editing Keys .....

5. Sample EFONT Editing Session .....

1

INTRODUCTION

If you take a close look at your screen, you will see that each letter or character on the screen is made up of many small dots. The arrangement of these dots gives each character its distinctive shape. If you could change the arrangements of the dots, you could create letters and characters that are not found in any of the character sets supplied with your operating system. The EFONT font editor lets you create these custom characters.

With EFONT, you can change the appearance of the letters and characters that appear on the screen, or you can design your own character sets from scratch. You can load the characters that you create into your operating system so they appear onscreen as you work with an application or language program. (EFONT is especially useful with the graphics package, GRAFIX.) In addition, you can print your edited character set if you have a dot-matrix printer.

Before running EFONT, make sure that your operating system includes a standard keyboard and character set. (If you want to load your edited character set onto a program or application diskette, you also need to have a copy of the operating system configuration program.)

EFONT CAPABILITIES

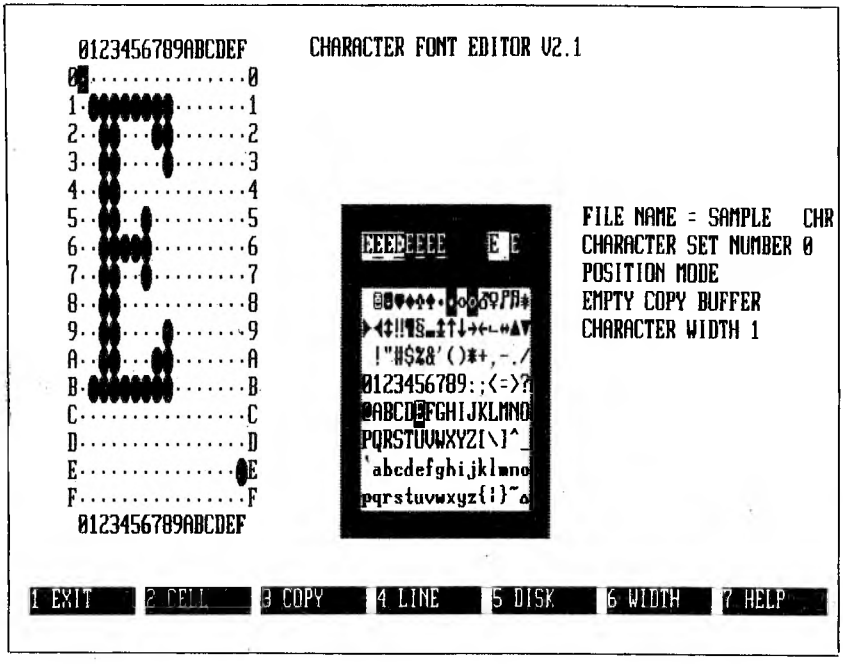
The EFONT font editor is a menu-driven program. The following menu options are available:

- Cell mode: Makes changes to the currently displayed character. The character can be rotated, mirrored, put into reverse video, or erased.
- Copy: Moves a range of characters from one place to another within a character set, or from one character set to another.
- Line mode: Controls the insertion and deletion of horizontal and vertical lines. Using this mode, you can change the horizontal size of a character or change that character's position within the character matrix display.
- Disk mode: Loads and stores your new or modified character sets onto a diskette. Disk mode is also used to edit the header. (The header contains important information used by the operating system configuration program.)
- Width mode: Constructs proportionally sized characters.
- Exit: Takes you from the Disk or other editing menu back to the Main menu. If you press the Exit key at the Main menu, you return to the operating system.

2

THE WORK SCREEN AND THE MAIN MENU

To start EFONT, boot your operating system. When the A > prompt appears on your screen, type **EFONT** and press Return. EFONT displays:



This display is the "work screen"—the place where you edit an existing character set or create a new one. The work screen has the following parts:

- Character cell—An enlarged view of the character being edited. It contains one of the two cursors present in the work screen. The character is displayed on a grid with 16 columns and 16 rows (for 256 possible positions). A period marks each spot where a row and a column intersect. If you put the cursor on one of these periods, you

can enter a dot that becomes part of the character you're editing. You can also remove a dot.

- Character set table—Consists of two parts: the character set display and the attribute field.

  The **character set display** shows the character set you are editing (character set 0, unless you change it). Each time you change the character set, the work screen and the characters in the character set display also change. The character set display also contains the other cursor present in the work screen. The position of this cursor determines which character is displayed in the character cell and in the attribute field.

  The **attribute field** shows the character pointed to by the cursor in the character set display. That character is shown in several ways: reverse video, underlined, half-intensity, and all combinations of the three. Two additional displays of the character being edited are at the right side of the attribute field: one in normal video and the other in reverse video. These displays are larger than the others, and are used to view characters larger than the normal text mode. (Characters up to 16 dots wide by 16 dots high can be displayed; the normal size is 10 by 16 dots.)

- Status area—Consists of the five lines of information you see at the center right of your screen. Each line tells you something about the current edit.

  The first line is the filename of the character set being edited; the second is the number of that character set; the third line shows the edit mode; the fourth line tells the status of the copy buffer; and the last line tells you the width of the character under edit (useful when making proportionally spaced character sets).

- The Main menu—The row of numbered functions at the bottom of the screen. The Main menu is the first one you see when you begin a session with EFONT; each of the other menus returns you to the Main menu.

  Each function on the Main menu is controlled by a like-numbered key at the top of your keyboard (the function keys). Each function displays its own menu when you press its function key.

3

FUNCTIONS ACCESSED THROUGH THE MAIN MENU

EXIT 3.1

The Exit key is at the left of the Main menu. If you press it, the screen clears and displays two new function keys:


```
NO     YES
```

If you press no, the Main menu reappears. If you press yes, you exit EFONT and return to the operating system.

CELL MODE 3.2

The Cell key is the second function key on the Main menu. It lets you alter the character displayed in the character cell. When you press the Cell key, this menu appears at the bottom of the screen:

```
MENU   REVERSE   MIRROR   ROTATE   CLEAR      HELP
```

The Menu key is an exit key; it returns you to the Main menu. (This is true for the Menu key on each of the menus.) The last function key on this (and any other) menu is a Help key. When you press it, the work screen disappears and you will see a screen of directions on how to use the function keys in the current menu.

The Reverse key reverses the display in the character cell—all of the "on" dots turn off, and all of the "off" dots turn on.

The Mirror key rotates the character in the character cell on its vertical axis, producing a "mirror image" of that character.

If you press the Rotate key, the character in the character cell rotates 90 degrees clockwise. You can use the Rotate function to create sideways character sets.

The Clear key erases the character in the character cell.

3.3 COPY

The third function key on the Main menu is the Copy key. It lets you move characters from one location to another. You can use Copy to move characters from one character set to another, for example, or within a character set to exchange the positions of the upper- and lowercase letters.

When you press Copy, the Main menu is replaced by the Copy menu:

```
MENU      RANGE      COPY                HELP
```

The Menu key returns you to the Main menu.

The Range key lets you identify a group of characters (range) that you want to copy to another location. To do this, you must define the starting character of the range and limit the range with an ending character.

Copy moves a range of characters from one place to another. To use Copy:

1. Define the range of characters to be moved. Mark the first character of the range with the Range key. Then mark the end of the range by moving the character set cursor to the character immediately after the last character in the range. Then press Range again. The range you've marked is highlighted with bright video. Make sure that it includes all the characters you want to move; if it doesn't, redefine the range.
2. After you define the range, the fourth line in the status area of the work screen tells you that the copy buffer is full. (The copy buffer holds up to 256 characters.) An error message appears if you try to move a larger range of characters.
3. When the range is correctly defined, move the character set cursor to the place to which you want to move the range of characters. Then, press Copy to copy your range to the new location.

The remaining key on the Copy menu displays Help information.

LINE MODE 3.4

The Line key controls the insertion and deletion of horizontal or vertical lines. It lets you change the horizontal size of a character or that character's position within the character cell. When you press the Line key, the Line menu appears:

```
MENU  INSERT H  DELETE H  INSERT V  DELETE V  HELP
```

The Menu key returns you to the Main menu.

The Insert H key scrolls the lines between the cursor location and the bottom of the character cell matrix down one position. At the same

time, it inserts a row of periods at the cursor location.

The Delete H key deletes the row of dots at the cursor position and scrolls up all the rows below the cursor position. At the same time, it adds a row of periods at the bottom of the character cell.

Insert V inserts a column of periods at the cursor position. All columns to the right of the cursor position scroll one position to the right. The rightmost column is erased.

The Delete V key erases the column of periods including the cursor position and all columns to the right of the cursor scroll left one position. At the same time, this key adds a column of periods at the right edge of the cell.

3.5 DISK MODE

The fifth key on the Main menu is the Disk key. It lets you load and store any character set you create onto diskette. The Disk key also lets you edit the header. When you press the Disk key, the Disk menu replaces the Main menu:

```
MENU    LOAD    SAVE    HEADER   CHDIR    HELP
```

As before, the Menu key returns you to the Main menu.

The Load key loads any existing character into the active character set shown on the work screen. To load a character set:

1. Return to the Main menu and choose a number for the character set you want to load. (To do this, press the plus (+) key or minus (−) key until the desired number appears in the status area of the work screen. These numbers range from 0 to 7; in most cases, 2 through 7 are empty.)

WARNING: Don't select character set 0 unless you actually want to replace the system character set.

2. Press the Disk key to call the list of filenames of the available character fonts.
3. Use the cursor movement keys (described in Chapter 4) to move the highlight to the name of the character set you want to load. Then, press the Load key. The new character set loads, and you return to the Main menu.

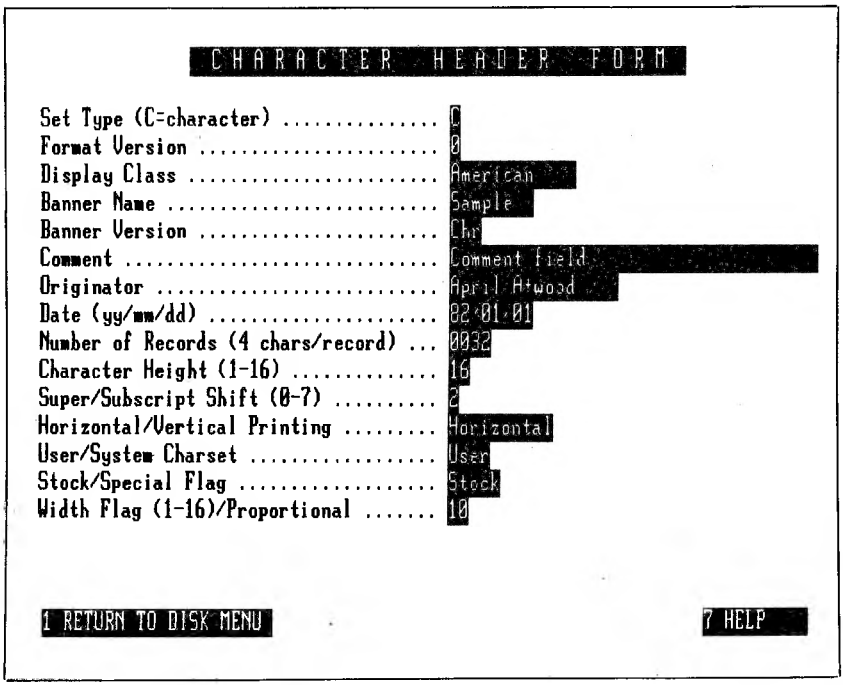
The Save key saves any new or edited character set on disk. If you press Save, the screen clears, displays the filename of the character set being edited, and then replaces the Disk menu with the following:


```
DISK      NAME OK                                HELP
```

If you want to change the filename of your character set, use the cursor keys to position the cursor in the filename field and type new characters. (The Backspace and Delete keys erase characters preceding the cursor.) When the name is correct, press the Name OK key to save the character set on the disk. When the set is saved, you return to the Disk menu.

If you don't want to save your character set, press the Disk key to return to the Disk menu.

The Header key allows you to edit the header block. (This block is saved at the same time you save a character set file. It contains important information for the system configuration program and the GRAFIX package.) Pressing the Header key clears the current screen display and replaces it with the header form and the Header menu:



There are three types of fields in the header form:

- The first nine are ASCII fields. They are changed in the same way as the filename fields in the Disk Save function described above.
- Three form increment fields contain numeric data. These are: Character Height, Super/Subscript Shift, and Width Flag. You can change the numbers in these fields by pressing any key.
- There are three toggle fields: Horizontal/Vertical Printing, User/System Charset, and Stock/Special Flag. When the cursor is in one of these fields, you can choose an option by pressing any key.

Use the up- and down-cursor keys if you want to move from one header block field to another. The cursor-left and cursor-right keys move from one character to another within an ASCII field.

The CH DIR key displays all the subdirectories under the currently active directory. Use this key to find a file (a character set file, for example) that you stored in another directory.

Press CH DIR, and use the cursor keys to highlight and load the subdirectory you want. To return, press CH DIR again, and select .. Repeat this procedure until you are back at the original directory.

Press Return To Disk Menu when all of the header fields are correct. The Help key displays a listing of the fields and their types.

WIDTH MODE 3.6

Use the Width key when you construct proportionally sized characters. The width of a character dictates how close the next character in a word can appear. If you press the Width key, a new menu appears:

```
MENU    AUTO        MANUAL                  HELP
```

The Menu and Help keys work as described earlier. The Auto function key tells EFONT to automatically set the width of the character under edit. If you press Manual, the width of the character under edit is increased. (The character width is listed in the fifth line in the status area of the screen. If you want proportional characters, be sure to set the width field in the header form to proportional.)

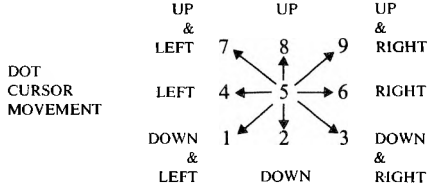
4

EDITING KEYS

CURSOR KEYS 4.1

The EFONT program has several cursor keys that operate in most edit modes.

- The cursor arrow keys (←, →, ↑, ↓) move the character set cursor to a new character.
- The number keys on the numeric pad move the cursor within the character cell. (The Help key on the Main menu shows you an example of how the number keys move this cursor.) The number keys are positioned around the 5 key in the same way that they move the cursor.



- The Return key and the Enter key (on the 10-key pad) are toggles for the dots in the character cell. Pressing one of these keys either makes a dot appear at a blank location or erases an existing dot.
- The Backspace key restores the character cell to its original condition. This key is a big help when you want to erase your changes and start the edit process over with the original character.

4.2 DOT EDIT MODES

The Dot Edit mode is displayed in the third line of the status area. You change the mode by using the Equals (=) key on the 10-key pad. The character set under edit is changed by using the Plus (+) key or the Minus (−) key on the 10-key pad.

EFONT has four edit modes:

- Position mode: This is the safest of the four modes because it is non-destructive—it lets you move the dot cursor around the character cell without changing any dots. The cursor keys function normally.
- Reset mode: Dots are turned off when the cursor passes over them, regardless of their current status.
- Toggle mode: Dots beneath the cursor change to the opposite status when you move the cursor: "On" dots are turned off; "off" dots are turned on.
- Set mode: Dots are turned on when the cursor passes over them, regardless of their current status.

5

SAMPLE EFONT EDITING SESSION

You are now familiar with the various menus, function keys, and editing keys that EFONT uses. In this chapter, you will use these menus and keys to create a character set. This example takes you through the process step by step. If you're unsure of any step, check the previous chapters for details.

1. Boot your operating system and type **EFONT** to load the font editing program.
2. When the work screen appears, press Disk at the Main menu to call the directory of character set filenames.
3. Use the cursor arrow keys to move the reverse video highlight over NORMAL.CHR.
4. At the Disk menu, press Load to load the Normal character set into EFONT. (The letters on your screen display remain the same, since the Normal character set is also used by the SAMPLE.CHR set automatically loaded when the work screen appears.)
5. For practice, copy character set 0 of the Normal font, rather than editing the original. Press Menu to return to the Main menu, then press Copy. The Copy menu appears at the bottom of your screen.
6. The character set cursor should be at the first character of character set 0. Press the Range key to set the beginning of the range at the first character.
7. Using the cursor arrow key, move the cursor over ALL the characters in the Normal character set. When you reach the last character, press the right-arrow key one more time. This puts the cursor on the first character in character set 1.

8. Press Range again to define the range of characters to copy. If you move the character set cursor back one space, you see all of character set 0 highlighted in bright video. The copy buffer line in the status area now reads "COPY BUFFER FULL."

9. Using the cursor arrow keys again, move the cursor to the upper left character in character set 1.

10. Press the Plus key on the 10-key pad. Character set 2 displays. (This set is blank.)

11. Press Copy. The character range you defined earlier (all of character set 0) appears in the character set display. Then, press Menu to return to the Main menu.

12. Use the cursor arrow keys to move the character set cursor to the uppercase A. The uppercase A appears in the character dot matrix.

13. Using the cursor movement keys and the Enter key on the 10-key pad, add or delete dots until you're satisfied with the new look of the uppercase A. When you have finished editing the character, go on to uppercase B, and then to each uppercase letter in the character set display.

14. When you have edited all the uppercase characters, press Save at the Disk menu to save your edited character set on the EFONT diskette.

15. Now, EFONT asks you to verify or change the name of your edited character set. Use the cursor arrow keys to move the cursor to the filename field, and enter the name NEWONE.CHR. Then, press Name OK to record the filename on the diskette.

16. If you press Disk, you can see that the filename NEWONE.CHR is now in the directory.

You have edited and saved a character set; now you must make it available for use with your system and application diskettes. To do this, you need an operating system configuration diskette (available from your dealer). Follow these instructions:

1. Put the EFONT diskette in drive A and the operating system configuration diskette in drive B.
2. Using the operating system's Copy command, copy the file NEWONE.CHR from drive A to drive B.
3. Remove the EFONT diskette, and move the system configuration diskette into drive A.
4. Re-boot your operating system. When the first system configuration menu appears, select "Generate a New Operating System" and press the Return key.
5. Select the appropriate keyboard at the Keyboard table, and press the Return key.
6. Answer Yes when the program asks if you want a second character set.
7. When the next display appears, choose "NEWONE" as your second character set. Then press the Return key.
8. Select the other elements of your operating system as you are asked for them.
9. When the "Current Configuration" display appears, accept the configuration as listed and press the Return key.
10. When the next screen appears, select "User Entered Filename" and press Return.
11. Enter NEWSYS and press the Return key.
12. Answer Yes when the program asks if you're sure you want to write your operating system to NEWSYS. Then press the Return key.
13. Insert a program or application diskette into drive B. When the BOOTCOPY program prompt appears on your screen, copy your operating system onto the diskette in drive B.

14. Your edited character set is now the second character set in your operating system; however, you need one more step before you can use that new character set. When you load your new operating system into your computer, press the Shift key and enter ALT-N while at the operating system level. This loads your edited character set into the operating system. If you want to use the other character set in the operating system, enter ALT-O without pressing the Shift key.

INDEX

Cell mode, 3-1
Character cell, 2-1
Character set, display of, 2-2
Character set table, 2-2
Character width, 3-7
Copy, 3-2 to 3-3
Cursor movement, 4-1

Disk mode, 3-4

Edit, status of, 2-2
Edit modes, 4-2
Exit, 3-1

Function keys at main menu
  cell mode, 3-1
  copy, 3-2 to 3-3
  disk mode, 3-4
  exit, 3-1
  line mode, 3-3

Header block, 3-5 to 3-6

Invoking EFONT, 2-1

Line mode, 3-3
Lines, insert and delete, 3-3 to 3-4
Load character set, 3-4 to 3-5

Move characters, 3-2 to 3-3

Proportional spacing, 2-2, 3-7

Sample editing session, 5-1 to 5-4
Save a character set, 3-5
Status area, 2-2

Work screen, 2-1

KEYGEN

First VICTOR printing February, 1983.
Second VICTOR printing December, 1983.

ISBN 0-88182-037-7 Printed in U.S.A.

CONTENTS

1. Introduction 1-1

2. KEYGEN Menus
   - 2.1 The Load Menu 2-2
     - 2.1.1 Keyboard Layout 2-3
     - 2.1.2 Character Set 2-4
     - 2.1.3 Keyboard Map 2-4
   - 2.2 Header Information 2-4

3. Creating a New Keyboard
   - 3.1 Assign Menu 3-2
   - 3.2 Assigning a Key 3-3
     - 3.2.1 Assigning from the Character Set 3-4
     - 3.2.2 Type In 3-5
     - 3.2.3 Assigning Attributes 3-5
     - 3.2.4 Assigning a Function 3-7

4. Completing KEYGEN
   - 4.1 Saving a Keyboard 4-1
   - 4.2 Using the New Keyboard 4-2

APPENDIXES

A: Creating a .KBD File A-1

B: Keyboard Considerations B-1

INDEX Index-1

FIGURES

2-1: KEYGEN Menus 2-1
2-2: Load Menu 2-2
2-3: Main Menu 2-5
2-4: Header Menu 2-6
3-1: Assign Menu 3-2
3-2: Alter Menu 3-3
3-3: Function Menu 3-7

TABLES

3-1: Attributes 3-6
3-2: Function Key Options 3-8
B-1: Keycode and Characters B-2

CHAPTERS

1. Introduction 1

2. KEYGEN Menus 2

3. Creating a New Keyboard 3

4. Completing KEYGEN 4

Appendix A. Creating a .KBD File ..... A

Appendix B. Keyboard Considerations ..... B

1

INTRODUCTION

KEYGEN is a program named for its "key generating" or customizing capacity. It lets you customize a keyboard to fit your needs. You can, for example, combine a particular keyboard display with a German character set to produce a German keyboard. You can move certain commands to more easily remembered keys, or configure a single key to enter a string of characters.

You can use KEYGEN to create several different keyboards and save them in files. Then, when you need a special configuration, simply load the appropriate file.

KEYGEN is a menu driven program, with displays that prompt you for necessary responses. And on-screen help is available from every menu.

Note: Printers vary in their ability to support different character sets. Check your printer manual to find out which sets it will print.

Using KEYGEN

Begin using the program by creating a KEYGEN disk. Copy the following files from your tool kit disks onto a formatted diskette or hard disk volume.

1. KEYGEN.EXE and KEYGEN.DAT from disk 2.
2. Files with the .KBD suffix (display files) from disk 3.
3. Files with the .KB suffix (keyboard files) from disk 3.
4. Files with the .CHR suffix (character files) from disks 2 and 3.

Next, log on to the drive or volume with the KEYGEN files. At the operating system prompt, type:

```
keygen
```

KEYGEN comes up and displays the first menu.

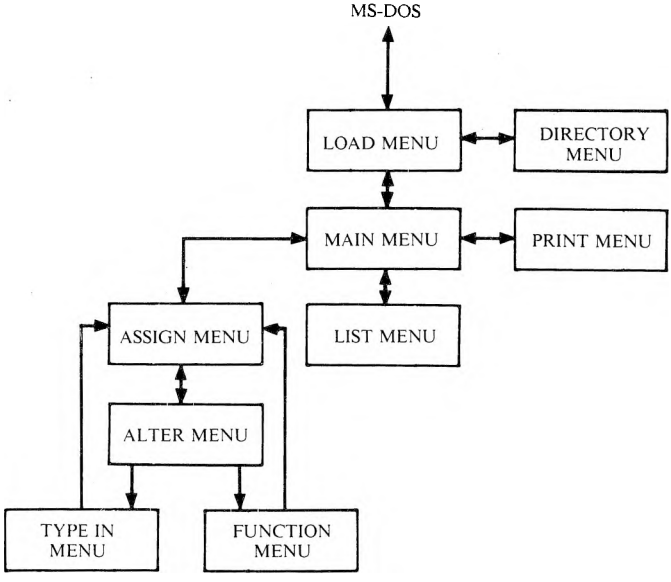
2

KEYGEN MENUS

KEYGEN is organized with a tree-like structure. When you first load the program, you are at the topmost level. From that level, you move down and branch out, depending on the options you choose.

Figure 2-1 illustrates the movement through KEYGEN menus.

Figure 2-1: KEYGEN Menus



At every level KEYGEN displays a function key menu which lists the options available at that point. Some of these options are the same from menu to menu, and others differ.

Function key 1, for example, moves you up through KEYGEN. If you press 1 EXIT from the first menu, you return to the operating system. If you press 1 MENU- in the succeeding levels, you return to the previous menu. Function key 7 HELP always displays on-screen information which explains the available choices.

The remaining function keys—2 through 6—perform tasks specific to the level you are on, and then take you to the next logical level.

2.1 THE LOAD MENU

To customize a keyboard, you must first load three components:

1. A keyboard layout (filename extension .KBD)
2. A character set (filename extension .CHR)
3. A keyboard map (filename extension .KB)

Use the Load Menu to choose your "starter set". This menu appears when you load KEYGEN.

Figure 2-2: Load Menu

A keyboard layout file (with the extension .KBD) contains the information to draw a diagram of a specific keyboard. KEYGEN supports a variety of keyboard structures; your *MS-DOS User's Guide* has a listing of all the available keyboards and character sets.

Use the cursor arrow keys to move through the keyboard layout files. Highlight the one you want and press 2 LOAD.

Changing Drives and Directories

If the file you are looking for is on another drive or volume, change drives by pressing 3 DRIVE.

KEYGEN prompts you for the drive or volume name to change to. Type in the letter of the drive you want. KEYGEN then displays the files in the current working directory of that drive (which is now the default drive for KEYGEN). Use the arrow keys to highlight the file you want and press 2 LOAD.

If the keyboard layout file you want is in another directory of the default drive, press 4 CH DIR. KEYGEN then displays the directories available in the current working directory (which is now the default directory). It also displays the current working directory name in the lower left corner of the screen. (If you are in a subdirectory, KEYGEN lists a directory named ".." which is the parent directory of the one you are in.)

Move the cursor to highlight the directory you want and press 2 ACCEPT. KEYGEN lists the appropriate files in the directory. Next, highlight the file you want, and press 2 LOAD.

KEYGEN allows you to keep up to 130 files of a single type in a directory. Your screen, however, displays only 45 files at a time. To see the rest of the files, you can use:

- 5 PAGE + to scroll to the next "page" (screen) of a long directory.
- 6 PAGE − to scroll back to a previous screen.

Note: It is a good idea to keep all your .KBD, .CHR, and .KB files in one directory. This technique allows you to stay in the same directory throughout a KEYGEN session.

2.1.2 CHARACTER SET

After you load a keyboard layout, KEYGEN lists the character set files (extension .CHR) available in the default directory. A character set file contains the font information that defines each character.

Notice the function key menu does not change. You can highlight and load one of the files on the screen, or change drives and/or directories to find the file you want. (You can, of course, use a file with a character set you designed yourself using EFONT, a program in the *Graphics Tool Kit*.)

2.1.3 KEYBOARD MAP

When you finish loading a character set, KEYGEN displays the keyboard map files (filename extension .KB) available in the default directory. A keyboard map file has the information that connects a specific key on your keyboard to a character or function.

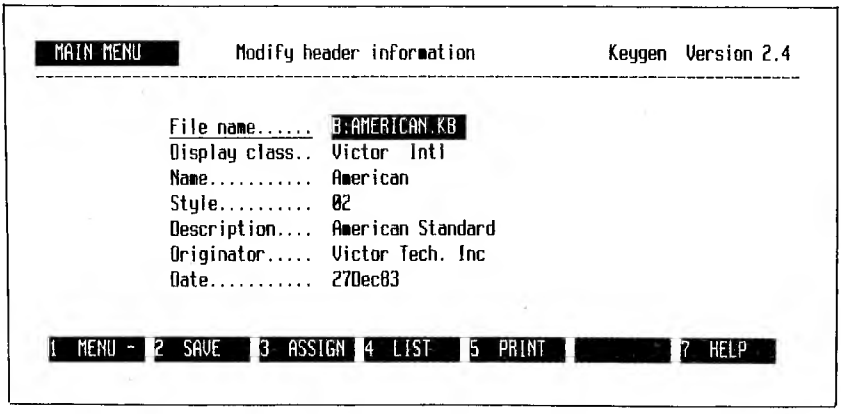
Again, you can highlight and load one of the displayed files or you can go to another drive or directory to find and load the one you want.

2.2 HEADER INFORMATION

Loading a keyboard map file takes you down to the next level in KEYGEN—to the Main Menu and the header information display.

You gain access to customizing options through the Main Menu. From this menu you can go down to the Assign Menu, or you can save your file and exit KEYGEN. First, though, use the Main Menu screen display to modify the header information.

Figure 2-3: Main Menu

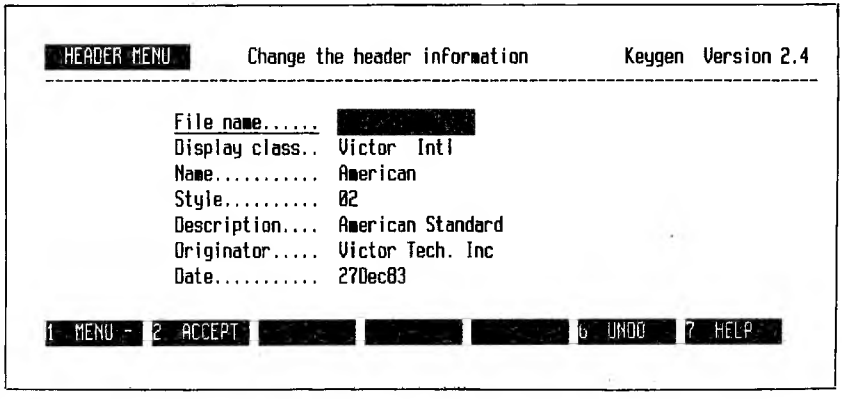


You see the header whenever you enter KEYGEN and go through the Load Menu. The header information should describe the keyboard you are about to create.

- Filename: the name you will use to load your keyboard. Use any name with a .KB extension.
- Display class: initially it is the keyboard display you loaded from the Load Menu; leave this as is or change it so that it is more descriptive to you.
- Name: your name.
- Style: version or style number of that particular keyboard; use it to differentiate between small changes in a particular keyboard.
- Description: what the keyboard does.
- Originator: source of software or design.
- Date: date of last modification.

To make header changes, first use the arrow keys to move the underline. When you reach a line you want to change, begin typing the new information. As soon as you begin typing, the Header Menu appears.

Figure 2-4: Header Menu



You can press:

- 2 ACCEPT to accept or enter a change.
- 6 UNDO to return to a previous entry. Use this key to correct a mistake. (You can also use the Backspace key to erase characters to the left of the cursor.)

Each time you press 2 ACCEPT you return to the Main Menu. You can repeat the process—move the underline and type a new entry—as many times as necessary. When you finish modifying your header, you are ready to move down to the Assign Menu and begin customizing your keyboard.

3

CREATING A NEW KEYBOARD

The Main Menu offers four choices (aside from MENU- and HELP):

- 2 SAVE saves the keyboard you created. You will use this function key later, after you finish assigning your keys.
- 3 ASSIGN calls up the Assign Menu.
- 4 LIST displays a listing of the keyboard map as it exists in the keyboard map file you loaded.
- 5 PRINT prints a listing of the keyboard map.

Both 4 LIST and 5 PRINT call up the List Menu. From this menu you can:

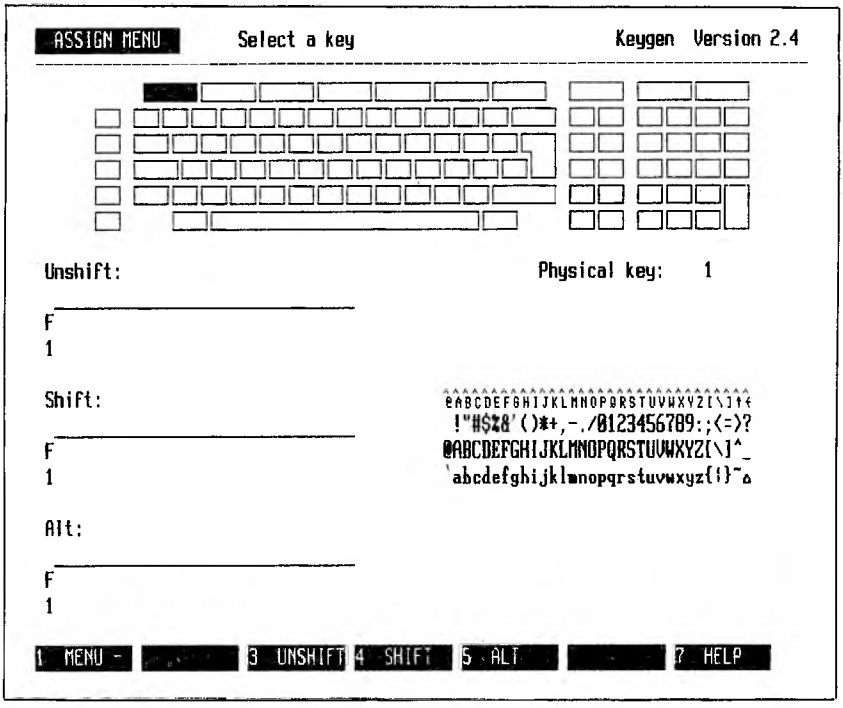
- 3 STOP
- 4 CONTINUE
- 5 PRINT

If you are displaying a list, 4 CONTINUE scrolls the list one screen at a time. 5 PRINT starts printing the list, beginning with the section currently on your screen. You can, therefore, print any part of a full listing. When you finish printing a list, KEYGEN takes you back to the Main Menu.

ASSIGN MENU 3.1

When you press 3 ASSIGN from the Main Menu, KEYGEN displays the Assign Menu.

Figure 3-1: Assign Menu



The screen displays the keyboard and the character set you loaded from the Load Menu. Below the keyboard you see various representations of keys, which correspond to the keyboard map you loaded. For each highlighted key, KEYGEN displays:

- The unshifted value—the character value is underlined and the ASCII value (hex) appears on the two lines below the character value.
- The shifted value—the character value is underlined and the ASCII value (hex) appears on the two lines below the character value.
- The ALT value—the character value is underlined and the ASCII value (hex) appears on the two lines below the character value.
- The attributes, such as lock, caps lock, repeat, and local.
- The physical key number (decimal).

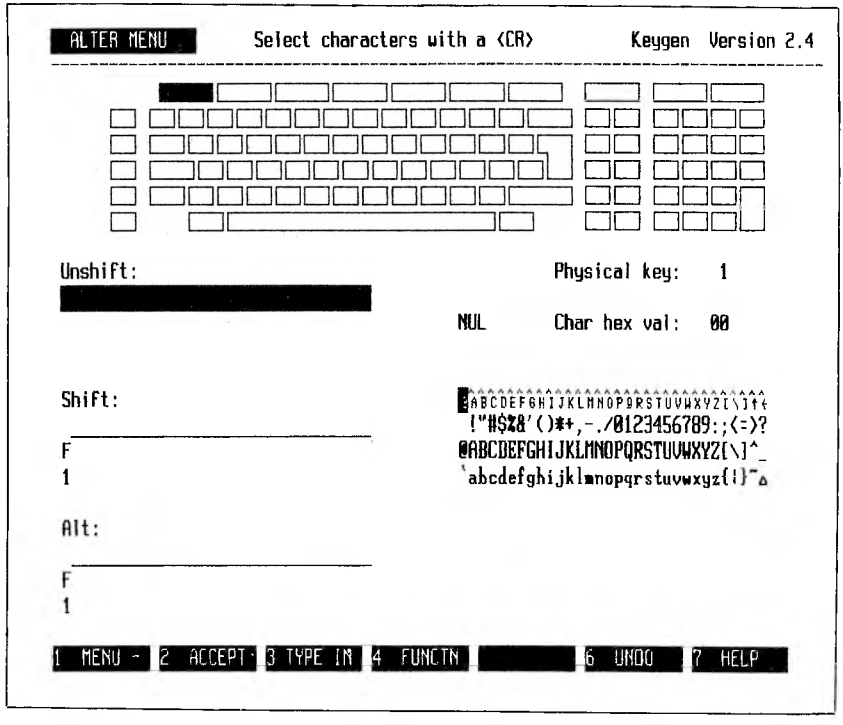
ASSIGNING A KEY 3.2

Begin a key assignment by using the direction keys to move through the keyboard until you find the key you want to change. Once your key is highlighted, choose a mode from:

- 3 UNSHIFT
- 4 SHIFT
- 5 ALT

Each of these choices brings up the Alter Menu.

Figure 3-2: Alter Menu



On this screen, a long highlight replaces the previous value for the chosen mode, and the cursor is in the character set. At the far right, the screen displays the hex value of the character pointed to by the cursor. If the cursor points to a control character, its name is displayed to the left of the hex value. In addition, a new menu appears at the bottom of the screen.

- 2 ACCEPT enters your selection and takes you back to the Assign Menu.
- 3 TYPE IN lets you type in your own character or string of characters. Pressing this key displays the Type In Menu.
- 4 FUNCTION lets you assign an attribute or a function to a key.
- 6 UNDO erases your last entry, and displays the previous value.

3.2.1 ASSIGNING FROM THE CHARACTER SET

To use the displayed character set to assign characters, first use the arrow keys to move through the character set display. When you have highlighted the character you want, press the Return key. The new character appears on the highlighted line, with the hex value below it.

You can assign up to 32 characters to each key; repeat the procedure until all the characters you want are in the long highlight. Then enter the assignment by pressing 2 ACCEPT.

KEYGEN then takes you to the Assign Menu. You can repeat the procedure—highlight a key, choose a mode, move through the character set and assign—until you configure the whole keyboard.

TYPE IN 3.2.2

KEYGEN offers a second method of assigning a string of characters to a key. This second method is, in most cases, the easiest to use. At the Assign menu, highlight a key on the keyboard. Then press 3 TYPE IN. KEYGEN displays the Type In Menu. You can then choose:

- 2 ACCEPT. Use this key to tell KEYGEN your assignment is complete.
- 3 CHARS. Use this key to tell KEYGEN you are making a key assignment in characters. You can assign up to 32 characters to each key.
- 4 HEX. Use this key to make an entry in hexadecimal. This number represents the character's ASCII value. For ASCII values below 10H you must type a leading zero. You can assign up to 32 characters to each key.
- 6 UNDO. Use this key to return to the previous value. (When entering hex values, you can also use the Backspace key to erase letters to the left of the cursor.)

Type in either the characters or hexadecimal values you want the key to represent. When you finish, enter the string by pressing 2 ACCEPT. KEYGEN takes you back to the Assign Menu. You can now work on another key, or assign an attribute to the same key.

ASSIGNING ATTRIBUTES 3.2.3

Four attributes can be assigned to keys. Table 3-1 describes the possible attributes.

Table 3-1: Attributes

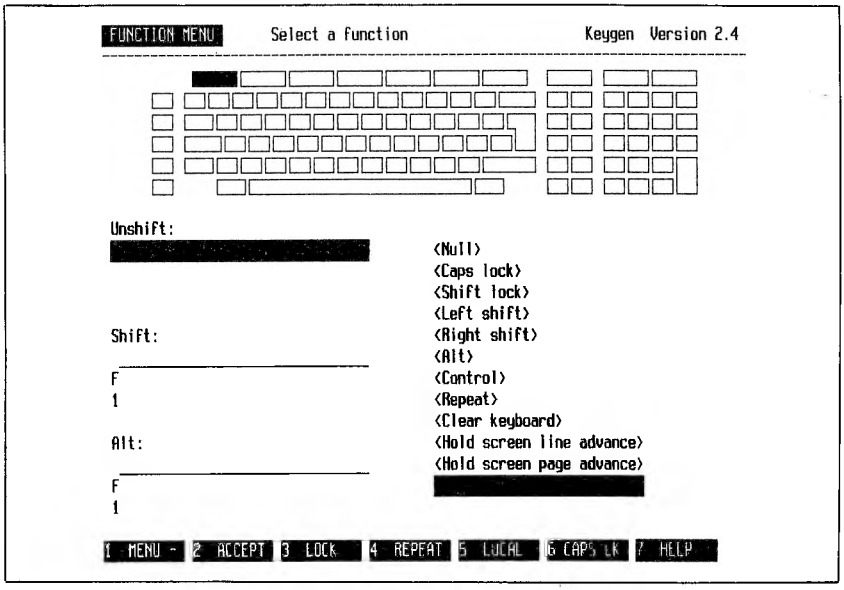
ATTRIBUTE	DESCRIPTION
Caps Lock	The Caps Lock attribute can be assigned to a key's unshifted mode. When this attribute is assigned to it, a key's unshifted mode is affected by the CAPS LOCK function key if you toggle the CAPS LOCK function key "on"; when you type the key's unshifted mode, you enter its shifted mode. For example, suppose you assign "e" and the Caps Lock attribute to the unshifted mode of physical key 33. You also assign "E" to the shifted mode of physical key 33. When the CAPS LOCK function key is "on", and you press physical key 33, you type an "E". (Standard keyboards usually assign the Caps Lock to letter keys only.)
Shift Lock	The Shift Lock attribute is the same as the Caps Lock attribute except that it is implemented when the SHIFT LOCK function key is toggled "on". (Standard keyboards usually assign the Shift Lock attribute to the regular typewriter keys.)
Repeat	Any mode of a key can be assigned the Repeat attribute. When this attribute is assigned to a mode of a key, the key (in that mode) repeats if you press it down for more than a half second.
Local	Any mode of a key can be assigned the Local attribute. When this attribute is assigned to a mode of a key, the key (in that mode) generates a code that is sent directly to the display interface, bypassing all application programs. A key mode with the Local attribute is implemented instantly and unconditionally. The screen brightness key, for example, alters the brightness without interfering with your program.

To assign an attribute, press 4 FUNCTION at the Alter Menu (after you have highlighted the key and selected the mode at the Assign Menu). The screen display indicates the existing attributes for the mode you have selected. The menu displays the attribute options. Add or delete an attribute by pressing the corresponding function key, which toggles the attribute on/off:

- ▶ 3 LOCK
- ▶ 4 REPEAT
- ▶ 5 LOCAL
- ▶ 6 CAPS LK

After you assign the attributes, return to the Assign Menu by pressing 1 MENU-. Do **not** press 2 ACCEPT. (You use 2 ACCEPT to make function key assignments.)

Figure 3-3: Function Menu



ASSIGNING A FUNCTION 3.2.4

You can assign a function rather than a character to a key. First highlight a key and select a mode, from the Assign Menu. Then press 4 FUNCTION. The right half of the screen displays the function key options. Use the arrow keys to highlight the function you want and press 2 ACCEPT.

After you press 2 ACCEPT, you return to the Assign Menu. Repeat the procedure until you have assigned all function keys you want to use.

Table 3-2 describes function key options.

Table 3-2: Function Key Options

FUNCTION	EFFECT
Null	Disables the selected mode of the key (i.e., the mode has no effect).
Caps Lock	This function key implements the Caps Lock attribute of the unshifted mode of a key; if you press the Caps Lock function key and a key with the Caps Lock attribute you enter the key's shifted mode. This function is usually assigned to the unshifted mode of key 54.
Shift Lock	This function key implements the Shift Lock attribute of the unshifted mode of a key; if you press the Shift Lock function key and a key with the Shift Lock attribute you enter the key's shifted mode. This function is usually assigned to the alternate mode of key 54.
Left Shift	When used with a second key, Left Shift implements the shifted mode of the second key. Left Shift is usually assigned to key 74.
Right Shift	When used with a second key, Right Shift implements the shifted mode of the second key. Right Shift is usually assigned to key 86.
ALT	This function key implements a key's alternate mode. On standard keyboards the control characters (00H to 1FH) are assigned to the alternate mode of keys that have characters 40H through 7FH assigned to their unshifted mode. For example, characters 40H and 00H are assigned to the unshifted and alternate modes of one key and characters 41H and 01H are assigned to the unshifted and alternate modes of another key.
Control	Reserved for further expansion.
Repeat	When used with a second key, the Repeat key causes that key to repeat.
Clear Keyboard	Empties the keyboard buffer so you can enter an immediate system interrupt.
Hold Screen Line Advance	Advances screen one page when in Page Advance "hold screen mode".
Hold Screen Page Advance	Advances screen one line when in Line Advance "hold screen mode".
No change at all	"No change at all" causes the existing function assignment to remain in effect. It is the default option when you display the Function Menu; if you display the Function Menu and press 2 ACCEPT, no function assignment is made unless you move the highlight from the "No change at all" option.

Note: Hold screen mode is described in "Display Driver Specifications" in the *MS-DOS User's Guide* (see ESC ^).

4

COMPLETING KEYGEN

SAVING A KEYBOARD 4.1

Once you finish configuring your keyboard, you must save it in a new keyboard file. This file will be stored under the name you gave it in the header on the Main Menu display.

All of the configuration procedures return you to the Assign Menu. To save a file, however, you must be at the Main Menu. Press 1 MENU- and move up one level to the Main Menu. From the Main Menu, press 2 SAVE. KEYGEN displays the message:


```
Saving.......
```

and then

```
File saved on disk.
```

You are ready to exit KEYGEN. Press 1 MENU- and 1 EXIT. The Reminder Menu asks if you are sure you want to leave KEYGEN. Press:

- 1 NO if you want to stay in the program and make further changes.
- 2 YES if you are finished. This key returns you to the operating system.

4.2 USING THE NEW KEYBOARD

To use your new keyboard, you must load it with MODCON, a utility described in the *MS-DOS User's Guide.*

Put the disk containing MODCON into a drive or copy it onto a hard disk volume; default to that drive or volume. Put your KEYGEN disk with the new keyboard file in the (second) floppy drive. Type:

modcon <new keyboard filename> <old keyboard filename> (cr)

MODCON loads the new keyboard, and saves the previous keyboard in a file called TEMP.KB.

Note: It is useful to copy the MODCON utility onto your KEYGEN disk; you can then easily load new keyboards from one drive or volume.

A

CREATING A .KBD FILE

KEYGEN uses a .KBD (keyboard display) file to draw a picture of a keyboard. Your *Applications Programmer's Tool Kit II* includes several different .KBD files. If you have a keyboard that is not represented by one of these files, you can create your own .KBD file.

To make a .KBD file, you need the following:

1. MAKE_KBD.EXE: This program (on your tool kit disk) combines two other files needed to construct a .KBD file.
2. KEYGEN.CHR: This file (on your tool kit disk) contains 32 control characters and a special set of graphics characters which are used to draw the keyboard display itself.
3. .KEY: You must write this file yourself. It is a list of key descriptions.

MAKING A .KEY FILE A.1

The .KEY file has the instruction set that KEYGEN.EXE uses to draw a keyboard and move through it. Two lines of data in the file correspond to each of the 104 logical keys on the keyboard. Data is ordered according to the logical key number, so that the entry for key 0 occupies the first two lines in the file, the entry for key 1 occupies the next two lines, and so on.

MAKE_KBD.EXE forms a .KBD file with KEYGEN.CHR and the .KEY file that you write. MAKE_KBD.EXE accepts data for 104 keys only. If your keyboard has more than that number, the additional keys are not included.

If a physical key has more than one logical key assigned to it, then the data for each of these logical keys is identical. For example, your computer has three logical keys which map together to form the carriage return key: 46, 66, and 87. Each key is listed in its chronological place in the .KEY file, and each has the same information.

You must enter the data in your .KEY file in the following order:

1. Address: KEYGEN uses window I/O, described in the "WINDOWIO" files on your tool kit disk. The window where the keyboards are drawn is "key_window". This window covers the entire width of the screen from the second line of the screen to the seventh.

   The address value you specify in .KEY is a single whole number that describes where in key_window you want the top left border of the key to begin. Because a screen line contains 80 columns (0–79), an address of 86 occurs in the sixth column of the second row of the window.

2. Four Neighboring Keys: To move through the keyboard map, KEYGEN must know which physical keys are to the left, right, above, and below the current key. List the logical key numbers of the keys above and below and to the left and right in that order.

   When a particular key has no neighbor in some direction, indicate this by entering an arbitrary negative number to stop the search for a neighbor. Or, enter the number of the key on the other side of the keyboard. This last method causes the cursor highlight to "wrap around" and begin again in the same row or column on the other side of the keyboard.

3. Newstart: When a key occupies a single row, enter an arbitrary negative number for this value. If a key occupies more than one row, however, use the address of the leftmost part of the second row for Newstart.

4. How Many Chars: Keys are drawn from a special graphics character set in KEYGEN.CHR. How Many Chars specifies the number of graphics characters needed to draw the key. If a key occupies two rows, specify the number needed to construct the top row of the key you are describing. MAKE_KBD.EXE will then compute (for itself) the number of characters occurring on the second row.

5. Appearance of the Key: KEYGEN.CHR has 24 graphics characters for drawing keys. 12 of these are the reverse-video equivalents of the other 12. Use the nonreverse-video characters, 0 through 11, to describe the construction of a key. Use EFONT (or KEYGEN) to look at these KEYGEN.CHR characters.

   Character sets are illustrated in the *MS-DOS User's Guide*. The 33rd character listed in the character sets is the first of these special graphics characters. This character is considered to be 0 for this purpose, and looks like "[". Determine the value of the other characters in the set by adding 1 to the value of the one to the left.

   List the values of the characters you need to "draw" the key, beginning at the left and moving right. If the key occupies more than one row, begin the description of the second row immediately after the description of the first row, on the same line in the .CHR file.

   End your description with an arbitrary negative number and a carriage return.

The following example is the Return key description in the EXAMPLE.KEY file. This key is described by logical key numbers 46, 66, and 87. Thus, this description occurs on lines 92–93, 132–133, and 154–155 of the EXAMPLE.KEY file.

```
290 45 47 25 86 368 4
3 7 7 4 0 1 9 8 8 6 -1
```

290 is the screen address which marks the upper left corner of the key. 45, 47, 25, and 86 are the keys to the left and right of, above, and below this key. The left corner of the second row of the key's picture begins at 368 in the window (Newstart).

The 4 describes how many special graphics characters are needed to draw the top row of the key. The first row takes the characters 3, 7, 7, and 4. The second row (not included in the 4) needs 0, 1, 9, 8, 8, and 6. The −1 indicates that the list is finished.

A.2 COMBINING THE FILES

Combine the files by running MAKE_KBD.EXE. At the system prompt, type:

```
make_kbd
```

Answer the next prompt with your .KEY filename. The program outputs a file called OUT.KBD. Rename this output file so that the new name is descriptive of the file, and end it with the extension .KBD.

B

KEYBOARD CONSIDERATIONS

This appendix gives you the character or function generated by each key position on the keyboards available with your computer. Table B-1 lists each key position in shifted, unshifted, and Alternate modes.

For each key position, Table B-1 lists one of the following:

- The actual character (letter, number, or symbol) generated by that key.
- The hexadecimal codes of the character generated by that key (such as F4).
- The function generated by that key (such as BKSP, DEL, ALT, and ESC).

Some listings include both the code and the character generated (such as F8 ° for key number 0).

For supplementary information, consult the following appendixes of the *MS-DOS 2.1 User's Guide*:

- Appendix A illustrates the physical keyboards available for your computer.
- Appendix B shows the key positions (logical key numbers) on the standard keyboards for the portable and desktop computers.
- Appendix C lists the hexadecimal code for each character in all the character sets.
- Appendix D contains a list of both the Escape sequences and Alternate sequences.

Table B-1: Keycodes and Characters

KEY POSITION	STAND	DOMESTIC	FRENCH	BRITISH	GERMAN	ITALIAN
0 unshifted	F8 °	F8 °	F8 !	F8 °	!	F8 °
0 shifted	F8 °	F8 °	F8 !	F8 °	!	F8 °
0 alternate	F8 °	F8 °	F8 !	F8 °	!	F8 °
1 unshifted	F1 ±	F1 ±	F1 ±	F1 ±	F1 ±	F1 ±
1 shifted	F1 ±	F1 ±	F1 ±	F1 ±	F1 ±	F1 ±
1 alternate	F1 ±	F1 ±	F1 ±	F1 ±	F1 ±	F1 ±
2 unshifted	F2 ≥	F2 ≥	F2 ≥	F2 ≥	F2 ≥	F2 ≥
2 shifted	F2 ≥	F2 ≥	F2 ≥	F2 ≥	F2 ≥	F2 ≥
2 alternate	F2 ≥	F2 ≥	F2 ≥	F2 ≥	F2 ≥	F2 ≥
3 unshifted	F3 ≤	F3 ≤	F3 ≤	F3 ≤	F3 ≤	F3 ≤
3 shifted	F3 ≤	F3 ≤	F3 ≤	F3 ≤	F3 ≤	F3 ≤
3 alternate	F3 ≤	F3 ≤	F3 ≤	F3 ≤	F3 ≤	F3 ≤
4 unshifted	F4	F4	F4	F4	F4	F4
4 shifted	F4	F4	F4	F4	F4	F4
4 alternate	F4	F4	F4	F4	F4	F4
5 unshifted	F5	F5	F5	F5	F5	F5
5 shifted	F5	F5	F5	F5	F5	F5
5 alternate	F5	F5	F5	F5	F5	F5
6 unshifted	F6 –	F6 –	F6 –	F6 –	F6 –	F6 –
6 shifted	F6 –	F6 –	F6 –	F6 –	F6 –	F6 –
6 alternate	F6 –	F6 –	F6 –	F6 –	F6 –	F6 –
7 unshifted	F7 ≅	F7 ≅	F7 ≅	F7 ≅	F7 ≅	F7 ≅
7 shifted	F7 ≅	F7 ≅	F7 ≅	F7 ≅	F7 ≅	F7 ≅
7 alternate	F7 ≅	F7 ≅	F7 ≅	F7 ≅	F7 ≅	F7 ~
8 unshifted	F8 °	F8 °	F8 !	F8 °	!	F8 °
8 shifted	F8 °	F8 °	F8 !	F8 °	!	F8 °
8 alternate	F8 °	F8 °	F8 !	F8 °	!	F8 °
9 unshifted	F9	F9	F9	F9	F9	F9
9 shifted	F9	F9	F9	F9	F9	F9
9 alternate	F9	F9	F9	F9	F9	F9
10 unshifted	FA	FA	FA	FA	FA	FA
10 shifted	FA	FA	FA	FA	FA	FA
10 alternate	FA	FA	FA	FA	FA	FA
11 unshifted	ESC	ESC H	ESC	ESC	ESC	ESC
11 shifted	ESC	ESC E	ESC	ESC	ESC	ESC
11 alternate	ESC	ESC E	ESC	ESC	ESC	ESC
12 unshifted	\	\	23 £	\	`	*
12 shifted	F8 °	F8 °	$	*Null*	`	A8 §
12 alternate	E3	E3	$	*Null*	`	A8 §
13 unshifted	!	!	&	!	!	9C `
13 shifted	!	!	1	!	!	!
13 alternate	!	!	1	!	!	!
14 unshifted	2	2	7B é	2	2	82 é
14 shifted	(a	(a	2	(a	"	2
14 alternate	<	<	2	<	"	<

B

KEY POSITION	STAND.	DOMESTIC	FRENCH	BRITISH	GERMAN	ITALIAN
15 unshifted	3	3	"	3	3	"
15 shifted	#	#	3	3	40 §	3
15 alternate	>	>	3	>	40 §	>
16 unshifted	4	4	'	4	4	'
16 shifted	$	$	4	23 £	$	4
16 alternate	F9	F9	4	23 £	$	#
17 unshifted	5	5	(	5	5	(
17 shifted	%	%	5	%	%	5
17 alternate	ALT-X	ALT-X	<	%	%	{
18 unshifted	6	6	5D §	6	6	_
18 shifted	9B ¢	9B ¢	6	$	&	6
18 alternate	AA	AA	6	$	&	[
19 unshifted	7	7	7D è	7	7	8A è
19 shifted	&	&	7	&	\	7
19 alternate	^	^	7	^	7	]
20 unshifted	8	8	!	8	8	^
20 shifted	*	*	8	*	(	8
20 alternate	`	`	*	`	<	`
21 unshifted	9	9	5C ç	9	9	87 ç
21 shifted	(	(	9	(	)	9
21 alternate	{	{	9	{	>	{
22 unshifted	0	0	40 à	0	0	85 à
22 shifted	)	)	0	)	=	0
22 alternate	}	}	0	}	=	}
23 unshifted	-	-	)	-	7E	)
23 shifted	_	_	5B °	_	?	F8 °
23 alternate	~	~	>	~	?	~
24 unshifted	=	=	-	=	#	-
24 shifted	+	+	_	+	'	+
24 alternate	\	\	–	\	`	\
25 unshifted	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H
25 shifted	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H
25 alternate	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H
26 unshifted	ESC H	DB	ESC H	ESC H	ESC H	ESC H
26 shifted	ESC E	CB	ESC E	ESC E	ESC E	ESC E
26 alternate	ESC z	DB	ESC z	ESC z	ESC z	ESC z
27 unshifted	DEL	DEL	DEL	DEL	DEL	DEL
27 shifted	DEL	CC	DEL	DEL	DEL	DEL
27 alternate	DEL	DC	DEL	DEL	DEL	DEL
28 unshifted	=	=	=	=	=	=
28 shifted	FD	FD	FD	FD	FD	FD
28 alternate	FD	=	FD	FD	FD	FD
29 unshifted	%	%	%	%	%	%
29 shifted	%	%	%	%	%	%
29 alternate	%	%	%	%	%	%
30 unshifted	/	/	/	/	/	/
30 shifted	/	/	/	/	/	/
30 alternate	/	/	/	/	/	/

KEY POSITION	STAND.	DOMESTIC	FRENCH	BRITISH	GERMAN	ITALIAN
31 unshifted	*	*	*	*	*	*
31 shifted	*	*	*	*	*	*
31 alternate	*	*	*	*	*	*
32 unshifted	ESC (	BD	ESC (	ESC (	ESC (	ESC (
32 shifted	ESC)	CD	ESC)	ESC)	ESC)	ESC)
32 alternate	ESC)	DD	ESC)	ESC)	ESC)	ESC)
33 unshifted	ALT-I	ALT-I	ALT-I	ALT-I	ALT-I	ALT-I
33 shifted	ALT-I	ALT-I	ALT-I	ALT-I	ALT-I	ALT-I
33 alternate	ALT-I	E9	ALT-I	ALT-I	ALT-I	ALT-I
34 unshifted	q	q	a	q	q	q
34 shifted	Q	Q	A	Q	Q	Q
34 alternate	ALT-Q	ALT-Q	ALT-A	ALT-Q	ALT-Q	ALT-Q
35 unshifted	w	w	z	w	w	z
35 shifted	W	W	Z	W	W	Z
35 alternate	ALT-W	ALT-W	ALT-Z	ALT-W	ALT-W	ALT-Z
36 unshifted	e	e	e	e	e	e
36 shifted	E	E	E	E	E	E
36 alternate	ALT-E	ALT-E	ALT-E	ALT-E	ALT-E	ALT-E
37 unshifted	r	r	r	r	r	r
37 shifted	R	R	R	R	R	R
37 alternate	ALT-R	ALT-R	ALT-R	ALT-R	ALT-R	ALT-R
38 unshifted	t	t	t	t	t	t
38 shifted	T	T	T	T	T	T
38 alternate	ALT-T	ALT-T	ALT-T	ALT-T	ALT-T	ALT-T
39 unshifted	y	y	y	y	z	y
39 shifted	Y	Y	Y	Y	Z	Y
39 alternate	ALT-Y	ALT-Y	ALT-Y	ALT-Y	ALT-Z	ALT-Y
40 unshifted	u	u	u	u	u	u
40 shifted	U	U	U	U	U	U
40 alternate	ALT-U	ALT-U	ALT-U	ALT-U	ALT-U	ALT-U
41 unshifted	ı	i	i	i	i	i
41 shifted	I	I	I	I	I	I
41 alternate	ALT-I	ALT-I	ALT-I	ALT-I	ALT-I	ALT-I
42 unshifted	o	o	o	o	o	o
42 shifted	O	O	O	O	O	O
42 alternate	ALT-O	ALT-O	ALT-O	ALT-O	ALT-O	ALT-O
43 unshifted	p	p	p	p	p	p
43 shifted	P	P	P	P	P	P
43 alternate	ALT-P	ALT-P	ALT-P	ALT-P	ALT-P	ALT-P
44 unshifted	AB	AB	^	*Null*	7D ü	8D ì
44 shifted	AC	AC	7E ¨	*Null*	5D Ü	=
44 alternate	AC	AC	7E ¨	*Null*	ALT-]	=
45 unshifted	]	]	~	]	+	$
45 shifted	[	[	~	[	*	&
45 alternate	ALT-]	ALT-]	~	ALT-]	*	$
46 unshifted	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
46 shifted	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
46 alternate	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*

KEY POSITION	STAND.	DOMESTIC	FRENCH	BRITISH	GERMAN	ITALIAN
47 unshifted	ESC @	BE	ESC @	ESC @	ESC @	ESC @
47 shifted	ESC L	CE	ESC L	ESC L	ESC L	ESC L
47 alternate	ESC O	DE	ESC O	ESC O	ESC O	ESC O
48 unshifted	ESC K	BF	ESC K	ESC K	ESC K	ESC K
48 shifted	ESC M	CF	ESC M	ESC M	ESC M	ESC M
48 alternate	ESC M	DF	ESC M	ESC M	ESC M	ESC M
49 unshifted	7	7	7	7	7	7
49 shifted	7	7	7	7	7	7
49 alternate	7	7	7	7	7	7
50 unshifted	8	8	8	8	8	8
50 shifted	8	8	8	8	8	8
50 alternate	8	8	8	8	8	8
51 unshifted	9	9	9	9	9	9
51 shifted	9	9	9	9	9	9
51 alternate	9	9	9	9	9	9
52 unshifted	-	-	-	-	-	-
52 shifted	-	-	-	-	-	-
52 alternate	-	-	-	-	-	-
53 unshifted	ESC p	ESC p	ESC p	ESC p	ESC p	ESC p
53 shifted	ESC q	ESC q	ESC q	ESC q	ESC q	ESC q
53 alternate	ESC q	ESC	ESC q	ESC q	ESC q	ESC q
54 unshifted	*Caps Lock*	*Caps Lock*	*Caps Lock*	*Caps Lock*	*Caps Lock*	*Caps Lock*
54 shifted	*Caps Lock*	*Caps Lock*	*Caps Lock*	*Caps Lock*	*Caps Lock*	*Caps Lock*
54 alternate	*Shift Lock*	*Shift Lock*	*Shift Lock*	*Shift Lock*	*Shift Lock*	*Shift Lock*
55 unshifted	a	a	q	a	a	a
55 shifted	A	A	Q	A	A	A
55 alternate	ALT-A	ALT-A	ALT-Q	ALT-A	ALT-A	ALT-A
56 unshifted	s	s	s	s	s	s
56 shifted	S	S	S	S	S	S
56 alternate	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S
57 unshifted	d	d	d	d	d	d
57 shifted	D	D	D	D	D	D
57 alternate	ALT-D	ALT-D	ALT-D	ALT-D	ALT-D	ALT-D
58 unshifted	f	f	f	f	f	f
58 shifted	F	F	F	F	F	F
58 alternate	ALT-F	ALT-F	ALT-F	ALT-F	ALT-F	ALT-F
59 unshifted	g	g	g	g	g	g
59 shifted	G	G	G	G	G	G
59 alternate	ALT-G	ALT-G	ALT-G	ALT-G	ALT-G	ALT-G
60 unshifted	h	h	h	h	h	h
60 shifted	H	H	H	H	H	H
60 alternate	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H	ALT-H
61 unshifted	j	j	j	j	j	j
61 shifted	J	J	J	J	J	J
61 alternate	ALT-J	ALT-J	ALT-J	ALT-J	ALT-J	ALT-J
62 unshifted	k	k	k	k	k	k
62 shifted	K	K	K	K	K	K
62 alternate	ALT-K	ALT-K	ALT-K	ALT-K	ALT-K	ALT-K

KEY POSITION	STAND.	DOMESTIC	FRENCH	BRITISH	GERMAN	ITALIAN
63 unshifted	]	]	]	]	]	]
63 shifted	L	L	L	L	L	L
63 alternate	ALT-L	ALT-L	ALT-L	ALT-L	ALT-L	ALT-L
64 unshifted	:	:	m	:	7C ö	m
64 shifted	:	:	M	:	5C Ö	M
64 alternate	:	ALT-^	ALT-M	:	ALT-\	ALT-M
65 unshifted	"	"	7C ù	"	7B ä	97 ù
65 shifted	"	"	%	"	5B Ä	%
65 alternate	"	"	%	"	ALT-{	%
66 unshifted	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
66 shifted	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
66 alternate	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
67 unshifted	ALT-J	E4	ALT-J	ALT-J	ALT-J	ALT-J
67 shifted	ESC I	E6	ESC I	ESC I	ESC I	ESC I
67 alternate	ESC xA	ESC xA	ESC xA	ESC xA	ESC xA	ESC xA
68 unshifted	ESC 8	E5	ESC 8	ESC 8	ESC 8	ESC z
68 shifted	ESC 9	E7	ESC 9	ESC 9	ESC 9	ESC z
68 alternate	ESC yA	ESC yA	ESC yA	ESC yA	ESC yA	ESC yA
69 unshifted	4	4	4	4	4	4
69 shifted	4	4	4	4	4	4
69 alternate	4	4	4	4	4	4
70 unshifted	5	5	5	5	5	5
70 shifted	5	5	5	5	5	5
70 alternate	5	5	5	5	5	5
71 unshifted	6	6	6	6	6	6
71 shifted	6	6	6	6	6	6
71 alternate	6	6	6	6	6	6
72 unshifted	+	+	+	+	+	+
72 shifted	+	+	+	+	+	+
72 alternate	+	+	+	+	+	+
73 unshifted	ESC 0	ESC 0	ESC 0	ESC 0	ESC 0	ESC 0
73 shifted	ESC 1	ESC 1	ESC 1	ESC 1	ESC 1	ESC 1
73 alternate	ESC 1	ESC 1	ESC 1	ESC 1	ESC 1	ESC 1
74 unshifted	*Left shift*	*Left shift*	*Left shift*	*Left shift*	*Left shift*	*Left shift*
74 shifted	*Left shift*	*Left shift*	*Left shift*	*Left shift*	*Left shift*	*Left shift*
74 alternate	*Left shift*	*Left shift*	*Left shift*	*Left shift*	*Left shift*	*Left shift*
75 unshifted	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
75 shifted	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
75 alternate	*Null*	*Null*	*Null*	*Null*	*Null*	*Null*
76 unshifted	z	z	w	z	y	w
76 shifted	Z	Z	W	Z	Y	W
76 alternate	ALT-Z	ALT-Z	ALT-W	ALT-Z	ALT-Y	ALT-W
77 unshifted	x	x	x	x	x	x
77 shifted	X	X	X	X	X	X
77 alternate	ALT-X	ALT-X	ALT-X	ALT-X	ALT-X	ALT-X
78 unshifted	c	c	c	c	c	c
78 shifted	C	C	C	C	C	C
78 alternate	ALT-C	ALT-C	ALT-C	ALT-C	ALT-C	ALT-C

KEY POSITION	STAND.	DOMESTIC	FRENCH	BRITISH	GERMAN	ITALIAN
79 unshifted	v	v	v	v	v	v
79 shifted	V	V	V	V	V	V
79 alternate	ALT-V	ALT-V	ALT-V	ALT-V	ALT-V	ALT-V
80 unshifted	b	b	b	b	b	b
80 shifted	B	B	B	B	B	B
80 alternate	ALT-B	ALT-B	ALT-B	ALT-B	ALT-B	ALT-B
81 unshifted	n	n	n	n	n	n
81 shifted	N	N	N	N	N	N
81 alternate	ALT-N	ALT-N	ALT-N	ALT-N	ALT-N	ALT-N
82 unshifted	m	m	,	m	m	,
82 shifted	M	M	?	M	M	?
82 alternate	ALT-M	ALT-M	?	ALT-M	ALT-M	?
83 unshifted	.	.	;	.	.	;
83 shifted	.	.	.	.	;	.
83 alternate	.	ALT-\	.	.	;	.
84 unshifted	.	.	;	.	.	;
84 shifted	.	.	/	.	;	/
84 alternate	.	ALT-]	/	.	;	/
85 unshifted	/	/	=	/	-	95 ò
85 shifted	?	?	+	?	_	!
86 alternate	?	?	+	?	_	!
86 unshifted	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*
86 shifted	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*
86 alternate	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*	*Rt. shift*
87 unshifted	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M
87 shifted	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M
87 alternate	ALT-M	ALT-J	ALT-M	ALT-M	ALT-M	ALT-M
88 unshifted	ESC A	ESC A	ESC A	ESC A	ESC A	ESC A
88 shifted	ESC A	ESC A	ESC A	ESC A	ESC A	ESC A
88 alternate	ESC xB	ESC xB	ESC xB	ESC xB	ESC xB	ESC xB
89 unshifted	ESC B	ESC B	ESC B	ESC B	ESC B	ESC B
89 shifted	ESC B	ESC B	ESC B	ESC B	ESC B	ESC B
89 alternate	ESC yB	ESC yB	ESC yB	ESC yB	ESC yB	ESC yB
90 unshifted	1	1	1	1	1	1
90 shifted	1	1	1	1	1	1
90 alternate	1	1	1	1	1	1
91 unshifted	2	2	2	2	2	2
91 shifted	2	2	2	2	2	2
91 alternate	2	2	2	2	2	2
92 unshifted	3	3	3	3	3	3
92 shifted	3	3	3	3	3	3
92 alternate	3	3	3	3	3	3
93 unshifted	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M
93 shifted	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M
93 alternate	ALT-M	ALT-J	ALT-M	ALT-M	ALT-M	ALT-M
94 unshifted	*Repeat*	*Repeat*	*Repeat*	*Repeat*	*Repeat*	*Repeat*
94 shifted	*Repeat*	*Repeat*	*Repeat*	*Repeat*	*Repeat*	*Repeat*
94 alternate	*Repeat*	*Repeat*	*Repeat*	*Repeat*	*Repeat*	*Repeat*

KEY POSITION	STAND.	DOMESTIC	FRENCH	BRITISH	GERMAN	ITALIAN
95 unshifted	*ALT*	*ALT*	*ALT*	*ALT*	*ALT*	*ALT*
95 shifted	*ALT*	*ALT*	*ALT*	*ALT*	*ALT*	*ALT*
95 alternate	*ALT*	*ALT*	*ALT*	*ALT*	*ALT*	*ALT*
96 unshifted	*Space*	*Space*	*Space*	*Space*	*Space*	*Space*
96 shifted	*Space*	*Space*	*Space*	*Space*	*Space*	*Space*
96 alternate	*Space*	*Space*	*Space*	*Space*	*Space*	*Space*
97 unshifted	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S
97 shifted	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S
97 alternate	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S	ALT-S
98 unshifted	ESC D	ESC D	ESC D	ESC D	ESC D	ESC D
98 shifted	ESC D	ESC D	ESC D	ESC D	ESC D	ESC D
98 alternate	ESC xC	ESC xC	ESC xC	ESC xC	ESC xC	ESC xC
99 unshifted	ESC C	ESC C	ESC C	ESC C	ESC C	ESC C
99 shifted	ESC C	ESC C	ESC C	ESC C	ESC C	ESC C
99 alternate	ESC yC	ESC yC	ESC yC	ESC yC	ESC yC	ESC yC
100 unshifted	0	0	0	0	0	0
100 shifted	0	0	0	0	0	0
100 alternate	0	0	0	0	0	0
101 unshifted	00	00	00	00	ALT-(a	00
101 shifted	00	00	00	00	ALT-(a	00
101 alternate	00	000	00	00	ALT-(a	00
102 unshifted	.	.	.	.	.	.
102 shifted	.	.	.	.	.	.
102 alternate	.	.	.	.	.	.
103 unshifted	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M
103 shifted	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M
103 alternate	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M	ALT-M

B

INDEX

..., 2-3
.CHR, 2-4
.KBD, 2-2, A-1
.KEY, A-1

ACCEPT, 2-3
ALT, 3-8
ALT value, 3-2
Alter Menu, 3-3
Assign Menu, 3-2
Attributes, 3-2, 3-5 to 3-6

Backspace, 3-5

Caps Lock, 3-6, 3-8
Character set, 2-4
Characters, assigning, 3-4
Clear keyboard, 3-8
Control, 3-8
Cursor arrow keys, 2-3

Directories, 2-3 to 2-4
Display class, 2-5
DRIVE, 2-3
Drives, 2-3 to 2-4

EFONT, 2-4
EXIT, 2-2

Filename, 2-5
Files, 2-3
Function, 3-7 to 3-8
Function Menu, 3-7

Header information, 2-4 to 2-6
Header Menu, 2-6
HELP, 2-2
Hexadecimal, 3-5
Hold screen, 3-8

Keyboard map, 2-4
KEYGEN disk, 1-1
KEYGEN.CHR, 1-1, A-1
KEYGEN.DAT, 1-1
KEYGEN.EXE, 1-1
KEYGEN.KB, 1-1
KEYGEN.KBD, 1-1
Keys, assigning, 3-3 to 3-4

Left Shift, 3-8
List Menu, 3-1
LOAD, 2-3
Local, 3-6

MAKE_KBD.EXE, A-1
Main Menu, 2-4, 3-1
Menus
  Alter, 3-3
  Assign, 3-2
  Function, 3-7
  Header, 2-6
  List, 3-1
  Load, 2-3
  Main, 2-4, 3-1
  Reminder, 4-1
MODCON, 4-2
Modes, 3-3

Null, 3-8

Physical key number, 3-2
Printers, 1-1

Reminder Menu, 4-1
Repeat, 3-6
Right Shift, 3-8

SAVE, 3-1
Saving, 4-1

Scroll, 2-3
Shift Lock, 3-6, 3-8
Shifted value, 3-2
Style, 2-5

TEMP.KB, 4-2
TYPE IN, 3-5

UNDO, 2-6
Unshifted value, 3-2

SYSGEN

Second VICTOR printing November, 1983.

ISBN 0-88182-088-1 Printed in U.S.A.

CONTENTS

1. Operating System Generation
   - 1.1 Introduction 1-1
   - 1.2 Using SYSGEN 1-1
   - 1.3 Selection Menus 1-3
     - 1.3.1 Keyboard Tables 1-3
     - 1.3.2 Character Set Selection 1-4
     - 1.3.3 Alternate Character Set 1-4
     - 1.3.4 Translation Tables 1-4
     - 1.3.5 Primary Printer Selection 1-5
     - 1.3.6 Secondary Printer 1-5
     - 1.3.7 Serial Port Configuration 1-6
     - 1.3.8 Banner Skeleton Selection 1-6
     - 1.3.9 Logo Selection 1-6
     - 1.3.10 Current Configuration 1-7
     - 1.3.11 Writing the Operating System Out 1-7

2. System Operation
   - 2.1 Program Files 2-1
   - 2.2 Batch Files 2-2
   - 2.3 System Selection Files 2-2
     - 2.3.1 Keyboard Table File 2-3
     - 2.3.2 Character Set Files 2-3
     - 2.3.3 Banner Skeleton File 2-4
     - 2.3.4 Logo Files 2-5
   - 2.4 Files Generated by SYSGEN 2-5
   - 2.5 Instruction Files 2-6

APPENDIXES

A. Character Sets A-1
B. SYSGEN Diskette Contents B-1

TABLES

2-1: System Selection File Extensions 2-3
2-2: Information Displayed by Character Set Tables 2-4

CHAPTERS

1. Operating System Generation 1

2. System Operation 2

Appendix A: Character Sets ..... A

Appendix B: SYSGEN Diskette Contents ..... B

1

OPERATING SYSTEM GENERATION

INTRODUCTION 1.1

SYSGEN is a system generation program that lets you generate a custom MS-DOS operating system. You can configure operating system components for Standard, International, British, French, Italian, and German variations (see Appendix A).

Configurable system components include character set, alternate character set, keyboard, logo and banner selection. User-configured I/O components include primary printer, secondary printer and serial communications port.

USING SYSGEN 1.2

The SYSGEN program is on diskettes 3 and 4 of your Programmer's Tool Kit. PTK 3 has the tool and batch files, and PTK 4 has the .OBJ files (see Appendix B).

Before you begin system selection and generation, use DISKCOPY to duplicate the system generation diskettes. Store the original system diskettes in a safe place. Next, make sure you have a formatted diskette or hard disk volume that you have set up with either HDSETUP or AUTOSET. If you need additional information, refer to your *Operator's Reference Guide.*

Load SYSGEN with these steps:

1. Boot up your system with MS-DOS.
2. Copy the MS-DOS files COMMAND.COM and CONFIG.SYS from your operating system diskette onto your PTK disk 3. SYSGEN will need these files to make the new system diskette.
3. Remove your operating system diskette and place the program disks on any two drives.
4. Default to the drive with PTK 3 and type **SYSGEN**.

SYSGEN signs on and prompts you for the date and time.

SYSGEN now begins displaying menus. The first menu gives you three choices:

- Generate a New Operating System
- Modify an Existing Operating System
- Help—Display Instructions

The "Generate a New Operating System" option takes you through the entire process of creating a new configuration. If you make an error during this process, the configuration acceptance display at the end of the program lets you correct it.

The "Modify an Existing Operating System" option lets you change an existing configuration. When you select an existing control file for modification, SYSGEN displays the configuration specified by that control file. You can then change any part of that configuration without going through an entire reselection process.

"Help—Display Instructions" is an on-line instruction display. It explains all the program options and how to select them.

Select a menu choice by pressing the space bar until the option you want is highlighted. Then press Return. After you enter your selection from the menu, the next menu appears. Repeat this procedure until you have completed all the menu options.

The next prompt asks you for the type of operating system you want (i.e., floppy disk, hard disk) and the location (drive numbers) of the PTK disks. Follow the onscreen example to answer this prompt.

The last prompt asks you which drive you want to copy your new system onto. Take out PTK 4, insert your formatted diskette, and type the drive number. Or, type the destination hard disk volume number. SYSGEN copies your customized operating system onto the specified drive.

NOTE: You can exit SYSGEN at any time by pressing ALT-C. This command returns you to the operating system.

SELECTION MENUS 1.3

The following options appear in menus. Press the space bar to move the highlight, and press Return to select the highlighted option.

KEYBOARD TABLES 1.3.1

The keyboard table defines the codes generated or keyboard functions executed when a key is pressed. Each selection is described by banner name, display class, descriptive comment, and filename.

- The banner name is the name of the character set (including version number) displayed in the banner. Character sets are illustrated in Appendix A.
- The display class describes the graphic subset (hex 21 through hex 7E) of the character set. It helps you avoid incompatible combinations of character set and keyboards. For example, the International display class defines hex 23 as the crosshatch (#), and the British class defines the same hex as the British monetary sign.

▶ The filename is the name of the file containing the character set.

The standard keyboard tables are provided on your configuration diskette. You can, however, generate your own keyboard tables using the KEYGEN utility (also available in your Programmer's Tool Kit).

1.3.2 CHARACTER SET SELECTION

This menu lets you select a character set. The Character Set Selection menu has the same format as the Keyboard Tables menu.

1.3.3 ALTERNATE CHARACTER SET

An alternate character set lets application programs display an entirely different character set of 128 or 256 characters. If you include an alternate character set in a configuration, you decrease the available memory by up to 8K bytes.

1.3.4 TRANSLATION TABLES

This option applies only to users with French keyboards.

PRIMARY PRINTER SELECTION 1.3.5

The Primary Printer Selection menu sets the name of the default printer of the logical device LST:. The choices are:

- Serial printer (UL1:)
- Parallel printer (LPT:)

You can change these values at run time.

If you select a serial printer, SYSGEN displays the menu for serial port configuration. Refer to the Serial Port Configuration menu for more details.

SECONDARY PRINTER 1.3.6

You can select a secondary printer as well as a primary printer. If you choose a serial printer as the secondary printer, the next menu you see is the Serial Port Configuration menu.

The primary and secondary printers cannot both be parallel printers because the system supports only one parallel port.

1.3.7 SERIAL PORT CONFIGURATION

This menu lets you set the baud rate, stop bits, and parity of the two serial ports. You can find information on these settings in the user menu for each device that you want to connect to a serial port.

The baud rate choices are 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2000, 2400, 3600 and 4800. The choices for stop bits are 1, 1.5 and 2.

The choices for parity are even, odd, and none. You can also set the parity bit with software for transmission; parity is not checked on incoming characters.

1.3.8 BANNER SKELETON SELECTION

The banner skeleton is a framework that holds the logo and configuration information displayed in the banner. For each configuration, the banner skeleton contains a different character set, keyboard, and other information.

1.3.9 LOGO SELECTION

The logo is a unique set of graphics characters that form the logo display. Normally, the logo is displayed as part of the banner when the operating system is loaded. If you've generated a system without logo characters, you must select a banner without a logo.

CURRENT CONFIGURATION 1.3.10

This menu displays the current configuration. The first selection in the upper box starts the process of writing the intermediate files onto disk. The second choice starts the selection process from the beginning. The items displayed in the lower box are the values of the current configuration.

When you select an item to modify, the menu for that item appears.

After you make the modification, the updated Current Configuration menu re-appears. If you choose "Accept the Current Configuration," SYSGEN displays the menu for writing the operating system out.

WRITING THE OPERATING SYSTEM OUT 1.3.11

This menu displays the current intermediate files. You can select an existing file, or you can enter a new filename by selecting the "User Entered File" option.

After you make your choice, SYSGEN asks you to confirm it. If you answer No, control returns to the Current Configuration menu. If you choose to write the operating system to the specified file, SYSGEN begins writing it; the job can take several minutes.

2

SYSTEM OPERATION

This chapter describes the operation of the system selection program. It includes:

- SYSGEN program files
- Batch files
- Selection files
- Files generated by SYSGEN

PROGRAM FILES 2.1

SYSGEN is a batch file that runs five other program files:

1. SYSELECT creates files for the customized parts of your operating system. This program makes the system component selections.
2. BIN2REL.EXE and LINK.EXE generate the operating system. BIN2REL.EXE converts binary image files to relocatable object module format files.
3. LINK.EXE combines component system files into an operating system. It collects the files SYSGEN selected and created and links them into a single file called MSDOS.EXE.
4. SYSLOC.EXE reformats the MSDOS.EXE file into the proper format for SYSCOPY.EXE. Invoke SYSLOC.EXE by typing **SYSLOC MSDOS .SYS**. (Note the space between MSDOS and .SYS.)
5. SYSCOPY.EXE writes the operating system to diskette or hard disk. It also copies the boot tracks onto the diskette or hard disk volume.

2.2 BATCH FILES

SYSGEN has six operating system generation batch files:

- 1.BAT (for a floppy disk system)
- 2.BAT (for a hard disk system)
- 3.BAT (for a floppy disk system with an IEEE driver)
- 4.BAT (for a hard disk system with an IEEE driver)
- 5.BAT (for the portable)
- 6.BAT (for the portable with an IEEE driver)

Each batch file runs the same five program files. The difference between the batch files is that each tells the linker to link different object files, depending on the configuration.

For example, if you have a hard disk system and you want to use the IEEE 488 driver in your operating system, choose the 4.BAT file (when SYSGEN prompts you for your operating system). This file tells the linker which object modules to link for your system.

2.3 SYSTEM SELECTION FILES

The SYSGEN diskette directory has information on keyboards, character sets, translation tables, banner skeletons and the logo. You can find these files by searching the directory for their file extensions.

Table 2-1: System Selection File Extensions

FILE TYPE	EXTENSION
Keyboard	.KB
Character set	.CHR
Banner skeleton	.BNR
Translation table	.XLT
Logo	.LGO

SYSGEN expects a particular format for each file type. Errors occur if other file types use any of the extensions in Table 2-1, or if you modify the format of a file type.

KEYBOARD TABLE FILE 2.3.1

Files with the .KB extension are keyboard table files. These files contain information on the keyboard code sent to the processor when a key is pressed. The files also have information on the keyboard table name, version number, origin, date of origin, and display class displayed by the system selection program.

CHARACTER SET FILES 2.3.2

Files with the .CHR extension contain character set tables. These tables contain data corresponding to the dot matrix displayed by each character on the keyboard. The tables also contain information on the character set name, version number, origin and date of origin, and display class.

SYSGEN displays most of this information to help you select the correct character set. Table 2-2 shows the format of the information in the first sector of these files.

The banner name and version are the name and version number of the character set placed in the banner.

Table 2-2: Information Displayed by Character Set Tables

TYPE OF INFORMATION	LENGTH (BYTES)
File type (K = keyboard; C = character)	1
Format version	1
Display class	12
Banner name	8
Filler (a space)	1
Comment	35*
Originator**	16
Date (yy/mm/dd)**	8
Length**	4
Unused**	51

* 31 bytes are displayed by SYSGEN
** Not displayed by SYSGEN

2.3.3 BANNER SKELETON FILE

Files with the .BAN extension are banner skeleton files. The banner is information displayed during system boot, including the logo and configuration information. The banner skeleton is a set of ASCII strings containing the escape sequences and characters needed to display the logo and configuration information.

SYSGEN displays the banner files available for selection. After you specify the keyboards and character sets, SYSGEN makes a copy of the banner. The first sector of the banner specifies the chosen keyboards and character sets.

When SYSGEN writes a custom banner, the first sector has this format: The first byte is zero, followed by 0DH 0AH. Next you see the length of the file in decimal (with a leading and a trailing space), followed by 0DH 0AH.

If the first byte is not zero, SYSGEN does not customize the banner. If data in the first sector is not "recognizable," SYSGEN uses default locations during custom banner generation.

The location of the keyboard name and character set name follow the same format as the file length. If the file length is 639 characters, the keyboard name is at byte 502, and the character set name is at 541. The first 24 bytes of the banner file are shown below (in hex):

```
00 0D 0A 20 36 33 39 20 0D 0A S0 35 30 20 0D
0A 20 35 34 31 20 0D 0A
```

LOGO FILES 2.3.4

Like the character set, the logo contains data that corresponds to a set of special characters. These characters represent the set of dots in the logo. If the size of the logo is nonstandard, the first byte must contain its length in sectors. SYSGEN supports sixteen-sector logo files.

2.4 FILES GENERATED BY SYSGEN

**.CTL Files*

The primary output of SYSGEN is a control file containing the specifications of the operating system. Use the "Modify an Existing Operating System" option to modify existing control files.

**.SPR Files*

SYSGEN generates an *.SPR file for each operating system you select. This file contains system parameter data to be loaded into the operating system.

**.BNR Files*

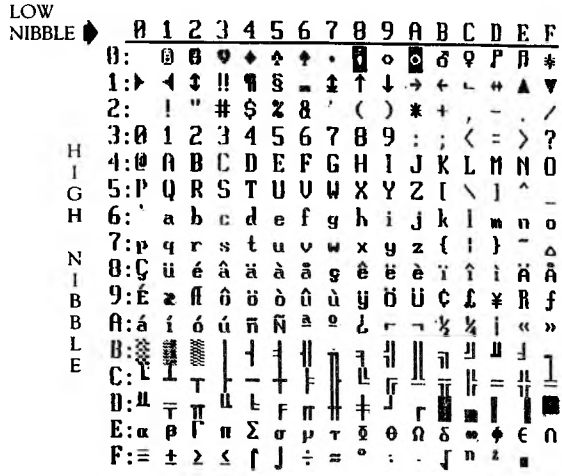
SYSGEN generates a *.BNR file (banner file) each time you select an operating system. This file is a customized version of the selected banner skeleton file.

2.5 INSTRUCTION FILES

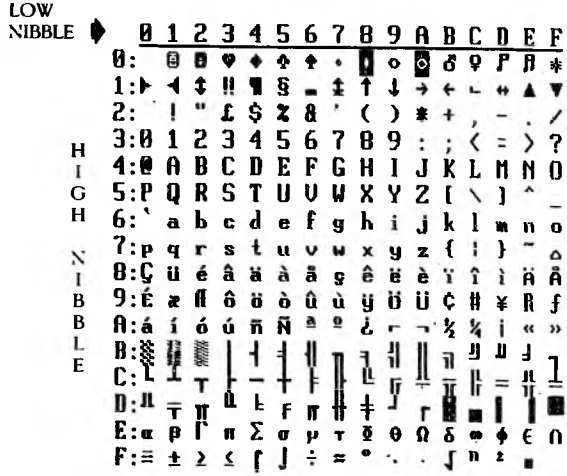
The file in the SYSELECT.HLP program contains information that tells you how to use SYSGEN.

CHARACTER SETS

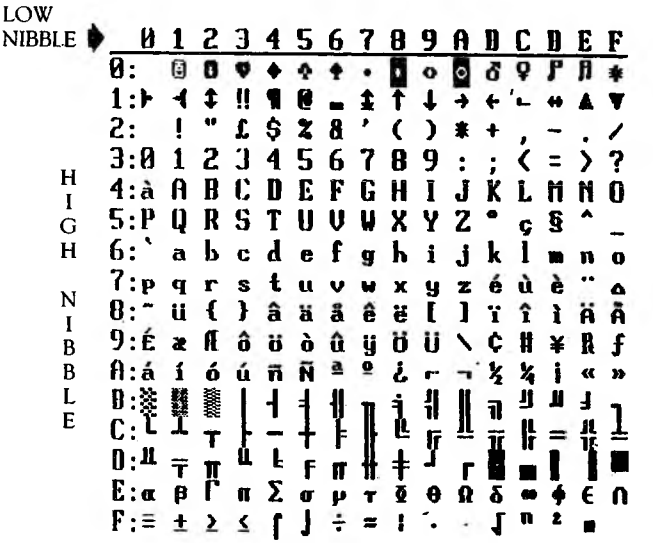
International Character Set



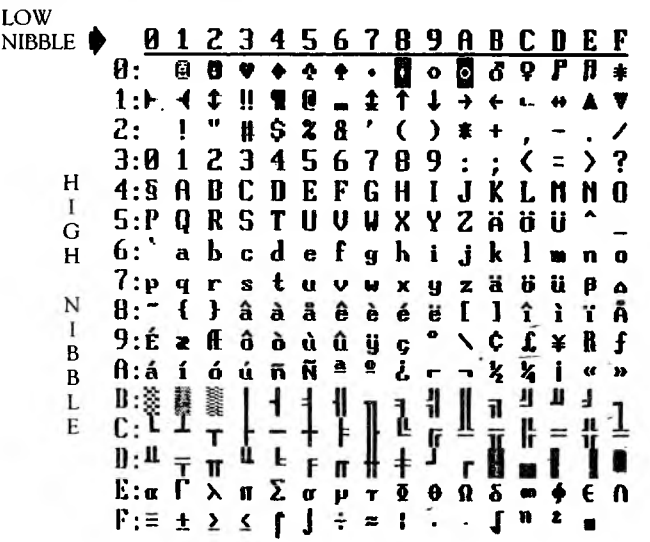
British Standard Character Set



French Standard Character Set



German Standard Character Set



B

SYSGEN DISKETTE CONTENTS

The files and programs described in this appendix are contained on the SYSGEN diskettes.

SYSGEN.BAT	Boot up batch file
PROMPT	SYSGEN prompt
MAKEMS.BAT	MS-DOS MAKE batch file
1.BAT 2.BAT 3.BAT 4.BAT 5.BAT 6.BAT	Configuration batch files
SYSELECT.EXE	Syselect program
SYSGEN.HLP	SYSGEN help
BIN2REL.EXE	BINary to RELocatable Converter
LINK.EXE	MS Linker
SYSLOC.EXE	System Locator
SYSCOPY.EXE	System copy utility
AMER02.KB FRENCH01.KB GERM03.KB ITALIN02.KB VICTOR.KB	Keyboard files

GERMAN.BAN ITALIAN.BAN VICNOLGO.BAN FRENCH.BAN VICTOR.BAN	Banner files
ITALIN02.CHR FRENCH01.CHR GERM02.CHR VINTL01.CHR	Character set files
FRENCH01.XLT NOTRANS.XLT	Translate tables
VICTOR.LGO	Logo
VICTOR.BNR VICTOR.CTL VICTOR.SPR	SYSGEN configuration information for diskette
BANNER.OBJ CHARSET.OBJ KEYS.OBJ SYSPAR.OBJ XLATE.OBJ	SYSGEN generated objects

INDEX

.BNR, 2-3
.CHR, 2-3
.KB, 2-3
.LGO, 2-3
.OBJ files, 1-1
.XLT, 2-3

ALT-C, 1-3

Banner, 1-6
  name, 1-3
Batch files, 1-1
Baud rate, 1-6

Character set, 1-3
  alternate, 1-4
  selection, 1-4
COMMAND.COM, 1-2
CONFIG.SYS, 1-2

Files
  *.BNR, 2-6
  *.CTL, 2-5
  *.SPR, 2-6
  banner, 2-4 to 2-5
  character set, 2-3 to 2-4, A-1
  keyboard table, 2-3
  logo, 2-5
  system selection, 2-2 to 2-3

Graphic subset, 1-3

Help, 1-2
  instruction files, 2-6

Keyboard tables, 1-3 to 1-4

Loading, 1-2
Logo, 1-6

Menus, 1-2, 1-3
Modify, 1-2

New Operating System, 1-2

Operating System, 1-2
  writing out, 1-7

Parity, 1-6
Printer
  parallel, 1-5
  primary, 1-5
  secondary, 1-5
  serial, 1-5

Selection menus, 1-3 to 1-7
Serial port configuration, 1-6
Stop bits, 1-6

Translation tables, 1-4

MODCON Utility

VICTOR Publications
380 El Pueblo Road
Scotts Valley, CA 95066
(408) 438-6680

First VICTOR printing February, 1983.

ISBN 0-88182-006-7 Printed in U.S.A.

CONTENTS

1. Overview 1-1

2. Operating Environment 2-1
   2.1 Character Sets, Keyboard Tables, and Translate Tables . 2-1
   2.2 MODCON Operation 2-2
   2.3 Command Syntax 2-2
   2.4 Error Conditions 2-3

3. Using MODCON 3-1
   3.1 Commands 3-1
   3.2 Applications 3-2

CHAPTERS

1. Overview 1

2. Operating Environment 2

3. Using MODCON 3

1

OVERVIEW

MODCON is designed to take advantage of one of the most powerful features of your computer—its flexibility. With MODCON, you can modify the configuration of the operating system with respect to the keyboard table, character set, and translate table. When used with EFONT and KEYGEN, MODCON provides you with a personalized environment not possible on other machines.

MODCON lets you select a new keyboard table, translate table, and/or character set before entering an application program. The current set(s) can be saved and restored when you exit the application program.

MODCON is supported by the following operating systems:

- MS-DOS: V1.25/2.5 or later
- CP/M: V1.1/2.4 and later

If you try to use MODCON with earlier versions of either operating system, an error message appears.

Note: Translate table functionality is available with MS-DOS V1.25/2.6 and later.

2

OPERATING ENVIRONMENT

Before you can use MODCON, you must set up any keyboard and character files and translate tables.

CHARACTER SETS, KEYBOARD TABLES, AND TRANSLATE TABLES 2.1

Character sets are obtained by:

- Selecting a set provided on the system selection (SYSELECT) diskette included in the Programmer's Tool Kit.
- Selecting a graphics character set provided with CHARGRAF in the Graphics Tool Kit.
- Modifying the current set (or any available set) using the EFONT character-font editor.

Keyboard tables are obtained by:

- Selecting a table provided on the system selection (SYSELECT) diskette included in the Programmer's Tool Kit.
- Modifying the current table (or any available table) using the keyboard table editor KEYGEN.

Translate tables are associated with character sets requiring dead key sequences. These are provided for each appropriate language on the SYSELECT diskette.

2.2 MODCON OPERATION

If you request it, MODCON saves the current keyboard table or translate table and/or character set in a file on the drive you specify. The keyboard file and translate table is 2K bytes and the largest character set file is 10K bytes. All header fields are initialized with blanks except for the following:

- TYPE is K (keyboard), C (character).
- VERSION is 0.
- BANNER NAME is the file name you specify.
- FILE SECTOR COUNT is the appropriate value.

Translate table header fields are set to nulls.

Any valid keyboard or character file created by KEYGEN, EFONT or MODCON—or taken from a system selection, graphics or other diskette—can be loaded from a file and set as the active keyboard table or character set.

Translate tables are processed only when a character set is being processed. If there is a translate table with the same name as the character set you have selected, that translate table is automatically included in your configuration. If no such translate table exists, a translate table is not included.

When you save a character set, any active translate table is automatically saved on the same diskette. The translate table has the same file name as the character set file, and the extension .XLT.

2.3 COMMAND SYNTAX

To invoke MODCON, type:

MODCON <command>

The command portion of the invocation can have any of these formats (items enclosed in brackets are optional):

<source file>[.<source ext.>]<save file>[.<save ext.>]
<source file>[.<source ext.>]
***<save file>[.<save ext.>]**

where:

<source file> is the name of the file(s) that contain the sets to be made active. If you enter an asterisk (*), no new sets are made active and the configuration remains unchanged.

<source ext.> is .KB for keyboard or .CHR for character set. If the extension is omitted, then both keyboard and character files are made active.

<save file> is the name of the file(s) used to save the currently active keyboard table and/or character set. If you use MODCON to set a new configuration, <save file> is optional.

<save ext.> is .KB for keyboard or .CHR for character set. If the extension is omitted, then both keyboard and character sets are saved.

NOTES: (1) The <source ext.> option is independent from the <save ext.> option.
(2) Translate tables can be acted on only if a character set is being processed.

ERROR CONDITIONS 2.4

Any of the following errors cause MODCON to terminate prematurely. When this happens, none of your new configuration is saved, and the operating system is unchanged. Any BDOS or BIOS errors are returned in the normal manner.

CANNOT OPEN FILE <filename>

Make sure the file exists on the specified drive.

DISK FULL

Make room for the new file(s). Keyboard files and translate tables need 2K bytes; a character set file needs 10K.

DIRECTORY FULL

Make room for one or two new entries.

INVALID FILE EXTENSION

Specify the proper extension (.KB or .CHR).

INVALID DELIMITER

The correct delimiter is an asterisk (*).

SYSTEM ERROR

Run system diagnostics.

OPERATING SYSTEM MISMATCH

Reboot with the correct version of the operating system.

DISK ERROR <filename>

Use a different diskette.

3

USING MODCON

This chapter shows how MODCON is used both at the command level and to create a prepackaged set of programs intended for end-users.

COMMANDS 3.1

The following examples show how commands are used:

- **MODCON GERM01 G02SAVE**
  This saves the current keyboard table and character set in files G02SAVE.KB and G02SAVE.CHR on the default drive. The data in GERM01.KB and GERM01.CHR become the new active sets.

- **MODCON M01 GERM01**
  This does the same as the previous example except that the original GERM01.KB and .CHR files are overwritten in the save operation.

- **MODCON AUST01**
  This sets the new keyboard table and character sets, both named AUST01. The previous keyboard table and character sets are not saved.

- **MODCON B:GRAPHIC.CHR SAVE.CHR**
  This saves the current character set in SAVE.CHR on the default drive. The new active character set is GRAPHIC.CHR on drive B. The keyboard tables are not changed.

- **MODCON * BRIT01.KB**
  This saves the current keyboard table in file BRIT01.KB on the default drive while leaving it as the active keyboard. The character set is not changed.

- **MODCON FRENCH.KB B:SWEDISH.CHR**
  This saves the current character set in file SWEDISH.CHR on drive B, while leaving that character set active. It also sets the keyboard table from the default drive file FRENCH.KB, and overwrites the existing keyboard table without saving it.

3.2 APPLICATIONS

The following example shows how a series of commands in a batch file (MS-DOS) or submit file (CP/M-86) set up a dedicated keyboard for a WordStar word processing session. The original keyboard is restored when WordStar is terminated.

MODCON WORDAMER.KB SAVED.KB	The American keyboard is set as the dedicated keyboard for the WordStar session. The original keyboard is saved so it can be restored at the end of the session.
WS	WordStar is invoked.
MODCON SAVED.KB	The original keyboard is restored.
DEL SAVED.KB	This deletes the original keyboard file (MS-DOS).
[ERA SAVED.KB]	This deletes the original keyboard (CP/M-86).