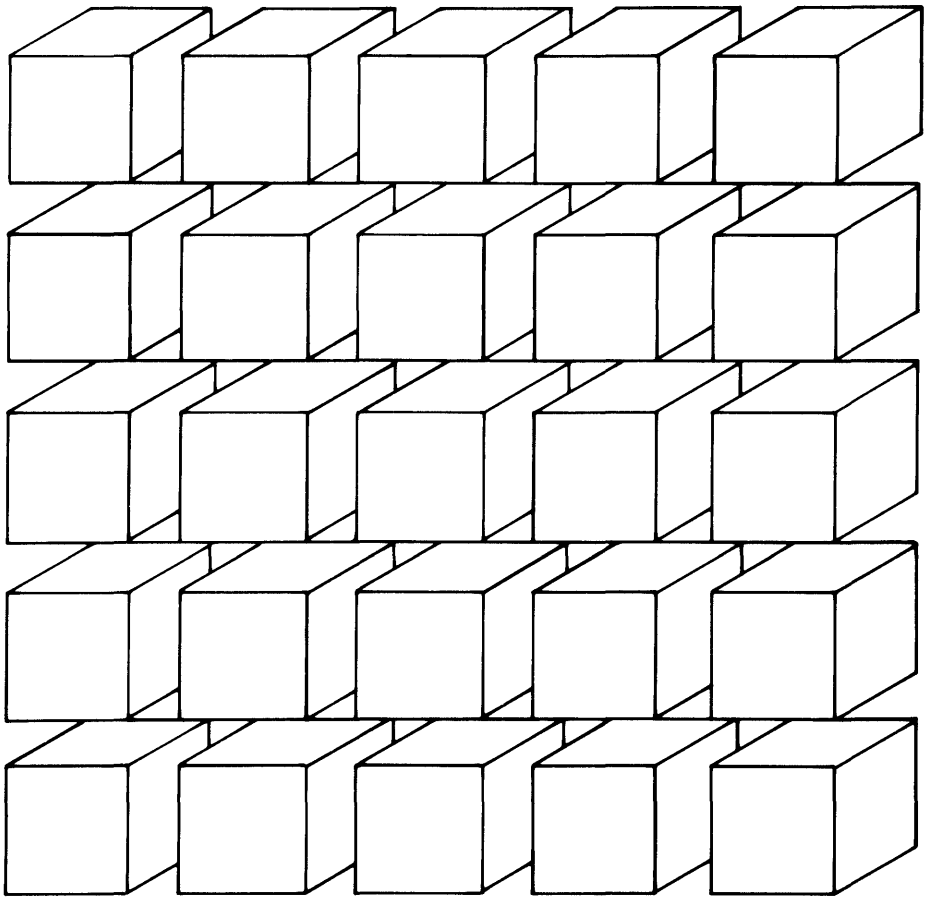
Business BASIC Technical Concepts (AOS/VS, AOS, RDOS, DOS)



Ordering No. 093-705004

Printed in the United States of America
Revision 00, March 1983

Notice

Business BASIC Technical Concepts
(AOS/VS, AOS, RDOS, DOS)
093-705004

Revision History:

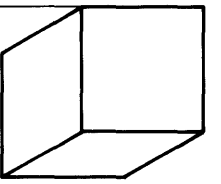
Original Release - March 1983

Addendum 086-000060-00 - August 1983

Effective with:

Business BASIC Rev. 7.0 (RDOS/DOS)
Business BASIC Rev. 3.0 (AOS and AOS/VS)

Table of Contents



Chapter 5
Terminal Types

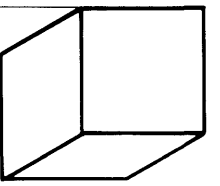
Chapter 6
Using Table Files with the FM Utility

Chapter 7
Screen Handling

Chapter 8
Logical File Database Structure

Appendix A
Memory Management for Business BASIC Systems

Preface



Scope

This manual is one of a set which includes *Business BASIC Statements, Commands, and Functions* and *Business BASIC Subroutines, Utilities, and BASIC CLI.* These three manuals replace the *Business BASIC Directory.* This set of manuals is intended as a reference for experienced Business BASIC programmers. It is assumed that the reader is familiar with Business BASIC and has read the *Guide to Using Business BASIC.*

This manual is to be used by Business BASIC programmers using the following operating systems: AOS/VS, AOS, RDOS and DOS. Separate manuals are available for Business BASIC on MP/OS.

The Business BASIC Technical Concepts manual presents technical information on file structures, INFOS®, File Maintenance, and Screen Maintenance, as well as information on terminal characteristics and many aspects of program development.

Organization

This guide is divided into seven chapters.

Chapter 1 is an overview of program development.

Chapter 2 describes how to use the Business BASIC CLI and the AOS CLI. The BASIC CLI program simulates a CLI environment and, therefore, allows you to use operating system features without terminating Business BASIC execution.

Chapter 3 describes the file organization, file structure, and file access methods used in Business BASIC.

Chapter 4 describes the types of data and computations used in Business BASIC.

Chapter 5 describes terminal types and terminal characteristics.

Chapter 6 presents a tutorial on the use of the File Maintenance utility with table files.

Chapter 7 describes the Screen Maintenance utility.

For further information about the Business BASIC language and your particular operating system, consult the Related Documents section of this guide.

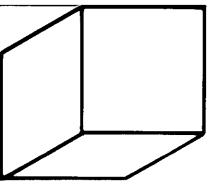
Typographical Conventions

The Business BASIC asterisk prompt (*) indicates the beginning of a command or program statement. To enter a statement or command, press NEWLINE or CR, depending on the system.

User input is distinguished from system response by these typefaces:

```
This is user input.
This is system response.
```

Chapter 1
Program Development



Two important concepts are covered in this chapter: **working storage,** which is the portion of memory you use to develop programs and where programs that are ENTERed or LOADed are stored, and the **common area,** a portion of memory you use to temporarily store information while changing programs.

Working Storage and the Common Area

Working storage is an area of the computer's memory that holds your Business BASIC program and data. The remaining memory contains the operating system (AOS, AOS/VS, RDOS, or DOS) and the Business BASIC interpreter. The program in working storage is in save file format. In this format, programs are compressed to save space. For example, each keyword is assigned a number so that the number rather than the larger alphanumeric word can be stored. One consequence of this compression is that in-line comments (starting with a colon) are stripped as they are entered.

A line you enter as:

```
* 10 A=12   :one dozen apples
```

will be shown on the screen when you use the LIST command as:

```
* LIST
0010  LET A=12
```

NOTE:

The Business BASIC syntax checker inserts optional keywords such as the LET statement shown above.

To preserve a program's comments, use the REM keyword or keep one copy of the program in ASCII format. The EDIT utility, discussed later in this chapter, is also used to add comments to a program.

The common area is a 512-byte area of memory used to store information and pass it from one program to a subsequently running program. The common area is directly accessed by the GETCM.SL subroutine, described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual, and by the BLOCK READ and BLOCK WRITE commands described in the *Business BASIC Commands, Statements, and Functions* manual.

The Business BASIC system provides a great deal of flexibility in creating, editing, and running a program. Commands that are used for this purpose are listed in Table 1-1. They are described in the *Business BASIC Commands, Statements, and Functions* manual or in the *Business BASIC Subroutines, Utilities and BASIC CLI* manual.

Command	Description
CHAIN	Execute a program from the current one without returning.
CLI	Execute the Business BASIC CLI.
CON	Continue an interrupted or stopped program.
ENTER	Merge program statements from disk to working storage.
ERASE	Delete program statements in working storage.
LIST	Display program statements in working storage.
LOAD	Load a program from disk to working storage.
NEW	Clear working storage.
POP	Terminate the Business BASIC CLI and clear the common area.
QUIT	Terminate the Business BASIC CLI without clearing the common area.
RENUMBER	Renumber program statements
REPLACE	Replace a program and variable values on disk.
RUN	Execute a program.
SAVE	Save a new program and variable values on disk.
SIZE	Display size of program in working storage.
SWAP	Execute a program from the current one and return.

Table 1-1 Commands for Creating, Editing, and Running a Program

Command	Output File Form	Effect	Command for Reusing
LIST	ASCII	Does not preserve variable values and last line number executed. Allows you to display results of typing and editing in sequential line number order. Saves comments identified by REM. Does not save comments started with :.	ENTER
SAVE	binary	Saves a program in a new disk file. Preserves variable values and last line number executed. Saves comments identified by REM. Does not save comments started with :.	LOAD, RUN, CHAIN, SWAP
REPLACE	binary	Saves a program in an existing disk file. Preserves variable values and last line number executed. Saves comments identified by REM. Does not save comments started with :.	LOAD, RUN, CHAIN, SWAP

Table 1-2 Differences among Program-Saving Commands

Saving a Program

To save a Business BASIC program in a disk file, issue a LIST, SAVE, or REPLACE command. While each of these commands saves all program statements from working storage, the type of output file created differs. Both SAVE and REPLACE create a binary formatted program file; LIST creates an ASCII character formatted file. Although a binary file takes less storage space and is faster to access than an ASCII file, it can be used only by Business BASIC programs. Differences among the three program-saving commands are summarized in Table 1-2.

Optimized Code

The OPT utility (described in the Business BASIC Subroutines, Utilities, and *BASIC CLI* manual) optimizes the size and runtime of a SAVEd program. There are two levels, 0 and 1. Level 0 removes REM statements and optimizes GOTO addresses, which include GOSUB, ON/THEN, and RESTORE.

Level 1 performs the same functions as level 0 and optimizes string, multiple LET statements, and arithmetic LET statements (except those which include functions or assign numeric values to string variables). Level 1 offers speed improvements of from 15 to 25 percent over nonoptimized programs.

File Extensions

Certain extensions have special meaning to Business BASIC. To prevent confusion, you should use these extensions only for the special file types that they are intended to describe, although Business BASIC does not restrict their use. Table 1-3 lists these extensions and their meanings.

Extension	Meaning
.TB	A table file. Used with the FM utility.
.SL	A Business BASIC subroutine source or listing file in ASCII format.
.S*n*	A screen file, used with the Screen Management utility, where *n* is the terminal type.
.SF	Indicates a utility source file listing that has comments. .SF files are contained in the $DOC directory, which was supplied with your Business BASIC software.
.SP	Indicates a commented utility source file module which is accessed by one or more .SF files. .SP files are contained in the $DOC directory, which was supplied with your Business BASIC software.

Table 1-3 File Extensions

Editing Business BASIC Programs

Business BASIC provides several file editing and formatting capabilities.

With the Business BASIC EDIT utility you can insert, delete, move, replace, and make other changes to text files. The files may contain descriptive text, Business BASIC program statements, or statements written in other programming languages. The EDIT utility can also be used to add comments to the end of Business BASIC program statements as a documentation aid. In addition, comments can be extracted to create a documentation file.

Another utility, the DOC utility, allows you to:

- Add comments to the end of Business BASIC program statements as a documentation aid.
- Extract comments from a program to create a documentation file.
- Output text files automatically formatted by Business BASIC according to your specifications.
- Automatically scan all your files and produce a table of contents.

A detailed description of both DOC and EDIT is presented in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Accessing and Running a Program

You can bring a program from a disk file to working storage for editing and/or running. LOAD and ENTER are two commands that bring a program to working storage without running it.

Several command formats allow you to run programs. You can run a program currently in working storage with the immediate mode RUN command, or you can simultaneously load and run a program by typing:

RUN "program name"

To facilitate program development, you can interrupt program execution (by pressing ESC) to inspect variable values and possibly change program statements. You may restart execution of an interrupted program immediately following the last line processed with the immediate mode CON command or at a particular line with the immediate mode command

RUN line number

Current variable values are preserved when a program is interrupted and restarted with a line number. If you start a program by typing RUN without a line number, the program restarts at its first line and resets all variables.

Three other commands are available for running a program: CHAIN, SWAP, and CON. These commands may be used in immediate mode. However, CHAIN and SWAP are generally used as program statements to execute a stored program from within the program in working storage. The main difference between CHAIN and SWAP is that CHAIN does not return control to the program originally in working storage while SWAP does. Flow of program control resulting from the CHAIN and SWAP commands is shown in Figures 1-1 and 1-2, respectively.

You may also run any SAVEd or REPLACEd program by typing the program name after the Business BASIC CLI prompt.

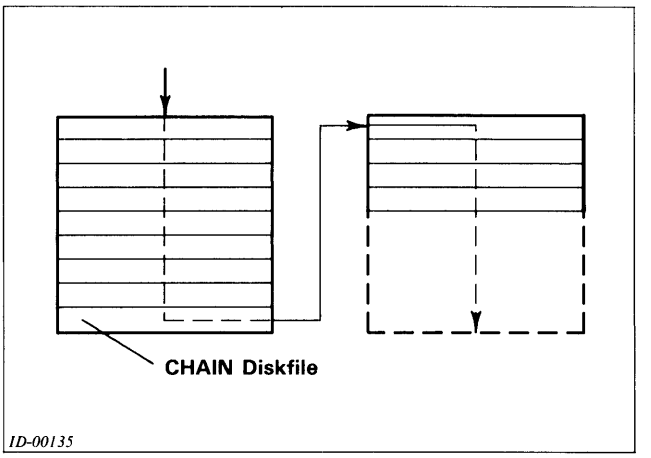


Figure 1-1 *Flow of Program Control with CHAIN Command*

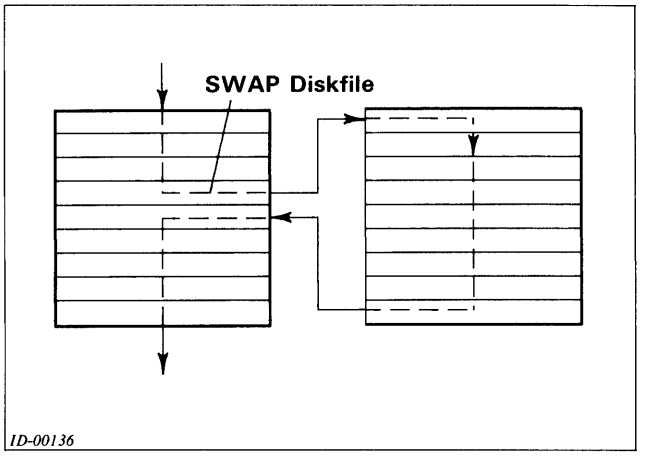


Figure 1-2 *Flow of Program Control with SWAP Command*

Command Format	Usage	Location of Program Executed	Program in Working Storage after Execution	Variable Values Preserved from Last Execution?
RUN	immediate	working storage	original	no
RUN line #	immediate	working storage	original	yes
RUN "program"	immediate	disk file	new	no
CHAIN "program"	program statement or immediate	disk file	new	no
CHAIN...THEN GOTO...	program statement or immediate	disk file	new	yes
CHAIN...THEN CON	program statement or immediate	disk file	new	yes
SWAP "program"	program statement or immediate	disk file	original	no
SWAP...THEN GOTO...line #	program statement or immediate	disk file	original	yes
SWAP...THEN CON...	program statement or immediate	disk file	original	yes
CON	immediate	working storage	original	yes

Table 1-4 Differences among Program Execution Commands

NOTE:
The SWAP "program command can also be executed simply by typing "program; the SWAP command itself does not have to be typed.

Chaining Overhead (RDOS/DOS only)

When a CHAIN statement is being executed, the following operations occur:

- The specified program file is opened.
- The file contents are checked to verify that it is a SAVEd file.
- Any memory blocks used to store the program segment currently LOADed or RUNning are released to the free memory pool.
- Any memory blocks used to store the data segment currently LOADed or RUNning are released to the free memory pool.
- The header portion of the SAVEd file is read in to determine how much memory is needed for program and data segments.
- The program segment of memory is expanded (request memory allocation to read in program segment, read in program segment from file, initialize certain page 0 locations to reflect newly loaded program segment size and address).
- The data segment of memory is expanded (request memory allocation to read in data segment of memory, read in data segment from file, initialize certain page 0 locations to reflect newly loaded data segment's size and address).

All of the above steps are of relatively low cost, except for the expansion of the program and data segments.

In the case of a CHAIN, each segment grows beyond the nearest 2KB boundary (currently zero since all memory for each segment was just deallocated). This requires the execution of a .SWAP primitive in the assembly language source code. As a result of executing the .SWAP primitive, the task scheduler steps in, starts up the swapper, swaps out the job that just requested more memory, places it on the swap-in queue, and takes control again.

NOTE:
The 2KB boundary is applicable to window mapping systems only. For a swapping system this boundary would be 512 bytes.

The next step is for the scheduler to find a job to run. This job will be the one that is doing the CHAIN (the one that just requested to expand either its program or data segment). If sufficient memory is not available for that job, other jobs must be forced out. When enough memory is free to expand the appropriate segment, the job is loaded in and run. Once this has taken place for the second time (first for the program segment, and then for the data segment), the job will run.

The .SWAP primitive must be executed to expand the user's segments, as the memory allocation scheme is intrinsically part of the swapper. A segment may grow beyond a 2KB boundary only at the time a job is swapped in. In allocating new portions of memory, the additional memory used must be adjacent to the segment being expanded. In addition, the program and data segments must reside in one contiguous area for each job.

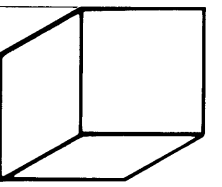
Push Space Limits (RDOS/DOS only)

You must have sufficient space in the BASIC.PS file (located in $SYS or $SY3) for the Business BASIC SWAP statement/command to work. By default, this file is three times the size of BASIC.SW (swap file), which is enough space for three SWAP statements/commands (two nested within the first). If you want a greater nesting limit for SWAPs, use the RDOS/DOS CLI CCONT command to create the BASIC.PS file with a size specified by this formula:

size of BASIC.PS = (max pages per job + 1) * (max number of nested SWAPs) * (max number of jobs)

The sysgen parameter MAXIMUM PROGRAM SIZE (512-BYTE PAGES) controls the size of the swap and push files (BASIC.SW and BASIC.PS). The RDOS/DOS BASIC sysgen procedure is described in *Business BASIC System Management*.

Chapter 2
The Business BASIC CLI



The Business BASIC Command Line Interpreter (CLI) is a program written in Business BASIC that acts as an interface between Business BASIC and your operating system (AOS, AOS/VS, RDOS, or DOS). The BASIC CLI program simulates a CLI environment and, therefore, allows you to use operating system features without terminating Business BASIC execution.

This chapter describes how to start and stop the Business BASIC CLI, use CLI commands and switches, run SAVEd Business BASIC programs from the CLI, and access the AOS CLI on AOS and AOS/VS systems.

BASIC CLI commands are preceded by the exclamation point (!) prompt. To execute a CLI command you can:

- Type the exclamation point prompt and the command. This method can be used only if you are in keyboard mode. After the command has been executed the asterisk prompt will reappear.
- Use RUN "CLI", SWAP "CLI", CHAIN "CLI" or !CLI to make the CLI operational. This method can be used in keyboard mode or from within one of your programs. This will start the BASIC CLI and display the exclamation point prompt. Every line you enter will be interpreted as a CLI command, and keywords that are not CLI commands will cause errors. The exclamation point will reappear after each command is executed. To terminate CLI operation and restore keyboard mode or a previously running program use the QUIT, POP, or CHAIN command.

The BASIC CLI commands are listed in Table 2-1. Each command is described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

When you execute a BASIC CLI command in keyboard mode, the CLI program is SWAPped and the command is passed through the common area. To preserve the information in the common area, or to execute more than one CLI command and get a faster response, run the CLI program itself, using the RUN, CHAIN, SWAP, or !CLI commands.

You must have access to the CLI program to RUN, CHAIN, or SWAP to the BASIC CLI. In RDOS/DOS, your system manager must give you access to the BASIC CLI. In AOS or AOS/VS, you must have R access to the BASIC CLI program to LOAD, RUN, CHAIN, or SWAP to it. In AOS, you also need the ability to create a son process to execute some of the BASIC CLI commands.

Differences between the RUN, CHAIN and SWAP commands are outlined in Chapter 1, Table 1-4.

CLI commands have the format

!command/global-switches argument-1 argument-n/local-switches

Arguments are automatically stored in the Business BASIC common area and may be read with the BLOCK READ or GETCM.SL keywords.

You can use global and local switches with BASIC CLI commands. A global switch defines what the command does to all of its arguments. For example:

```
!DELETE/C MATRIX, COMPARE
```

The /C switch appended to the DELETE command causes a confirmation prompt to be displayed before each listed file is deleted. This prevents you from accidentally deleting a file.

A local switch defines what the command does to a specific argument. For example:

```
!DELETE 06-11-82/B
```

The /B switch appended to the argument 06-11-82 above deletes all files in the working directory created before 06-11-82.

Using the BASIC CLI in Keyboard Mode

You can execute a CLI command from keyboard mode by typing an exclamation point and the command. The system will then SWAP to the CLI and pass the command through the common area. When the command is finished executing, it will pop you back into keyboard mode. However, when you use the immediate CLI execution, your common area is used and the response time is slow. To preserve your common area, or to execute more than one CLI command and get a faster response, use the CLI utility (!CLI) instead of the immediate CLI execution.

APPEND	Appends a file to another file.
AOS	Enables you to access the AOS CLI (AOS and AOS/VS only).
ASG	Assigns a device for exclusive use (RDOS/DOS only).
ATTACH	Attaches your terminal to a detached job (RDOS/DOS only).
BLDCOM	Builds a documentation file for PRTCOM.
BUILD	Builds a command file (RDOS/DOS only).
BYE	Terminates AOS, AOS/VS or RDOS/DOS CLI.
CCONT	Creates a contiguous file (RDOS/DOS only).
CDIR	Creates a subdirectory (RDOS, AOS, AOS/VS) or directory (DOS).
CHAIN	Executes a program.
CHATR	Changes a file's attributes (RDOS/DOS only).
CHLAT	Changes a file's link access attributes (RDOS/DOS only).
CLI	Executes the BASIC CLI.
CPART	Creates a secondary partition (RDOS/DOS only).
CRAND	Creates a random file (RDOS/DOS only).
CREATE	Creates a file (AOS, AOS/VS) or sequential file (RDOS/DOS)
DELETE	Deletes a file, directory, or partition.
DIR	Changes the current directory.
DISK	Displays the amount of disk space used and remaining.
DOC	Prints a series of text input files.
DOCTOC	Prints a table of contents.
DUMP	Copies one or more disk files in DUMP format to an output file.
EDIT	Creates and/or edits a text file.
EQUIV	Renames a device (RDOS/DOS only).
FPRINT	Dumps a file to the line printer or terminal.
FDUMP	Fast dumps one or more files to mag tape (RDOS/DOS only).
FILCOM	Compares two files word by word.
FILES	Displays names of files in current directory.
FLOAD	Fast loads FDUMPed files (RDOS/DOS only).
FM	File maintenance package.
GDIR	Displays the current directory name.
GQUE	Gets the default queue name.
GSDIR	Displays the current system directory name (RDOS/DOS only).
GSYS	Displays the current system name (RDOS/DOS only).
GTOD	Displays the time and date.
INIT	Initializes a device (RDOS/DOS only).
LINK	Links an alternate name to a file.
LIST	Lists information for files in the current directory.
LOAD	Reloads dumped files.
LSTCOM	Compares two listing files character by character.
LSTMERGE	Merges two listing files.
MDIR	Displays the master directory name (RDOS/DOS only).
MOVE	Moves files from one directory to another.
POP	Exits from the BASIC CLI and clears the common area.
PORTS	Displays jobs on the system.
PRINT	Prints a file on the line printer.
PRTCOM	Prints a BLDCOM documentation file.

Table 2-1 Basic CLI Commands (continues)

QUIT	Exits from the BASIC CLI but retains the common area.
RENAME	Renames files.
REWIND	Rewinds a magnetic tape (AOS, AOS/VS only).
SDIR	Sets the system directory (RDOS only).
SLINE	Selects a line (terminal) and attaches to it (RDOS/DOS only).
SPDIS	Disables device spooling (RDOS only).
SPEBL	Enables device spooling (RDOS only).
SPKILL	Deletes the spool queue (RDOS only).
SQUE	Sets the default queue.
START	Starts a detached job (RDOS/DOS only).
STAT	Displays the status of all jobs (processes).
TABLE	Prints a program cross-reference.
TCOPY	Copies from tape to tape.
TFER	Copies a file between tape and disk.
TPRINT	Prints the tuning report (RDOS only).
TYPE	Displays a file on your terminal.
UNLINK	Removes link entries from a directory.
VFU	Edits a format control file for a data channel line printer.
XFER	Copies one file to another file.

Table 2-1 Basic CLI Commands

It is preferable to SWAP to the CLI if you currently have a program sitting in the program buffer and wish to preserve it. When the CLI is exited via the POP or EXIT command, the system returns to keyboard mode with the program intact. When you SWAP to the CLI, all files are closed. Upon return from the CLI these files must be reopened and repositioned to their previous point from the SWAP.

Using the BASIC CLI from a Program

For some applications you might want to SWAP to the BASIC CLI from a Business BASIC program and pass a command line to the BASIC CLI. To do this, use the common area as you would a utility you can only SWAP to. Build your command line in a string, pass the string to the common area, and SWAP to the BASIC CLI.

If you are using RDOS/DOS Business BASIC, this command line string must start with the substring "CLI.CM<0>". When you SWAP to the BASIC CLI, the system scans the first 7 bytes of the common area looking for the substring "CLI.CM<0>". If it finds this substring, it expects to find a series of commands separated by semicolons following the substring. These commands can be BASIC CLI commands with optional switches, filenames of BASIC programs, or utility programs. An example of a Business BASIC program that builds a simple command line and SWAPs to the BASIC CLI follows:

```
0010  DIM X$(512)
0020  LET X$= "CLI.CM<0>","FILES;GTOD;POP",FILL$(0)
0030  BLOCK WRITE X$
0040  SWAP "CLI"
0050  PRINT "CLI OPERATION PERFORMED"
0060  STOP
```

This program sends the command line in X$ to the BASIC CLI via the common area. You can have only one 512-byte string in the common area. The BASIC CLI then executes the FILES utility, the GTOD command, and the POP command, which returns control to the original program at line 50. To make the BASIC CLI return control to the original program, your last command must be POP or QUIT.

You can use indirect file conventions (@ under RDOS/DOS and [] under AOS-AOS/VS) in the BASIC CLI but only at one level. For example, @filename@ may not occur within a file that was used as an indirect file. You can use parentheses to repeat commands with different arguments, but you cannot nest indirects within parentheses or vice versa. Before using these conventions, consult the user's manual for your operating system.

Running a Program from the CLI

You may run any SAVEd Business BASIC program from the Business BASIC CLI by typing the program's filename after the CLI's exclamation point prompt. The CLI swaps to the file you have specified and returns the prompt after the program has been executed. The GETCM.SL subroutine allows you to create CLI commands that use keyword and argument switches. This subroutine is described in detail in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Using the AOS CLI

If you are using Business BASIC on an AOS or AOS/VS system, the !AOS command creates a new son process running the AOS CLI and gives you the AOS CLI right parenthesis prompt. Your process running Business BASIC is its father process. The format of the AOS command is:

!AOS

Once you have the parenthesis prompt, you can type an AOS CLI command or macro.

If you change your environment while in the son process (for example, you DIR to another directory or change your searchlist), the new environment is effective only while you are in the son process. When you return to your original (father) process with a BYE command, you also return to your original environment. For example:

```
!GDIR
:UDD:COCHRAN:BASICMAN
!AOS
)DIR :UDD:COCHRAN
)DELETE TEST
)BYE
!GDIR
:UDD:COCHRAN:BASICMAN
```

In the above example, you start in directory BASICMAN, execute the AOS command, move to directory COCHRAN, delete file TEST, and then BYE back to the BASIC CLI to return to directory BASICMAN.

In Figures 2-1, 2-2, and 2-3, a user (with user name MAUREEN) creates a son process running the AOS CLI, checks its current environment, changes SEARCHLIST and DIRECTORY settings, returns to the father process, and checks the original environment. The user also uses the STAT program to display other system processes.

```
* !AOS
AOS CLI   REV xx.xx 29-SEP-82 16:31:25


) CURRENT
LEVEL              0
SUPERUSER          OFF
SUPERPROCESS       OFF
SCREENEDIT         ON
SQUEEZE            OFF
CLASS1             ERROR
CLASS2             WARNING
TRACE
VARIABLES          0    0    0    0    0
                   0    0    0    0    0
LISTFILE           @LIST
DATAFILE           @DATA
DIRECTORY          :UDD:MAUREEN
SEARCHLIST         :UDD:MAUREEN, :PER, :UTIL, :UTIL:$SYS, :UTIL:$LIB
DEFACL             MAUREEN.OWARE
STRING
PROMPT
CHARACTERISTICS    /605X/LPP=24/CPL=80
                   /ON/ST/EBO/ULC/WRP
                   /OFF/SFF/EPI/BBT/SPD/RAF/RAT/RAC/NAS/OTT/EOL/UCO/LT/FF/EBI
M/MOD/TO/TSP/PBN/ESC/FIQ/NHL
)
```

Figure 2-1 Creating a Son Process

At this point, the son's process environment is the same as the father's. The information displayed by CURRENT applies to both processes. MAUREEN now changes the son's environment settings for DIRECTORY and SEARCHLIST (see Figure 2-2).

```
)DIR :UDD:MARK
)SEARCHLIST :UDD:COMMON, :PER, :UTIL
)CURRENT
LEVEL          0
SUPERUSER      OFF
SUPERPROCESS   OFF
SQUEEZE        OFF
CLASS1         ERROR
CLASS2         WARNING
TRACE
VARIABLES          0    0    0    0    0
                   0    0    0    0    0
LISTFILE       @LIST
DATAFILE       @DATA
DIRECTORY      :UDD:MARK
SEARCHLIST     :UDD:COMMON, :PER, :UTIL
DEFACL         MARK.OWARE
STRING
PROMPT
CHARACTERISTICS    /605X/LPP=24/CPL=80
                   /ON/ST/EBO/ULC/WRP
                   /OFF/SFF/EPI/BBT/SPD/RAF/RAT/RAC/NAS/OTT/EOL/UCO/LT/FF/EBI
M/MOD/TO/TSP/PBN/ESC/FIQ/NHL

)
```

Figure 2-2 Changing the Son Process's Environment

The son's environment now has a different searchlist and current directory. We can create other levels of environment within this son process using the AOS CLI PUSH command, and then return to the original level using the AOS CLI POP command. To return to the father process running Business BASIC, MAUREEN must BYE from the son process as follows:

```
)BYE
AOS CLI TERMINATING 29-SEP-82 17:00:12
```

MAUREEN is now in Business BASIC's keyboard mode. To check the current user environment settings,

MAUREEN can use the !AOS command to get back to AOS CLI and then execute the AOS CLI CURRENT command once again (see Figure 2-3).

```
* !AOS
AOS CLI   REV xx.xx 29-SEP-82 16:31:25

)CURRENT
LEVEL           0
SUPERUSER       OFF
SUPERPROCESS    OFF
SQUEEZE         OFF
CLASS1          ERROR
CLASS2          WARNING
TRACE
VARIABLES           0    0    0    0    0
                    0    0    0    0    0
LISTFILE        @LIST
DATAFILE        @DATA
DIRECTORY       :UDD:MAUREEN
SEARCHLIST      :UDD:MAUREEN,:PER,:UTIL,:UTIL:$SYS,:UTIL:$LIB
DEFACL          MAUREEN.OWARE
STRING
PROMPT
CHARACTERISTICS     /60SX/LPP=24/CPL=80
                    /ON/ST/EBO/ULC/WRP
                    /OFF/SFF/EPI/BBT/SPD/RAF/RAT/RAC/NAS/OTT/EOL/UCO/LT/FF/EBI
M/MOD/TO/TSP/PBN/ESC/FIO/MNL
```

Figure 2-3 *Checking the Original Environment*

If MAUREEN uses the !AOS command to move to AOS CLI and a WHO command to display what process (PID number) she is, she will get the PID number of the new son process running the AOS CLI. For example:

```
* !AOS
)WHO
PID: 16 MAUREEN  016   :CLI.PR
)BYE
AOS CLI TERMINATING  29-SEP-82 17:30:12
*
```

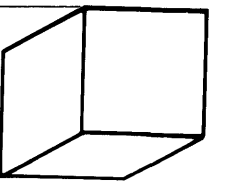
However, if she executes the !AOS CLI TREE command, more useful information is displayed:

```
* !AOS
AOS CLI REV xx.xx 29-SEP-82 16:31:25
)TREE
PID: 16, FATHER: 13, SONS:
)BYE
*
```

This tells the user that the current process (the new son process created by the !AOS command) is PID 16, and its father process (the original Business BASIC process that executed the !AOS command) is PID 13.

For more information about AOS-AOS/VS processes, see *Introduction to the Advanced Operating System* and *Learning to Use Your Advanced Operating System.* For more information about the AOS CLI, see the *Command Line Interpreter User's Manual* for AOS.

Chapter 3
File Input/Output



This chapter discusses file organization, structure, and access methods.

File organization refers to the method used to store a file on disk. Three basic types of internal file organization exist for storing RDOS/DOS data files in Business BASIC: sequential (RDOS only), random (RDOS/DOS), and contiguous (RDOS/DOS). In addition, AOS and AOS/VS systems have some specific storage characteristics.

File structure refers to the divisions within a data file. Five terms are used when discussing file structures:

- Physical file, which is a unit of data storage recognized by the operating system. Data in these files is usually in binary or ASCII format, but it may be stored in any format.
- Subfile, which is a division of a file that is recognized by Business BASIC. A Business BASIC physical file may be divided into one or more subfiles.
- Master file, which is a physical file that contains subfiles.
- Logical file, which is a subfile or a physical file that contains no subfiles.
- Linked-available-record format (or dynamic record allocation), which is a method of disk storage that allows a file to reuse space left by deleted records. Each deleted record contains a pointer that points to the next available record in the file.

Types of access common in Business BASIC include getting information from a file and putting information into memory (reading or inputting), putting information from memory into a file (writing or outputting); determining the position (number) of a record, determining the location of the file pointer, and moving the file pointer. Sequential, random, and indexed sequential access methods are discussed in this chapter.

File Organization

Sequential File Organization (RDOS/DOS only)

In a sequentially organized file, information is stored on the disk in groups of 512 bytes. (A disk block contains 512 bytes of information.) The last 2 bytes of each 256-word block are reserved and contain a pointer to the next block of data in the file. If you are building a sequential file, the system simply appropriates the next available disk block when it needs additional space and constructs a pointer to that block. The system finds the necessary space and maintains the pointers. Since sequential files appropriate space as it is needed, you can always append records to a sequential file.

Figure 3-1 shows how a sequentially organized file might look on the disk. After processing any given block of a sequential file, the system may step either to the previous block or to the next block in the series. To access a record in the tenth block of a sequential file after accessing a record in the first block, the system would have to read the nine intervening blocks — a time-consuming process. Random or contiguous files provide faster record access.

Contiguous File Organization

Contiguous files have a rigid structure, but they provide the fastest access to data. Figure 3-2 provides an example of contiguous file organization.

Contiguous files are composed of a fixed number of disk blocks that are physically adjacent, or contiguous, on the disk. The operating system can easily calculate the location of a record and access it directly. These files cannot be expanded or reduced in size, since by definition they occupy a fixed series of disk blocks. The disadvantage of using contiguous files is that you must allocate all the disk space that the file will require when you create the file. In addition, a contiguous file cannot be created unless the required number of contiguous disk blocks is available.

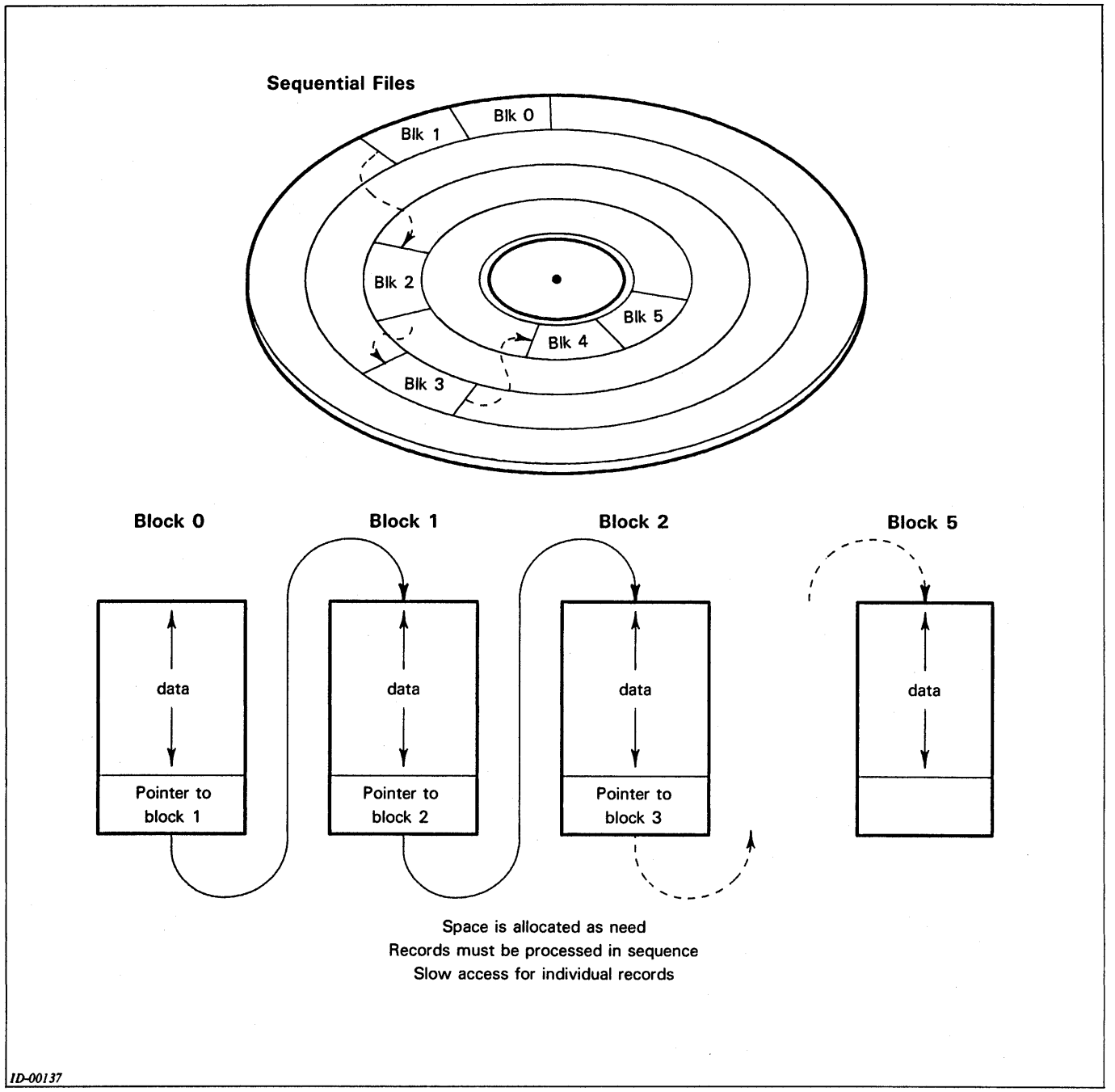


Figure 3-1 *Sequential File Organization*

Random File Organization

Random file organization (Figure 3-3) provides the best combination of flexible file structure and easily accessible data. Random files do not require that you read all intervening records when positioning to a specific record in the file. The operating system maintains a small sequential file that contains pointers to data blocks in the random file. The operating system needs only to read this small file to find the location of the data block containing the desired record. It then accesses this block directly, resulting in faster access of records.

Random files need slightly more disk space than sequential files because of the file of pointers that the system maintains. But, random files offer fast, direct access. A random file allocates space as it needs it, and allows you to append records to it at will.

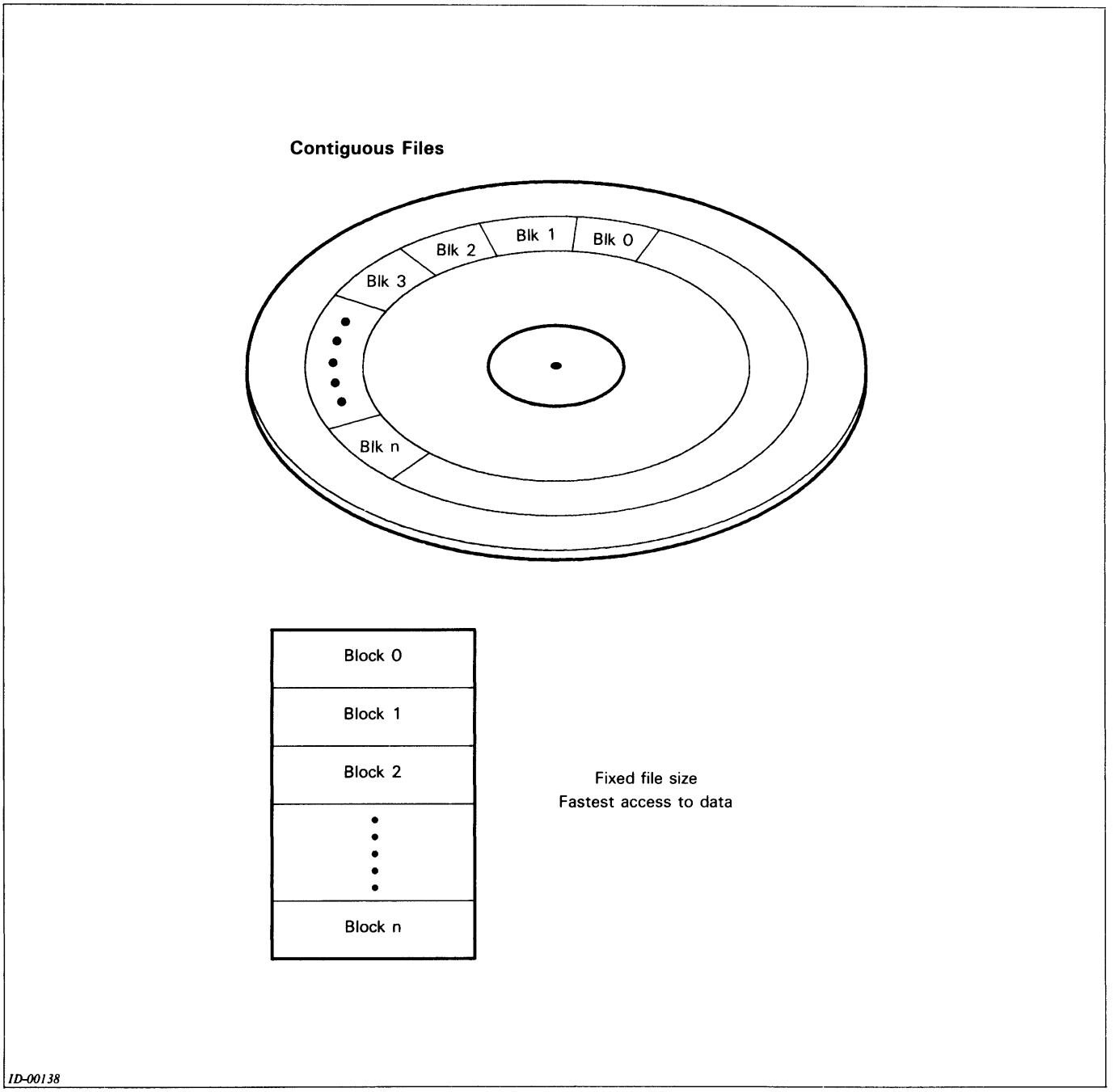


Figure 3-2 *Contiguous File Organization*

AOS Files

Every AOS-AOS/VS disk file is built from one or more 256-word (512-byte) disk blocks. The system ties together disk blocks within files by employing a hierarchical index.

The basic unit of storage in this file organization is a file element consisting of one or more contiguous blocks (blocks with sequential physical addresses). The file element size is specified when a file is created. The system allocates file space in multiples of file elements. Thus a file with an element size of eight grows in units of eight blocks.

Consider a file created with an element size of two. Initially, this file consists of two contiguous blocks (see Figure 3-4). If file storage requirements exceed the two blocks, an index block is allocated that contains the addresses of each of the file elements. The index element provides no data storage of itself, and each index element is one block (not one file element size) in length.

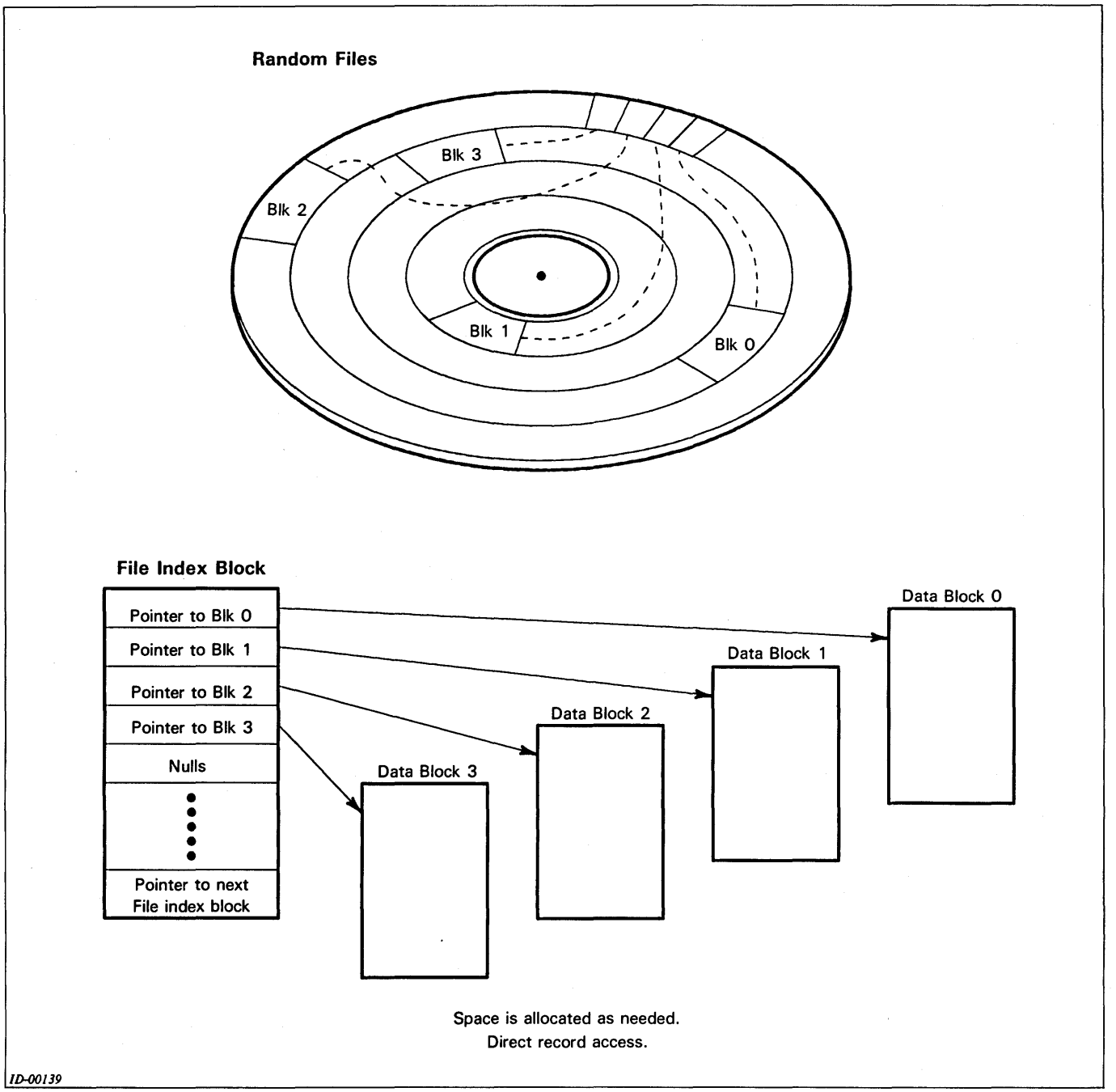


Figure 3-3 *Random File Organization*

As data storage requirements grow, more file elements are allocated and added to the list maintained in the index. If the index space becomes exhausted, another level of indexing is added. Pointers in the topmost level index element then point to further index elements.

In general, you will never be aware of index elements; the system stores and retrieves information so that you need not be concerned with the actual linkage between disk blocks. However, you should be aware of file organization, since it determines the tradeoffs in designing file structures. For example, large file element sizes could be used for creating data storage in contiguous disk areas that do not need indexes; these could be accessed quickly. Smaller file element sizes require more disk accesses but permit more efficient utilization of disk storage.

The largest possible disk file would contain 2**23 disk blocks. However, the total disk storage in a system is never completely available for user data storage. This is because some storage is always required by the system for disk bootstrap and block allocation structures.

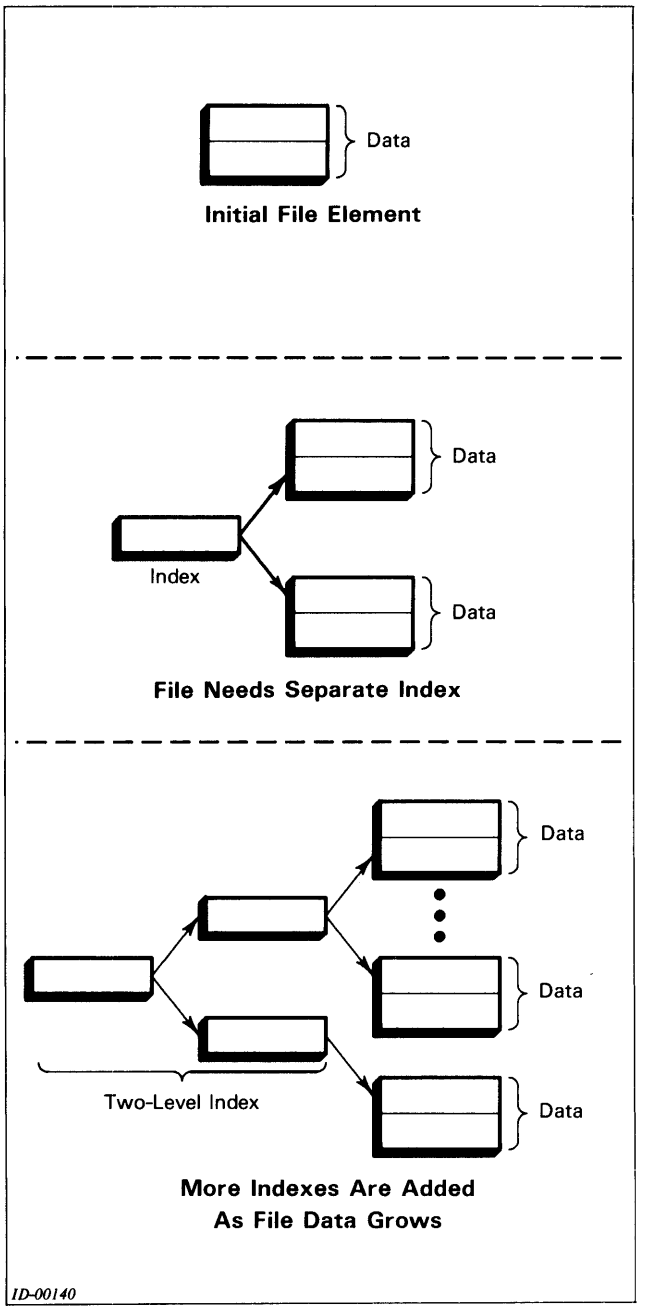


Figure 3-4 *Stages in AOS File Growth*

File Access

This section describes input/output procedures to data files with sequential, contiguous, and random internal file organization. Access commands for these files are listed in Table 3-1.

Typical business applications require quick access to records with a key (indexed sequential files). In addition, to use deleted record space efficiently, Business BASIC file design provides a linked-available-record format. For access to linked files and indexed files, the commands listed in Table 3-1 are supplemented by Business BASIC commands and the INITINDEX.SL subroutine, as well as Data General-supplied utilities, which are listed as follows:

KFIND	IBUILD
KNEXT	TBUILD
KADD	XBUILD
KDEL	INDEXBLD
INDEXCALC	INITFILE

These commands, programs, and the INITINDEX.SL subroutine are discussed briefly later in this chapter. In addition, the K commands are discussed in the *Business BASIC Commands, Statements, and Functions* manual. The utilities listed above and the INITINDEX.SL subroutine are discussed in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Sequential and Random Access

Files can be read and written in random or sequential mode. In random mode, the POSITION FILE command may be used to move the file pointer, and any record may thus be accessed immediately after any other record.

In sequential mode, only the next record can be accessed; the POSITION FILE command does not work. Since random mode allows for faster access of files, it is usually used for files containing records that require constant updating, such as employee personnel records. Sequential mode is used for files to which new records are constantly being added; for example, a list of company products. Modes are determined when the file is opened.

To read to or write from a data file with sequential, random, or contiguous organization, you must open it in either random or sequential mode for access on a channel. This is done by using the OPEN FILE command. To open subfiles, linked files, or ISAM files, use the OPEN FILE command or the OPEN utility.

Keyword	Purpose
BLOCK READ	Input data from the common area (512 bytes).
BLOCK READ FILE	Input data from a file, in multiples of 512 bytes.
BLOCK WRITE	Output data to the common area (512 bytes).
BLOCK WRITE FILE	Output data to a file in multiples of 512 bytes.
CLOSE FILE	Close a specific file.
CLOSE	Close all opened files.
DATA	Specify values for variables in a READ statement.
DELAY	Delay program execution.
EOF	Check for end of file.
GPOS	Determine the current file pointer position.
INPUT	Input ASCII data from the terminal.
INPUT USING	Input ASCII data from the terminal, allowing an error and a terminator trap.
INPUT FILE	Input ASCII data from a file.
INPUT FILE USING	Input ASCII data from a file, allowing an error and a terminator trap.
OPEN FILE	Open a file.
PAGE	Set number of characters per output page line.
POSITION FILE	Position the file pointer.
PRINT	Output ASCII data to the terminal.
PRINT USING	Output ASCII data according to a format to the terminal.
PRINT FILE	Output ASCII data to a file, terminal, or device.
PRINT FILE USING	Output ASCII data according to a format to a file, terminal, or device.
READ	Read values from a DATA statement.
READ FILE	Input binary data from a file.
RESTORE	Reset DATA list pointer.
TAB	Set number of characters per print zone.
TINPUT	Input ASCII data from the terminal within a fixed amount of time.
TINPUT USING	Input ASCII data from the terminal within a fixed amount of time, allowing an error and a terminator trap.
WRITE FILE	Output binary data to a file.

Table 3-1 Input/Output Commands

The following discussion explains the OPEN FILE command as well as the concept of a channel and access modes. Later in this chapter more specific explanations are given of I/O to simple sequential and random files as well as the procedures used for I/O to subfiles, linked files, and ISAM files.

Types of OPEN

A Business BASIC program can open a data file opened in two possible ways, exclusive and shared. The mode is specified with the OPEN statement. In RDOS/DOS systems other users can READ exclusively opened files but cannot WRITE to them. In AOS/VS and AOS systems, other users cannot READ or WRITE to exclusively opened files. A file opened in shared mode may be used concurrently by any or all of the current users in the system.

Channels

In Business BASIC, **channel** describes the line of communication from your working storage area in memory to a data file or device. Each file you read from or write to in a Business BASIC program must be assigned to a unique channel number. When the file is opened for access, a channel number is associated with that filename. The file is subsequently referred to by its channel number rather than by its name.

Each user has 16 channels available for file input/output, ranging from 0 through 15. In addition, channel number 16 refers to the terminal. Any file assigned to channel 16 will read data from and write data to the terminal.

Using the OPEN FILE Command

In Business BASIC, the OPEN FILE command can be used as a command or a statement. When a file is opened for access, the following must be specified:

1. The file to be opened. This may be represented by the actual filename in quotes, or by a string variable that contains the actual filename.
2. The channel number. This may range from 0 to 15. Channel 16 is used to specify that data input or output should be via the terminal. All subsequent references to that file will be by the channel number rather than by the filename.

   Any channel number not already associated with another filename may be used. The next sequentially available channel number is customarily used, although you are not restricted to using it. The UCHANS utility tells you which channels are in use.

   If you specify a channel number that is already associated with another file, an error condition will occur.
3. The mode of access. This, in turn, specifies the following file characteristics:
   - type of open — exclusive or shared
   - type of access — random or sequential

Mode	Type of Open	Type of I/O	Type of Access	Rule
0	exclusive	input/output	random	Create if file does not exist.
1	exclusive	output only	sequential	Delete file, then create file.
2	exclusive	output only	sequential	Append to existing file, or create file.
3	shared	input only	sequential	File must exist.
4	shared	input only	random	File must exist.
5	shared	input/output	random	File must exist.
6	exclusive	input/output	random	File must exist.
7	exclusive use of mag or cassette tape	input/output	MTDIO access (sequential)	Tape must exist, tape file can be created, deleted, or appended to.

Table 3-2 File Access Modes

- type of I/O — input, output, or both
- rules governing the file's existence — should it be deleted if it already exists, created if it does not exist, or must it exist?

By choosing the correct mode, you can manipulate the file characteristics. Table 3-2 summarizes the eight possible file access modes.

Closing a File

After a file has been processed, it must be closed. Closing a file disassociates that filename from the channel to which it is assigned. You should close a file when you are finished using it. You must also close a file if you want to reopen it in a different mode.

CLOSE may be used as a statement or a command, and has two forms. The first form simply contains the word CLOSE, and will close all files currently open to the user. The second form is CLOSE FILE channel number. It will close only the particular file associated with that channel number.

Once a file has been closed, the channel number associated with that file is free and can be reused to open another file. The file itself can be reopened on a different channel and in a different mode.

Most programs that access files close the files when they finish using them or at the end of the program. If you forget to close a file and try to open it, an error condition (stating the file is already open) will result. If this occurs you can issue a CLOSE or a CLOSE FILE command while in keyboard mode, and then continue with your program. In order to avoid this error, it is good practice to include a CLOSE statement at the beginning of your program, before any OPEN FILE statements are issued.

End of File

End of file (EOF) is a Business BASIC function used in sequential files. The function is used to detect whether the last READ FILE statement executed has read in the last data record, and the end of file has been reached.

The format of the end of file function is:

EOF (channel number)

The end of file function returns a value of 1 if it detects an end of file or if the channel is not open. A value of 0 is returned if the end has not yet been reached. EOF is normally used in reading sequential files, when reaching the end of the data will signal the end of processing. In addition, it can be used with an IF/THEN statement to make a conditional transfer when the end of the file has been reached.

NOTE:

End of mag tape is indicated by a status word in MTDIO.

I/O to Sequential, Random, or Contiguous Files

To write to a sequential, random, or contiguous file, put the following steps in your program:

1. Dimension the string variable(s) and numeric arrays to be written.
2. Assign values to the variables to be written.
3. Open the file.
4. Write the variable or array to the file using WRITE FILE or PRINT FILE. The number of bytes written is determined as follows:
   - For numeric variables, by the variable's precision.

- For string variables, by the dimension of the variable. If PRINT FILE is used, the amount written will be less than the dimensioned length of the variable if the string contains either a <12> or a <13>, which represent NEWLINE and CR, respectively.

5. Use POSITION FILE to move the file pointer.

   If a file has random or contiguous organization, you may access individual records directly by their relative positions within the file. This is done by positioning the file pointer to the beginning of the record, or portion of a record, to be accessed. If records are fixed in length, the algorithm

   record number * record length

   is used to position the file pointer. The file pointer moves during a record access, and at the conclusion of the operation it is positioned to the byte immediately following the last byte that was read or written. You can also check the current position of the file pointer using the GPOS (get position) function. This is particularly useful if a file contains variable length records.

6. Repeat steps 4 and 5.
7. Close the file using the CLOSE command.

NOTE:

Steps 5 and 6 can be used only if the file has been opened for random access.

To read from a file, include the following steps in your program:

1. Dimension the string variables and arrays that will receive the information.
2. Open the file. The file pointer is at the first byte in the file.
3. Get the information from the file using either READ FILE or INPUT FILE. The number of bytes transferred is determined in step 1 for strings and by precision for arrays.
4. Use the EOF function (described earlier in this section) to check if the end of the file has been read. Attempting to read past the end of the file causes an error.
5. Use POSITION FILE to move the file pointer.

   If a file has random or contiguous organization, you may access individual records directly by their relative positions within the file. This is done by positioning the file pointer to the beginning of the record, or portion of a record, to be accessed. If records are fixed in length, the algorithm

   record number * record length

   is used to position the file pointer. The file pointer moves during a record access, and at the conclusion of the operation it is positioned to the byte immediately following the last byte that was read or written. You can also check the current position of the file pointer using the GPOS (get position) function. This is particularly useful if a file contains variable length records.

6. Repeat steps 3 through 5.
7. Close the file.

NOTE:

Steps 4, 5, and 6 are optional. POSITION FILE may only be used if the file is open for random access.

File Access Summary

To read from or write to a data file, the file must first be opened for access and its filename associated with a channel number. Once the file has been opened, records can be read from or written to it. The number of bytes of data that are read in or written out is determined by the size of the precision of the variable you supply. When reading or writing records, the file is referred to by its channel number rather than by its filename. If the file being read is a sequential file, then the end of file function will normally be used to determine when the end of the data has been reached. Once a data file has been processed, the filename must be disassociated from its channel number by a CLOSE [FILE] statement.

Random or contiguous files may also be processed sequentially. To do this, simply position the file pointer to byte 0 of the file, and then continue processing as if the file were organized sequentially. Remember that the file pointer moves with each read or write, and will be automatically positioned and ready to access the next record. You may also position the file pointer to any position within the file and read the file sequentially from that point.

If a file is random or contiguous, you may access individual records directly by their relative positions within the file. This is done by positioning the file pointer to the beginning of the record, or portion of a record, to be accessed. If records are fixed in length, then the algorithm

record number * record length

is used to position the file pointer. See the explanation of this procedure above.

Subfiles, Linked Files, and Index Files

The sequential, contiguous, and random access files described in the preceding sections are useful when data is not frequently added to and then deleted from a file, when records do not contain many different pieces of data, and when fewer than 16 files are simultaneously used by a single program.

The next sections describe the more complex methods of data file organization for subfiles, linked files, and index files.

Subfiles

Subfiles let you simultaneously use an unlimited number of files in the same program. (Sequential, contiguous, and random files are limited to 16 per program.)

A Business BASIC physical file may be divided into one or more subfiles. Each subfile has a fixed size and begins where the previous subfile ends. Figure 3-5 shows a file of inventory data that is divided into three subfiles for the different types of inventory items maintained: parts, perishable items, and items on order.

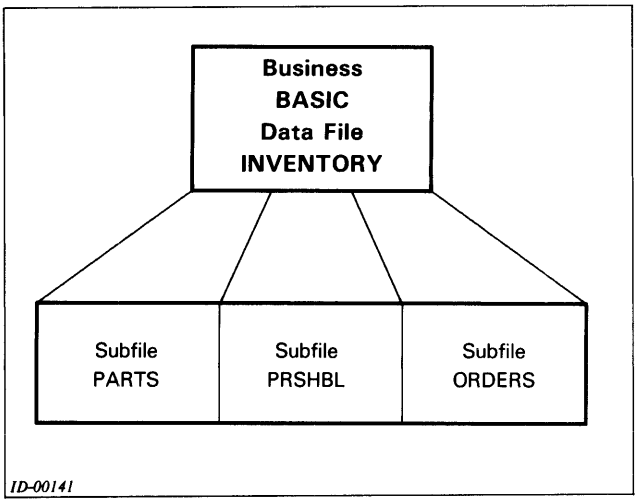


Figure 3-5 *Subfiles Within a Data File*

An unlimited number of subfiles may be processed in a single program. If you have 26 customer account files, for example, you could access each from the same program. Without subfiles, you would be continually closing and opening files.

The PARAM File

Since the operating system does not recognize subfiles, Business BASIC needs a way to find subfiles that you reference in your programs. To do this, Business BASIC uses the PARAM file.

The PARAM file contains information about the size and location of subfiles and is necessary for the use of subfiles, linked files, and ISAM files.

When a program requires information from the PARAM file, the information is extracted from the PARAM file and put in the file characteristics (C1) array.

The C1 array contains information used for computing the position of records within a subfile. The C1 array is two dimensional, with n rows and four columns, where n is the number of physical files and subfiles used in your program.

The PARAM file consists of 42-byte records. Table 3-3 describes the record structure. The first entry (record number 0) is reserved for information describing PARAM itself. Beginning with record number 1, there is one record for each file and subfile.

Starting Byte	Number of Bytes	Description
0	2	Status indicator, always equal to 1.
2	10	Subfile name or physical file name if not a subfile.
12	10	Filename of physical file that holds subfile, or filename of physical file if not a subfile.
22	4	Byte pointer to start of subfile or physical file.
26	2	Record length of subfile or physical file.
28	4	Last record number of subfile or physical file.
32	4	Number of last record containing data.
36	6	Unused.

Table 3-3 PARAM File Record Contents

An empty PARAM file is in the $LIB directory (or $LIB3 if you are using triple precision Business BASIC), which was supplied with your Business BASIC system. It contains data in record 0 followed by 100 blank records. You may use one PARAM file for all programs, or you may have as many as one PARAM file per directory. If you use more than one PARAM file, and use an AOS or AOS/VS system, be sure that the directory containing the PARAM file you want is on your searchlist. Also, be sure that the correct PARAM file is being used, or parts of your database may be destroyed.

To make a PARAM entry for a new file, use the INITFILE utility. INITFILE operates with a dialog in which you supply information describing your file in response to prompts. You may also use the Business BASIC file maintenance utility (described later in this chapter) or write the values to PARAM with output statements (see Table 3-1).

If you receive an error message indicating the PARAM file does not exist, you must create it as follows:

- Create the PARAM file using the CCONT, CRAND, or CREATE commands, or the DBGEN program described later in this chapter.
- Initialize record 0 using the file maintenance utility described later in this chapter.
- Use INITFILE and supply information describing each of your physical and subfiles.

A tutorial illustrating how to create a PARAM file and use the INITFILE utility is presented in Chapter 6.

Linked-Available-Record Format

Linked-available-record format (or dynamic record allocation) is a method of disk storage that allows a file to reuse space left by deleted records. Each deleted record contains a pointer that points to the next available record in the file. This storage method is an efficient use of disk space because it allows deleted record space to be reused.

Records of a linked file contain both user data and information used by Business BASIC for dynamic allocation. The first record of the file is a header, which contains information Business BASIC needs to process linked files. Each subsequent data record contains 2 bytes of information used by Business BASIC.

The first two bytes of a data record indicate whether or not the record has been deleted. If a record has not been deleted, the second through the last bytes contain user data. Counting begins with byte 0. Bytes 2 through 5 of a deleted record point to the next record available for receiving data. Thus, deleted records are chained together. Figure 3-6 shows an example of a deleted record chain.

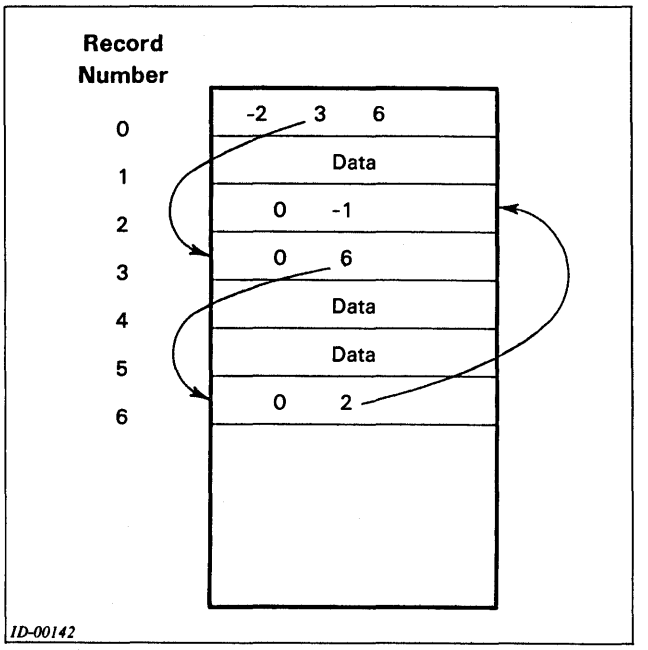


Figure 3-6 Deleted Record Chain for a Linked File

The header record (record 0) of a dynamically allocated file contains the information needed to use linked-available-record access. You do not need to access record 0 because the subroutines provided with Business BASIC for linked-available-record access do it for you.

Business BASIC supplies the GETREC.SL, DELREC.SL, and POSFL.SL subroutines to access record 0 and do the linked-available-record access. DELREC.SL deletes a record and puts it on the linked-available-record chain. GETREC.SL uses record 0 in a linked-available-record file to find the next available record. POSFL.SL positions you to any record or any byte within a record. You need the C1 array to use these subroutines. The C1 array is described later in this chapter.

ISAM Files

An indexed sequential access method (ISAM) file structure contains records with a unique key that allows selected records to be read quickly. The actual data for the ISAM file is contained in a database file. Another file, called an index file, contains the key and pointer to the record number in the database file. Usually, an index file and its corresponding data file are referred to collectively as an ISAM file.

In Business BASIC, an ISAM file consists of a data file and one or more index files. (When using the DBGEN and FM utilities a maximum number of three index files are allowed.) The index files contain keys that point to the data files and records within them. Consequently, you must perform I/O on a data file and at least one index file to accomplish ISAM access.

Index Files

An index file provides fast access to information in a data file independent of the information's physical location. An index file does not contain user data but instead contains keys, which point to entries in a data file.

Additional index file characteristics are listed as follows:

- Each index file contains only one type of key. For example, an index file could contain an overdue balance key that points to a customer accounts receivable data file.
- More than one index file may point to the same data file. To continue the previous example, a second index file whose keys identify dormant accounts could also point to the same customer accounts receivable data file.
- Duplicate key values may optionally be permitted to exist. These are keys with identical values that point to different data file entries.
- The key values are maintained in sorted order. Thus you may read sequentially through a data file by several different keys without the time-consuming task of sorting the data file. For example, you may read through an accounts receivable data file by zip code, by customer name, or by credit limit code.

The ISAM statements used with index files are: KADD (add a key), KFIND (find a key), KNEXT (find the next key), and KDEL (delete a key). These statements use block 0 of an index file, so you have to follow the appropriate conventions when creating your index file. The K commands are described in the Business BASIC Commands, Statements, and Functions manual.

Index files differ from data files in their internal makeup. Instead of being organized in records, they contain fixed-length blocks of 512 bytes each.

Block 0 of an index file is a header block that contains access information. Table 3-4 describes the contents of an index file's block 0. The keywords for creating and initializing index files write the required information to block 0.

Starting Byte	Number of Bytes	Description
0	2	Number of bytes per entry (key plus pointer).
2	2	Number of keys per block.
4	2	Last available block.
6	2	Next available block.
8	2	Number of level 0 block location.
10	2	Disk blocking factor (percentage). This factor is not in a percentage format in the header block. For example, if a disk blocking factor of 50% is specified, and the maximum number of keys per block is 36, the number 18 would be shown in the header block.
12	1	Bit flag for automatic file locking when the data entry is accessed.
13	1	Bit flag for allowing duplicate keys.
14	498	Unused.

Table 3-4 Contents of Index File Block 0

The blocks that make up an index file are created and maintained by Business BASIC when you use any of the keywords for creating and initializing index files or when you add a new key or delete an existing key with the KADD and KDEL keywords.

By definition a block always contains at least one available entry. When an addition uses the last available entry, the block is subdivided into two new blocks.

NOTE:

When adding keys with the KADD statement, and the block needs to split, the blocking factor is automatically 50%.

Creating and Initializing Index Files

Business BASIC provides one utility for calculating index file parameters, four for creating index files, and one utility and one subroutine for initializing them.

INDEXCALC computes and prints information for the index, including the maximum number of keys per index block, number of keys per block with the user specified blocking factor, number of blocks at level 0 (one two, etc. for as many levels as needed), number of sectors in the index, and number of sectors in the associated data file.

IBUILD creates an index file from a sorted data file or a sorted temporary file.

TBUILD creates a temporary (or tag) file from a linked (data) file or an index file. A tag file is sorted faster than

a data file, and thus facilitates creating an index file (which is maintained in sorted order).

XBUILD creates an index file with a disk blocking factor of approximately 75% from a linked-available-record data file. Use INDEXBLD to create an index file with a different blocking factor.

INDEXBLD creates an index file from an unsorted data file, tag file, or another index file, or re-creates an index file with a new blocking factor. INDEXBLD uses IBUILD, TBUILD, XBUILD, KADD, and QFILESORT.

INITFILE initializes an index file or linked file by writing the first record with header information, writing one or more additional records, and optionally making an entry in the PARAM file.

INITINDEX.SL (a subroutine) performs the same actions as INITFILE except one: it does not make an entry into the PARAM file.

Each utility or subroutine above is described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual. In addition, a tutorial illustrating the use of INITFILE and INDEXCALC is presented in Chapter 6.

File Characteristics (C1) Array

Besides the PARAM file and the index file (if you use one), there is another entity that stores information about your data files: the file characteristics (C1) array. Use the array with the file access routines GETREC.SL, DELREC.SL, and POSFL.SL.

The C1 array must be two dimensional with n-1 rows and 3 columns, where n is the number of logical files (subfiles and/or physical files) to be used in the program. Arrays begin with 0, not 1; therefore, if you have a C1 array with two rows, you can open three files. They will have logical file numbers 0, 1, and 2. The column dimensions correspond to the following:

0 Contains the channel number on which you will open your logical file. Your logical file must be a physical file, not a subfile. Logical file number 0 can also contain the channel number of the physical file on which you will open each subfile as a logical file. (Channel number is the number you associate with a file for all file access.)

1 Contains the byte offset to record 0 of the logical file (that is, the subfile starting location in the physical file), or 0 if the file is a physical file rather than a subfile.

2 Contains the file size; that is, the number of the last record in the logical file, whether it is a subfile or a physical file.

3 Contains the record size; that is, bytes per record of logical file, whether it is a subfile or a physical file.

Use the OPEN utility or the FINDFILE.SL subroutine (both described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual) to put information in your C1 array or you can use LET statements as in Figure 3-7. Both the OPEN utility and the FINDFILE.SL subroutine will "look up" the information you need from the PARAM file to build your C1 array.

In the sample program in Figure 3-7, the record size, channel number, or file size of any file can be changed by changing the values in the C1 array. With a C1 array, you can use the following access routines to modify a record:

- GETREC.SL to access an available record in order to write to it
- POSFL.SL to position to any record in the logical (subfile or physical) file
- DELREC.SL to delete any record in the logical file

```
0010 DIM C1(2,3)          :Dimension C1 to be a 3*4 array.
0020 LET C1(0,0)=2        :File 0 (SUB1) opened on channel 2,
0030 LET C1(0,1)=51200    :and begins at byte 51201 (block 100).
0040 LET C1(0,2)=1200     :and contains a max of 1200 records,
0050 LET C1(0,3)=48       :with 48 bytes in each record.
0060 LET C1(1,0)=2        :File 1 (SUB2) also opened on channel 2
                          :because SUB1 and SUB2 are in same
                          :physical file MASTER, opened on
                          :channel 2.
0070 LET C1(1,1)=0        :SUB2 begins at byte 1 of MASTER,
0080 LET C1(1,2)=400      :contains 400 records
0090 LET C1(1,3)=128      :with 128 bytes in each record.
0100 LET C1(2,0)=3        :File 2 (PHYS) opened on channel 3,
0110 LET C1(2,1)=0        :starts at byte 1 since it is a
0120 LET C1(2,2)=100      :physical file and contains a max of 100
0130 LET C1(2,3)=50       :records with 50 bytes in each record.
0140 REM -- NOW WE OPEN OUR FILES
0150 LET C%=C1(0,0)       :Channel number now in C%.
0160 OPEN FILE(C%,0),"MASTER":This will allow access to both SUB1
                          :and SUB2.
0170 LET C2%=C1(2,0)      :Channel number of PHYS now in C2%.
0180 OPEN FILE(C2%,0),"PHYS"
.
.
.
```

Figure 3-7 C1 Array

Building a C1 Array with FINDFILE.SL

FINDFILE.SL is a subroutine that builds a C1 array without opening your files. It leaves column 0 (channel number) blank and returns the next available channel number to you in a variable (C%). Its entry point is line 7800 (that is, GOSUB 7800). FINDFILE.SL gets the necessary information from the PARAM file. If a PARAM file does not exist, FINDFILE.SL asks you for the byte offset, record size, and file size, but it does not create a PARAM entry.

To use FINDFILE.SL, first supply FINDFILE.SL with the subfile name and the logical file number in response to its questions. Then create a subfile name in variable

X$. Your logical file number corresponds to the row in the C1 array (the logical file number 2 is put in row 2 of C1, logical file number 0 is put in row 0 of C1, etc.), and you assign your logical file number to variable F%. You can then GOSUB to FINDFILE.SL at line 7800. However, you must have entered FINDFILE.SL in your program (see ENTER in the *Business BASIC Commands, Statements, and Functions* manual).

When control returns to the main program, you will have a C1 array with the 0 column blank, a variable X$ with the physical filename for the logical file (subfile or physical file), and a variable C% with the next available channel number you can use with an OPEN FILE statement. Figure 3-8 uses FINDFILE.SL to fill the C1 array with the same values used in Figure 3-7.

```
0010 DIM C1(2,3)          :Dimension C1 to be a 3*4 array.
0020 DIM X$(10)           :Dimension X$ to max filename size.
0030 LET X$="SUB1"        :SUB1 is a subfile in MASTER,
0040 LET F%=0             :and is logical file 0.
0045 DIM T9$(42)          :Dimension T9$ to 42 bytes.
0050 GOSUB 7800           :Go to FINDFILE.SL subroutine.
0060 GOSUB 0200           :Go to verification routine.
0070 LET X$="SUB2"        :SUB2 is in MASTER,
0080 LET F%=1             :and is logical file 1.
0090 GOSUB 7800           :Go to FINDFILE.SL subroutine.
0100 GOSUB 0200           :Go to verification routine.
0110 LET X$="PHYS"        :PHYS is a physical file,
0120 LET F%=2             :and is logical file 2.
0130 GOSUB 7800           :Go to FINDFILE.SL subroutine.
0140 GOSUB 0200           :Go to verification routine.
0150 STOP
0200 REM -- VERIFICATION ROUTINE
0210 PRINT X$             :Prints name of physical file.
0220 LET C%=C1(F%,0)      :Assign channel number to 0 column.
0230 PRINT C1(F%,0),C1(F%,1),C1(F%,2),C1(F%,3)
0240 RETURN
```

Figure 3-8 *Filling the C1 Array*

You may also use the OPEN utility to find a channel for your file and to supply the information you need for the C1 array.

Positioning to a Record

When you OPEN a file with the OPEN utility, you put the channel number in the C1 array. The C1 array's row dimension corresponds to the logical file number. You position to the record using POSFL.SL, which requires the logical file number and two variables with assigned values in your program: F% for the logical file number, and R1 for the record number you want. Optionally, you can specify a byte location in the record R1 with the variable V%.

There are two entry points to POSFL.SL: you can GOSUB to 9610 to position to a record R1; or you can specify V% and GOSUB to 9612 to position to a byte within record R1, where V%=0 refers to the first byte of the record. After executing, POSFL.SL returns with three values: C% for the channel number of the file to be used with READ FILE or WRITE FILE; R9 for the byte position in the physical file at the start of record R1; and R8 for the byte position of the start of record R1 in the logical file (different from R9 if the logical file is a subfile). You can use R9 or R8 with a POSITION FILE statement, followed by a WRITE FILE statement, to do a fast rewrite of the record.

In Figure 3-9, we position to a record, read the record, and use POSITION FILE with R9 to go back to rewrite the record. Note that we did not delete the code that opens the files and fills the C1 array.

```
0010 DIM X$(512),C1(2,3),REC$(48)      :Record size of SUB2 is 48.
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$                    :Send logical file info into common.
0040 SWAP "OPEN"                       :OPEN will return X$ with C1 array.
0050 BLOCK READ X$                     :Retrieve info from common area.
0060 LET K=1                           :Pointer to first element in string.
0070 FOR I=0 TO 2                      :For each logical file, 0, 1 and 2,
0080   FOR J=0 TO 3                    :and for each dimension of C1 array,
0090     LET C1(I,J)=ASC(X$(K,K+3))    :extract element, put in C1 array.
0100     LET K=K+4                     :Bump pointer 4 bytes.
0110   NEXT J
0120 NEXT I
0130 INPUT "RECORD NUMBER OF SUB2 TO BE REWRITTEN: ",NUM
0140 LET R1=NUM                        :Give POSFL.SL a record number R1.
0150 LET F%=1                          :F% used by POSFL.SL for logical file.
0160 GOSUB 9610                        :Position to record R1 in file F%
                                       :using POSFL.SL, returns C%, R8 and R9.
0170 READ FILE (C%),REC$               :C% is channel number.
0180 DIM NEWREC$(48)                   :For new record.
.                                      :Code to build NEWREC$ for new record.
.
.
0210 POSITION FILE (C%,R9)             :Use R9 returned by POSFL.SL to position
0220 WRITE FILE (C%),NEWREC$           :and rewrite record R1.
.
.
.
```

Figure 3-9 *Positioning to a Record with POSFL.SL*

In Figure 3-10, we use POSFL.SL to position to a byte within record R1. V% is the byte position relative to 0 (that is, V%=0 is the first byte of the record). Because we want to read only the part of the record specified from byte position BYT to the end, we read the data into a substring rather than into the whole string. Then we set the substring's size to start at the byte location BYT plus 1 because records begin at byte 0 and strings begin at byte 1.

```
.
.
0130 INPUT "RECORD NUMBER OF SUB2 TO BE REWRITTEN: ",NUM
0135 INPUT "BYTE OFFSET, OR 0 IF BEGINNING OF RECORD: ",BYT
0140 LET R1=NUM
0145 LET V%=BYT
0150 LET F%=1
0160 GOSUB 9612  :Alternate entry to POSFL.SL if using V%
0170 READ FILE (C%),REC$(BYT+1,48)
.
.
```

Figure 3-10 *Positioning to a Byte with POSFL.SL*

Deleting a Record

DELREC.SL automatically updates record 0 and then deletes the record by setting its status (first 2 bytes) equal to 0. The deleted record then joins the chain of deleted and available records so eventually you can reuse that space. DELREC.SL requires the variables R1 and F% to be assigned as well as the C1 array. You enter DELREC.SL at entry point 8600. DELREC.SL does not return any values. It uses scratch variables X0%, Y0%, X0, Y0, and W0. You cannot use these variable names for other variables in your program. When DELREC.SL executes, it calls POSFL.SL to position to the desired record (see Figure 3-11).

Writing a Record

Use GETREC.SL to get an available record to write to. GETREC.SL finds an available record from the deleted record chain. It then updates record 0 and returns the record number to you in R1. Use this record number with POSFL.SL to position to the record before writing to it.

GETREC.SL allocates a new record in random files if the deleted record chain contains no more records. If you run out of deleted records in a contiguous file, and you've used up all the space allotted to the contiguous file, you have to copy your contiguous file into a larger contiguous file or to a random file.

Like DELREC.SL, GETREC.SL needs a variable, F%, for the logical file number and a C1 array. It uses the same scratch variables DELREC.SL uses, with the addition of the Z0 variable. Remember that you cannot use those variable names for other variables in your program. GETREC.SL returns the variable R1 with the record number you use with POSFL.SL. After executing GETREC.SL (8400 is the only entry point), execute POSFL.SL to position to the record using the R1 value returned by GETREC.SL. Then use WRITE FILE to modify the new record.

Figure 3-11 shows a partial update session. The record is deleted using DELREC.SL, and a new one is added using GETREC.SL. This is a good update technique if placement of new records does not matter. If you have an index to the data file, placement is unimportant as long as you KADD the new key to the index.

Adding a Key

After using GETREC.SL to get an available record, POSFL.SL to position to it, and WRITE FILE to write the record, use KADD to add the new index key that will point to the new record.

```
.
.
.
0130 INPUT "RECORD OF SUB2 TO BE DELETED: ",NUM
0140 LET R1=NUM
0150 LET F%=1                  :SUB2 is logical file 1 in C1 array.
0160 GOSUB 8600                :Go to DELREC.SL to delete record.
0170 GOSUB 8400                :Go to GETREC.SL to find next available record.
0180 GOSUB 9610                :Go to POSFL.SL using R1 returned by GETREC.SL.
0190 WRITE FILE (C%),NEWREC$   :C% is channel number returned by POSFL.SL.
.
.
.
```

Figure 3-11 Deleting a Record with DELREC.SL

KADD adds the key entry (string and record pointer) to the index file described in the descriptor string. KADD searches the index to find the proper location for the key. If you did not allow duplicate keys in the file and KADD finds one, it sends the record pointer variable back to you with a value less than or equal to 0 and does not add the new key.

Figure 3-12 shows a partial program that writes a new data record to an ISAM file. Both the index and data files are cataloged in the C1 array, and we use GETREC.SL and POSFL.SL to write the new data record.

```
0010 DIM DESC$(18),BUF$(544)    :Strings used with KADD.
.                               :Code for C1 array and OPEN utility goes here
.                               :MASTER and INDEX files are in C1 array
.                               :Fill DESC$ descriptor string here.
0200 DIM KEY$(10),REC$(48)      :Key length 10 bytes, record length 48 bytes
0210 INPUT "TYPE KEY: ",KEY$
0220 INPUT "TYPE DATA: ",REC$   :Input record data and key.
.                               :Code to validate data and key goes here.
.
.
0300 LET F%=2                   :Logical file 2 is data file.
0310 LOCK 2,"MASTER",0,C1(F%,3) :Lock record 0 of data file MASTER.
0320 GOSUB 8400                 :Go to GETREC.SL to get available record,
                                :record number returned in R1.
0330 UNLOCK 2                   :Unlock lock #2.
0340 GOSUB 9610                 :Position to record R1 using POSFL.SL.
0350 WRITE FILE (C%),REC$       :Write new record using C% returned by
                                :POSFL.SL.
0360 KADD DESC$,BUF$,KEY$,R1    :Add new key.
0370 IF R1<=0 THEN GOTO 0500    :If duplicates are not allowed and KADD found
.                               :one, or if an error occurred, KADD returns
.                               :R1 with a value of 0 or less.
```

Figure 3-12 Writing a Record and Adding a Key

The partial program in Figure 3-12 uses automatic locking for the index file and LOCK/UNLOCK for the data file. The DESC$ descriptor string and BUF$ buffer string were shown in previous examples. Note that we use KFIND to find the record pointer R1 for use in the program.

Finding a Key and Reading a Record

KFIND searches the index file (described in the descriptor string) for a match to the key you supply. If KFIND finds an exact match, it returns the data record pointer associated with the key. If KFIND cannot find the exact key, it finds a key with a value slightly greater than the specified key, and returns the negative value of the record pointer associated with the key found. Suppose you specify key ABC and there is no ABC, but two keys, ABCA and ABCB exist. KFIND will find the key ABCA, and a subsequent key KNEXT will find ABCB. If KFIND cannot find a key value equal to or greater than the key you supply, it will return a value of 0 for the record pointer.

After finding the key (the data record pointer associated with the key), use POSFL.SL (described above) to position to the record and READ FILE to read it. Figure 3-13 illustrates KFIND and READ FILE.

```
.                                   :Code to set up files,
.                                   :dimension strings, C1 array.
.                                   :DESC$ descriptor string must be filled
                                    :with information about the index file.
0500 INPUT "KEY FOR RECORD TO BE DELETED: ",KEY$
0510 KFIND DESC$,BUF$,KEY$,R1       :Find key and R1 record pointer.
0520 IF R1<=0 THEN GOTO 0700        :If record does not exist,
                                    :go to 700.
0530 LET F%=8                       :Logical file number for data file.
0540 GOSUB 9610                     :POSFL.SL uses F% and R1.
0550 LOCK 3,"MASTER",R8,C1(F%,3)    :R8 returned by POSFL.SL.
0560 READ FILE(C%),REC$             :C% returned by POSFL.SL.
.                                   :Do record checking here.
.
.
0600 KDEL DESC$,BUF$,KEY$,R1        :Delete key entry from index file.
0610 LOCK 4,"MASTER",0,C1(F%,3)     :LOCK record 0 of data file.
0620 GOSUB 8600                     :Delete record from data file.
0630 UNLOCK                         :Safe to unlock both deleted record and
.                                   :record 0.
.
.
```

Figure 3-13 *Finding a Key and Reading a Record*

Finding the Next Key and Reading the Next Record

To find the next key in sequence, use KNEXT after KFIND. The first KNEXT will not work unless you do a KFIND before it. However, you can do subsequent KNEXTs after the first one without doing a KFIND each time. KNEXT needs the buffer string (BUF$ in these examples), so do not change the buffer string between KFIND and KNEXT or between KNEXTs.

KNEXT returns the record pointer associated with the key found. If KNEXT reaches the end of the index, it returns a 0 for the record pointer.

Use KNEXT to read an index sequentially from a given key. First use KFIND with a null key to return the first key in the index. Then use KNEXTs to reach the key you want. You can also use KNEXT to find all occurrences of duplicate keys.

After reaching the desired key, use POSFL.SL to position to the data record indicated by the record pointer, and do a READ FILE to read the record found. Figure 3-14 illustrates this procedure.

When you access two index files using KNEXTs, you need a separate buffer string (BUF$ in this example) for each index file. KADD, KDEL, KFIND, and KNEXT are all described in the Business BASIC Commands, Statements, and Functions manual.

```
.                                   :Code to set up C1 array, use OPEN
utility.
.                                   :set up DESC$ string, etc. goes here.
.
0100 LET KEY$=""                    :Null key to get first entry in index file.
0110 KFIND DESC$,BUF$,KEY$,R1       :Find first key entry.
0120 LET R1=ABS(R1)                 :Negative R1 expected for approximate match.
0130 GOTO 0150                      :Process key and pointer for first record
                                    :before calling KNEXT.
0140 KNEXT DESC$,BUF$,KEY$,R1       :Get next key in sequence.
0150 IF R1=0 THEN GOTO 0600         :If at end of index, go to 600.
0160 LET F%=3                       :Logical file number for data file.
0170 GOSUB 9610                     :POSFL.SL uses F%, positions to data record
                                    :R1.
0180 READ FILE (C%),REC$            :C% returned by POSFL.SL; read record.
.                                   :Process record here.
.
.
0200 GOTO 0140                      :Go back to KNEXT to get next key.
.
.
.
```

Figure 3-14 *Finding the Next Key and Reading the Next Record*

Block Input/Output

Use BLOCK READ FILE and BLOCK WRITE FILE to transfer 512-byte blocks (sectors). The size of the variable you use with a block statement determines the number of blocks transferred. Your variable must be able to hold at least 512 bytes -- that is why you can use only string variables and arrays with BLOCK READ and BLOCK WRITE.

Block I/O is faster than conventional I/O because it uses no operating system buffer. Since regular I/O uses buffers, you cannot mix block and regular I/O. Block I/O can use random (direct) access. In the BLOCK READ FILE or BLOCK WRITE FILE statement, specify the block number in the file where the I/O operation will start. You have to read/write sequentially from that point. However, you can also perform a sequential access from the first block in the file. Remember, GPOS (the statement that returns the current file pointer position) does not work when you do block I/O.

BLOCK READ FILE reads into a string variable or array as many full blocks as the string variable or array can hold. If you previously DIMensioned string variable A$ to 1536 bytes, BASIC transfers 3 blocks; for a 1535-byte string variable, BASIC transfers only 2 blocks.

Array elements can hold 4 bytes each, so array A needs to be DIMensioned to at least 128 items for 1-block transfers, 255 for 2 blocks, etc. Two-dimensional arrays are allowed; that is, array A with 128 elements could be DIMensioned as A(3,31) with 4 rows and 32 columns (arrays begin at element 0 -- row 0, column 0). If the string variable or array is larger than the amount of data in the block, it will be filled with nulls. These rules also apply when you write strings or arrays to a file with a BLOCK WRITE FILE statement.

BLOCK READ and BLOCK WRITE (that is, without the keyword FILE) are special cases of block I/O that write to or read from a common area set up for each terminal. The length of the common area for each job is 1 block (512 bytes) and is always open and ready for use. You can put data into the common area with a BLOCK WRITE or BLOCK WRITE FILE (16) statement, since channel number 16 is a dummy channel for your terminal. You can then SWAP, CHAIN, or RUN a new program, and your data is still there in the common area for you to retrieve with a BLOCK READ or BLOCK READ FILE (16) statement. This is the easiest way to pass information to other programs. But you can pass only 512 bytes of information at a time, and the information must be in the form of a string or array. You should always fill the strings or arrays with nulls when you write them (if smaller than 512, fill to 512). You will always need a 512-byte string or array to read to the common area, and if the data is smaller than 512, BLOCK READ will pad the string or array on the right with nulls.

In Figure 3-15, the common area is used to pass information to and retrieve file information from the OPEN utility program. In Figure 3-16, the file IN is copied to the file OUT by transferring 16 blocks at a time. More information about BLOCK READ and BLOCK WRITE can be found in the *Business BASIC Commands, Statements, and Functions* manual.

```
0010 DIM X$(512)                    :For one block.
0020 LET X$="SUB1,5,SUB2,5,PHYS,6";FILL$(0)
0030 BLOCK WRITE X$                 :Zip-- into common area.
0040 SWAP "OPEN"
0050 BLOCK READ X$                  :We read what OPEN sent to common area.
.
.
.
```

***Figure 3-15** Passing Information Through the Common Area*

```
0010 DIM A$(8192)                   :For 16 blocks.
0020 OPEN FILE(0,4), "IN"           :Open files.
0030 OPEN FILE(1,0), "OUT"
0040 LET B=0                        :Initialize block number.
0050 BLOCK READ FILE(0,B),A$        :Read up to 16 blocks starting at B.
0060 BLOCK WRITE FILE(1,B),A$(1,LEN(A$)) :Write blocks just read.
                                    :but only write current length of A$ in
                                    :case we have less than 16 blocks of data
0070 IF EOF(0) THEN GOTO 0100       :Test for end of file
0080 LET B=B+16                     :Bump starting block number.
0090 GOTO 0050                      :Go back and read 16 more blocks.
0100 CLOSE                          :If at end of file, close all files.
0110 STOP
```

***Figure 3-16** Transferring Blocks of Data*

Simplifying ISAM: The FM and DBGEN Programs

The commands and utilities described above provide a great deal of flexibility in the design and use of index files. However, they can become complicated. Business BASIC provides two programs designed to save you the trouble of writing programs to maintain an ISAM database.

The File Maintenance (FM) and Data Base Generator (DBGEN) programs, while powerful, have certain limitations. FM does not allow checking the validity of entries, and since FM is a generalized program, it may run more slowly than programs you could write for your particular purposes. The DBGEN program has similar limitations. Both the FM and the DBGEN programs limit you to three indexes per data file.

FM

FM is an interactive program written in Business BASIC that allows you to read, add to, and delete from data files, and automatically adjusts up to three index files associated with each data file. In addition, FM allows you to generate quick reports on the contents of a file or some portion of a file.

To perform these operations, FM requires information about the data files and associated index files: the names of the files, the size and number of the records in each of these files, and so forth. This information is stored in a table file; one table file is required for each data file used with the FM program.

When you run FM, the program gets information about the files you wish to maintain from the PARAM file. Therefore, data and index files must be initialized before

using FM to maintain them. Each data file requires a table file, which can be created interactively with FM.

With table files in order, FM proceeds by displaying screens of information and requesting input. Many responses require only a single keystroke.

The FM utility is further described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

DBGEN

DBGEN is an interactive program written in Business BASIC. It prompts you for the information needed to create a database of one data file with up to three index files, and then initializes the files and constructs a table file automatically. The DBGEN program is further described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Constructing an ISAM Database

To construct and use an ISAM database you must:

- Decide on the data record format — how many fields per record, and how many bytes per field — and the number of data records you need. You must also decide what your index entries will be and how many indexes you need.
- Create and initialize a PARAM file (add record 0 to the PARAM file with FM) if one does not exist.
- Create and initialize the data file and index file(s), and make entries for them in the PARAM file. You must also update the "highest record used" in PARAM record 0.
- Fill in the index and data files; that is, put the value for each key and data record in the appropriate file.
- Examine, print, and change the values for keys and data records.

Business BASIC provides three ways to perform the programming steps described above.

(1) For the greatest amount of flexibility, use the K keywords (KFIND, KNEXT, KADD, KDEL) and the linked file commands discussed earlier in this chapter. These enable you to create a database to your exact specifications, and to use and modify it however necessary.

(2) To save programming time, use the DBGEN program to create a database of one data file with up to three index files. Then use FM or the K keywords to build the files.

(3) You can use the FM program to save the trouble of writing a program to use and maintain an existing database.

Table 3-5 lists the steps to follow and the Business BASIC keywords used to create an ISAM database. Each step can be accomplished in one of at least two ways. In the table, keywords that have the same net effect are separated by the word or, keywords that are used together are grouped together, and those that accomplish the same step in different ways are listed in the same column. As you become proficient in using ISAM files you will appreciate the flexibility these keywords offer.

NOTE:

If you create a database using the commands listed in Table 3-5, place index files and their associated data files in different directories, and if possible in different disk devices. This will improve response time.

Rebuilding ISAM Files: ICOMPRESS

The ICOMPRESS utility is used to restructure an index file.

When you add and delete keys from an index file, the multileveled ISAM key structure dynamically expands, but it does not dynamically contract. Even though Business BASIC uses space created by deleted keys in the same range, after continued adding and deleting of keys the file may become full, even though there seems to be ample room for the key entries in the index file.

This problem occurs only when a large number of keys are deleted and are replaced with new keys that have a different range of values than the deleted entries. If the keys deleted and replaced are of random nature, the problem would not occur. Although an empty block would not be reallocated, it would be reused by new keys of the same range as the old ones.

ICOMPRESS solves this problem by compressing the keys and creating a new random file. This new file can be the original (therefore it would be overwritten) or another file (leaving the original as a backup). Make a backup copy of the original index file on tape or diskette before running ICOMPRESS.

ICOMPRESS can also be used to free up disk space for adding more keys to the ISAM key structure of a file. This use of the ICOMPRESS utility is discussed in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Rebuilding Linked-Available-Record Files: RELINK

If you have a data file with linked-available-record format whose deleted-record chain is destroyed, the RELINK utility should be used to re-create the deleted-record

Step	Keywords	Comments
Create PARAM file	CRAND or OPEN FILE	You must build record 0 using FM.
	DBGEN	If PARAM file is needed, DBGEN automatically creates it and fills record 0.
Create and initialize data file	CRAND or OPEN FILE	Requires subsequent use of INITFILE to initialize.
Calculate index file parameters	INDEXCALC	Returns information needed to initialize index.
Amend PARAM, create and initialize data file	INITFILE	Creates, initializes file; makes PARAM file entry. INITFILE requires information from INDEXCALC.
	INITINDEX.SL	Subroutine that requires information from INDEXCALC.
	DBGEN	Calculates parameters and initializes index. Does not require INDEXCALC.
Build table file for FM	TABBUILD MOVETABREC file I/O commands	These commands can be faster than FM in some cases.
	FM	Interactively constructs table file.
	IBUILD	Builds index from sorted index or data file.
Construct index	INDEXBLD	Builds an index interactively.
	XBUILD	Builds index from a data file.
	KADD, KDEL, KFIND, KNEXT	Together these commands allow you to build an index, key by key.
	FM	Automatically constructs index from a data file.
Construct data file	File I/O commands	Listed in Table 3-1.
	FM	Interactively constructs data file, key by key.
Access and modify database	Commands to construct indexes and data files	Requires extensive programming to locate records and update keys.
	FM	Interactively finds records and updates data files; automatically updates indexes.

Table 3-5 Steps and Keywords Used in an ISAM Database

chain. RELINK also should be run if a system crash occurs while a data file with linked-available-record format is being updated.

RELINK is a utility used to re-create a deleted-record chain for a data file. RELINK is described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual. The RELINK utility searches the PARAM file for the logical filename you give it. If it does not find a PARAM entry, RELINK asks you for the byte offset in the physical file to record 0, the record size, and the maximum number of records. RELINK then searches for records with a 0 status (first 2 bytes of the record equal 0) and puts them on the new deleted-record chain.

INFOS Files (AOS and AOS/VS only)

INFOS® is an indexing system that organizes and manages records of information. AOS and AOS/VS Business BASIC provides an interface to the AOS INFOS system. This system is discussed in the *INFOS® System User's Manual* (093-000152).

INFOS and Business BASIC

You cannot create an INFOS file with AOS or AOS/VS Business BASIC; you must use the AOS INFOS utility ICREATE.

After you create the INFOS II files, you can use Business BASIC INFOS II statements to manipulate the INFOS II files with all the functionality described in the *INFOS® System User's Manual.* We describe the INFOS II interface, which consists of these statements, below. For a complete description of each statement, see the *Business BASIC Commands, Statements, and Functions* manual.

Statements

The Business BASIC INFOS II interface consists of a set of statements that behave like regular Business BASIC commands, though their format is somewhat different. You can use these statements as commands and program

statements. Table 3-6 lists the INFOS II statements with brief descriptions.

Statement	Description
DBCLOSE	Closes an open INFOS II file.
DBDELETE	Deletes a key and/or record.
DBGET	Gets values returned by INFOS II.
DBOPEN INFOS	Opens an INFOS II index.
DBREAD	Reads from an INFOS II file.
DBREINSTATE	Reinstates a logically deleted record.
DBRELEASE	Releases locks and/or position.
DBRETRIEVE HIGHKEY	Retrieves the high key.
DBRETRIEVE KEY	Retrieves the key.
DBRETRIEVE SIDEF	Retrieves a subindex definition.
DBRETRIEVE STATUS	Returns the INFOS II status.
DBREWRITE	Rewrites an INFOS II database record.
DBSET	Permanently sets an INFOS II parameter.
DBSUBINDEX DEFINE	Defines a subindex.
DBSUBINDEX DELETE	Deletes a subindex.
DBSUBINDEX LINK	Links a subindex.
DBSUBINDEX LINKINIT	Initializes a link subindex string.
DBSUBINDEX LINKSET	Sets parameters in link subindex string.
DBWRITE	Writes to an INFOS II file.

Table 3-6 INFOS II Statements

Argument Pairs

Each INFOS II statement has a set of arguments called an argument pair. Argument pairs, comparable to switches, alter the statements. Some argument pairs are required for particular statements, while others are optional or not applicable. Some argument pairs are:

ACCESS=REL means relative access

DUPKEY=NO means the specified key is not a duplicate

MOTION=BACK means the direction of relative motion is backward

The INFOS II statements are designed to allow easy access to INFOS files. Where possible, the statements take default values for argument pairs. You can redefine the default values by using the DBSET statement.

Creating an INFOS II File

When you create an INFOS II file, two logical structures are defined: an index and a database. INFOS II automatically links these two structures to form a single ISAM file and, using AOS or AOS/VS, allocates space and physically constructs the file.

To create an INFOS II file, use the AOS INFOS II utility ICREATE. You must execute ICREATE from the CLI. Type:

ICREATE "filename"

where **filename** is the name of the index file. If you do not specify the filename, INFOS II prompts you for it. Figure 3-17 shows a sample ICREATE dialog. See the *INFOS® System User's Manual* for a complete discussion of ICREATE.

```
****** INFOS FILE CREATION      6/16/82 9:54:15 ******

NAME OF FILE TO BE CREATED: ACCOUNTS
ACCESS METHOD (I=ISAM, D=DBAM) [D] I

****** DEFINE INDEX FILE ******

        PAGE SIZE (BYTES) [2048]: 2048
   ROOT NODE SIZE [2042]: 2042
   MAXIMUM KEY LENGTH [255]: 10
   ALLOW DUPLICATE KEYS IN THIS INDEX? (Y OR [N]): N
   ENABLE SPACE MANAGEMENT? (Y OR [N]): N
   ENABLE KEY COMPRESSION? (Y OR [N]): N
   OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]): N

   ****** DEFINE INDEX VOLUME(S) ******

          NUMBER OF VOLUMES TO DEFINE [1]: 1
     VOLUME 1 NAME [VOL01]: VOL01
          SPECIFY MAXIMUM SIZE? (Y OR [N]): N
          SPECIFY FILE ELEMENT SIZE? (Y OR [N]): N

****** DEFINE DATABASE FILE ******

        DATABASE FILE NAME [ACCOUNTS.DB]: ACCOUNTS.DB
   PAGE SIZE (BYTES) [2048]: 2048
   ENABLE SPACE MANAGEMENT? (Y OR [N]): N
   ENABLE DATA RECORD COMPRESSION? (Y OR [N]): N
   OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]): N

   ****** DEFINE DATABASE VOLUME(S) ******

          NUMBER OF VOLUMES TO DEFINE [1]: 1
     VOLUME 1 NAME [VOL01]: VOL01
          SPECIFY MAXIMUM SIZE? (Y OR [N]): N
          SPECIFY FILE ELEMENT SIZE? (Y OR [N]): N
```

Figure 3-17 ICREATE Dialog

Accessing INFOS II Files

To access an INFOS II index file, you must open it using the DBOPEN statement from keyboard mode. Type

```
* DBOPEN INFOS "filename",string$
```

where filename is the name of the INFOS II file (named with ICREATE), and string$ is the name of the channel on which the INFOS II file is opened.

The string is called the channel string. It is similar to the channel number used in Business BASIC statements (such as PRINT FILE, READ FILE, WRITE FILE). DBOPEN initializes the channel string, which you use

throughout the program to refer to the INFOS II file. The following example illustrates DBOPEN.

To open an INFOS II file called ACCOUNTS on channel string MASTER$, type

```
* DBOPEN INFOS "ACCOUNTS", MASTER$
```

Now you can use any INFOS II statement to manipulate the file.

Two common statements are DBWRITE and DBREAD. DBWRITE writes records and/or keys to an INFOS II file; DBREAD reads records and/or keys from an INFOS II file.

To write a record to ACCOUNTS, type

```
* DBWRITE MASTER$
```

where MASTER$ is the channel string referring to the INFOS II file ACCOUNTS. DBWRITE rewrites a record to an INFOS II database.

To read a record from ACCOUNTS type

```
* DBREAD MASTER$
```

where MASTER$ is the channel string referring to the INFOS II file ACCOUNTS.

To read the previous record in ACCOUNTS and store it in a string variable called MDATA$, type

```
* DBREAD MASTER$, ACCESS=REL, MOTION=BACK, REC=MDATA$
```

ACCESS=REL indicates relative access (versus keyed). MOTION=BACK indicates that the direction of relative motion is backward because the system reads the previous record. MOTION can be used with relative access only. REC=MDATA$ indicates that MDATA$ is the record string because it receives the data from the record that the system reads.

Other common INFOS II operations include deleting keys (DBDELETE) and reinstating logically deleted records (DBREINSTATE). You may also wish to perform more advanced INFOS II features. These include defining subindexes (DBSUBINDEX DEFINE), retrieving subindex definitions (DBRETRIEVE SIDEF), and linking subindexes (DBSUBINDEX LINK). Two INFOS II statements, DBSET and DBGET, are not directly related to any particular INFOS II function. DBSET lets you set default values for subsequent program statements; DBGET retrieves values returned by previous statements.

Assembly Language Subroutines

One advantage of Business BASIC is its ability to call assembly language subroutines and, therefore, its more direct interaction with the operating system. You can call assembly language subroutines if your system manager included them in a file called USERSUBS.RB when he generated your system. The USERSUBS.OB (AOS and AOS/VS) or the RDOS/DOS macroassemblers generate executable machine language code.

Assembly language subroutines perform tasks that Business BASIC cannot do. Improper use of assembly language subroutines, system calls, or task calls can crash the system. Only assembly language programmers who have learned RDOS/DOS, AOS, or AOS/VS assembly language should use this interface.

You must submit all assembly language subroutines to the system manager at system load time. In RDOS/DOS, the subroutines should be in the file USERSUBS.RB. In AOS and AOS/VS, the subroutines should be in the file USERSUBS.OB (or filename.OB, where filename is the name for your subroutine file). The system inputs this file to the relocatable loader when it creates the BASIC executable file. With your subroutines, you must include a subroutine table that has the entry point SBRTB.

The subroutine table is a list of all assembly language subroutines available to a BASIC program. For each assembly language subroutine, the interface requires a four-word list containing the subroutine number, the subroutine entry point, the number of arguments, and the argument control word. We describe the subroutine table with the UCALL statement/command in the *Business BASIC Commands, Statements, and Functions* manual. Figure 3-18 shows a call to an assembly language subroutine, the subroutine table, and the subroutine itself.

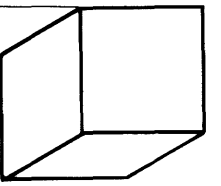
```
0010 REM         PROGRAM TO TEST UCALL NUMBER 1:
0020 REM         SINGLE PRECISION CUBE GENERATOR
0030 LET TIMES=0
0040 LET CUBE=0
0050 FOR N=1 TO 29
0060   UCALL 1,N,CUBE
0070   IF CUBE<>N*N*N THEN PRINT "ERROR!!!!"
0080 NEXT N
0090 FOR N=30 TO 40
0100   UCALL 1,N,CUBE
0110   IF CUBE<>0 THEN PRINT "ERROR!!!"
0120 NEXT N
0130 LET TIMES=TIMES+1
0140 PRINT "I'VE TRIED ";TIMES;" TIMES"
0150 GOTO 0040
```

Assembly language subroutine number 1 could be the following:

```
                .TITLE  USERSUBS          ; BASIC assembly subroutines
                        .ENT    SBRTB             ; Entry point: SBRTB
                        .NREL                     ; Normal relocatable code
                        .ENTO   OCALL

; Subroutine Table

SBRTB:          1                                 ; Subroutine number
                CUBE                              ; Subroutine entry point
                2                                 ; Number of arguments
                2B1+3B3                           ; 1st arg may be exp:
                                                  ; 2nd must be a variable
                -1                                ; End of table.

; The following re-entrant subroutine will return the value of the
; specified number cubed, at single precision machine language speed.

; The number in the expression must be in the range between 1 and 29
; inclusive, or a zero will be returned to represent an error state.

; Recall that when our routine is entered, AC2 will contain a pointer
; to the argument list and the return address that follows it


CUBE:           LDA             3,0,2             ; 3 -> value to be cubed
                LDA             0,1,2             ; 0 -> return variable
                STA             0,TR1             ; Save the pointer
                STA             2,TR2             ; Address of arg 2
                ISZ             TR2
                ISZ             TR2               ; Return address
                LDA             0,0,3             ; High order of value

                MOV#            0,0,SZR           ; Must equal zero
                JMP             ERROR
                LDA             0,1,3             ; Low order of value
                LDA             1,D29             ; Max low order value
                SUBZ#           0,1,SNC           ; Must be <= 29
                JMP             ERROR
                MOV             0,1               ; 1 gets factor
                MOV             0,2               ; So does 2
                SUB             0,0               ; Clear mul offset
                MUL                               ; Square
                MUL                               ; Cube
EXIT:           LDA             2,TR1             ; 2 -> return variable
                STA             1,1,2             ; Store low order
                STA             0,0,2             ; Store high order (ie 0)
                JMP             @TR2

D29:            29.

ERROR:          SUB             0,0               ; Clear 0
                SUB             1,1               ; Clear 1
                JMP             EXIT

                END
```

Figure 3-18 *Example of a UCALL Subroutine*

Chapter 4
Data and Computations



Business BASIC provides two types of data: string and numeric. String data may be any character or combination of characters that the keyboard can generate; for example, letters, digits, and special characters. Numeric data consists of integers.

Although numeric data is limited to integers, Business BASIC also has formatting techniques that allow numeric precision to be maintained and decimal points printed in numbers.

Numeric Data

A numeric constant (also called a numeric literal) is written as a signed or unsigned decimal number within the allowable range. Neither commas nor periods are permitted. Examples of numeric constants are:

59
-771083
+941

Numeric data also includes numeric variables, arrays, and expressions.

Numeric Variables

A numeric variable is a data item to which a numeric value is assigned during program execution. Numeric variables, also referred to as simple numeric variables, are denoted by a single letter or a letter followed by one to five letters or digits. Examples include:

A
A3
NUM1
OUTPUT

The percent sign and the pound sign (#) have special meaning when either is the last character in a variable name. On input or output operations, a numeric variable may be restricted to transferring 16 bits instead of 32 by ending its name with a percent sign. However, such a variable still occupies one 32-bit word.

Numeric Arrays

An array is an ordered set of values. Each member of the set is called an array element.

An array may have one or two dimensions. Elements of one-dimensional arrays form rows; elements of two-dimensional arrays form rows and columns. Figure 4-1 illustrates one- and two-dimensional arrays.

You reference an array element by specifying the array's name followed by one or two subscripts enclosed in parentheses or square brackets. Subscripts are variables or expressions that identify a specific row and/or column position. Note that the first element of a one-dimensional array and the first column of a two-dimensional array are always referenced with a subscript of zero.

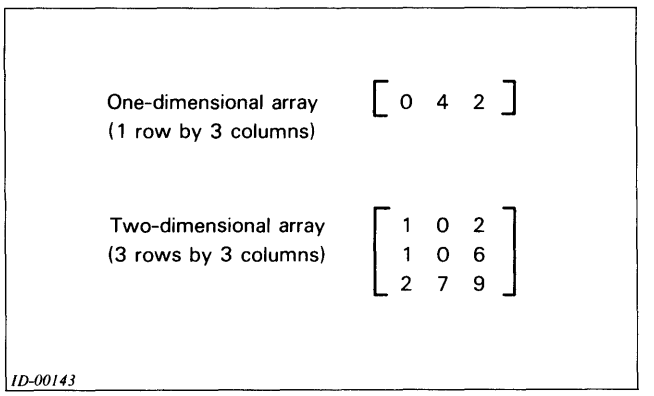


Figure 4-1 *One- and Two-Dimensional Arrays*

To use an array in a program, you must declare its name, the number of dimensions, and the number of elements in a dimension statement. The DIM command declares a numeric array and is described in the *Business BASIC Commands, Statements, and Functions* manual.

Default Dimension Declaration

You may use an array without declaring it and have default dimension values assigned by Business BASIC. An undeclared one-dimensional array is automatically

allocated 11 elements; an undeclared two-dimensional array is automatically allocated 11 rows and 11 columns, or 121 elements. You will conserve storage by using a dimension statement to explicitly declare an array requiring fewer than the default number of elements.

Referencing Array Elements

Each of the elements of an array is identified by the name of the array followed by a subscript enclosed in either parentheses or square brackets. a subscript indicates the location of the element within the array and may be a number, a numeric variable, or an expression. A subscript must evaluate to a number between zero and the value declared in the array's dimension statement. That value serves as the upper bound the subscript may assume.

One-dimensional array elements are referenced with a single subscript. For example, elements of an array declared as DIM B(5) are referenced as:

B(0), B(1), B(2), B(3), B(4), B(5)

Two-dimensional arrays are referenced with two subscripts separated by commas. For example, some elements of an array declared as DIM E(24,5) are :

E(I-3,5)
E(O,J*K)
E(24,RND(6))

In the third example above, RND(6) invokes a system-supplied function to compute a random number between zero and five (the RND function is described in detail in the *Business BASIC Commands, Statements, and Functions* manual).

Redimensioning During Program Execution

You may "redimension" a previously declared array by issuing an immediate mode dimension statement with different dimension values. An array can be redimensioned only to the same or fewer number of elements. For example, you may redimension an array declared as B(2,3) to B(1,2) or B(3,2), but not to B(3,3).

Redimensioning alters the way in which you reference array elements but does not alter the amount of storage occupied by the array. You may access elements of a redimensioned array using different subscripts. For example, you might redimension a two-dimensional array to have only as many elements as a single row and then use a single subscript instead of a double one to access those elements. Subscript references outside the newly defined range of subscripts cause an error.

Numeric Expressions

A numeric expression is any combination of numbers, numeric variables, array elements, and numeric functions linked together by arithmetic operators and parentheses. Table 4-1 summarizes the Business BASIC numeric operators.

Operator	Operation	Example
+	addition	A(4) + 9
-	subtraction	M - B(INDEX)
*	multiplication	C * 6
/	division	X/Y(COUNT +2)
^	exponentiation	Z^3

Table 4-1 Business BASIC Numeric Operators

A Business BASIC numeric expression containing several operators is evaluated according to the precedence rules given in Table 4-2.

Sequence	Operation
1	unary minus or plus
2	exponentiation
3	multiplication
4	addition and subtraction

Table 4-2 Rules of Numeric Operation Evaluation

When two operators are of equal precedence, evaluation proceeds from left to right. For example:

Z-A + B*C^D

1. C^D is evaluated.
2. B is multiplied by the result of step 1.
3. A is subtracted from Z.
4. The result of step 2 is added to the result of step 3.

You may change the order of evaluation by enclosing expressions in parentheses. An expression in parenthesis evaluated first. If parentheses are nested, the innermost parenthetical operation is evaluated first. For example:

Z-((A+B)*C)^D

1. A+B is evaluated.
2. The result of step 1 is multiplied by C.
3. The result of step 2 is raised to the power D.
4. The result of step 3 is subtracted from Z.

Handling Decimals

All numbers in Business BASIC are 4- or 6-byte integers, depending on your system's precision. To accomplish decimal arithmetic you should use the following procedures:

- To add decimals, multiply by an appropriate power of 10.
- To drop decimals, divide by an appropriate power of 10.
- To multiply, remember that the number of decimal places in the multiplier, plus the number of decimal places in the multiplicand equals the number of decimal places in the product.
- To divide, remember that the dividend (the top number) must have more decimal places than the divisor (the bottom number) -- as many as the number of decimal places that are needed in the quotient (or answer).

```
: X=.73 and Y=47.6
:
0010 DATA 73,476,73,476
0020 READ X,Y
0030 LET X=X+Y*10                    :for 2 decimal places
0040 PRINT USING "E6.2,T10,Z",X
0050 READ X,Y
0060 LET Y=(X+(5*SGN(X)))/10+Y       :for 1 decimal place
:
:Note the rounding that adjusts for the correct sign
:    -73.65 rounded = -73.7 not -73.6
:     88.05 rounded = 88.1 not 88.0
:
0070 PRINT USING "E5.1",Y
```

Figure 4-2 *Adding and Dropping Decimals*

```
: X=.73 and Y=47.6
: X * Y = 34.748
:
0010 LET X=73
0020 LET Y=476
:
: Z will have 1 decimal place
:
0030 LET Z=((X*Y) + (50*SGN(X))*SGN(Y))/100
0040 PRINT USING "E6.1,T10,Z",Z
:
: Z will have 2 decimal places
:
0050 LET Z=((X*Y)+(5*SGN(X))*SGN(Y))/10
0060 PRINT USING "E6.2",Z
```

Figure 4-3 *Multiplying Decimals*

```
: X=64.73  Y=47.6
: X/Y=1.78
: Y/X= .561
: Z will always have 1 decimal place
:
0010 LET X=6473
0020 LET Y=476
0030 LET Z=X/Y                                    :unrounded
0040 PRINT USING "E6.1,T10,Z",Z
0050 LET Z=((X*10)/Y)+SGN(X)*SGN(Y)*50/10         :rounded
0060 PRINT USING "E6.1,T10,Z",Z
0070 LET Z=Y*100/X                                :unrounded
0080 PRINT USING "E6.1,T10,Z",Z
0090 LET Z=(Y*1000/X) + (SGN(X)*SGN(Y))*500/10    :rounded
0100 PRINT USING "E6.1",Z
:
: Note that in the unrounded examples above, the answer
  is not correct
: If any sort of accuracy is
  involved it is important to round
```

Figure 4-4 *Dividing Decimals*

An example of adding and dropping decimals is presented in Figure 4-2; multiplying decimals is presented in Figure 4-3. Figure 4-4 gives an example of dividing decimals.

Predefined Numeric Instructions

The Business BASIC functions and subroutines for performing numeric computations are summarized in Table 4-3. The functions may be used in any Business BASIC statement permitting a numeric expression and are described in the *Business BASIC Commands, Statements, and Functions* manual or in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

String Data

A string is a combination of characters that may include letters, digits, spaces, binary values, and special characters. A string constant (also called a string literal) is written within quotation marks as in this example:

"Data General Corporation"

All blank spaces within the quotation marks are significant. The delimiting quotation marks are not printed if the string literal is output.

You may include a special character or control code in a string literal by enclosing its ASCII decimal equivalent value in angle brackets in the form <n>, where n is between 0 and 255. The angle brackets do not appear when a string containing the ASCII character is output. For example:

"AT THE SOUND OF THE TONE <7>"

causes the string "AT THE SOUND OF THE TONE" to be displayed and the terminal bell to ring because 7 is the ASCII code for bell. Appendix E lists ASCII codes.

Format	Usage	Description
ABS	function	Compute absolute value of a number.
AND	function	Compute logical AND of two numeric expressions.
ASC	function	Use equivalent ASCII numeric value for a string.
LEN	function	Compute number of characters currently in a string.
MAX	function	Find larger of two numbers.
MIN	function	Find smaller of two numbers.
MOD	function	Find remainder after dividing two numbers.
OR	function	Compute logical OR of two numeric expressions.
POS	function	Determine position of substring in a string.
RANDOMIZE	statement	Initialize random number generator.
RND	function	Produce a random number.
SGN	function	Determine sign of a number.
SHFT	function	Shift bits left or right.
SQR	function	Compute square root of a number.
VAL	function	Convert a string to a number.
VAL.SL	subroutine	Convert a string to a number.
MUL.SL	subroutine	Calculate a percent of a number.
DIV.SL	subroutine	Calculate a percentage.

Table 4-3 Predefined Numeric Instructions

String data includes string literals and string variables. String variables, referencing strings and substrings, and string usage are described in the following pages.

String Variables

A string variable is a data item to which a string value is assigned during program execution. The equivalent ASCII code for each string character is stored in binary representation in 1 byte (8 bits) of a string variable.

To use a string variable in a program, first use the DIM statement to declare its name and the maximum number of characters it may have. The DIM statement is described in the *Business BASIC Commands, Statements, and Functions* manual.

Referencing String Variables and Substrings

You may reference an entire string by specifying the name of a string variable. You may also reference certain characters (substring) by specifying the string variable's name followed by a subscript enclosed in either parentheses or square brackets. A subscript indicates the byte location (each character occupies 1 byte) for the beginning of the substring and, optionally, the end. A subscript may be either a number, numeric variable, or expression. It must evaluate to a number between 0 and the value declared in the string's dimension statement. Table 4-4 lists different forms of string references.

String Variable	Refers to
A$	Entire string.
A$(2)	Second through last character.
A$(R)	Rth through last character.
A$(3,7)	Third through seventh character.
A$(I,J)	Ith through Jth character, where I<=J. If I>J, this is a null string.
A$(0)	The position immediately following the last character in A$ (a string may actually contain fewer than the maximum number of characters allowed).

Table 4-4 String References

String Usage

A string expression (either a literal or variable reference) may be used in assignment statements, the PRINT statement, input statements, read statements, and in relational expressions in IF statements. Some examples are given in Table 4-5.

Statement	Action
PRINT A$(1,4)	Prints first four characters of A$.
LET B$= "RESULTS ARE:"	Assigns a string literal to B$.
IF A$(I)=B$(J) GOTO 10	Transfers to statement on line 10 if the Ith character of A$ is the same as the Jth character of B$.
INPUT C$, D$(1,1)	Lets you type in a string literal for C$ and a single character for D$.

Table 4-5 Sample Uses of String Expressions

Business BASIC does not support "string arrays." However, you can get around this restriction by creating a string large enough to accommodate all the data that would normally be placed in a string array, and then load the string with the proper data. The following algorithm should be used for this procedure:

data = array-name[(element# - 1) * element size + 1
(element# - 1 * element size + element size)]

or:

data = array-name[(element# - 1) * element size + 1
element# * element size]

Multidimensional string arrays are achieved by using the same procedure. This procedure is used in Figure 4-5 to open six files without using six different OPEN statements.

```
0010 DIM FLS$[36]
0020 LET FLS$="APPLESPEACHSPEARS ORANGEAPRCOTGRAPES"
0030 FOR I=1 TO 6
0040   OPEN FILE[I.0],FLS$[(I-1)*6+1,(I-1)*6+6]
0050 NEXT I
```

***Figure 4-5** Creating a "String Array"*

Concatenating Strings

Strings may be concatenated in an assignment statement by specifying string expressions separated with commas. For example, Figure 4-6 produces the following when the program is executed:

```
THE YEAR IS 1982, THE MONTH IS JUNE

0090 DIM A$(16),BJ(23),C$(50)
0100 LET A$="THE YEAR IS 19XX"
0105 LET B$="THE MONTH IS XXXXXXXXX"
0110 LET C$=A$(1,14),"82",B$(1,13),"JUNE"
0015 PRINT C$
```

***Figure 4-6** Concatenating Strings*

String Assignments

You may store characters into different locations of a string with an assignment statement. Some examples follow in Table 4-6.

Assignment Statement	Action
LET A$=B$	Contents of A$ are replaced with the contents of B$.
LET A$=""	Contents of A$ are replaced with a null string.
LET A$=A$,B$	Contents of B$ are appended to the current contents of A$.
LET A$(0)=B$	Contents of B$ are appended to the current contents of A$.
LET A$=B$,A$	Produces unpredictable results because A$ no longer exists at the point it is to be appended.
LET A$=4+5	Contents of A$ are replaced with the string literal 9.

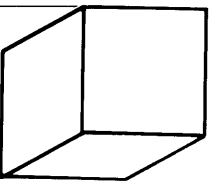
Table 4-6 Assigning Characters to Different Locations within a String

The Data General-supplied string functions provide additional string assignment capabilities. These functions, which can be used as assignment statements or commands, are listed in Table 4-7. They are described further in the Business BASIC Commands, Statements, and Functions manual.

Format	Description
CHR$	Convert a number to a string.
FILL$	Store the same character in all or part of a string.
TRUN$	Truncate a string.
CRM$	Cram (store) three string bytes into two.
UCM$	Expand two crammed string bytes to three.
VAL	Convert a string to a number.

Table 4-7 String Functions

Chapter 5
Terminal Types



In Business BASIC, a terminal refers to a DASHER® display terminal or equivalent device used to log on. When you sign onto the system through the HELLO program, the program tests for the type of terminal being used. If the HELLO program cannot determine the terminal type, it will ask you for a code number identifying the terminal. The terminal type codes are listed in Table 5-1.

Model	Type Code
TELETYPE	0
H.P. 2640/44/45	1
DASHER HARDCOPY and DEC-WRITER	2
6012	3
ADM-1 and ADM-2	4
HAZELTINE 2000, MOD1	5
6052/3, D100/200	6
ADM-3	7
Terminal type 8	8
Terminal type 9 (hardcopy)	9

Table 5-1 Terminal Type Codes

NOTE:

Type codes 2 and 6 are used by RDOS, DOS, AOS, and AOS/VS. Type codes 8 and 9 are used by AOS and AOS/VS only for performance.

NOTE:

The D200/100, 6052/3, and other DASHER® terminals are supported by Data General. The 6012 and teletype terminals are Data General category 2 products.

Data General does not support non-DG terminals. However, source listings of the drivers for non-DG terminals are provided in a subdirectory named NBASIC which was supplied with your system. You are warned, however, that these source listings may not be up-to-date or support current standards for these terminals. To use these source listings, first assemble them using the MACRO assembler. Then use the editor to modify the Business BASIC .CM file and to reload the system.

The program listed below can be used to display your terminal type:

```
10 LET T=0
20 STMA 1,0,T
30 PRINT T
```

AOS, AOS/VS, RDOS, and DOS DASHER® displays are type 6 terminals. This type uses byte-oriented character handling.

Terminal types 8 and 9 are terminal interfaces that provide line-oriented character handling. Terminal type 8 is designed for DASHER 605x compatible terminals. It lets you use AOS-AOS/VS Screen Management primitives, which are listed in Table 5-2. Terminal type 9 is designed for terminals with the HARDCOPY characteristic.

Terminal types 8 and 9 should improve system response because they use fewer system calls than terminal type 6. The standard type 6 interface executes two system calls for each character input: the first call reads the character from the terminal, and the second call echoes the character back to the terminal. This approach is necessary to maintain a high compatibility between AOS, AOS/VS, RDOS, and DOS Business BASIC. Terminal types 8 and 9 require one or two calls for each line of characters input.

Terminal type 8 also allows you to bypass the dot editor by redisplaying any line that contains an error. For example:

```
* 10 DIMM A$(10)
ERROR 2 - STATEMENT OR COMMAND SYNTAX IS INVALID
* 10 DIMM A$(10)
  -
```

The cursor position (indicated by the hyphen) is under the 1.

By pressing CTRL-A the cursor is positioned to the end of the buffer.

```
* 10 DIMM A$(10)
       -
```

By pressing CTRL-B the cursor is positioned to the end of the previous word.

```
* 10 DIMM A$(10)
      -
```

By pressing CTRL-B again the cursor is positioned under the second M.

```
* 10 DIMM A$(10)
    -
```

Typing a space now corrects the line and you can press the NEWLINE key.

```
* 10 DIM A$(10)
```

Another feature of terminal type 8 is that the contents of the edit buffer can be displayed on the screen by using the appropriate Screen Management primitive. For example:

```
* LOAD "TEST"
* LIST
0010 INPUT A
0020 PRINT A
*
```

If you press CTRL-F, 0020 would appear on the screen. If you press CTRL-A, 0020 PRINT A would appear on the screen.

Key	Function
CTRL-A	Move to the end of the character string.
CTRL-B	Move to the end of the previous word.
CTRL-E	Enter/exit the insert character mode.
CTRL-F	Move to the beginning of the next word.
CTRL-H	Move to the beginning of the character string (HOME).
CTRL-K	Erase everything to the right of the cursor (ERASE EOL).
CTRL-M	Erase everything to the right of the cursor (CR).
CTRL-X	Move to the right one character.
CTRL-Y	Move to the left one character.

Table 5-2 Screen Management Primitives

Converting a Terminal to Type 8 or 9

If your account ends in 2 or 6 when you normally sign on to AOS or AOS/VS Business BASIC, you may add a global /E switch on your command line. For example:

X BBASIC/C/S=:UDD:BBASIC:NEWS/E

If you normally sign on with a different account type, or have modified the HELLO program for your own use, the following commands must be issued:

STMA 2,0,x
PRINT @(-19)

where x is 8 or 9. These steps cause the terminal interface to be invoked.

NOTE:

Your AOS or AOS/VS system must be generated to allow the interface to terminal type 8 or 9. See your system manager about generating the system with this option.

Type 8 Restrictions

1. Line cancel key is always CTRL-U, even if you tell the interpreter to make it otherwise.
2. The delete character is always the DEL key.
3. If control characters are allowed for input, you must precede the control characters listed below with CTRL-P:

   CTRL-C CTRL-Q

   CTRL-O CTRL-S

   CTRL-P CTRL-T

This action will enable these characters to be accepted into the input buffer.

4. If you are allowing control characters on input, and you input a CTRL-D, the input is terminated and the value of SYS(10) is not changed.
5. If you are allowing only uppercase characters (each lower-case character input is translated into upper-case), and you define your unpend character to be an uppercase alphabetic (for example, A), inputting the string 1a2bA echos to the terminal as 12B and unpending takes place. The buffer, however, contains the string 1A2B. This is due to a restriction within AOS.
6. If you are allowing control characters on input, they are echoed on the screen with carets (for example, CTRL-W as ^W). This affects any programs using the cursor positioning keys.
7. If you are allowing control characters, the control characters listed in Table 5-2 cannot be put into your buffer. These characters are Screen Management primitives and have special meaning to terminal type 8 users.
8. When inputting characters, if you enter an unpend character when a CTRL-E operation is in progress, the characters to the right of the cursor will not be moved back over to the left to fill the blank at the cursor position before the unpend key was pressed.
9. If a CTRL-E operation is in progress and a character is input that should be ignored (for example, a control character), the CTRL-E is terminated, the cursor is placed at the end of the buffer, and the last character in the buffer is displayed on the screen twice, even though it is in the buffer only once.
10. If the characters input span more than one line on the screen, the cursor position at the end of a read or input statement is left wherever the cursor was when the unpend key was pressed. If the characters input do not span more than one line, the cursor is left at the greater of the two: (a) the ending column position or (b) the starting column number + number characters input.

Type 9 Restrictions

1. Control characters are always allowed, even if you issue the Business BASIC statement "STMA 7,2".
2. For non-auto-unpend inputs, auto-unpending occurs when the buffer is overfilled. For example, if the maximum input is five characters and a sixth character is typed in, the read is terminated and the sixth character is lost.
3. Line cancel key is always CTRL-U, even if you tell the interpreter to make it otherwise.
4. The delete character is always the DEL key.
5. If control characters are allowed for input, you must precede the control characters listed below with CTRL-P:

   CTRL-C CTRL-Q

   CTRL-O CTRL-S

   CTRL-P CTRL-T

   This action will enable these characters to be accepted into the input buffer.
6. If you are allowing control characters on input, and you input a CTRL-D, the input is terminated and the value of SYS(10) is not changed.
7. If you are allowing only uppercase characters (each lower-case character input is translated into upper-case), and you define your unpend character to be an uppercase alphabetic (for example, A), inputting the string 1a2bA echoes to the terminal as 12B and unpending takes place. The buffer, however, contains the string 1A2B. This is due to a restriction within AOS.
8. If you are allowing control characters on input, they are echoed on the screen with carets (for example, CTRL-W as ^W). This affects any programs using the cursor positioning keys.

Terminal Control Operations

Terminal control operations allow you to:

- Position the cursor on the display screen.
- Modify terminal characteristics such as the number of characters input or page width.
- Initiate terminal actions such as starting and stopping a blinking display or clearing the display screen.

Terminal control operations may be used in ASCII input and output statements (PRINT, PRINT USING, INPUT, and INPUT USING). Each begins with the commercial at sign (@) and is followed by one or two coded values.

The format for positioning the cursor is

@(line-number,column)

where line-number and column are between 1 and the maximum number of lines and character positions, respectively, that the display screen can hold.

The format for initiating terminal action is

@(item)

where item is a code for the action to be taken. Table 5-3 lists the available terminal action codes.

The format for modifying terminal characteristics is

@(item,value)

where item is a code for a terminal characteristic that is to be set to value. Table 5-4 lists the available terminal characteristics codes. Terminal control items -1 to -15 are described further in the *Business BASIC Commands, Statements, and Functions* manual.

The following statement uses terminal control operations to display a blinking underscored message:

```
0065 PRINT @(-40);@(-52);"SYSTEM DOWN FROM 12 TO
2";@(-53);@(-41)
```

Code	Description
-20	Move cursor to the first character position on the top line (home position) of the display screen.
-21	Move cursor right one column.
-22	Move cursor down one column.
-23	Move cursor left one column.
-24	Move cursor up one column.
-25	Sound the bell or audible alarm.
-26	Tab to next tab stop.
-27	Return cursor to first character position of the current line.
-28	Move cursor to the beginning of the next line.
-30	Clear the entire screen (or generate a formfeed in a file).
-32	Clear the screen to the end of the line.
-38	Start displaying text in low intensity.
-40	Start displaying blinking text.
-41	End displaying blinking text.
-42	Set page mode.
-43	Turn off page mode.
-52	Begin underlining text.
-53	End underlining text.

Table 5-3 Terminal Action Codes

Item Number	Description
-6	Primary interrupt character. See also STMA 4,6.
-7	Secondary interrupt character. See also STMA 4,7.
-8	Page width. See also PAGE; STMA 4,8.
-9	Tab size. See also TAB; STMA 4,9.
-10	Maximum number of characters allowed on the next input request. It is reset to 132 whenever the program stops. A negative value has the same effect as a positive one. See STMA 4 for how to input a positive value.
-19	Reset most terminal characteristics (-1 to -15) to their default values.

Table 5-4 Terminal Characteristics Codes

Terminal Control Function Library (RDOS/DOS only)

For systems that define their own CRT modules, code and data that deal with the terminal control function have been placed in a library, CRT.LB. The new CRT module must be edited into CRT.LB using the library file editor before generating the Business BASIC system (BSG).

Placing Terminal Control Function in an Overlay (RDOS/DOS only)

The code and data that handle the terminal control function (@) can be placed into an overlay. This is done by answering the BSG question CHANGE DEFAULT OVERLAYS positively and the third question, CRT MODULES, negatively. Unless these two questions are answered in this manner, the terminal control function code and data will be placed into main memory, which has been the case in previous revisions of Business BASIC.

Hardcopy Terminals as Line Printers (RDOS/DOS only)

There are various methods for making a terminal (receive only and send/receive) into a lineprinter on either an ALM line or the secondary TTY controller. Two general techniques are documented below.

The easiest method employs the STMD statement. The format of the command is:

STMD n,p,r,STR$

where n should be set to 1 in this application, p is the port number to send the message to, r should be set to 0, and

STR$ is the string to be sent. Attach the hardcopy terminal to the appropriate line. Business BASIC numbers the ALM ports from 2 to (x+1), where x is the number of genned-in lines. The secondary TTY controller is port 1.

The text string sent must contain all control characters, including the carriage return at the end of the line. If you assign the text to STR$, then execute:

STR$(0)="<13>"

This appends a carriage return to the string. Two hardcopy devices on two ALM lines can serve simultaneously as line printers. Each would be sent messages using the STMD with the appropriate port number.

A more elaborate method is to generate a detached job that attaches to the hardcopy terminal. This job can then receive information from other jobs using the STMB 13 command. The information could be text strings that are printed out or data that is then processed by the special printing job before being printed. This method could be used to create a spooler, but it requires an additional job and the associated overhead.

STMA User System Call

STMA is a system call that allows you to examine and modify many aspects of a job. You can examine and change terminal characteristics, terminal types, and examine account information. You can also pass information between programs, check for IKEYs (interrupts) and status flags, set echo/no echo and message/no message flags, and reset FOR/NEXT and GOSUB/RETURN stacks if they are nested up to their maximum limits. Finally, the STMA system call enables you to translate any code into your code (or ASCII default) on input and output, as well as perform ASCII/EBCDIC translations. All STMA system calls are described in the *Business BASIC Commands, Statements, and Functions* manual.

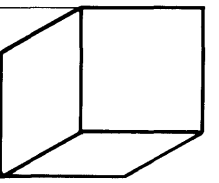
Resetting Terminal Characteristics to Default Values

The -19 terminal characteristics code listed in Table 5-4 can be used to set most of a terminal's characteristics back to their default values. However, in some instances, you may want to reset all terminal characteristics. The STMA user system calls can be used for this purpose (see Figure 5-1).

```
0010 STMA 7,0          :turn on echo
0020 STMA 7,1          :do not allow lower-case
0030 STMA 7,2          :do not allow control characters
0040 STMA 7,3          :enable column counter
0050 STMA 7,4          :allow messages to be sent
0060 STMA 7,5          :enable IKEYs
0070 PRINT @(-19)      :resets terminal control items -1 to -15 to their
                       :default values for your terminal
```

Figure 5-1 *STMA User System Calls*

Chapter 6
Using Table Files with the FM Utility



This chapter presents instructions on how to set up a table file and maintain it and how to use the FM utility with table files.

A table file must exist for a corresponding data file that is to be maintained with FM. It holds information about the file, specifies the appearance of displays used for data input to and output from the file, specifies a standard textual heading that appears at the top of a display, and contains a list of identification numbers for the users authorized to access a file.

Before you can set up a table file with FM, you must perform the following steps, which are described in this chapter:

1. Create a database.
2. Run INDEXCALC to determine the number of keys per index block.
3. Create a PARAM file, which contains general information needed for ISAM files. (The PARAM file is also discussed in Chapter 3.)
4. Run INITFILE to initialize the files and enter the necessary information into the PARAM file.

After you have performed these steps, you may use FM to set up a table file and maintain its corresponding data file. Setting up a table file and maintaining its data file are also discussed in this chapter.

Throughout this chapter, we will use the same example to illustrate the procedures.

Creating a Database

Suppose an employer has a list of employees. The list includes employee last name, first name, telephone extension, and employee identification number. The employer wants to have two key fields: last name and first name.

The name of the table file is EMP.TB. It will include a data file called EMP. Each record in EMP has four fields: LAST, FIRST, EXT, and EMPID. Two of the fields are key fields: LAST and FIRST.

Figure 6-1 shows a sample record in the data file. Before you can build a table file, you must know the record size and number of records in the data file. In our example, the records are 52 bytes long, and the file contains a maximum of 100 records.

LAST	FIRST	EXT	EMPID
Wadsworth	Elizabeth	5733	11111

ID-00144

Figure 6-1 *Record in Data File EMP*

Using INDEXCALC

Once you know what the database will be, use INDEXCALC to calculate certain information needed for the index file(s). INDEXCALC calculates the number of sectors in the index, a value you will use when you initialize the files.

The values returned by INDEXCALC vary depending on key length, record length, the number of records in the file, and the index blocking factor.

In our example there are two key fields; both have a length of 10 bytes. The other values — record length, number of records in the file, and index blocking factor — are also the same for both keys. Therefore, INDEXCALC needs to be run only once. If the keys were of different lengths INDEXCALC would have to be run twice. The dialog in Figure 6-2 is for a 10-byte key.

```
* "INDEXCALC

BYTES PER KEY: 10
BYTES PER RECORD: 52
MAXIMUM NUMBER OF RECORDS: 100
INDEX BLOCKING FACTOR (%): 50

36 MAXIMUM KEYS PER INDEX BLOCK
6 BLOCKS AT LEVEL 0
1 BLOCKS AT LEVEL 1
8 SECTORS IN INDEX
11 SECTORS IN FILE
DO YOU WISH TO CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y OR N)? N"
```

Figure 6-2 Running INDEXCALC for a 10-Byte Key

To run INDEXCALC, first RUN or SWAP to the INDEXCALC utility from the Business BASIC prompt by typing:

```
* RUN "INDEXCALC"
```

or

```
* "INDEXCALC
```

The system then prompts for information about the key as shown in Figure 6-2. The prompts and their explanations are listed below.

BYTES PER KEY — Enter the length of the key field in bytes. Here FIRST and LAST are both 10 bytes in length. (Business BASIC adds 4 to this number for the character pointers, so the actual key field size is 14).

BYTES PER DATA RECORD — Enter the record length in bytes. We enter 52. Calculate this value L by:

L = 2 (for dynamic record allocation) + (length of each field) + (extra)

If you enter an odd number, Business BASIC rounds it up by 1; for example, 29 would become 30.

Figure 6-3 shows the record length for a record in EMP.

MAXIMUM NUMBER OF RECORDS — Enter a positive integer for the maximum number of records in the data file.

INDEX BLOCKING FACTOR — Enter an integer from 1 to 100 for the blocking factor. The integer is treated as a percent; for example, 50 means a blocking factor of 50%. If you are using KADDs to build an index, you must use a blocking factor of 50.

The system then displays the following information:

36 MAXIMUM KEYS PER INDEX BLOCK — Each index block can hold up to 36 keys.

6 BLOCKS AT LEVEL 0 — There are 6 blocks at level 0, the level that points to the database (see Figure 6-4).

1 BLOCKS AT LEVEL 1 — There is 1 block at level 1 (see Figure 6-4).

8 SECTORS IN INDEX — The index has 8 sectors: 1 header block (block 0), 1 block at level 1, and 6 blocks at level 0 (see Figure 6-4).

11 SECTORS IN FILE — The data file has 11 sectors. Calculate this value N by:

N = (bytes per record) * (maximum number of records)/512

and round N up to the next integer. In our example:

N = 52 * 100/512

N = 5200/512 = 10.015 = 11

DO YOU WISH TO CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y OR N) — Enter YES, NEWLINE (AOS and AOS/VS), or CR (RDOS/DOS) to run INDEXCALC for another file. Since there are no other key sizes on which to run INDEXCALC, you should type NO to end INDEXCALC and return to keyboard mode.

Creating and Initializing the PARAM file

When you are creating a table file, be sure a PARAM file exists. To create a PARAM file in your own or another directory, create the file from the CLI (BASIC, RDOS/DOS, AOS, or AOS/VS) and initialize it (set up record 0) with FM. Figure 6-5 gives an example of the PARAM file and its contents.

To create a PARAM file from the BASIC CLI, type

```
* !CRAND PARAM
```

Next, initialize the file. You cannot initialize the file with the INITFILE utility since INITFILE references the PARAM file. You must use FM, a utility that maintains Business BASIC files. FM is explained later in this chapter.

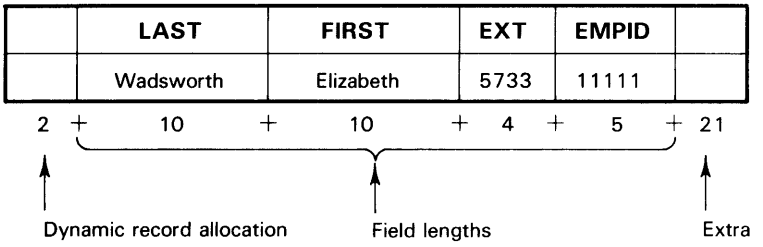
RUN or SWAP to FM by typing:

```
* RUN "FM"
```

or

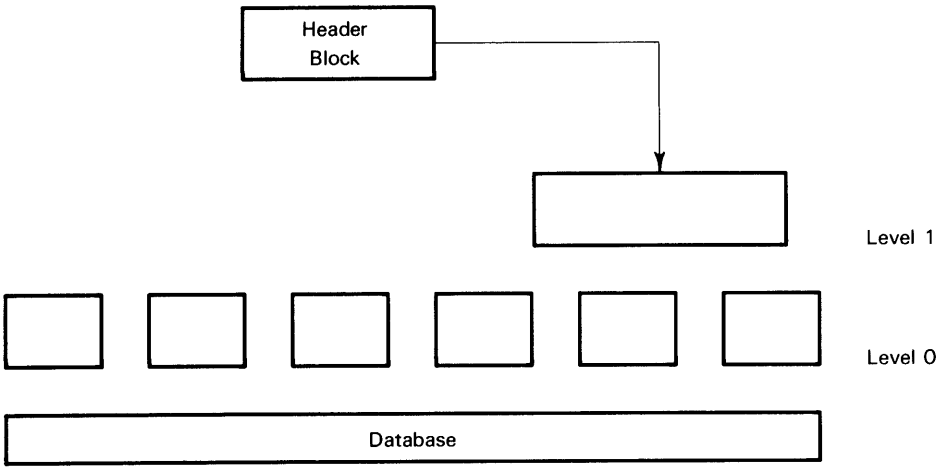
```
* "FM
```

Business BASIC displays the screen shown in Figure 6-6 and prompts for the filename.



ID-00145

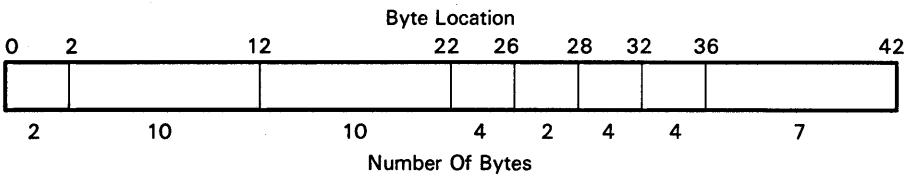
Figure 6-3 *Record Length of a Data Record for EMP*



ID-00146

Figure 6-4 *Index Block Structure for FIRST and LAST*

PARAM File



Byte Locations	Descriptions
0 - 1	Status indicator, always equal to one.
2 - 11	Subfile name or physical file name if not a subfile
12 - 21	Filename of physical file that holds subfile, or filename of physical file if not a subfile.
22 - 25	Byte pointer to start of subfile or physical file.
26 - 27	Record length of subfile or physical file.
28 - 31	Last record number of subfile or physical file.
*32 - 35	Number of last record containing data.
36 - 42	Unused.

*Note: This portion of the PARAM file is automatically updated when you add records to the PARAM file via the INIFILE utility. If you add records to the PARAM file using FM, you must update this count yourself.

ID-00147

Figure 6-5 *File Information in PARAM File*

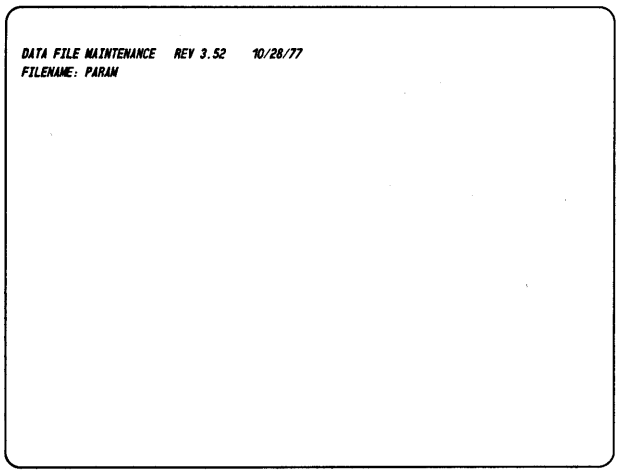


Figure 6-6 *File Maintenance Screen*

FILENAME — Enter the name of the file you wish to run FM on. We enter PARAM so we can initialize the PARAM file (set up record 0).

After you enter the filename, FM displays the screen shown in Figure 6-7.

To initialize the PARAM file, you must set up record 0. To do this press the ADD function key (F4). (Function keys for FM are described in Table 6-2 and in the Business BASIC Subroutines, Utilities, and BASIC CLI manual.) Then enter data in each field as shown in Figure 6-7 and explained below. Press NEWLINE (AOS and AOS/VS) or CR (RDOS/DOS) after each entry.

When you have entered data in all the fields, press the STOP function key (F10) to exit from FM and return to keyboard mode.

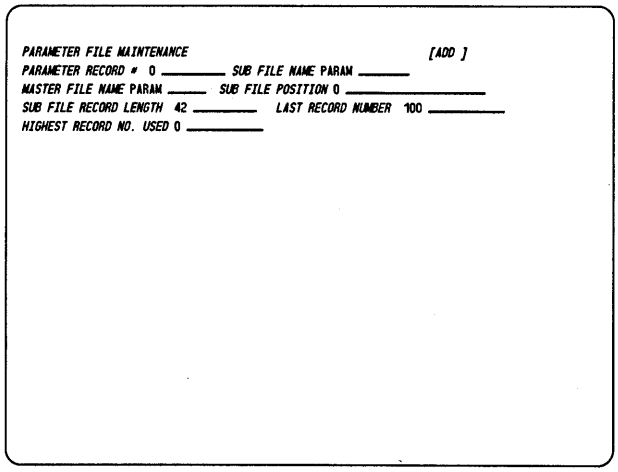


Figure 6-7 *Setting Up Record 0 of the PARAM File*

The prompts and their explanations are listed below.

PARAMETER RECORD # — Enter 0 to create record 0 of the PARAM file. When you are creating a PARAM file, you must set up record 0 to initialize the PARAM file. Record 0 describes the PARAM file itself.

SUB FILE NAME — Enter PARAM. (There is no subfile, so this is the same as the entry for MASTER FILE NAME in the next field.)

MASTER FILE NAME — Enter PARAM as the name of the master file.

SUB FILE POSITION — Enter 0 since there is no subfile (thus no byte offset).

SUBFILE RECORD LENGTH — Enter 42. All records in a PARAM file have 42 bytes.

LAST RECORD NUMBER — Enter an arbitrary number N for the maximum number of records in PARAM. We enter 100.

HIGHEST RECORD NO. USED — Enter 0, the highest record number used in PARAM thus far. Every time you make an entry in PARAM with the INITFILE utility, this value automatically increases. If you add records yourself with FM, you must update this count in PARAM with FM each time you add a record. For example, if you enter three more records, the highest record number used is 3.

This prompt is displayed only if you are referencing record 0. In the other records in the PARAM file, FM ignores this field and sets the value to 0.

To print the contents of the PARAM file after it is initialized, use the FPRINT command from the Business BASIC CLI.

Once you have created and initialized the PARAM file, you are ready to enter records in it. Note that records in the PARAM file are definitions of your physical files and subfiles. To enter, or catalog, them in the PARAM file, use the INITFILE utility.

Using INITFILE

INITFILE initializes all linked-available record files by performing two functions:

1. Makes an entry for the file in the PARAM file.
2. Creates block 0 if the file is an index file, or record 0 if the file is a data file.

The INITFILE utility can be used as a CLI command, or as a program to which you may RUN, CHAIN, or SWAP. After invoking the program you are prompted for information describing the file to be initialized. The exact dialog varies depending on your answers to the questions.

You must catalog all your files, physical and logical, in the PARAM file. When you run FM, or any time a file is opened, Business BASIC reads PARAM first. Therefore, records of all data files and index files must exist in PARAM or the system will generate execution errors.

For the EMP.TB example INITFILE must be run four times:

1. For the index file FIRST.
2. For the index file LAST.
3. For the data file EMP.
4. For the log file EMP.LG.

This procedure is explained in the following pages.

Initializing FIRST and LAST

An index file contains the key field(s) used to index data in a corresponding data file. In our example the data is indexed by FIRST and LAST; therefore, two index files are created.

To do this, RUN or SWAP to the INITFILE utility and answer the dialog as shown below. These entries are used to update the PARAM file shown in Figure 6-5.

```
* RUN "INITFILE"

INDEX (0), DATA (1), STOP (2) 0
LOGICAL FILE NAME FIRST
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES
MASTER FILE NAME: FIRST
BYTE OFFSET TO RECORD 0: 0
MAXIMUM NUMBER OF INDEX SECTORS: 8
BYTES PER KEY 10
BLOCKING FACTOR (% PERCENT): 50
DUPLICATE KEYS ALLOWED? YES
INDEX (0), DATA (1), STOP (2) 0
```

Figure 6-8 *Creating Index File FIRST with INITFILE*

The prompts displayed in Figure 6-8 and their explanations are listed below.

INDEX (0), DATA (1), STOP (2) — Enter 0 to INITFILE an index file, 1 for a data file, and 2 to return to keyboard mode. We enter 0 to INITFILE the index file FIRST.

LOGICAL FILE NAME — Enter the logical filename of the index file. The logical file is the name of the file you use in the program. We enter FIRST.

FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? — Enter YES to create an entry for the file in PARAM. FM asks the following questions only if you are making a PARAM entry. FM uses the answers to make the PARAM entry for EMP.

MASTER FILE NAME — Enter the name of the master file. If the index file is not a subfile, this is the same name as the logical file name. If the index file is a subfile, the name of the master file may be different.

BYTE OFFSET TO RECORD 0 — Enter 0 if there is no byte offset (files that are not subfiles, or the first subfile in a master file). If there is a byte offset (subfiles), enter N where

N = (maximum number of index sectors) * 512

Every block in an index file is automatically set to 512 bytes.

MAXIMUM NUMBER OF INDEX SECTORS — Enter the number of sectors in the file, which is the SECTORS IN INDEX value in INDEXCALC (see Figure 6-9).

```
* RUN "INDEXCALC"

BYTES PER KEY: 10
BYTES PER RECORD: 52
MAXIMUM NUMBER OF RECORDS: 100
INDEX BLOCKING FACTOR ( % ): 50

36 MAXIMUM PER INDEX BLOCK
6  BLOCKS AT LEVEL 0
1  BLOCKS AT LEVEL 1
8  SECTORS IN INDEX
11 SECTORS IN FILE
DO YOU WISH TO CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y OR N)? N

* RUN "INITFILE"

INDEX (0), DATA (1), STOP (2) 0
LOGICAL FILE NAME FIRST
FILE NOT IN PARAM FILE!  DO YOU WISH TO ADD? YES
MASTER FILE NAME: FIRST
BYTE OFFSET TO RECORD 0: 0
MAXIMUM NUMBER OF INDEX SECTORS: 8
BYTES PER KEY 10
BLOCKING FACTOR (% PERCENT): 50
DUPLICATE KEYS ALLOWED? YES
INDEX (0), DATA (1), STOP (2) 0
```

***Figure 6-9** Using Results of INDEXCALC with INITFILE*

BYTES PER KEY — Enter the key length in bytes. We enter 10.

BLOCKING FACTOR (% PERCENT) — When adding keys to an index block, BASIC fills the first index block to 100% of its capacity. If you then try to add more keys, BASIC splits the information in half and places half of it in the second block leaving each block half full. Then, when it tries to add the next key using a KADD, it inserts the new key into the first or second block, choosing the location of the new key so that the original sorting remains in ascending order. If you are using KADDs to build an index, the system automatically uses a blocking factor of 50%. You can change the blocking factor if you use IBUILD or INDEXBLD to rebuild an index. You can specify a larger or smaller factor; a high blocking factor makes the index more dense than a smaller one.

When using KADD commands (and a blocking factor of 50%) the actual overall blocking factor is approximately 75%.

DUPLICATE KEYS ALLOWED? — Enter YES to allow duplicate keys; enter any other value if you do not want duplicate keys. In this example, YES is entered because some employees may have the same first name. If you specify no duplicate keys and then attempt to enter a duplicate key when you are entering data into the data file, FM generates an error.

INDEX (0), DATA (1), STOP (2) — We enter 0 to INITFILE the index file LAST.

```
LOGICAL FILE NAME LAST
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES
MASTER FILE NAME: LAST
BYTE OFFSET TO RECORD 0: 0
MAXIMUM NUMBER OF INDEX SECTORS: 8
BYTES PER KEY 10
BLOCKING FACTOR (% PERCENT): 50
DUPLICATE KEYS ALLOWED? YES

INDEX (0), DATA (1), STOP (2) 1
```

***Figure 6-10** Creating Index File LAST with INITFILE*

To INITFILE index file LAST, follow the dialog in Figure 6-10. Data entry is explained below.

LOGICAL FILE NAME — Enter LAST as the name of the logical file.

MASTER FILE NAME — Enter LAST as the name of the master file.

INDEX (0), DATA (1), STOP (2) — Enter 1 to INITFILE the data file EMP next.

Initializing a Data File

```
INDEX (0), DATA (1), STOP (2) 1

LOGICAL FILE NAME EMP
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES
MASTER FILE NAME: EMP
BYTE OFFSET TO RECORD 0: 0
RECORD SIZE IN BYTES: 52
MAXIMUM NUMBER OF DATA RECORDS: 100
SHOULD FILE BE NULL FILLED? YES

INDEX (0), DATA (1), STOP (2) 1
```

***Figure 6-11** Creating Data File EMP with INITFILE*

To INITFILE a data file, follow the dialog in Figure 6-11, noting the following differences:

INDEX (0), DATA (1), STOP (2) — Enter 1 to INITFILE a data file.

LOGICAL FILE NAME — Enter EMP as the name of the logical file.

FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? — Enter YES to make a PARAM entry for EMP.

MASTER FILE NAME — Enter EMP as the name of the master file. Since the logical file and the master file are the same, logical file EMP resides in physical file EMP.

BYTE OFFSET TO RECORD 0 — Enter 0 since the logical file and the master file are the same (there is no subfile).

RECORD SIZE IN BYTES — Enter 52 as the record size in bytes.

MAXIMUM NUMBER OF DATA RECORDS — Enter 100 as the last record number.

SHOULD FILE BE NULL FILLED — Enter YES to fill the file with nulls.

INDEX (0), DATA (1), STOP (2) — Enter 1 to INITFILE the log file EMP.LG, shown below in Figure 6-12.

Initializing a Log File

A log file shows all additions and changes made to the data file. Log file names use an .LG extension, which is recommended but not required.

```
INDEX (0), DATA (1), STOP (2) 1

LOGICAL FILE NAME EMP.LG
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES
MASTER FILE NAME: EMP.LG
BYTE OFFSET TO RECORD 0: 0
RECORD SIZE IN BYTES: 70
MAXIMUM NUMBER OF DATA RECORDS: 100
SHOULD FILE BE NULL FILLED? NO
INDEX (0), DATA (1), STOP (2) 2
```

Figure 6-12 *Creating Log File EMP.LG with INITFILE*

To INITFILE a log file, follow the dialog in Figure 6-12. Data entry is explained below.

LOGICAL FILE NAME — Enter EMP.LG as the logical name of the log file.

MASTER FILE NAME — Enter EMP.LG as the master filename.

RECORD SIZE IN BYTES — Enter N where

N = (size of a data record) + 18

In our example,

N = 52 + 18 = 70

MAXIMUM NUMBER OF DATA RECORDS — Enter a number for the maximum number of records. We enter 100.

SHOULD FILE BE NULL FILLED — Enter NO. The file is not filled with nulls.

In summary, PARAM now contains:

- header record information
- information about two index files
- information about a data file
- information about a log file

To check this example, run FM on PARAM following the dialog in Figure 6-12. Then press the FIND function key and 1 to find record 1. The screen displays the information shown in Figure 6-13. To look at the rest of the PARAM records, press FIND NEXT or FIND and a record number (your function is echoed in the brackets in the upper-righthand corner of the screen). To exit from FM to keyboard mode, press STOP. Function keys are discussed later in this chapter.

```
PARAMETER FILE MAINTENANCE                                    [    ]
PARAMETER RECORD # 1 _____      SUB FILE NAME FIRST _____
MASTER FILE NAME  FIRST _____       SUB FILE POSITION 0 _____
SUB FILE RECORD LENGTH  512 _____   LAST RECORD NUMBER  8 _____
HIGHEST RECORD NO. USED  0 _____
```

Figure 6-13 *PARAM File Entry for EMP (created during INITFILE)*

When you have performed INITFILE on your index files, your data file, and your log file, you are ready to create a table file for your files.

Using FM to Maintain Data Files

FM is a utility that provides file maintenance functions for any data file for which a table file exists. It uses several Business BASIC utility programs to give you keyboard mode access to data and index files. With FM, you can:

- access records by record number or key (up to three keys)
- add, change, and delete records and/or keys
- define screen formats for data fields to appear on screen "pages" (or use screens set up in SM)
- set up data files in linked-available-record format or direct access format
- set up multiple screen formats/pages for up to nine different data record types

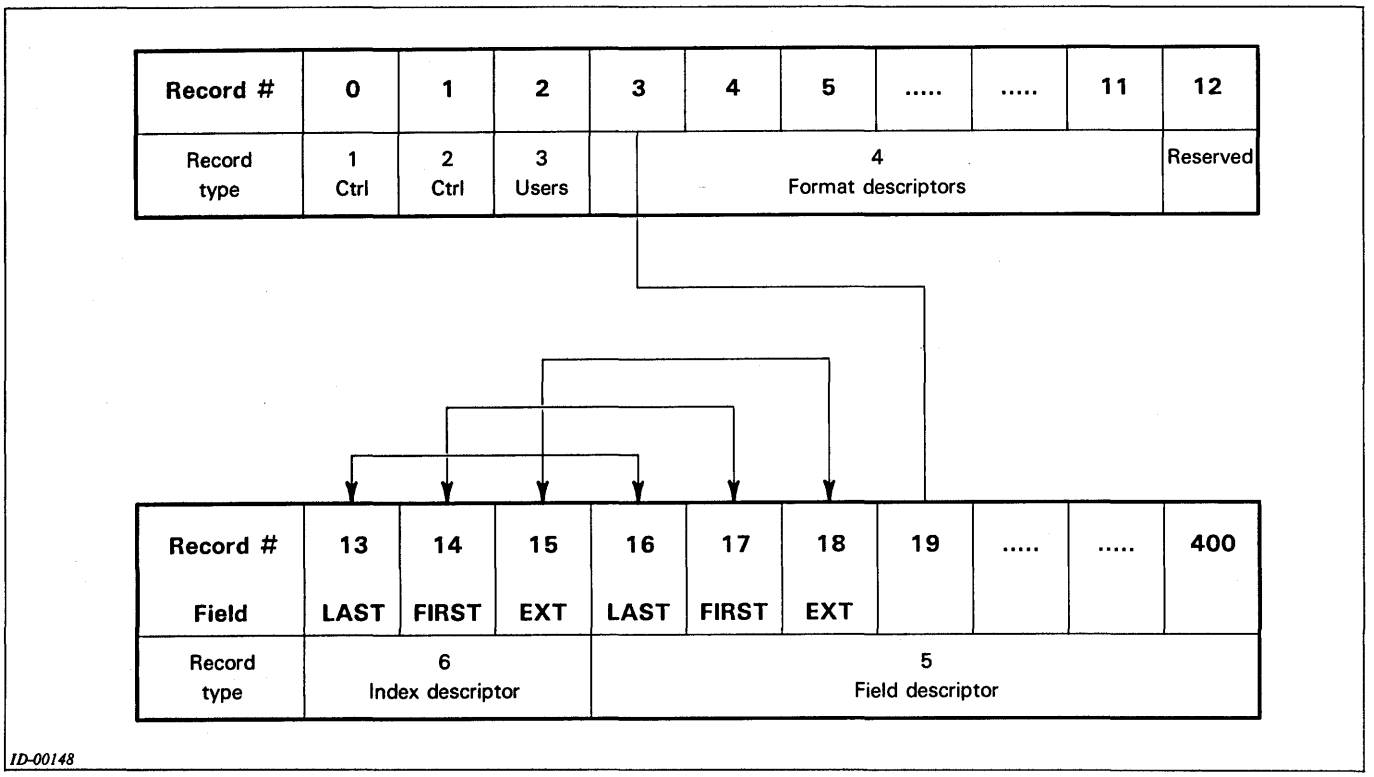


Figure 6-14 *Record Types in EMP.TB*

- specify user passwords

FM uses dynamic space allocation. It also handles file locking, so multiple users can access a single data file. You can use FM on table or data files.

To use FM on a data file, you must first set up a table file (filename.TB) to describe the data file.

Setting up a Table File

A table file consists of a data file, one to three indexes, and other information describing the data.

A table file uses six record types (formats) for entering this information. The record types are located at specific record numbers; they are set up to point to other records in the table file.

Figure 6-14 shows the records in table file EMP.TB, and Table 6-1 summarizes each record type.

Record Type	#	Name	Definition
1	0	ctrl	Contains names of data and index files, number of record types, number of fields, number of pages and pointers to other descriptions.
2	1	ctrl	Contains heading, record, and file size of data file, and optional log file and screen filenames.
3	2	user #	Contains eight valid user IDs and passwords.
4	3	format	Contains pointers to field descriptor records screen descriptor formats and number of pages for containing each data record.
5	usually 16-400	field descriptor	Contains screen formats for data records and keys.
6	usually 13-15	index descriptor	Points to field descriptor records containing formats for keys.

Table 6-1 Explanation of Record Types in EMP.TB

NOTE:

Record Types 5 and 6 can be anywhere in the table file as long as the pointer to the record type points to the first record of that type. In addition, subsequent records of that type follow the first record in sequential order.

You must INITFILE the data file and index files before using FM to set up a table file. INITFILE creates the files and makes entries for the data file and index files in PARAM.

In general, setting up a table file with FM involves three steps:

1. Creating the table file.
2. Executing FM.
3. Entering data into the six table file record types.

These steps are explained in the following sections.

Creating the Table File and Executing FM

When you have INITFILEd the data and index files (as described earlier in this chapter), you are ready to set up the table file.

First, create the table file by typing:

```
* !CRAND EMP.TB
```

Then RUN FM by typing:

```
* RUN "FM"
```

You can also use the CHAIN or SWAP command to execute FM. FM then displays the dialog in Figure 6-15.

```
DATA FILE MAINTENANCE  REV 3.52  10/28/77
FILENAME: FM.TB

DATA FILE NAME: EMP.TB
```

Figure 6-15 *Using FM on Table File EMP.TB*

Data entry is as follows:

FILENAME — Enter FM.TB, a master table file that already exists. The table file parameters and record descriptions are in FM.TB.

DATA FILE NAME — Enter the name of the table file in the form filename.TB.

Screen for a Record

FM then displays the screen in Figure 6-16. FM positions the cursor in the command brackets indicating it is ready for a command. After you enter the command, as described below, you can enter data into the table file.

```
FILE MAINTENANCE TABLE FILE                    [   ]

RECORD # _____
```

Figure 6-16 *FM Screen*

Entering Commands — Function Keys and Roll Mode

When the cursor is in the command brackets, you must enter a command. If you have a DASHER display terminal, use the file maintenance function keys listed in Table 6-2. (See the preface for information on ordering FM templates). Other terminals (as well as DASHER displays) can run the roll-mode version (FM.RM). If FM does not recognize your terminal, it will execute the roll-mode version.

Entering Data into a Table File

The general format for entering data is the following:

1. Press the FORMAT CHANGE key
2. Enter record type (1, 2, 3, 4, 5, or 6)
3. Press the ADD key.
4. Enter data for a field. If a field does not require data entry, press NEWLINE (AOS and AOS/VS) or CR (RDOS/DOS). Business BASIC automatically enters a 0 in numeric fields and leaves alphanumeric fields blank.
5. Press NEWLINE (AOS) or CR (RDOS).

Key	Command	Roll-Mode Command	Description
1			
2	FIND NEXT	NEXT	find next record
3	CHANGE	CHANGE	change a record
4	ADD	ADD	add a new record
5	FORMAT CHANGE	FORMAT	change record format
6	DELETE	DELETE	delete a record
7	PAGE CHANGE	PAGE	change page
10	STOP	STOP	stop FM
11	CLEAR ERROR	(not applicable) DISPLAY AUTO	clear an error display a record automatic display
12			previous field
NEW LINE (AOS and AOS/VS) CR (RDOS/DOS)			next field

Table 6-2 FM Function Keys

6. Repeat steps 3-5 to add records that are the same type.
7. Repeat steps 1-5 to add records of different types.
8. Press the STOP key.

Refer to the steps above to enter data for each record type in the following screens.

Type 1 Record — Control Record

```
FILE MAINTENANCE TABLE FILE                    [FORM]

RECORD # 1_____
```

Figure 6-17 Get a Type 1 Record

After you press the FORMAT CHANGE key and enter 1, the screen shown in Figure 6-18 is displayed. Press the ADD key and enter data as shown below.

```
FILE MAINTENANCE TABLE FILE                    [ADD]

#0 CONTROL RECORD # 0 _____  FILENAME EMP _____  TYPE (D,L) L
INDEX FILENAME LAST _____ KEY DESCRIPTOR RECORD # 13 ____
INDEX FILENAME FIRST _____ KEY 14 ___ INDEX FILENAME _____ KEY 0__
# RECORD TYPES 1 _____ INDICATOR 31416 ___ MAX FIELDS 6 ___ DEFAULT FORMAT 1 ___
DEFAULT PAGE 1 ___
```

Figure 6-18 Type 1 Record

Data Entry for Type 1 Record

0 CONTROL RECORD # — Enter 0. The control record is always located at record 0 in a table file. (See Table 6-1 for the records in a table file.)

FILENAME — Enter the logical filename you used in INITFILE on the data file (the logical filename in PARAM).

TYPE (D,L) — Enter L for linked-available-record format (uses deleted records) or D for direct (does not use deleted records).

INDEX FILENAME — Enter the logical filename you used in INITFILE of the index file (the logical filename in PARAM).

KEY DESCRIPTOR RECORD # — Enter 13, 14 or 15, the record number of the type 6 record that points to the

index named in INDEX FILENAME. This information shows how record 13 points to LAST, record 14 points to FIRST, and record 15 points to EXT.

INDEX FILENAME — Enter the logical filename used for the second key.

KEY — Abbreviation for KEY DESCRIPTOR RECORD # (see previous entry).

RECORD TYPES — Enter a number for the number of different record types (formats) you want for the data file. A table file can have up to nine record types.

If you enter 0, you risk overwriting active records when you ADD records because FM does not check for deleted records.

If you enter a negative number, FM does not automatically select a format for your record type. You must do this with the FORMAT CHANGE key.

INDICATOR — Enter 31416 (or whatever password your system manager selects).

MAX FIELDS — Enter a number N where:

N >= (key fields) + (data fields) + extra

We enter 6 based on: 3 keys + 1 data field + 2 extra

N is greater than or equal to the largest number of fields on any page. If the entry is less than the format you are using, FM generates an error.

DEFAULT FORMAT — Enter a number N where:

N = (record # of format descriptor record to be used first) - 2

FM uses the default format first. For example, in Figure 6-14 type 4 records start at record number 3. since we have only one type 4 record, record 3 contains the format we want FM to use first. Since FM adds 2 to the DEFAULT FORMAT entry, enter 3 - 2, or 1.

If you have more than one format (more than one type 4 record), FM uses the first format specified in this entry and will find the other type 4 records by following the formula

N + 2

where N is the record number that you give FM each time you press the FORMAT CHANGE key and answer the subsequent RECORD # query. Enter record numbers sequentially, or FM will not find the appropriate type 4 records in order.

DEFAULT PAGE — Enter 1 to display the first page of the table file when you begin running FM on your data file. In our example we need only one page to hold all the record formats (there is only one record format, specified in # RECORD TYPES). Even if you have more than one page, you will probably want to display your first page initially.

Type 2 Record — Control Record

Press the FORMAT CHANGE key to enter data for another record type, then enter 2 for the record type (see Figure 6-19).

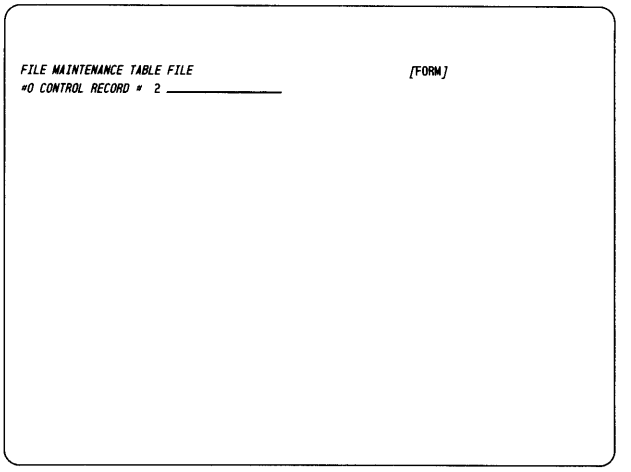


Figure 6-19 *Executing a Format Change for a Type 2 Record*

The screen in Figure 6-20 is displayed. Then press the ADD key and enter the data as shown below.

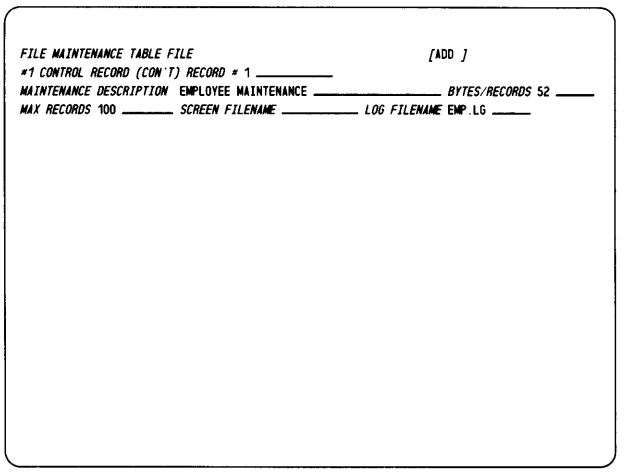


Figure 6-20 *Type 2 Record*

Data Entry for Type 2 Record

#1 CONTROL RECORD (CON'T) RECORD # — Enter 1 for the record number of the type 2 record. This record is always located in record 1.

MAINTENANCE DESCRIPTION — Enter a title description for the table file. This title appears at the top of the screen whenever you run FM on the table file.

BYTES/RECORDS — Enter the number in the PARAM file. FM uses this value if it cannot find the data file in PARAM.

NOTE:

If you use subfiles, you must catalog them in PARAM. Otherwise Business BASIC assumes they are physical files.

MAX RECORDS — Enter the number in the PARAM file. FM uses this value if it cannot find the data file in PARAM.

SCREEN FILENAME — This entry is optional. If you are using a screen, enter a screen name in the form filename.S6, where filename is the name of the data file. You must use the .S6 extension and create a screen with the Screen Maintenance (SM) utility. SM is discussed in Chapter 7.

If you are not using a screen, leave this field blank. FM will use the default FM screen.

LOG FILENAME — This entry is optional. If you have a log file, enter a log filename in the form filename.LG. This form is recommended but not required.

Type 3 Record — Valid Users

Press the FORMAT CHANGE key and enter 3 (see Figure 6-21).

```
FILE MAINTENANCE TABLE FILE                    [FOR ]
#1 CONTROL RECORD (CON'T) RECORD # 3 _____
```

Figure 6-21 Executing a Format Change for a Type 3 Record

The screen in Figure 6-22 is displayed. Then press the ADD key and enter the data as shown below.

```
FILE MAINTENANCE TABLE FILE                    [ADD ]

#2  VALID USERS RECORD # 2 _____ USER #1 0 _____ INITIALS _____
USER #2 0_____ INITIALS _____ USER #3 0_____ INITIALS _____
USER #4 0_____ INITIALS _____ USER #5 0_____ INITIALS _____
USER #6 0_____ INITIALS _____ USER #7 0_____ INITIALS _____
USER #8 0_____ INITIALS _____ USER #9 0_____ INITIALS _____
```

Figure 6-22 Type 3 Record

Data Entry for Type 3 Record

#2 VALID USERS RECORD # — Enter 2. The Type 3 record always resides in record 2 (see Table 6-1).

USER #1 — This entry is optional. If you want the file to be public, enter 0 (or press NEWLINE or CR).

If you want a password, enter a nonzero number of up to 10 digits to match with the initials entered in INITIALS (see next item).

INITIALS — Enter the initials of the user who must use the password number in the previous user number. AOS and AOS/VS systems treat characters 3 through 5 of USERNAME as the user initials. RDOS/DOS systems treat characters 3 through 5 of ACCOUNT as the user initials.

If you enter a password but leave this field blank, every user must use the password.

USER #2-USER #9 — You can specify up to nine passwords.

Type 4 Record — Format Descriptor

Press the FORMAT CHANGE key and enter 4 (see Figure 6-23). The screen in Figure 6-24 is then displayed. Press the ADD key and enter the data as shown below.

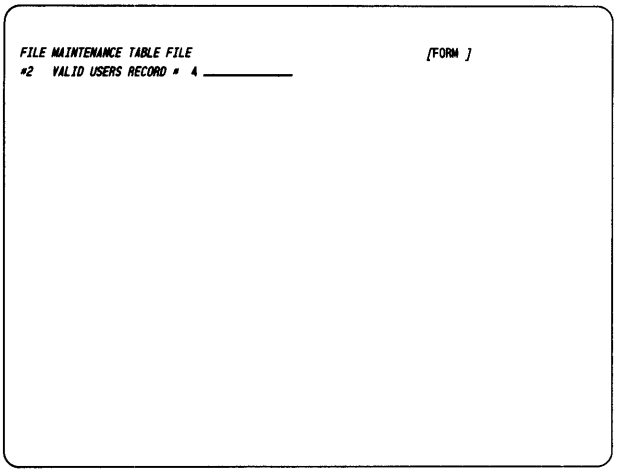


Figure 6-23 *Executing a Format Change for a Type 4 Record*

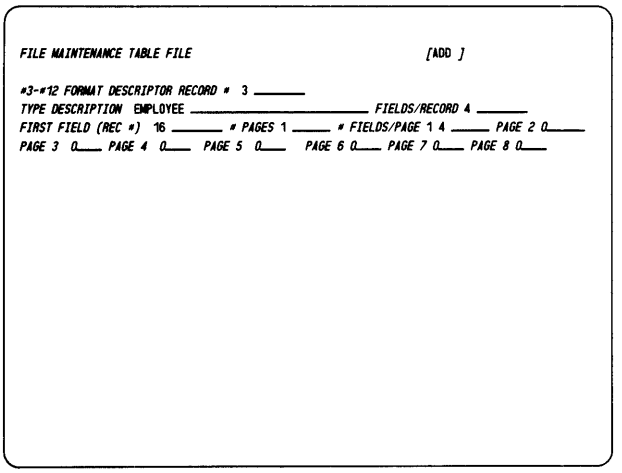


Figure 6-24 *Type 4 Record*

Data Entry for Type 4 Record

#3-#12 FORMAT DESCRIPTOR RECORD # — Enter 3 for the first format descriptor record. They start at record 3 and proceed sequentially to 11, allowing up to nine different formats.

TYPE DESCRIPTION — Enter an arbitrary name for the format you are using.

FIELDS/RECORD — Enter a number N for "total fields" displayed, where:

N = (all fields) - (key fields)

FM automatically displays key fields, so do not count them.

FIRST FIELD (REC #) — Enter the record number of the field descriptor record (type 5) that describes the first nonkey field. Type 5 records usually start at record 16. FM expects subsequent field descriptor records to follow consecutively.

PAGES — Enter a number from 1 to 8 for the number of pages required for this format. Each format can have up to 8 pages. Each page is a separate screen.

FIELDS/PAGE 1 — Enter the number of "total fields" on page 1 (see FIELDS/RECORD).

PAGE 2-PAGE 8 — Enter the number of "total fields" for each page of your format.

Type 5 Record — Field Descriptor

Press the FORMAT CHANGE key and enter 5. The screen in Figure 6-26 is displayed. Press the ADD key and follow the data entry instructions below.

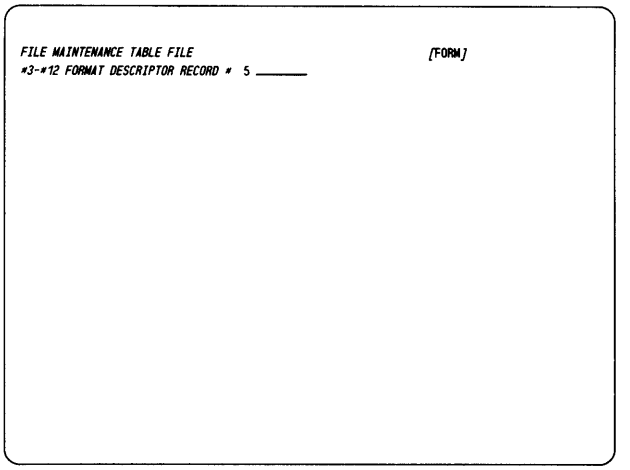


Figure 6-25 *Executing a Format Change for a Type 5 Record*

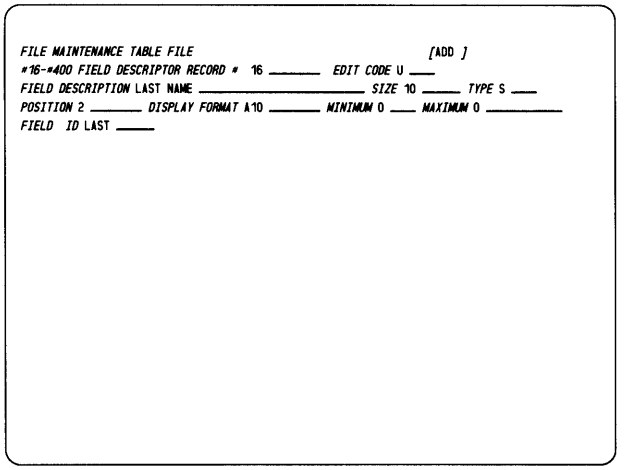


Figure 6-26 *Type 5 Record — LAST*

Data Entry for Type 5 Record

#16-#400 FIELD DESCRIPTOR RECORD # — Enter 16, which is the first field descriptor record. Enter consecutive record numbers, one field per record, in the order you want the fields displayed. There must be a field descriptor record for every field in the data record.

Record types 5 and 6 can be anywhere in the table file as long as the pointer to the record type points to the first record of that type and subsequent records of that type follow the first record in sequential order.

TYPE	Definition	Bytes/Character	Characters
C	crammed	2 bytes/3 characters	(uppercase) A-Z 0-9 comma, space period, dash
S	string	1	
I	integer	2 (single precision) binary (double precision)	-32768 to 32767
D	double integer	4 binary	-2147483648 to 2147483647
B	binary	1	-128 to 127
		or 3	-8388608 to 8388607
N	digit subfield (part of a larger number)		
F	bit flag field		
T	triple precision	3	
T	triple integer	6 binary	-14073748855328 to 14073748855327

Table 6-3 FM Character Types

EDIT CODE — Enter one of four edit codes:

- U — unedited alphanumeric, left justified
- N — edited numeric, left justified
- UR — unedited alphanumeric, right justified
- NR — edited numeric, right justified

FIELD DESCRIPTION — Enter up to 32 characters for the field prompt.

SIZE — Enter the length of the field in bytes.

TYPE — Enter a character type, shown in Table 6-3. Some combinations of character types are possible, such as CS, BF, or BN.

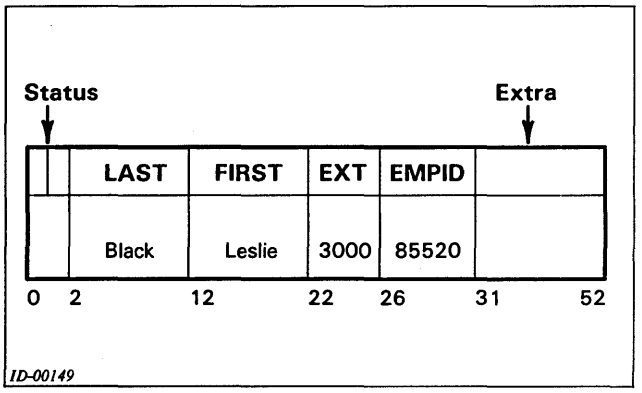


Figure 6-27 Sample Data Record

POSITION — Enter the starting byte of the field relative to byte 0. Do not enter a number less than 2. Business BASIC reserves the first 2 bytes in a data record for the status flag. Figure 6-27 shows a data record in sample data file EMP.

DISPLAY FORMAT — Enter a format code for how FM should display the field. (The format codes are similar to the PRINT USING formats; see PRINT USING.) Enter one of four formats: Fw.d, Aw, Bw.d, or Nw.d.

Fw.d displays a right-justified binary or integer field, where w is the maximum number of digits, including the decimal, and d is the number of digits to the right of the decimal.

Example:

Data	Field Length	Display Format	Ouput
99999	5	F6.2	999.99
100	3	F3.2	1.0

Aw displays a string or crammed field, where w is the maximum number of characters.

Example:

Data	Field Length	Display Format	Ouput
SMITH ARLING-TON	5 6 (crammed)	A5 A9	SMITH ARLINGTON

Bw.d displays a bit flag field, where w is the number of spaces setting the bit corresponding to 2 ^ d; w must always = 1, and d is 0 to 31.

Example:

Value N	Bit	Display Format
16 (10000)	4	B5.4

Nw.d displays a subfield, where w is the maximum number of digits, and d is the number of digits to the left of the rightmost digit.

Example:

Data	Field Length	Display Format	Output
123456 2468	6 4	N3.1 N2.2	345 24

MINIMUM — If you specified an unedited (U or UR) field in EDIT CODE, skip this entry. If you specified a numeric (N or NR) field, enter a minimum numeric value for the field. Do not enter decimal points; you define them in DISPLAY FORMAT. Table 6-3 shows minimum values for certain fields.

MAXIMUM — If you specified an unedited (U or UR) field in EDIT CODE, skip this entry. If you specified a numeric (N or NR) field, enter a maximum numeric value for the field. Do not enter decimal points; you define them in DISPLAY FORMAT. Table 6-3 shows maximum values for certain fields.

FIELD ID — Enter a field name up to six characters (usually a shortened form of the FIELD DESCRIPTOR entry).

In Figures 6-28 to 6-30, we create type 5 records for the remaining three fields. Press the ADD key to enter data.

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

#16-#400 FIELD DESCRIPTOR RECORD # 17 _____  EDIT CODE U _____
FIELD DESCRIPTION FIRST NAME _____ SIZE 10____ TYPE S ___
POSITION 12 ___ DISPLAY FORMAT A10 ____ MINIMUM 0 _____      MAXIMUM 0 ____
FIELD ID FIRST ____
```

Figure 6-28 *Adding a Type 5 Record — FIRST*

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

#16-#400 FIELD DESCRIPTOR RECORD # 18 _____  EDIT CODE N _____
FIELD DESCRIPTION TELEPHONE EXTENSION _____ SIZE 4 ____ TYPE D ___
POSITION 22 ___ DISPLAY FORMAT F4.0 ___ MINIMUM 0 _____      MAXIMUM 9999 ____
FIELD ID EXT _____
```

Figure 6-29 *Adding a Type 5 Record — EXT*

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

#16-#400 FIELD DESCRIPTOR RECORD # 19 _____  EDIT CODE N _____
FIELD DESCRIPTION IDENTIFICATION NUMBER _____
SIZE 2 _____ TYPE I ____
position 26 ___ DISPLAY FORMAT F5.0 ___ MINIMUM 0 _____
MAXIMUM 32767 ___
FIELD ID EMPID ____
```

Figure 6-30 *Adding a Type 5 Record — EMPID*

Type 6 Record — Index Descriptor

Press the FORMAT CHANGE key and enter 6, as shown in Figure 6-31. The screen in Figure 6-32 is displayed. Press the ADD key to enter data, following the data entry descriptions below.

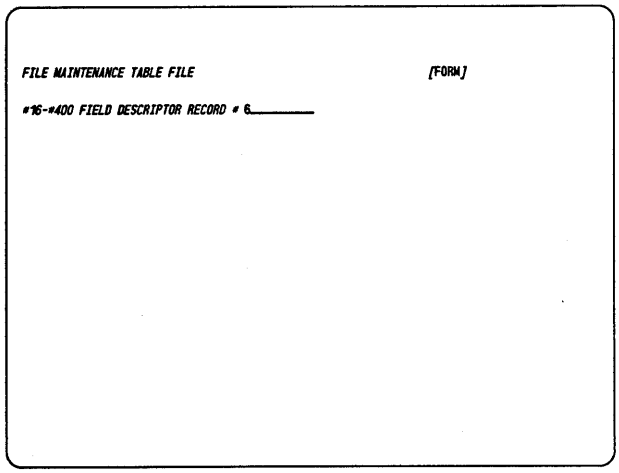


Figure 6-31 Getting a Type 6 Record

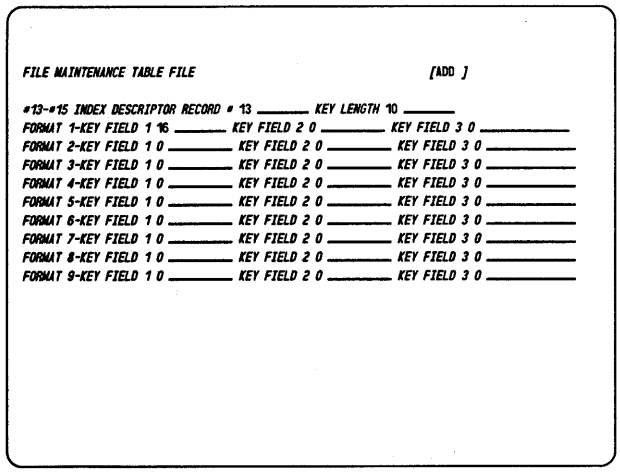


Figure 6-32 Type 6 Record

Data Entry for Type 6 Record

#13-#15 INDEX DESCRIPTOR RECORD # — Enter the record number. Type 6 records are usually record numbers 13-15. If this is the first type 6 record, enter the same record number you entered in the first KEY DESCRIPTOR RECORD # field of the type 1 record. The key descriptor record number points to the type 6 record, so the two entries must be the same.

Record types 6 and 5 can be anywhere in the table file as long as the pointer to the record type points to the first record of that type, and subsequent records of that type follow the first record in sequential order.

KEY LENGTH — Enter a number from 1 to 30 for the length of the key field in bytes. A key can have up to three fields. The maximum length of a key is 30 bytes. For example, a key can consist of three 10-byte fields or two 15-byte fields.

FORMAT 1-KEY FIELD 1 — Enter the record number of the type 5 record (usually (16-400) that describes the "major" key for the first key for format 1 records.

KEY FIELD 2 — Enter the record number of the type 5 record that describes the second key field for the first key for format 1 records. If the key has only one field, skip this entry.

KEY FIELD 3 — Enter the record number of the type 5 record that describes the third or "minor" key field for the first key for format 1 records. If the key has only two fields, skip this entry.

FORMAT 2-FORMAT 9 — Enter values as described above for each field in each key in each format.

In Figure 6-33, we create type 6 records for the second key. Press the ADD key, which allows you to enter data.

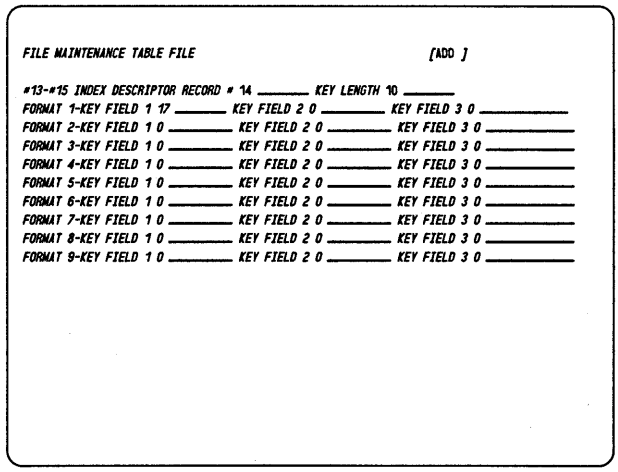


Figure 6-33 Adding a Type 6 Record — Second Key

Exit

You have now completed your table file EMP.TB. Press the STOP key to exit from FM and return to keyboard mode.

Using FM Table and Associated Files

Business BASIC provides five utilities to help you develop and use FM table files.

FMTABPRINT enables you to print the contents of an FM table file with descriptive comments from the associ-

Eileen Salls	5733	28316	Victoria Albano	3347	23789
Maureen Bulger	5214	2345	Peggy Lantana	5657	11430
Diane Boone	5732	111	Harold Brewer	3324	22444
Bob Viator	5771	222	Jeremiah Bradford	7776	27834
Elizabeth Abbott	5268	4567	Ronald Dasher	3645	10934
Brigitte McGovern	5787	4289	Elizabeth Plath	2668	2270
Mary Rogers	6763	250	Maurice Fowler	8652	2317
Karin Young	4633	123	Ruth Vorse	4851	211
Tom Brady	6406	1134	Gene Heston	2765	431
Jonathan Fortier	2344	7899	Cynthia Borek	3455	899
Justin DesRosiers	3457	1256	Chris Brophy	2066	15589
Rebecca McCarthy	3458	1275	Debbie Clark	9087	2265
Fran Markowitz	5443	2787	James Heffernan	3678	2580
Margie Strauss	5742	3778	David Panarese	7754	300
Kate Goodwin	2222	1455	Richard Giacomo	9733	2666
Sally Jones	4487	1096	Karen Black	3119	22114
Kathy Chen	5744	22178	Lisa Whittier	3778	112
Andy Hawley	5734	28946	Don Fradette	4811	1367
Mary Bouchard	7344	12345	Althea Tune	3321	17654

Table 6-4 Sample Data for EMP

ated comment file or with default comments if no comment file exists.

FMPRINT enables you to print selected records from a data file using display information from the corresponding table file and your response to a terminal-user dialog.

FMLOG enables you to print selected records from an FM log file using display information from the associated table file and your responses to a terminal-user dialog. A log file is automatically created when you use FM provided you have put the file's name into the LOG FILENAME field in the table file's type 2 record.

TABBUILD enables you (through a Business BASIC-user dialog) to create table file type 5 records for recurring fields of the same type. These fields are usually part of an array occurring in a data record.

MOVETABREC enables you to duplicate records in a table file. This utility is useful for defining multiple pages and/or formats where the same fields appear on more than one page or format.

These utilities are described in detail in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

To add data to the EMP file, which you have created, type:

```
* RUN "FM"
```

The system then displays the following prompt:

```
Data File Maintenance Rev 3.52 10/28/77
Filename:
```

Enter EMP in response to the filename prompt.

The screen in Figure 6-34 is displayed, which enables you to enter records into the employee data file.

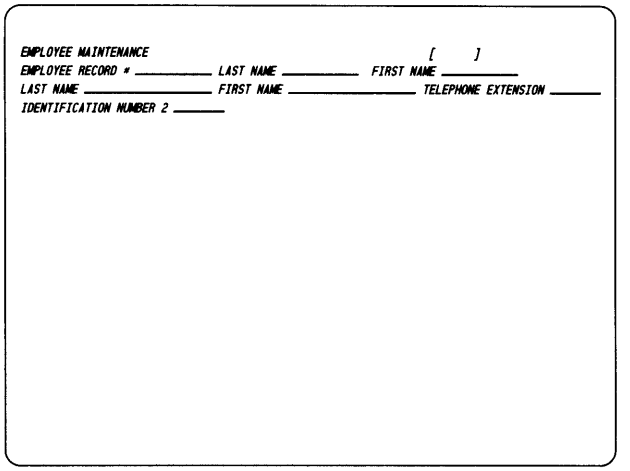


Figure 6-34 *Screen Used to Add Records to EMP*

The data that was entered into the EMP data file through FM is shown in Table 6-4. A decimal listing of the contents of the index file FIRST is shown in Figure 6-35. This listing can be produced with the FPRINT command.

```
      word    word    word    word    word    word    word
      zero     one     two   three    four    five     six

   0    14      36       8       4       3      18       1       0  ...$.............
  10     0       0       0       0       0       0       0       0  ................
  20     0       0       0       0       0       0       0       0  ................
...
 400    20       2   16716   21576   17729       0       0       0  ....ALTHEA......
 410    38   16718   17497       0       0       0       0      18  .&ANDY..........
 420 16975   16896       0       0       0       0       4   16978  BOB...........BR
 430 18759   18772   21573       0       0       6   17224   21065  IGITTE......CHRI
 440 21248       0       0       0      30   17241   20052   18505  S.........CYNTHI
 450 16640       0       0      29   17473   22089   17408       0  A........DAVID...
 460     0       0      33   17477   16962   18757       0       0  .....!DEBBIE....
 470     0      31   17481   16718   17664       0       0       0  ....DIANE.......
 500     3   17487   19968       0       0       0       0      37  ..DON..........%
 510 17737   19525   17742       0       0       0       1   17740  EILEEN........EL
                .                       .                       .
                .                       .                       .
                .                       .                       .
1000    19   65535   19265   21061   19968       0       0       0  ....KAREN.......
1010    35   19265   21065   19968       0       0       0       8  .#KARIN.........
1020 19265   21573       0       0       0       0      15   19265  KATE..........KA
1030 21576   22784       0       0       0      17   19529   21313  THY.........LISA
1040     0       0       0       0      36   19777   21063   18757  .........$MARGIE
                .                       .                       .
                .                       .                       .
                .                       .                       .
1330     0   64617   22784       0       2   64619   19762       0  ..1iY.....1kM2..
1340     0   64627   19761       0       0   64629   17408       0  ..1sM1....1uO...
1350     2   64631   19506       0       2   64633   19506       0  ..1wL2....1yL2..
1360     0   64635   20017       0       2   64637   18737       0  ..1}N1....1}I1..
1370     0   20017       0       2   64637   18737       0       0  ..N1....1}I1....
1400     2       0   19029   21332   18766       0       0       0  ....JUSTIN......
1410     1   65535   65535   65535   65535   65535       0       2  ................
1420 65535   65535   65535   65535   65535   65535   65535   65535  ................
1430 65535   65535   65535   65535   65535   65535   65535   65535  ................
...
```

Figure 6-35 FPRINT Decimal Listing of Index File FIRST

Explanation of FPRINT Listing

Header Block

Figure 6-35 illustrates a decimal listing of the contents of the index file named FIRST. Figure 6-4 illustrates all levels of an index block structure. The header block consists of lines 0 through 20 of this file. The contents of each word in the header block is explained in Table 6-5.

Block 1

After the thirty-sixth key is entered, block 1 (which starts at line 400) splits. It splits because earlier in this chapter (Figure 6-9) we had specified a maximum of 36 keys per block. Block 2 starts at line 1000.

Word 1 in block 1 contains a 20. This value represents the number of keys in block one. Word 2 points to the next available block. In this example a value of 2 is listed, which points to block 2.

Word	Explanation of Contents
0	Number of bytes per key entry. This is the key length in bytes, plus 4 bytes (needed to point to a data record or index block). A value of 14 is listed in this file because earlier in this chapter we specified a value of 10 bytes per key (using the INITFILE utility) in Figure 6-8.
1	Number of keys per block. This is the maximum number of keys that will fit into one block, which is equal to 508 bytes per entry. (There are 512 bytes in each block, four of which are used as a block heading). A value of 36 is listed in this file. This value was also specified using the INITFILE utility.
2	Last available block location. This is equal to the maximum number of blocks that this key file may contain minus 1. In this example we answered 8 in response to the MAXIMUM NUMBER OF INDEX SECTORS prompt in the INITFILE utility. However, this file may contain a maximum of 9 blocks.
3	Next available block location. This is a one-word pointer to the next block that may be allocated. When a new block is allocated, this number is incremented; thus blocks are allocated only from the end of the file.
4	Level 0 internal block location. This is a one-word pointer to the level 0 block, or more specifically, to the root of the multilevel tree. This root may be a level 0 block, or it may point to data, in which case there is no level 0 block.
5	Blocking factor. The blocking factor is used only with utilities such as IBUILD. When keys are KADDed to the file, each block may contain as many keys as it can hold (specified in word 1).
6	This word contains flags. Bit 14 is set if auto-lock is specified, and bit 15 is set if duplicate keys are allowed. In this example word 6 contains a 1 to indicate a flag is set. This flag is set because earlier in this chapter, we specified that duplicate keys were allowed (Figure 6-8).

Table 6-5 Contents of Words in Header Blocks

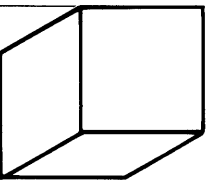
Block 2

Word 1 in block 2 contains a 19, which shows that 19 keys are in block 2. Word 2 in block 2 contains the value 65535, which is the decimal representation of -1. This value means that the next block is the last available block — the index block which starts at line 1400.

Index Block

The index block contains the number 2 in the first word. This indicates that two blocks that contain keys are in this index file. The second word contains a 0 to indicate that this is the last available block.

Chapter 7
Screen Handling



The Screen Maintenance (SM) utility builds screens with data fields that you use with your programs. The SM utility enables you to:

- Indicate the location of fields where the operator is to type in data and where your user program may output data.
- Indicate the textual portion of fields.
- Specify the type and characteristics of data to be input from and output to a field. For example, you may restrict a field to numeric data with an included decimal point.

You may specify several different screen formats, or "screens," with SM. Each screen is stored as a record in a screen file. Multiple screens may be stored in the same file. The screen may also be used by the Business BASIC File Maintenance program (see Chapter 6) to override default formatting of screens or to make a screen match a source document.

A typical screen has both high- and low-intensity fields. Some fields may even blink. You may design a screen to resemble a data entry or retrieval form, so that the operator has only to fill in certain areas. These areas are displayed in high intensity, while the prompts or instructions are low intensity. For example, you may write a program that allows an operator to input names and telephone numbers using the following screen.

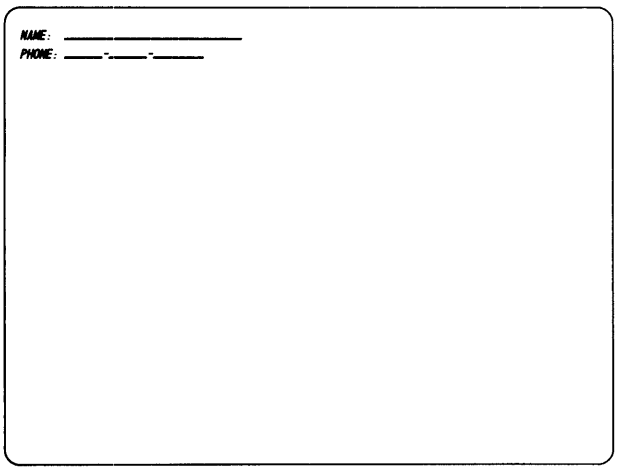


***Figure 7-1** Sample Program Screen*

NAME:, PHONE:, and the two hyphens on the second line are high-intensity fields, while the underscores are low-intensity ones. The operator can type data into the underscored fields.

High-intensity fields provide a form for positioning the operator to the correct location.

You define a screen by running SM, modifying an existing screen or typing a new one, and writing the screen to a file. You define the fields of a screen by using special terminal function keys and typing field formatting characters.

You may write a Business BASIC program to use one or more screens, process input from the terminal operator, display values in different fields, display error messages for incorrect data types by the operator, or make a field blink. You perform these actions with the four screen usage subroutines UNFORM.SL, FORM.SL, PROTFORM.SL, and BLNKFORM.SL.

UNFORM.SL produces unformatted screen input and output, including displaying a screen from the screen file, positioning the cursor to a field, reading data typed by the operator, outputting a value to a screen field, and locking the keyboard and displaying an error message.

FORM.SL produces formatted screen input and output converting a data value to the specifications you supplied when defining the screen.

PROTFORM.SL converts a high-intensity field to low intensity and vice versa.

BLNKFORM.SL starts and stops the blinking of a screen field.

Instructions for using these four subroutines, as well as SM, are presented in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Additional information about screen handling and cursor positioning is presented in Chapter 5.

Chapter 8
Logical File Database Structure

This chapter discusses the use of the logical file database structure and provides an example illustrating its use.

Logical Files

Business BASIC permits 16 physical disk files to be opened concurrently by a program. Because many applications require a greater number of files to be opened simultaneously, Business BASIC offers two methods of implementing logical files as subsections of physical files. Thus, by using logical files, a Business BASIC program may open what appears to be an unlimited number of data files within a single program. Each logical file has a fixed length within the physical file. Figure 8-1 illustrates an inventory data file that is divided into three logical divisions.

In the first method of handling logical subsections of files, the logical divisions of physical files are known as subfiles. A PARAM file is maintained which contains the descriptive information that a program needs to use the subfiles. In this context, a program extracts the file information from the PARAM file and places it in the file characteristics (C1) array. The C1 array contains information for computing the position of records within the subfile. Refer to Chapter 3 of this manual for a complete description of subfiles, the PARAM file, and the C1 array. This chapter also lists the statements, subroutines, and utilities which are used to create and manipulate subfile structures.

The second method of handling logical subsections of files is called the *logical file database structure*. The statements, subroutines, and utilities used to create and manipulate this file structure are described below.

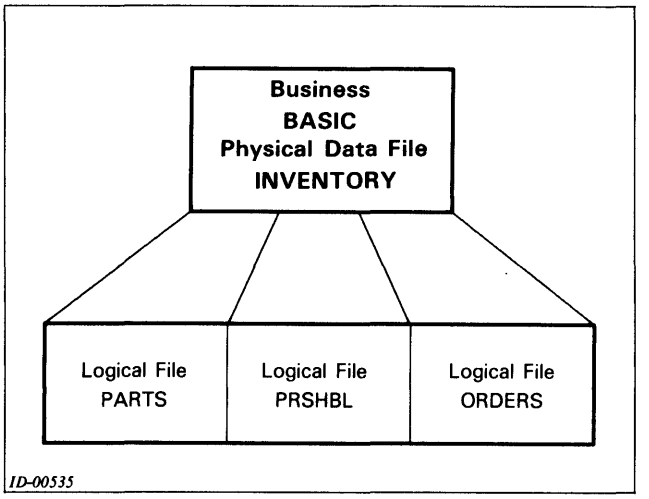


Figure 8-1 *Example of Logical Files within a Physical Data File*

Logical File Database Structure

The logical file database structure is a file set that includes a volume label file (with a .VL extension) and a database file (with a .DB extension). Logical files are implemented as links which point to the volume label file. The volume label file contains an entry with the information specified when the logical file was created, and maps the logical file to the database file which contains the actual data. Thus a simple logical file database (named CUST) for an indexed file with two indexes would contain the following two physical files:

CUST.DB
CUST.VL

and the following three logical files:

CUST
CUSTI1
CUSTI2

Note that each of these logical files would be linked to the volume label file, CUST.VL. An example of a logical file directory listing for an RDOS/DOS system, as well as an AOS/VS or AOS system is presented later in this chapter.

NOTE:

The logical file database structure eliminates the need for PARAM because the description of logical files is in the volume label file. A string variable called the logical file table string (LFTABL$), which is dimensioned by the user, replaces the C1 array.

The volume label file is used by the LOPEN FILE statement to determine the characteristics of a logical file being opened. This information is then copied into LFTABL$, which controls the operation of other logical I/O statements. A description of the contents of the volume label file and LFTABL$ are presented in Tables 8-2 and 8-3.

Logical files are created with the Logical File Utility (LFU) which is described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual. Logical files are allocated in 512-byte increments. This convention is used because logical index files are allocated in 512-byte blocks of entries.

Individual logical files may be defined as one of three types:

D direct random where the user handles all record assignments

L linked available records where record allocation assignments are made dynamically by Business BASIC

I index files which are maintained via the ISAM statements

The accessing of logical files differs only slightly from the accessing of physical disk files. The use of logical files is supported by the Business BASIC logical I/O statements listed in Table 8-1. A special form of the LOPEN statement allows physical files to be defined as logical files. This enables you to use the logical I/O statements with a file which was not created with LFU.

DELREC	Delete a logical record in a type L file.
GETREC	Allocate a logical record from a type L file.
KADD	Add a key entry to an index (type I) file.
KDEL	Delete a key entry to an index (type I) file.
KFIND	Find a key entry in an index (type I) file.
KNEXT	Return the next key entry from an index (type I) file.
LOCK/UNLOCK	Synchronize updating of files shared between programs.
LOPEN FILE	Open and/or define a logical file.
LREAD FILE	Position and read a logical record.
LWRITE FILE	Position and write a logical record.

Table 8-1 Logical I/O Statements

Note that LOPEN FILE, LREAD FILE, LWRITE FILE, GETREC, and DELREC are Business BASIC statements which are provided for I/O to logical data files. The KADD, KDEL, KFIND, and KNEXT statements may be used with logical data files as well as with data files cataloged in PARAM.

If you plan to use the logical file database structure, there is no need to use the PARAM file and C1 array. You should also be aware of a conversion utility called PARAMCON that converts a PARAM file database structure to a logical file database structure. Once converted, you may access the database with both the OPEN as well as the LOPEN FILE statement and all of the other logical I/O statements listed in Table 8-1. Note that there are a number of PARAMCON prerequisites regarding the length and names of data files. These prerequisites are listed in the PARAMCON description which is presented in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Volume Label File Format

The volume label (.VL) file contains records of 32 bytes each, which describe the size and location of the logical files in the database (.DB) file. An illustration of the contents of a record in the volume label file is described in Table 8-2.

Bytes	Description
0-9	Logical file name.
10-11	Starting sector number of a logical file in the .DB file.
12-13	Record length in bytes.
14	Record type (I, L, D).
15-17	Last valid record number of logical file.
18-27	Logical file name of next volume label file (unused).
28-29	Read access flags (unused).
30-31	Write access flags (unused).

Table 8-2 Contents of Records in Volume Label File

The volume label file is maintained by the LFU utility program, which uses the information to determine available file space as logical files are created. When LFU is used to delete a logical file, the name *DEL is placed in the logical file name field of the .VL file entry. LFU will reuse this space if another logical file of the same size is created.

The LOPEN FILE statement uses the volume label file to determine the characteristics of a logical file being opened. This information is copied into LFTABL$, which then controls the operation of the other logical I/O statements. For this reason, prior to the first LOPEN statement in a program, the string variable LFTABL$ must be dimensioned and filled with nulls to a length of at least 26 times the highest logical file number to be used.

Logical File Table (LFTABL$)

The logical file table string (LFTABL$) is used to store the definition of any logical file referenced by any logical I/O statement. Within LFTABL$ are entries of 26 bytes for each logical file which is opened with the LOPEN statement. The contents of a 26-byte record contained in LFTABL$ is illustrated below in Table 8-3.

Bytes	Description
1-2	Channel number on which file was opened with LOPEN.
3-6	Starting byte.
7-8	Flags.
9-18	Logical file name.
19-20	Record length in bytes of logical file.
21-24	Last valid record number of logical file.
25	File type (I, L, or D).
26	Next volume link.

Table 8-3 Contents of Records in LFTABL$

After a logical file has been LOPENed (and, therefore, described in LFTABL$) it may be referenced via any appropriate command: LREAD, LWRITE, KFIND, KADD, KDEL, KNEXT, GETREC, DELREC, or LOCK. In addition, you may access the information in LFTABL$ by using the LFDATA.SL subroutine. We advise that you do not alter the information in LFTABL$ since this alteration could cause undesired operation of the logical I/O statements.

Direct Random Access Files

Direct random access files (type D) are assumed to be fixed length records with no other restrictions.

Linked Available Record Files

Linked available record format (or dynamic record allocation) is a method of disk storage that allows a file to reuse space left by deleted records. Each deleted record contains a pointer that points to the next available record in the file. This storage method is an efficient use of disk space because it allows deleted record space to be reused.

Record 0 of a linked available record file (Type L) is reserved for information describing the next available record. Record 0 is always the same size as all other data records for a type L file. Therefore, the minimum record size is 6 bytes.

NOTE:

It is extremely important that the minimum record size of a type L file be >= 6 bytes. If this rule is violated, the information normally contained in bytes 2-5 of record 0 (see Table 8-4) will be corrupt.

When the record size is less than 10 bytes, all of the data records are initially linked together and then allocated from the deleted record chain.

The record 0 format is shown in Table 8-4.

Bytes	Description
0-1	Status flag (always equal to -2).
2-5	Record number of next available record (-1 if no records are on the deleted record chain).
6-9	Record number of last record used in the file (present only when the record size is >= 10 bytes).
10-13	Active record count, initially 0 (present only when record size is >= 14 bytes). This value is incremented by the GETREC statement and decremented by the DELREC statement.
14-end	Reserved (present only when record size is greater than 14 bytes).

Table 8-4 Contents of Record 0

The rest of a linked available record file is made up of data records. The format of an active data record is shown in Table 8-5. The format of a deleted data record is shown in Table 8-6.

Bytes	Content
0-1	Status flag (>0).
2-end	User data.

Table 8-5 Format of an Active Data Record

Bytes	Content
0-1	Status flag (<= 0; usually 0).
2-5	Record number of next deleted record in deleted chain or -1 if last value in chain.
6-end	Unused (old data from when record was active).

Table 8-6 Format of a Deleted Data Record

Index Files

Logical index files (type I) are allocated in 512-byte blocks of entries. The first block, block 0, is reserved as a header block which describes the index file. The format for block 0 is shown in Table 8-7. Remaining index file blocks contain key and pointer information, in the format shown in Table 8-8.

Bytes	Description
0-1	Number of bytes per entry (key and pointer).
2-3	Number of keys per block.
4-5	Last usable block number.
6-7	Next available block.
8-9	Level 0 block number (sometimes referred to as a root node).
10-11	Blocking factor.
12-13	Bit flags. If the rightmost bit of this flag is set to 1, duplicate keys may be added to the index.
14-end	Reserved.

Table 8-7 Contents of Block 0 for Index Files

Bytes	Content
0-1	Number of entries in the block.
2-3	Number of the next block in sequence.
4-end	Key entries — Consists of a key and a record pointer to the data file. Keys are fixed length in size, and must be an even number of bytes. Record pointers require 4 bytes and must be a positive signed number.

Table 8-8 Format of a Block Containing Keys

Creating a Logical File Database

Before you can set up a logical file database you must perform the following steps:

1. If your database uses index files, use the INDEXCALC utility to determine the number of keys per index block and the number of records (sectors) needed in the index file.
2. Use LFU to create the logical and physical files which make up the database.
3. Use the logical file I/O statements listed in Table 8-1 to access the database. Prior to the first LOPEN statement in a program, the string variable LFTABL$ must be dimensioned and filled with nulls to a length of at least 26 times the highest logical file number to be used. All of the logical I/O statements used are described in the *Business BASIC Commands, Statements, and Functions* manual.

INDEXCALC and LFU are described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual. In addition, a detailed example which explains all of the INDEXCALC prompts is presented in Chapter 6 of this manual.

Following is an example showing the steps in creating a small logical file database called NAMEDB. The database contains one logical data file called NAME, which is designed to hold 100 names and identification numbers. The database is indexed by two index files: ALPHAX and IDNDX. ALPHAX contains 6-byte alphabetic keys. IDNDX contains 2-byte numeric keys.

Since the two keys have different lengths, INDEXCALC must be run twice.

```
* !INDEXCALC
INDEXCALC VERSION 7.00

BYTES PER KEY: 2                          ID will be a two-byte numeric field.
BYTES PER RECORD: 24                      24 bytes for each NAME file record.
MAXIMUM NUMBER OF RECORDS: 100            NAME data file has 100 records.
INDEX BLOCKING FACTOR(%): 50              Specify blocking factor of 50%.


84 MAXIMUM KEYS PER INDEX BLOCK
42 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
3 BLOCKS AT LEVEL 0
1 BLOCKS AT LEVEL 1
5 SECTORS IN INDEX
5 SECTORS IN FILE

DO YOU WISH TO CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y OR N)? Y
BYTES PER KEY: 6                          Specify 6-byte alpha key for ALPHAX.
BYTES PER RECORD: 24
MAXIMUM NUMBER OF RECORDS: 100
INDEX BLOCKING FACTOR(%): 50


50 MAXIMUM KEYS PER INDEX BLOCK
25 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
4 BLOCKS AT LEVEL 0
1 BLOCKS AT LEVEL 1
6 SECTORS IN INDEX
5 SECTORS IN FILE

DO YOU WISH TO CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y OR N)? N
```

After using INDEXCALC to perform the required calculations, you must use LFU to create the physical database file NAMEDB contiguously with enough room for the logical files NAME, ALPHAX, and IDNDX. From the information displayed by INDEXCALC we know that the logical files NAME and IDNDX will each be 5 sectors long, and ALPHAX will be 6 sectors long. The three logical files add up to 16 sectors.

```
* !LFU PCREATE NAMEDB 16

* !LFU LCREATE NAME NAMEDB L 24 100 N
```

creates a logical file NAME of type L in the database NAMEDB with a record length of 24 bytes and a last usable record number of 100.

```
* !LFU LCREATE IDNDX NAMEDB I 4 2 50 N
```

creates a logical file IDNDX in the database NAMEDB with a last usable block of 4 (the calculated 5 sectors means 0 through 4). It has keys that are two bytes long, a blocking factor of 50%, and duplicate keys are not allowed.

```
* !LFU LCREATE ALPHAX NAMEDB I 5 6 50 Y
```

Creates a logical file ALPHAX in the database NAMEDB with a last usable block of 5. It has keys that are 6 bytes long, a blocking factor of 50%, and duplicate keys are allowed.

A listing of the definitions of all the above files in the database may be obtained with the LFU PLIST command as shown below.

```
* !LFU PLIST NAMEDB

   DB  FILE: NAMEDB
  FILE      FILE   STARTING      # OF       RECORD    LAST       # OF
  NAME      TYPE   SECTOR      SECTORS      LENGTH   RECORD     BYTES
NAME          L        0           5          24      100       2424
IDNDX         I        5           5         512        4       2560
ALPHAX        I       10           6         512        5       3072
                                  ------                        --------
                TOTAL SECTORS:    16                BYTES:      8056
```

If you used the !LIST command to list the files that were just created, the system would display the files shown below. All of the logical files that were created are linked to the NAMEDB volume label file. Note that in this example the files were created in a directory called MAIN. The example shows how the list would look on an RDOS/DOS as well as an AOS/VS or AOS system.

RDOS/DOS display:

```
ALPHAX              MAIN:NAMEDB.VL
IDNDX               MAIN:NAMEDB.VL
NAMEDB.DB
NAME                MAIN:NAMEDB.VL
NAMEDB.VL
```

AOS/VS and AOS display:

```
NAMEDB.DB          8192  DBF
NAMEDB.VL           512  VLF
NAME                     LNK  NAMEDB.VL
IDNDX                    LNK  NAMEDB.VL
ALPHAX                   LNK  NAMEDB.VL
```

Logical File I/O Sample Program

The files have now been created and initialized. A program is shown below that uses these files to add names to the database, inquire against the names using either the identification number index or alpha index, and delete names from the database.

```
:Program to add, delete, and find names in a linked logical data file called
:name, indexed by an ID number and ALPHA key.
:
0010 DIM LFTABL$[78],NAME$[20],BUF$[544],KEY$[6],REC$[24],X$[80]
0020 LET LFTABL$=FILL$(0)              :Initialize logical file table.
0030 LOPEN FILE[1,BUF$],"NAME"         :Open the name file.
0040 LOPEN FILE[2,BUF$],"IDNDX"        :Open the ID index.
0050 LOPEN FILE[3,BUF$],"ALPHAX"       :Open the ALPHA index.
0060 LET X=0
:Begin here for function which does not allow next and delete functions.
:
0090 LET RECNO=-1
:
:Begin here for normal function loop.
:
0100 INPUT "FUNCTION  FIND, NEXT, ADD, DELETE, STOP : ",@(-10,1),X$
0110 STRPOS X,"FNADS",X$      :Turn key into function number.
:
:Find, next, add, delete, and stop functions.
:
0120 ON X THEN GOTO 0200, 0500, 0600, 0800, 1000
0130 GOTO 0100                         :Not a function abbreviation
0200 REM FIND FUNCTION
0210 INPUT "ID NUMBER: ",ID            :Null or zero response goes to ALPHA.
0220 ON SGN(ID)+2 THEN GOTO 0200, 0400, 0230
0230 LET KEY$=CHR$(ID,2)               :Compose key string.
0240 LET CURNDX=2                      :Save in case of next function.
0250 KFIND CURNDX,BUF$,KEY$,RECNO
0260 IF RECNO>0 THEN GOTO 0300 :Read and display.
0270 REM NOT THERE
0280 PRINT "RECORD NOT FOUND"
0290 GOTO 0090
0300 REM READ & DISPLAY
0310 LREAD FILE[1,RECNO],REC$          :Read and extract fields for RECNO.
0320 UNPACK "JJA20",REC$,STATUS,ID,NAME$
0330 PRINT "ID:";ID," NAME: ";NAME$
0340 GOTO 0100
0400 REM USE ALPHA INDEX
0410 INPUT "ALPHA KEY (LLLLLI) : ",KEY$
0420 LET CURNDX=3                      :Save in case of next.
0430 KFIND CURNDX,BUF$,KEY$,RECNO
0440 IF RECNO=0 THEN GOTO 0270         :Not there.
0450 LET RECNO=ABS(RECNO)              :Don't care if not exact.
0460 GOTO 0300                         :Read and display.
0500 REM NEXT FUNCTION                 :Next operates on last index found.
0510 IF RECNO>0 THEN GOTO 0540 :  NEXT IS OK
0520 PRINT "FUNCTION REQUIRES FIND FIRST"
0530 GOTO 0090
0540 REM NEXT IS OK
0550 KNEXT CURNDX,BUF$,KEY$,RECNO
0560 IF RECNO>0 THEN GOTO 0300         :Read and display.
0570 GOTO 0270 :  NOT THERE
0600 REM ADD FUNCTION
0610 INPUT "ID NUMBER: ",ID
0620 LET KEY$=CHR$(ID,2)      :Check for duplicate ID.
```

```
0630 KFIND 2,BUF$,KEY$,RECNO          :Not allowed for this index.
0640 IF RECNO<=0 THEN GOTO 0670       :Shouldn't be there.
0650 PRINT "ID ALREADY EXISTS"
0660 GOTO 0090
0670 INPUT USING "","NAME: (LAST,FIRST) ",NAME$
0680 PACK "ZJJA20",REC$,1,ID,NAME$    :Compose REC$ from ID and NAME$.
0690 GETREC 1,RECNO                   :Allocate record in name file.
0700 LWRITE FILE[1,RECNO],REC$        :Write it out.
0710 LET KEY$=CHR$(ID,2)              :Add ID to index.
0720 KADD 2,BUF$,KEY$,RECNO
0730 GOSUB 0900                       :Compose alpha key.
0740 PRINT "ALPHA KEY IS ",KEY$
0750 KADD 3,BUF$,KEY$,RECNO           :Add ALPHA key to index.
0760 GOTO 0090                        :Done.
0800 REM DELETE FUNCTION     :Delete record following a find or next.
0810 IF RECNO<=0 THEN GOTO 0520       :OK to delete???
0820 DELREC 1,RECNO                   :Delete the name record.
0830 LET KEY$=CHR$(ID,2)              :Delete its ID from the index.
0840 KDEL 2,BUF$,KEY$,RECNO
0850 GOSUB 0900                       :Compose alpha key.
0860 KDEL 3,BUF$,KEY$,RECNO           :Delete its ALPHA index entry.
0890 GOTO 0090
0900 REM COMPOSE ALPHA KEY
0910 LET X=0                          :Init pointer into NAME$
0920 EXTRACT KEY$,NAME$," ,",X        :Extract last name
0930 LET KEY$[0]=FILL$(32)            :Pad with spaces
0940 EXTRACT KEY$[6],NAME$," ,",X     :Extract first initial

0950 RETURN
1000 REM STOP
1010 CLOSE
1020 END


* RUN

FUNCTION  FIND, NEXT, ADD, DELETE, STOP : A
ID NUMBER: 10
NAME: (LAST,FIRST)  DOE,JAMES
ALPHA KEY IS  DOEJ
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : A
ID NUMBER: 20
NAME: (LAST,FIRST)  SMITH,JACK
ALPHA KEY IS  SMITHJ
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : A
ID NUMBER: 30
NAME: (LAST,FIRST)  JONES,BOB
ALPHA KEY IS  JONESB
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : F
ID NUMBER: 10
ID: 10        NAME: DOE,JAMES
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : N
ID: 20        NAME: SMITH,JACK
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : N
ID: 30        NAME: JONES,BOB
```

```
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : N
RECORD NOT FOUND
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : N
FUNCTION REQUIRES FIND FIRST
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : A
ID NUMBER: 20
ID ALREADY EXISTS
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : A
ID NUMBER: 40
NAME: (LAST,FIRST)  SMITH,MARY
ALPHA KEY IS  SMITHM
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : A
ID NUMBER: 50
NAME: (LAST,FIRST)  LEWIS,ALFRED
ALPHA KEY IS  LEWISA
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : F
ID NUMBER: 10
ID: 10          NAME: DOE,JAMES
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : D
FUNCTION  FIND, NEXT, ADD, DELETE, STOP : S
```

Appendix A
Memory Management for Business BASIC Systems

The question is often asked, How many users will a given system support? This is not a simple question since many factors influence the answer. In order to understand the complexity of this question, you must understand how the various systems are organized in terms of memory management. After that, the issues of user program size and the number of users that can be run with acceptable performance can be addressed.

NOTE:

The term user window refers to the portion of logical (addressable) memory that is allocated for user programs.

Unmapped Systems (URDOS/DOS)

The maximum amount of memory available on an unmapped system is 64KB; therefore, all addresses are physical addresses. With DOS, only one ground is possible; URDOS (unmapped RDOS) supports two grounds, but this is not very practical. DOS is recommended over URDOS, since URDOS is larger.

In an unmapped swapping system, the full amount of user program space is available to any one job, unless the Business BASIC sysgen or the ACCOUNTING file restricts the program size. If programs are small enough, more than one job may be in the user window at a time. Any jobs which are not in the user window are on disk and must be swapped in. A user program consists of two parts: the program segment and the data segment. The memory assigned for these segments is 512-byte blocks. Typical user space is in the range 12 to 22 KB.

In an unmapped nonswapping system, the entire user program space gets partitioned (on a word basis), so that each of n jobs gets 1/n space. Beyond that the Business BASIC sysgen or ACCOUNTING file may specify a smaller program size. All jobs are in the user window at all times in their assigned partitions. Partitioning is done on a one-word basis. Also, assigning memory to a user program and data segment is on a one-word basis.

Bank Select Systems (CS/10 mod C3 only — DOS)

The bank selector is supported by Business BASIC on the CS/10 mod C3 only. It uses the DOS operating system, but Business BASIC manipulates the bank selector.

Although all jobs are in memory at all times, only one bank can be addressed at a time (see Figure A-2). When one bank is addressable, none of the others are. All five banks are of equal size (up to a maximum of 16 KB). If the user window left for bank 0 is greater than 16 KB, the portion exceeding 16 KB is "wasted" memory. (This will not often be the case.) If the window left for bank 0 is less than 16 KB, the portions of banks 1-4 in excess of bank 0 are "wasted" memory.

Mapped Systems (RDOS)

A mapped system is always a swapping system. It supports two grounds and memory is assigned in 1K word pages (2048 byte blocks). The system maps up to 32 1K word pages to produce an apparent 64KB continuous address space (logical memory). In actuality, these blocks are not necessarily contiguous (see Figure A-3). The amount of physical memory depends on the configuration.

The full amount of user program space is available to any one job, unless the Business BASIC sysgen or the ACCOUNTING file restricts the program size. Any job that is not in the user window is either in extended memory or on disk. Jobs are swapped in to extended memory, then mapped in to the user window. The memory for mapping is on a 1K word page basis, allocated separately for the program and data segments.

In a mapped system, the overlay area must be aligned on a 1K word page boundary. If the Business BASIC interpreter is a few words over a page boundary, the remainder of that page will be "wasted" in order that the overlay area can be properly aligned. For this reason, all sysgen choices should be made with care. The size of the interpreter must be weighed against the sysgen features.

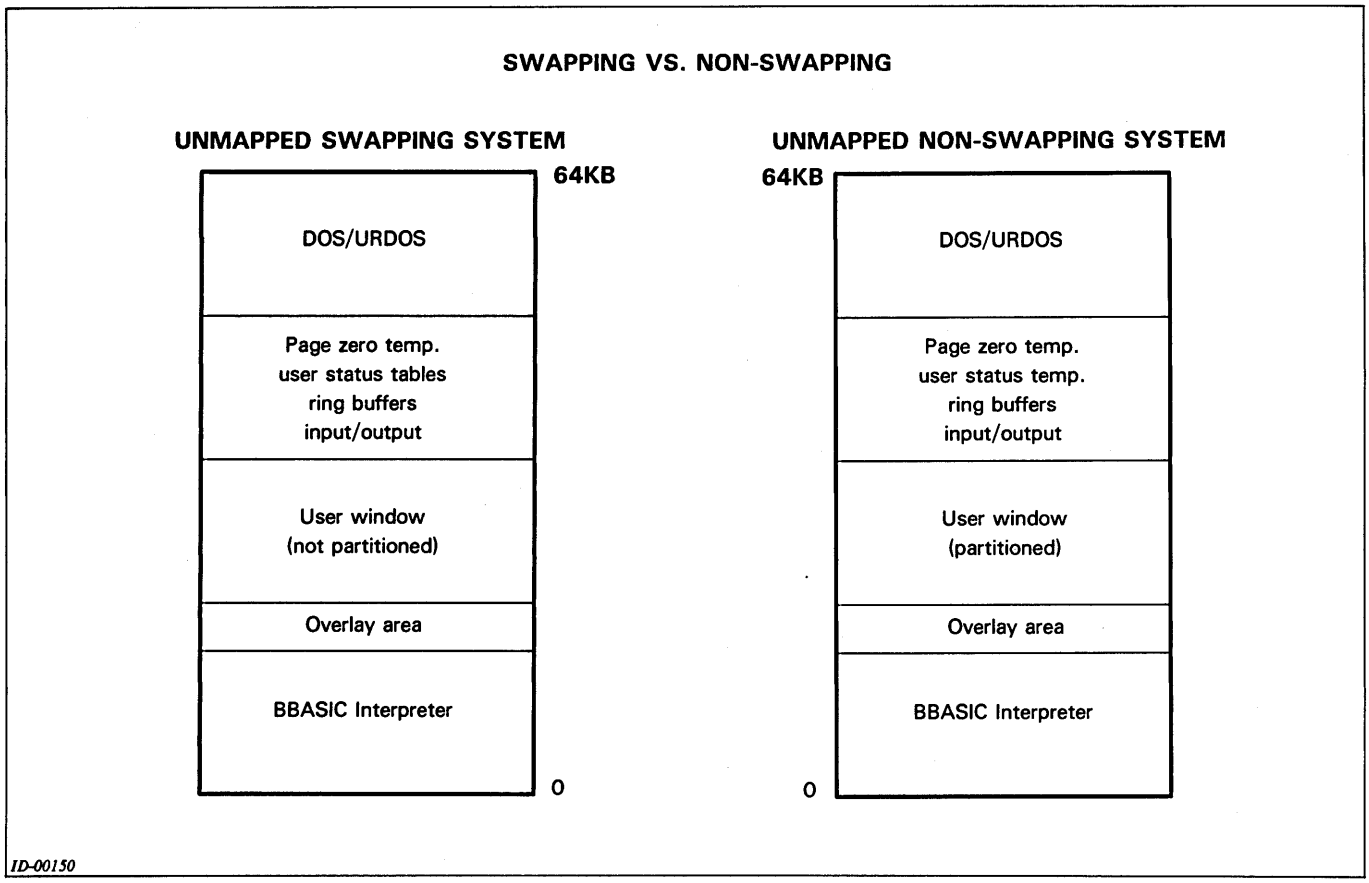


Figure A-1

If virtual overlays are chosen, the overlays are loaded into extended memory and mapped in as needed. If this option is not chosen, overlays will be swapped in from disk as needed.

NOTE:

The foreground can be used for the CLI or for another language, for example, Fortran, assembler, another copy of Business BASIC.

AOS and AOS/VS Systems

Each user executes with his own address space of up to 64 KB. The interpreter is divided into shared and unshared code. The actual size of each of these portions of the interpreter is revision dependent. These figures can be accessed from the AOS/AOS-VS utility PED. The use of shared Business BASIC code frees up additional memory to the system but not to the individual user (see Figure A-4).

Sizing Concerns under RDOS/DOS Business Basic

Size of the User Window

The size of the user window is dependent on:

(1) The size of URDOS/DOS (in an unmapped environment only).

(2) The size of the Business BASIC interpreter.

(3) Boundary alignment and memory considerations.

(4) The number of jobs being executed.

(5) Whether or not swapping was chosen (in an unmapped environment only).

(6) Maximum allowable size of 16 KB (in a CS/10 environment only).

For each additional job, memory must be allocated for another set of page 0 temporaries, another user table, and another set of input and output ring buffers. This memory subtracts from what would have been available to the user window. The page 0 temporaries and the user status

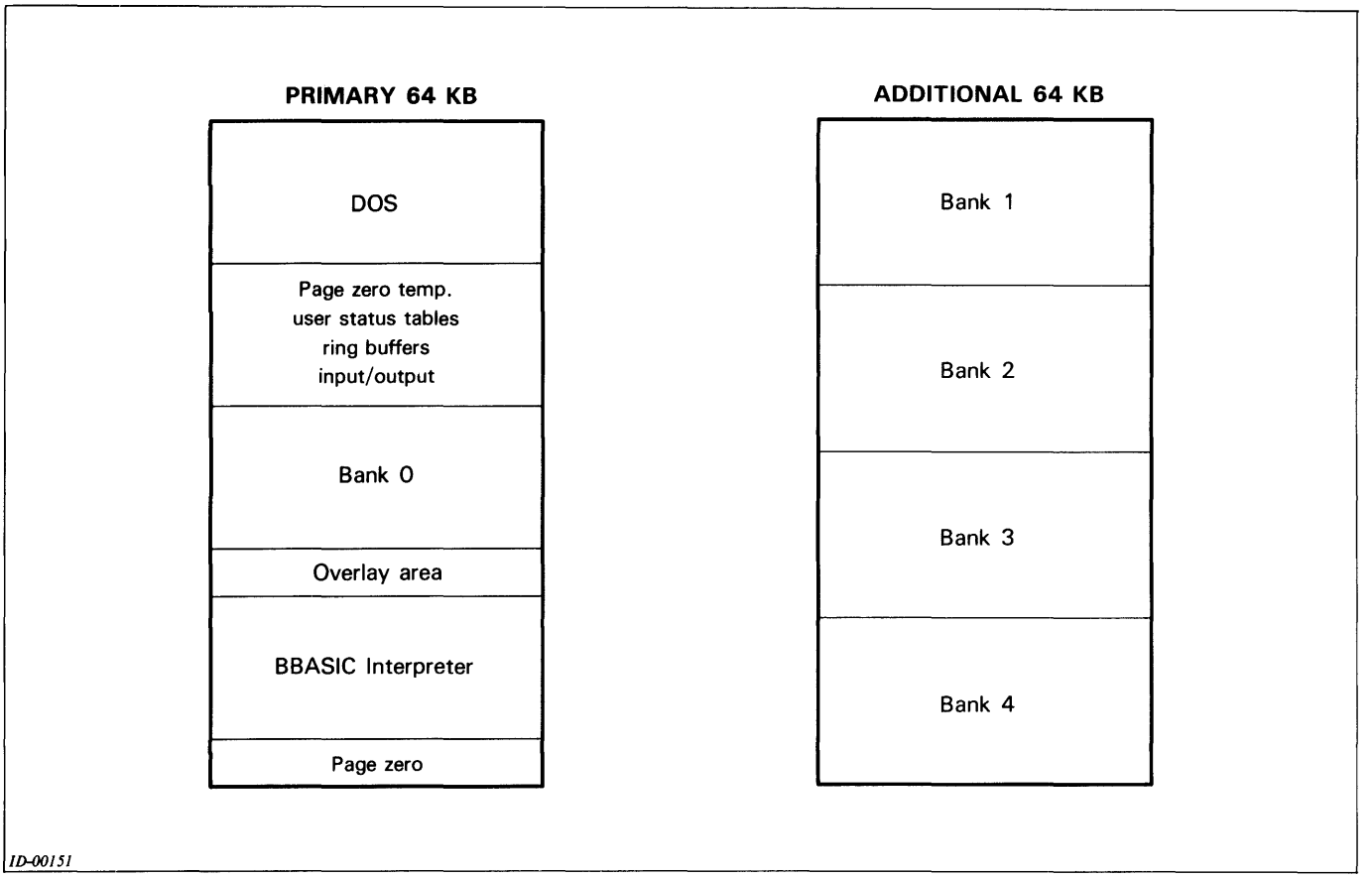


Figure A-2

table are a fixed size on a revision by revision basis. As of revision 5.10 (and the revisions which preceded it) the combined size is approximately 800 bytes per job. The input and output ring buffer sizes are chosen by the user at sysgen time. Whatever amounts are chosen will apply to every job. This cannot be changed on a per job basis.

Boundaries and rounding are an important consideration. In a mapped system where memory is on a 1K word page basis, the reduction of a few words from the interpreter or from the ring buffers, could in certain instances free up an additional page for the user window. This will have to be determined on an individual customer basis.

User Program Size

When a program is being developed, the designer should plan ahead for the maximum environment in which the program will be executed. This refers both to the maximum number of jobs that Business BASIC will execute and the sysgen features that will be chosen. In the unmapped swapping system and the mapped system, program size and the length of the user window will determine the number of jobs that will simultaneously fit into the user window. To illustrate the importance of the number of jobs simultaneously in the user window, consider what happens during the execution of a system call. While a job is executing a system call, it cannot be swapped/mapped out of the user window. If it is the only job in the user window, no other job gets serviced during this time. If, however, more than one job is simultaneously in the user window, then while one of the jobs is executing an I/O system call, another job can use the CPU.

In the CS/10 environment the restriction of not being mapped out during a system call also applies. However, in this situation, only one bank is ever addressed at a time. While a system call is being executed, no other job can ever get serviced — for example, I/O weight and CPU activity do not overlap.

Performance Considerations

Due to the variety of factors influencing performance, a fixed number of terminals that a given system will support cannot always be predicted. This is very application dependent. Users should not confuse the maximum number of physical connections with the maximum number of jobs which will produce acceptable performance. Some items to consider are:

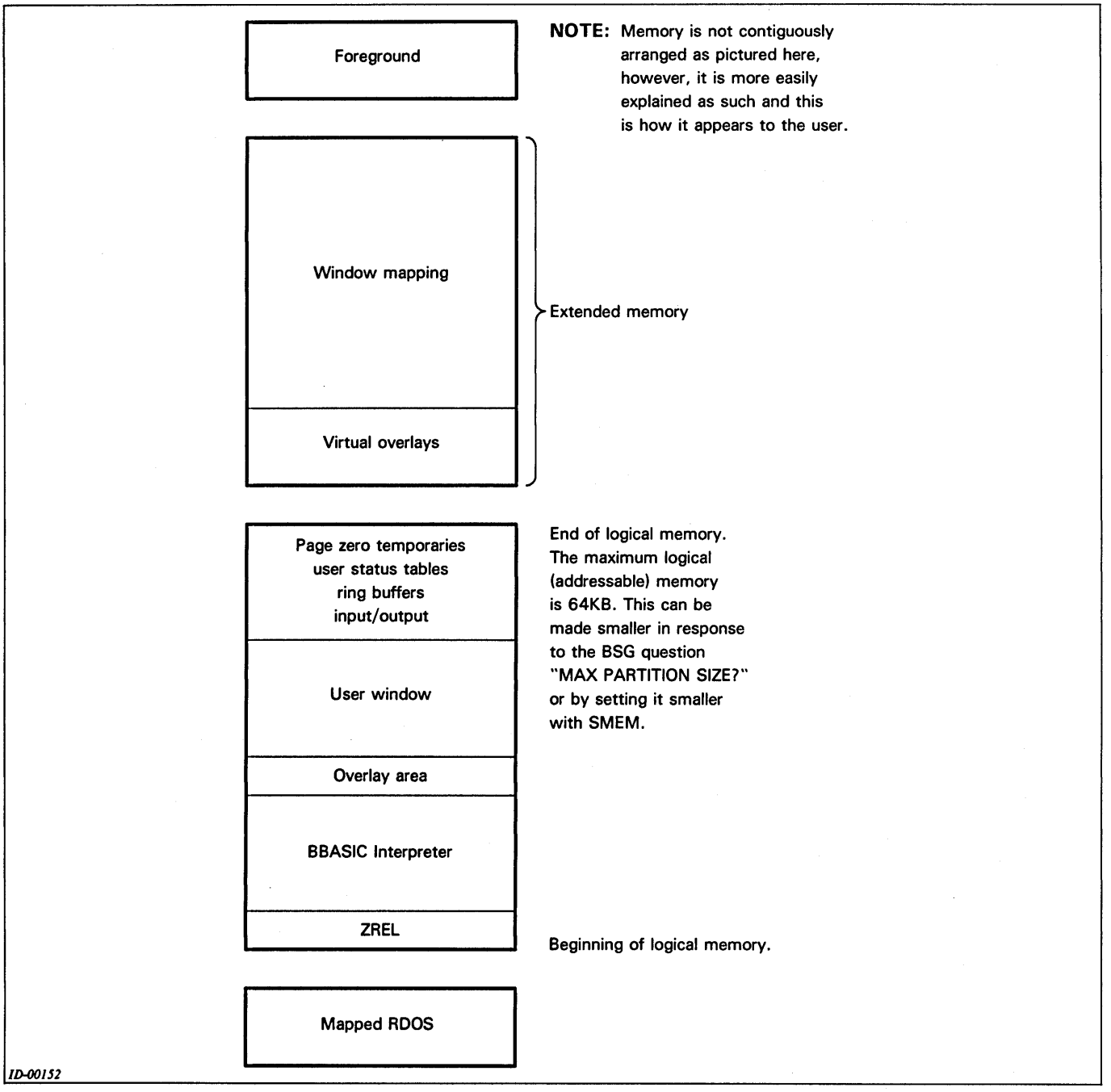


Figure A-3

(1) the hardware configuration

(2) the operating system

(3) the sysgen options for both the operating system and Business BASIC

(4) the size, design, and type of application programs

(5) the user definition of acceptable performance

(6) whether any of the tasks can be done during nonpeak hours

The techniques of SWAPping and CHAINing can be employed to create smaller program modules. The overhead involved in this process should also be considered. The greatest impact occurs in a multiple user environment when many SWAPs and CHAINs are being executed. The disk is kept busy servicing these SWAP and CHAIN statements, which takes away from the time it spends handling record I/O.

Users are urged to plan ahead for the maximum number of jobs they will want to support on a given system.

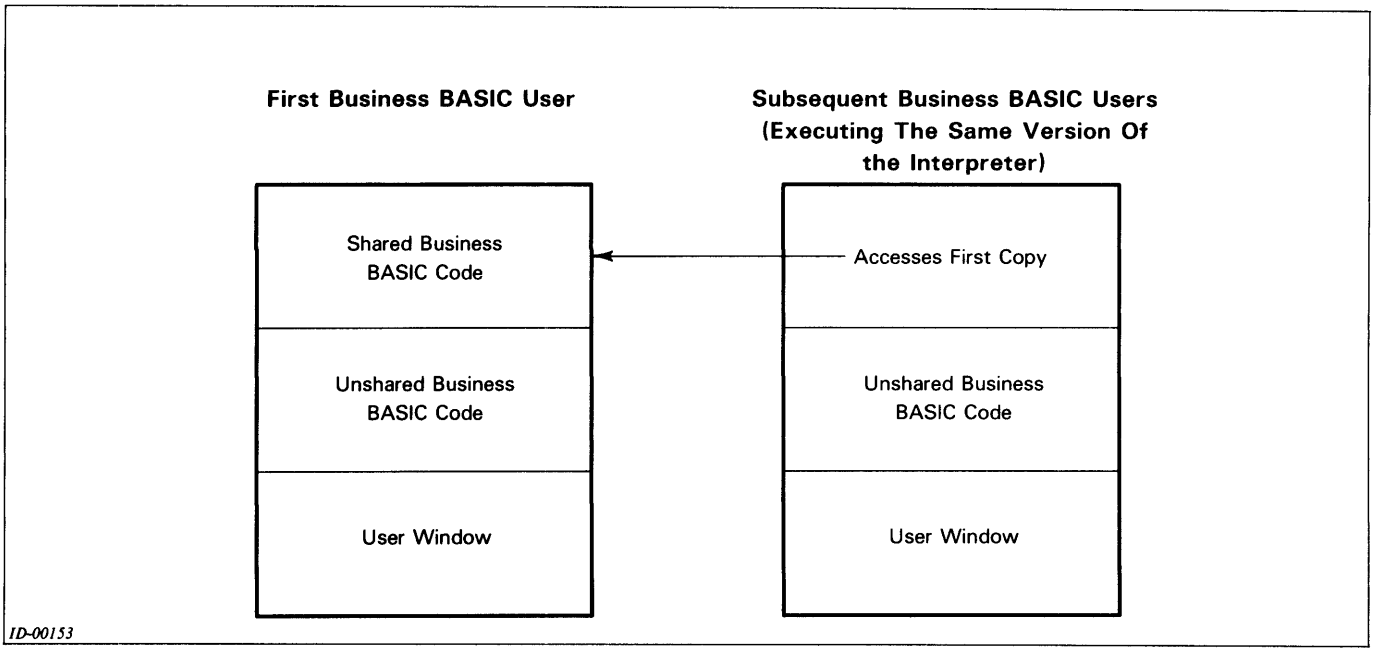


Figure A-4

Benchmarks, with the user's own software, are encouraged, especially for upgrading to large systems. Do not underestimate the task of system design and the importance of choosing the correct hardware, operating system, and language.

Data File Considerations

Careful placement, construction, and access of data files will lead to improved performance. Consider the following points:

(1) Contiguous files should be used where possible.

(2) Business BASIC has its own native ISAM file structure and access under all operating systems.

(3) Performance considerations include the hash frame size, the placement of files on disk, and the number of disk controllers.

Peripheral Support

Business BASIC .IDEFs its own multiplexor driver. Since the operating system does not control these ports, they cannot have device names that would be recognized by the operating system. For this reason, the only two ways to address a receive only printer on a Business BASIC multiplexor port are to detach a job to that port number, or to send that port a "message" via the STMD command.

Both interactive displays and printers may be connected to Business BASIC ports. Only Data General terminals/printers are supported. However, users may incorporate their own drivers for any non-DG terminals/printers.

Both direct and remote asynchronous connections are supported. Business BASIC has modem control logic for its lines.

The number of terminals and the number of jobs are not necessarily synonymous, since Business BASIC supports detached jobs (which execute independent of a terminal).

A multiplexor that is genned into Business BASIC may not be genned into the operating system. An additional multiplexor may be genned into the operating system, provided it is on a different device code. Business BASIC can access a printer on the operating system multiplexor by referring to the device name (QTY:x) or by using a reserved filename that is linked to the device name.

The second TTY may not be genned into both Business BASIC and the operating system.

Business BASIC relies on the operating system to control the other peripherals such as the magnetic tape drive, the disk drive, the system printer, and the paper tape reader.

Sizing Concerns under AOS-AOS/VS Business BASIC

Size of the User Window

In the AOS environment, the size of the user window is completely dependent upon the sysgen choices. There are only six sysgen options. Sample sysgens should be done to determine sizing, as there is no worksheet available. Typical sizes range from 13 to 28 KB.

User Program Size

When a program is being developed, the designer should plan ahead for any additional sysgen features that might be desired later. For example, if maximum-size programs were developed for a Business BASIC system that did not include the INFOS interface statements, they would need to be reduced in size if that option were to be chosen at a later date.

Performance Considerations

Due to the variety of factors influencing performance, the fixed number of terminals that a given system will support cannot always be predicted. This number is very application dependent. Users should not confuse the maximum number of physical connections with the maximum number of jobs which will produce acceptable performance. Some items to consider are:

(1) the hardware configuration

(2) the operating system

(3) the sysgen options for both the operating system and Business BASIC

(4) the size, design, and type of application programs

(5) the user definition of acceptable performance

(6) whether any of the tasks can be done during nonpeak hours

The techniques of SWAPping and CHAINing can be employed to create smaller program modules. The overhead involved in this process should also be considered. The greatest impact occurs in a multiple user environment when many SWAPs and CHAINs are being executed. The disk is kept busy servicing these SWAP and CHAIN statements, which takes away from the time it spends handling record I/O.

Users are urged to plan ahead for the maximum number of jobs they will want to support on a given system. Benchmarks, with the user's own software, are encouraged, especially for upgrading to large systems. Do not underestimate the task of system design and the importance of choosing the correct hardware, operating system, and language.

Data File Considerations

Careful placement, construction, and access of data files will lead to improved performance. Consider the following points:

(1) Large element sizes (which produce contiguous-like files) should be used where possible for large files.

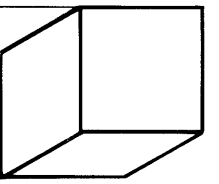
(2) Business BASIC has its own native ISAM file structure and access under all operating systems. In addition, AOS and AOS/VS have an interface to INFOS.

(3) Performance considerations include the hash frame size, the placement of files on disk, and the number of disk controllers.

Peripheral Support

Business BASIC relies on AOS-AOS/VS to support all peripherals.

Appendix B
Glossary



arithmetic operator

A symbol used in a numeric expression. The following symbols are valid operators: + (unary plus or addition), - (unary minus or subtraction) * (multiplication), / (division) and ^ (exponentiation). See also expression.

array

An ordered set of integer values. Each array element is stored according to the precision it is set at, or the default precision. Business BASIC allows you to use one-dimensional arrays (vectors) with columns only (each element is a column), and two-dimensional arrays (matrices) with rows and columns. Rows start at row 0, and columns start at column 0.

ASCII code

A character's decimal code number (unless octal is specifically stated). All printable characters, as well as nonprintable characters and other keys on your terminal's keyboard, have ASCII code numbers. See also characters.

attributes

All files can have attributes. The BASIC CLI command CHATR will change a file's resolution attributes (permanent, read-protected, and write-protected).

BASIC

Business BASIC. There are five types of Business BASIC: RDOS/DOS Business BASIC (double precision), RDOS/DOS Business BASIC (triple precision), AOS and AOS/VS Business BASIC (double precision), AOS and AOS/VS Business BASIC (triple precision), and ENTERPRISE® Business Basic. ENTERPRISE® Business BASIC allows multiple precision.

BASIC CLI (command line interpreter)

A utility program that emulates the RDOS/DOS CLI. You can RUN, SWAP, or CHAIN to the CLI and execute BASIC CLI commands interactively, or you can SWAP to the CLI from a program that passes the command through the common areas. The BASIC CLI prompt is an exclamation point.

binary value

A number that represents an integer value. Binary values are stored in 4 or 6 bytes, depending on the precision of your system. See also precision.

bit

A storage place in memory. Eight bits make 1 byte, and 2 bytes make 1 word. Use the AND, OR and SHFT functions to move bits around in a word or to compare the bits in one word to those in another word (sometimes called bit checking).

byte

Eight bits, from bit locations 0 to 7. It takes 1 byte to hold a character (the character's ASCII code number), 4 bytes to hold a number in a double precision system, and 6 bytes to hold a number in a triple precision system.

C1 array

See file characteristics array.

carriage return

Used to terminate a line in keyboard mode. A carriage return will also terminate any BASIC CLI command and an INPUT request from any program.

character

A character is stored as an ASCII code number in 1 byte (see also byte). It is different from a number, which is a binary value stored in 4 bytes (double precision) or 6 bytes (triple precision). Whenever we say character data, we mean string literals.

command

An instruction directing the system to do something immediately. You type BASIC commands in keyboard mode (asterisk prompt). Some BASIC statements can also be commands.

common area

A 512-byte storage area unique to each job (process). You use it to store information temporarily so that you can SWAP to another program that will pick up that information. To access the common area, use the BLOCK READ and BLOCK WRITE statements/commands. You can pass only strings and arrays that are 512 bytes long to and from the common area.

device name

A preassigned name for a device. Under AOS-AOS/VS, the names usually begin with an at sign. For example, @LPT is the line printer, @DPHn refers to a disk unit, and @DPZn refers to a diskette. Devices that are used with RDOS or DOS systems have four-character names that begin with a dollar sign or end with a colon. For example, $LPT is the line printer, and MT0: refers to a magnetic tape drive.

digit

See character.

directory

A large file that contains entries (pointers) called filenames. Under AOS or AOS/VS Business BASIC, directories are DIR-type files, and you refer to one using a pathname. Under RDOS/DOS, directory names have a .DR extension that you need to specify only if you are accessing the directory as a file. You do not have to specify the .DR extension when referring to the file in a pathname argument.

dynamic allocation

A method that allows files to reuse space left by deleted records. Business BASIC has subroutines (GETREC.SL, DELREC.SL, POSF.SL) that access and allocate dynamic file space for random files. See also linked-record format.

edit buffer

The keyboard edit commands .P, .A, .C, .E, and .I use a small edit buffer that contains the last statement LISTed or typed in, so that you can edit the statement. The EDIT utility uses a file as an edit buffer to hold lines of text while you modify them.

expression

A numeric expression consists of any combination of numbers, numeric variables, array elements, and numeric functions, linked together by arithmetic operators and parentheses. A string expression is a combination of string literals, string variables, substrings and string functions, separated by the concatenation operator (,).

file

Any collection of data. Business BASIC has program files, listing files, source files, text files, data files, subfiles of data files, index files, tag files, table files, log files, documentation files, and screen files. Directories and devices are also files. To access a file, use its filename.

file characteristics (C1) array

A Business BASIC convention necessary for using the GETREC.SL, DELREC.SL, and POSFL.SL subroutines, which provide fast subfile access. The C1 array contains the channel number of the physical file for each subfile, the byte offset in the physical file to the beginning of each subfile, the file size, and the record size.

filename

A name that refers to a file. Filenames in RDOS/DOS can have up to 10 alphanumeric characters and up to two extension characters after a period; filenames in AOS or AOS/VS can have up to 32 alphanumeric characters including underscores and extensions.

ISAM file

A file containing records that have a unique key, which allows fast reading for selected records. The actual data for the ISAM file is contained in a database file. Another file, called an index file, contains the key and a pointer to the record number in the database file.

interrupt key

The key(s) you press to interrupt a program or a command execution. It is either the ESC key, which sends a CTRL-C CTRL-A sequence to the computer, or a two-character sequence, CTRL-C CTRL-x, where x is any character except C, D, O, P, Q, S, T, U, or V.

IKEY

See interrupt key.

keyboard mode

In this mode you can type a command for immediate execution, or type program statements to create a program in working storage. If the asterisk prompt is displayed, you are in keyboard mode. All interaction takes place in this mode.

library

The Business BASIC library contains the prewritten subroutines and utilities. These subroutines and utilities are contained in the $LIB directory (or $LIB3 for triple precision systems). All users should have access to it.

linked-record format

A method of disk storage that provides dynamic record allocation for random files. It allows a file to reuse space left by deleted records. Each record contains a pointer that points to the next available record in the file. This storage method is an efficient use of disk space since it provides a way to reuse deleted record space.

listing file

A file created by the LIST command; i.e., an ASCII listing of the program currently in working storage. You can use the BASIC EDIT utility or any system editor to modify a listing file.

log file

A file containing a record of all access to a data file in FM (File Maintenance). Each data file can have a log file defined by its table file. The utility FMLOG maintains this log file.

logging off

Logging off the system is very easy. Just type BYE, and the system will log you off.

logging on

The process of identifying yourself as a valid user of the Business BASIC system. Logon procedures are described in the Business BASIC Guide.

NEWLINE

A key used as a terminator in keyboard mode (AOS-AOS/VS). On RDOS/DOS systems the CR key is used for this purpose.

number

The system stores a number as a binary value, whereas it stores a digit as a character. See also binary value.

numeric function

A function that always returns a numeric value. It can be used as a numeric expression in most statements/commands.

PARAM file

A file that contains information that you need in order to use subfiles and/or the OPEN utility program. The PARAM file for your system can be in the library ($LIB or $LIB3), or you can set up your own.

pathname

One or a series of directory names that specify the path through the directory tree structure from your current directory. To find a file in another directory in an RDOS/DOS system, specify the directory name (without the .DR extension) and the filename in the form directory name:filename.

To find a file in another directory in an AOS or an AOS/VS system, specify the path through the directory tree structure from your current directory. For example, from current directory @DPZ0:UDD:BOVE to the file TEST in user directory @DPZ0:UDD:ABS, the pathname would be ABS:TEST (up to the parent UDD, and down to ABS), or @DPZ0:UDD:ABS:TEST (up to the root, and down through UDD and ABS).

precision

The number of bytes used to store or transfer a numeric variable or array element. A double precision system stores all numeric values in 4 bytes (2 words, or 32 bits), and transfers single precision variables using only 2 bytes (1 word, or 16 bits), and double precision variables using 4 bytes (2 words, or 32 bits). Single precision variables have a percent sign appended to the variable name (for example, VAR%). Double precision variables have no sign appended to the name (VAR). Triple precision Business BASIC stores all numeric variables in 6 bytes (3 words), and transfers each variable according to its type: single precision variables with percent signs transfer only 2 bytes (1 word, or 16 bits), double precision variables with no signs transfer only 4 bytes (2 words, or 32 bits), and triple precision variables with a pound sign (#) transfer all 6 bytes (3 words, or 48 bits).

program file

A file created when you SAVE (or REPLACE) a program in working storage. The program file saves the program in interpretive code that is ready to execute. You can LOAD, RUN, CHAIN, and SWAP program files.

prompt

A character output to your terminal to signify that you must type something. The asterisk prompt signifies BASIC's keyboard mode. The BASIC CLI prompt signifies that you are running the BASIC CLI program. Other types of prompts include the line prompt (000:) in EDIT, and the command line prompt (>>) in SM.

random file

A file that allows random reading and writing and dynamic space allocation. The system maintains a small file of pointers to data blocks in the random file that find and directly access the block and the record within it. A file composed of contiguous elements is a random file.

record

A collection of related data fields treated as one unit.

record type

In FM (File Maintenance), you can define up to nine different record types. These records are the same length, but the first 2 bytes vary according to its type. These 2 bytes tell FM which type of format to use when displaying the record's page of screen fields. FM table files have six different record types.

relational operator

A symbol used to compare two expressions in an IF decision statement. The relational operators are: = (equal), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), and <> (not equal).

save file

An executable file. In BASIC, we call it a program file to differentiate a BASIC program from an executable operating system utility.

screen field

An area on your terminal's screen. Use Screen Maintenance (SM) to define screen fields in screen files so that you can use them with the FORM.SL, UNFORM.SL, and PROTFORM.SL subroutines to handle screen field input/output. A data field usually has a certain data type, such as alphanumeric, signed integer, etc, which corresponds to a screen field that would hold the data.

screen file

A file created by the Screen Maintenance (SM) utility; it usually has an .Sn extension, where n is the terminal type. A screen file can hold multiple screens. Your program can display the screen and input/output data in screen fields by using the FORM.SL, UNFORM.SL, and PROTFORM.SL subroutines.

searchlist

A list of directories that the CLI scans if it cannot find the file you specified in the current directory. You can display and change your searchlist with the CLI command SEARCHLIST. To add a directory to your searchlist, you must type its entire pathname from your working directory. The searchlist applies only to AOS/AOS-VS systems. See also pathname.

source file

An ASCII test file containing a LIST of a program (created by LIST, the EDIT utility, or some other program such as an editor). Usually a source file contains comments entered with the EDIT utility or any system editor, and a listing file is a program LISTed to file without comments. Source files usually have an .SF extension.

statement

An instruction in a program. Each BASIC statement has a line number, a BASIC keyword, and arguments.

string

A combination of characters (letters, digits, spaces, special characters, and sometimes binary values). You define a string literal by enclosing these characters in quotation marks (decimal ASCII code numbers are enclosed in angle brackets), and you assign the string literal to a string variable as if you were assigning a numeric expression to a numeric variable. For example, LET A$="ABC123.-+*<7>". A$ now contains a string of characters, digits, special symbols, and the ASCII code number 7 (for the terminal bell).

subfile

A logical file that exists within a physical file. To set up and use subfiles, you need a record in the PARAM file for the subfile and a file characteristics (C1) array in your program. You do not need subfiles if 16 channels are enough for your job.

subroutine

Any group of statements that you GOSUB to and RETURN from. Business BASIC provides prewritten subroutines in its library; these handle linked-available-record access, formatted screen input/output, percent calculations, string-to-number conversions and subfile access. These subroutines are source (listing) files with an .SL extension and are described in the Business BASIC Subroutines, Utilities, and BASIC CLI manual. You ENTER them into your program and GOSUB to them as you would to your own subroutines.

table file

A file that contains user IDs and passwords for each data file (not log-on passwords), screen formats for pages and data, and descriptors for keys. If you use File Maintenance (FM), you need a table file (with a .TB extension) to describe the multiple fields and pages of your datafile and types of data. Chapter 6 describes how to create a table file using FM.

tag file

A temporary index file that is accessed like a regular file. Its records are fixed in length, consisting of a key (string value) and record pointer (integer value); you will usually create it in sorted order using the TBUILD utility.

template

A character used to match several filenames. You can use a hyphen to represent any string of characters except a period, and an asterisk to represent any single character except a period. In RDOS or DOS Business BASIC systems you can use a plus to represent any string of characters in a filename or extension.

utility

A BASIC program that is SAVEd in the library for your use. You can RUN, CHAIN, or SWAP to some utility programs,; others require arguments passed through the common area, and you can only SWAP to them.

variable

An expression that represents a value. There are three kinds of variables: numeric, array, and string. You assign values to variables using LET, INPUT, READ and DATA, INPUT USING and TINPUT. Variable names must begin with a letter, and can have up to five alphanumeric characters after the first letter, plus a special sign ($, %, #) at the end.

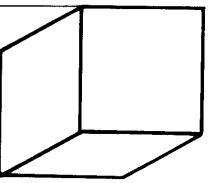
word

A unit measure for a memory storage space. Two bytes equal one word (where 8 bits equal 1 byte). String variables hold each character in 1 byte, so two characters would make up one word. See also bit, byte.

working storage

The portion of memory used to develop programs.

Appendix C
Running RDOS/DOS Programs on AOS-AOS/VS



You can run an RDOS/DOS-created program on an AOS or AOS/VS system. However, because AOS and AOS/VS function differently from RDOS/DOS, you need to make a few adjustments when transferring files from one system to another:

1. Use the RDOS.PR program to load RDOS/DOS files to be transferred to an AOS or AOS/VS system.
2. If you are going to run an RDOS/DOS program on an AOS or AOS/VS system, you must change any carriage returns embedded in string literals to NEW LINEs (that is, change all <13>s to <10>s. If you do not change the carriage returns, AOS will correctly interpret them, but it will not execute a form feed.
3. You will need to create an AOS macro called ELEMSIZE4.CLI:

   ```
   DEL/2=IGNORE ?_TMP_ELEMSIZE4.TM
   DEL/2=IGNORE ?_TMP_%1%
   FILESTATUS/ELEMENTSIZE/2=ABORT
   %1%
   CREATE/ELEMENTSIZE=4
   ?_TMP_ELEMSIZE4.TM
   COPY/APPEND/2=ERROR
   ?_TMP_ELEMSIZE4.TM %1%
   RENAME %1% ?_TMP_%1%
   RENAME ?_TMP_ELEMSIZE4.TM %1%
   FILESTATUS/ELEMSIZE %1%
   ```

To move a SAVE file or a nontext data file from RDOS/DOS to AOS or AOS/VS:

4. On RDOS/DOS machines, execute the following commands:

   ```
   INIT MT0
   DUMP/V MT0:0 filename
   RELEASE MT0
   ```

5. On AOS or AOS/VS machines, execute the following commands:

   ```
   X RDOS LOAD/V @MTA0:0 filename
   ```

To move a text file from RDOS/DOS to AOS or AOS/VS, steps 4 and 5 are as follows:

4. On RDOS/DOS machines, execute the following commands:

   ```
   INIT MT0
   DUMP/V MT0:0 filename
   RELEASE MT0
   ```

5. On AOS or AOS/VS machines, execute the following commands:

   ```
   X RDOS LOAD/V @MTA0:0 filename/C
   ELEMSIZE4 filename
   ```

   The ELEMSIZE4 command is necessary only if you intend to open the file in shared mode.

To move a SAVE file from AOS or AOS/VS to RDOS/DOS, steps 4 and 5 are:

4. On AOS or AOS/VS machines, execute the following command:

   ```
   X RDOS DUMP/V @MTA0:0 filename
   ```

5. On RDOS/DOS machines, execute the following commands:

   ```
   INIT MT0
   LOAD/V MT0:0 filename
   RELEASE MT0
   CHATR filename S
   ```

To move a text file or a nontext data file from AOS or AOS/VS to RDOS/DOS, steps 4 and 5 are different from above:

4. On AOS machines, execute the following command:

   ```
   X RDOS DUMP/V @MTA0:0 filename
   ```

5. On RDOS/DOS machines, execute the following commands:

   ```
   INIT MT0
   LOAD/V MT0:0 filename
   RELEASE MT0
   ```

When moving any files from AOS or AOS/VS to an RDOS/DOS system, make sure that the filename is a valid RDOS/DOS filename.

Compatibility Considerations

The following differences exist between Business BASIC under RDOS/DOS and AOS-AOS/VS. They should be kept in mind when creating programs that will be transported to AOS or AOS/VS systems.

1. The DETACHED job feature is not implemented.
2. STMD and MSG are not implemented.
3. The following STMC commands are not implemented:

STMC 2	STMC 23	STMC 29
STMC 3	STMC 24	STMC 30
STMC 4	STMC 26	STMC 31
STMC 10	STMC 27	STMC 32
STMC 33	STMC 46	STMC 51
STMC 34	STMC 47	
STMC 36	STMC 48	
STMC 45	STMC 49	

4. The Business BASIC CLI commands FDUMP and FLOAD are not implemented.
5. NEWLINE replaces CR.
   a. NEWLINE is the default primary unpend character.
   b. Text file records must end with NEWLINE.
   c. Programs that contain CR characters in string literals or in numeric variables must be changed to use NEWLINE instead.
6. IKEY characters are restricted.

   IKEY is either the ESC key or a two-character control sequence, CTRL-C CTRL-x, where x is any character except C, D, O, P, Q, S, T, U, V. ESC and CTRL-C CTRL-A are equivalent. Use of the BREAK key is not necessary with terminal type 8 or 9.

   Terminal I/O is done using binary reads of one character. For an IKEY sequence to be recognized, it will, therefore, be necessary to press the BREAK key before the control sequence.
7. STME statements are available under AOS or AOS/VS.
8. All input/output errors are reported with RDOS error codes. If an AOS or AOS/VS error cannot be translated to an equivalent RDOS/DOS error code, I/O error 60 occurs. A new function SYS(31) will return the true AOS/AOS-VS error number and AERM$ will return the AOS/AOS-VS error message.
9. A file that will be shared (mode 3, 4, or 5) must have an element size equal to a multiple of 4.
10. Terminal I/O.

    The current terminal type 6 is very compatible with RDOS/DOS Business BASIC type 6. This was obtained at a loss of performance. Terminal type 8 generally improves performance with a minor loss of compatibility.

    Output to the terminal is not performed immediately but is accumulated in a 512-byte buffer. The buffer is written to the terminal under the following conditions: an INPUT statement, a program change (such as SWAP), IKEY, a DELAY statement, STOP, END, or a full buffer. You can prematurely write the buffer to the terminal by executing the STMA 8, 5 statement.
11. File links.

    AOS Business BASIC handles links the way they are handled under RDOS/DOS.

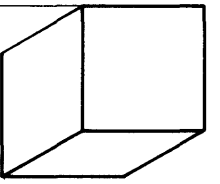
    Deleting a link will delete the resolution file. To delete a link, use UNLINK. Links should contain complete pathnames to the resolution file. For example, use CREATE/LINK A :UDD:USER:B, rather than CREATE/LINK A B.
12. New SYS function.

    SYS(30) returns a code indicating the precision of Business BASIC, and a code that indicates the host operating system.
13. Record locking.

    Record locking is slower on AOS and AOS/VS systems than on RDOS/DOS because the Business BASIC process must communicate with a "server" process (RLS) via the IPC mechanism.
14. File deletion.

    AOS and AOS/VS systems allow you to delete a file even if the file is open by yourself or another user. At the time of the DELETE call, AOS and AOS/VS systems mark the file for deletion. The file is not actually deleted until the final CLOSE of the file. RDOS/DOS returns an error on an attempt to delete an open file.

Appendix D
Keyword and Utility Summary



A keyword is a word whose meaning the Business BASIC interpreter recognizes or a command that the BASIC CLI recognizes. The EDIT, FM, SM, CFM, CSM, LFM, LFU, BASIC CLI, and DOC utilities are BASIC programs that recognize their own keywords. With the exception of the BASIC CLI commands, the keywords recognized by these utility programs are not listed here.

Table D-1 groups Business BASIC keywords by suggested applications. Commands, statements, and functions are described in the *Business BASIC Commands, Statements, and Functions* manual. Subroutines, utilities, and BASIC CLI commands are described in the *Business BASIC Subroutines, Utilities, and BASIC CLI* manual.

Keyword	Function	Type
AERM$	Returns native operating system and Business BASIC error messages	function
CHAIN	executes a utility or another program from your program	statement/command and BASIC CLI command
CON	continues execution of a program (for debugging)	command
DATA	Specifies data for variables in READ statements	statement
DBCLOSE	closes an open INFOS II file (AOS, AOS/VS only)	statement
DBDELETE	deletes a key and/or record (AOS, AOS/VS only)	statement
DBGET	gets values returned by INFOS II (AOS, AOS/VS only)	statement
DBOPEN INFOS	opens an INFOS II index (AOS, AOS/VS only)	statement
DBREAD	reads from an INFOS II file (AOS, AOS/VS only)	statement
DBREINSTATE	reinstates a logically deleted record from an INFOS II file (AOS, AOS/VS only)	statement
DBRELEASE	releases locks and/or position in an INFOS II file (AOS, AOS/VS only)	statement
DBRETRIEVE HIGHKEY	retrieves the high key in an INFOS II file (AOS, AOS/VS only)	statement
DBRETRIEVE KEY	retrieves the key in an INFOS II file (AOS, AOS/VS only)	statement
DBRETRIEVE SIDEF	retrieves a subindex definition in an INFOS II file (AOS, AOS/VS only)	statement
DBRETRIEVE STATUS	returns the INFOS II status (AOS, AOS/VS only)	statement
DBREWRITE	rewrites an INFOS II database record (AOS, AOS/VS only)	statement
DBSET	permanently sets an INFOS II parameter (AOS, AOS/VS only)	statement
DBSUBINDEX DEFINE	define a subindex for an INFOS II file (AOS, AOS/VS only)	statement
DBSUBINDEX DELETE	deletes a subindex for an INFOS II file (AOS, AOS/VS only)	statement
DBSUBINDEX LINK	links a subindex for an INFOS II file (AOS, AOS/VS only)	statement
DBSUBINDEX LINKINIT	initializes a link subindex string for an INFOS II file (AOS, AOS/VS only)	statement
DBSUBINDEX LINKSET	sets parameters in link subindex string for an INFOS II file (AOS, AOS/VS only)	statement
DBWRITE	writes to an INFOS II file (AOS, AOS/VS only)	statement
DEF	defines your own function	statement
DELAY	delays execution of the next statement or command	statement/command
DIM	dimensions arrays and strings	statement/command
END	optional program terminator	statement
ENTER	brings in/merges listing file	statement/command
ERM$	returns RDOS-compatible or Business BASIC error messages	function
FOR/NEXT	defines a program loop	statements
GOSUB/RETURN	executes a subroutine and return	statements
GOTO	goes to a specific statement	statement
GQUE	displays the default queue	BASIC CLI command
IF	conditional execution of a statement	statement

Table D-1 Program Development Keywords (continues)

Keyword	Function	Type
IF THEN ELSE	conditional execution of a statement	statement
LET	assigns a value to a variable	statement/command
LIST	displays program in working storage or creates listing file	command
LOAD	brings in a SAVEd program	command
NEW	clears working storage	statement/command
ON ERR	traps an error in your program	statement
ON GOTO ON GOSUB	conditional transfer to a statement or a subroutine	statements
ON IKEY	traps an interrupt in your program	statement
READ	assigns DATA values to variables	statement
REM	remark statements	statement
RENUMBER	renumbers program statements	command
RENUM	renumbers a program in segments and optionally removes REM statements	utility program and BASIC CLI command
REPLACE	replaces a SAVEd program with current program	statement/command
RESTORE	resets a DATA list pointer	statement/command
RNAM	renames program variables	utility program and BASIC CLI command
RUN	executes a program	command
SAVE	saves a program in a program file	statement/command
STOP	stops program execution	statement
SWAP	executes another program and return	statement/command
TABLE	prints a program cross- reference	BASIC CLI command
UCALL	calls an assembly language subroutine	statement/command
VAR	lists variables	utility program and BASIC CLI command

Table D-1 Program Development Keywords

Keyword	Function	Type
.(period)	sends contents of edit buffer to working storage	command
.A	appends to contents of edit buffer	command
.C	changes characters in edit buffer and sends to working storage	command
.E	changes characters in edit buffer	command
.I	changes entire contents of edit buffer	command
.P	displays contents of edit buffer	command
ERASE	erases statements in working storage	statement/command

Table D-2 Keyboard Editing Keywords

Keyword	Function	Type
ABS	absolute value of a number	function
AND	logical AND of two numbers	function
ASC	extracts a number from a string	function
BCDBIN.SL	converts an eight-byte packed decimal string to a binary number.	subroutine
BINBCD.SL	converts a binary number to an eight-byte packed decimal string.	subroutine
CHR$	puts binary value into string	function
CRM$	crams 3 bytes of string into 2 bytes	function
DAY.SL	used in the HELLO program to check the date and time (RDOS/DOS only)	subroutine
DIV.SL	calculates extended percentage	subroutine
DRCHK.SL	used in the HELLO program to check directory privileges (RDOS/DOS only)	subroutine
EXTRACT	extract the next field from a string using specified delimiters	statement
FILL$	fills a string with a value	function
FIX.SL	converts a floating-point number to a fixed point number	subroutine
FLOAT.SL	converts a fixed-point number to a floating point number	subroutine
LEN	determines length of string	function
MAX	determines larger of two numbers	function
MIN	determines smaller of two numbers	function
MOD	determines remainder after dividing two numbers	function
MUL.SL	calculates a percent of a number	subroutine
OR	logical inclusive OR of two numbers	function
PACK	Encode string and numeric information into a single string variable	statement/command
POS	determines position of substring in a string	function
QADD	quad-precision add	statement/command
QDIV	quad-precision divide	statement/command
QLOAD	loads the 4-byte value of a numeric expression into a string variable	statement/command
QMUL	quad-precision multiply	statement/command
QSTORE	converts a quad precision string into a double-precision variable	statement/command
QSUB	quad precision subtract	statement/command
RANDOMIZE	reseeds random number generator	statement/command
RFORM	allows the definition of record formatting to be done outside of the PACK and UNPACK statements	statement
RND	random number generator	function
SCANWHILE	scan a string while characters match a specified set	statement
SCANUNTIL	scan a string until characters in a specified set are encountered	statement
SGN	determines sign of a number	function
SHFT	logical bit shift of a number	function
SQR	determines square root of a number	function
STRPOS	find a string position in another string	statement/command
TRUN$	truncates a string	function
UCM$	uncrams a string	function
UNPACK	parse (syntactically analyze) a record string	statement/command
VAL	converts a string to a number	function
VAL.SL	converts a string to a number	subroutine
VALUE	converts a string to a number	statement/command

Table D-3 Data Manipulation Keywords

Keyword	Function	Type
BLOCK READ [FILE]	reads blocks from file or common area	statement/command
BLOCK WRITE [FILE]	writes blocks to file or common area	statement/command
CLOSE [FILE]	closes any or all files	statement/command
DELREC	deletes a record in an LOPENed linked-record file	statement/command
DELREC.SL	deletes a record in a linked-record file	subroutine
EOF	tests for end of file	function
GETLAST.SL	retrieves last useable record number for an LOPENed linked-record file	subroutine
GETREC	gets available record in an LOPENed linked-record file	statement/command
GETREC.SL	gets available record in a linked-record file	subroutine
GPOS	tests for file position	function
FM	invokes File Maintenance program	utility program and BASIC CLI command
FMLOG	prints a log file (FM)	utility program and BASIC CLI command
FMPRINT	prints a file maintained by FM	utility program and BASIC CLI command
FMTABPRINT	prints a table file (FM)	utility program and BASIC CLI command
FORM.SL	formatted screen input/output	subroutine
INPUT [FILE]	inputs characters from file or from terminal	statement/command
INPUT USING	inputs with error trap	statement/command
KADD	adds a key to an index file	statement/command
KDEL	deletes a key from an index file	statement/command
KFIND	finds a key in an index file	statement/command
KNEXT	finds the next key in an index file	statement/command
LFM	invokes Logical File Maintenance program	utility program and BASIC CLI command
LOCK	locks blocks of a file	statement/command
LOPEN FILE	open a logical file	statement/command
LREAD FILE	read a record from a logical file	statement/command
LWRITE FILE	write a record to a logical file	statement/command
MTDIO	magnetic tape input/output	statement/command
OPEN	opens several files including subfiles	utility program
OPEN FILE	opens a file or device for access	statement/command
POSFL.SL	positions to a record in a data file	subroutine
POSITION FILE	positions to a byte in a file	statement/command
PRINT [FILE]	outputs characters to a file, terminal, or device	statement/command
PRINT USING	outputs using a format	statement/command
PROTFORM.SL	protects/unprotects screen fields	subroutine
READ FILE	inputs data from a file	statement/command
SFORM.SL	Allows screen field I/O	subroutine
TINPUT [FILE]	timed INPUT from a file or terminal	statement/command
UNFORM.SL	unformatted screen I/O	subroutine
UNLOCK	unlocks blocks of a file	statement/command
WRITE FILE	outputs data to a file	statement/command

Table D-4 File Input/Output Keywords

Keyword	Function	Type
APPEND	appends one file to another	BASIC CLI command
BLDCOM	builds a documentation file	BASIC CLI command
BUILD	builds a command file	BASIC CLI command
CCONT	creates a file by element size	BASIC CLI command
CDIR	creates a directory	BASIC CLI command
CPART	creates a directory	BASIC CLI command
CRAND	creates a user data file	BASIC CLI command
CREATE	creates a user data file	BASIC CLI command
DBGEN	prompts a user for the information needed to create a database of one data file with up to three index files, and then initializes the files and constructs a table file automatically.	utility program and BASIC CLI command
DBGENT	triple precision version of DBGEN (described above); stores and retrieves full triple precision numeric fields	utility program and BASIC CLI command
DBMOVE	move a database from one directory to another	BASIC CLI
DELETE	deletes a file or directory	statement/command and BASIC CLI command
DOC	prints a series of text files	utility program and BASIC CLI command
DOCTOC	prints table of contents of DOC files	utility program and BASIC CLI command
EDIT	creates and edits text files (invokes the EDIT program)	utility program and BASIC CLI command
FILCOM	compares two files	BASIC CLI command
FILESORT	sorts a data file	utility program and BASIC CLI command
FINDFILE.SL	finds a subfile	subroutine
IBUILD	builds an index file	utility program and BASIC CLI command
INDEXBLD	builds or rebuilds index files	utility program and BASIC CLI command
INDEXCALC	calculates index file size	utility program and BASIC CLI command
INDEXPRT	prints an index file	utility program and BASIC CLI command
INITFILE	initializes index or data files	utility program and BASIC CLI command
INITINDEX.SL	initializes an index file	subroutine

Table D-5 File Manipulation Keywords (continues)

Keyword	Function	Type
IREBLD	rebuilds an index file	utility program and BASIC CLI command
LFDATA.SL	get a logical file description	subroutine
LFU	logical file utility used to create, delete, display, and rename files in a logical file database structure	utility program and BASIC CLI command
LINDEXBLD	builds an index file	utility program and BASIC CLI command
LINDEXPRT	prints an index file	utility program and BASIC CLI command
LINITINDEX.SL	initializes an index file that was LOPENed	subroutine
LRELINK	recreates the deleted record linked list in a linked-record file	BASIC CLI
LSTCOM	compares two listing files	BASIC CLI command
LSTMERGE	merges two listing files	BASIC CLI command
LXFER	copies one logical database file to another logical file	utility program and BASIC CLI command
MOVE	moves files to another directory	BASIC CLI command
MOVETABREC	copies FM table file records	utility program and BASIC CLI command
PARAMCON	converts a PARAM file database structure into a logical database file structure	utility program and BASIC CLI command
PRINT	prints a text file	BASIC CLI command
PRTCOM	prints a documentation file built by BLDCOM	BASIC CLI command
QFILESORT	quick-sorts a data file	utility program and BASIC CLI command
RELINK	recreates deleted-record chain of linked-record file	utility program and BASIC CLI command
RENAME	renames a file	statement/command and BASIC CLI command
TABBUILD	defines arrays for table file records (FM)	utility program and BASIC CLI command
TBUILD	builds a tag file BASIC CLI command	utility program and
TYPE	displays a text file	BASIC CLI command
VLPRINT	volume label print, displays information about a physical file	BASIC CLI command
XBUILD	builds an index file	utility program and BASIC CLI command
XFER	copies one file to another file	BASIC CLI command

Table D-5 File Manipulation Keywords

Keyword	Function	Type
@	terminal control and cursor positioning	expression used with PRINT statement/command
ASG	assigns a device for exclusive use (RDOS/DOS only)	BASIC CLI command
ATTACH	attaches a job to a terminal (RDOS/DOS only)	utility program and BASIC CLI command
CHAR	changes or displays device characteristics (AOS only)	statement/command
CSM	builds and maintains a screen file	utility program and BASIC CLI command
DIR	changes current disk directory	statement/command
DIR	changes current disk directory	BASIC CLI command
DIRS	displays all initialized directories (RDOS only)	utility program and BASIC CLI command
DUMP	dumps files to disk or tape	BASIC CLI command
DISMOUNT	releases a disk or diskette from its drive (AOS, AOS/VS only)	BASIC CLI command
EQUIV	renames a device (RDOS/DOS only)	BASIC CLI command
FDUMP	fast-dumps one or more files to mag tape (RDOS/DOS only)	BASIC CLI command
FLOAD	fast-loads FDUMPed files (RDOS/DOS only)	BASIC CLI command
FPRINT	dumps a file to line printer or output file	BASIC CLI command
FREE	Frees a device from exclusive (ASG) use (RDOS/DOS only)	utility program and BASIC CLI command
INIT	initializes a directory or device (RDOS/DOS only)	BASIC CLI command
LOAD	loads DUMPed files	BASIC CLI command
LSPEED	sets multiplexor line speed (RDOS/DOS only)	utility program and BASIC CLI command
MSG	sends a message to another terminal or process (RDOS/DOS only)	command
MTDIO	magnetic tape input/output	statement/command
OPEN FILE	opens a file or device for access	statement/command
PAGE	sets page width of terminal	command
RELEASE	releases a directory or device (RDOS/DOS only)	BASIC CLI command
REWIND	rewinds a magnetic tape (AOS, AOS/VS only)	AOS CLI command
SDIR	sets the system directory (RDOS only)	BASIC CLI command
SLINE	selects a line and attaches job to it (RDOS/DOS only)	BASIC CLI command
SM	builds screen file and manipulates screen	utility program and BASIC CLI command
SPDIS	disables spooling (RDOS/DOS only)	BASIC CLI command
SPEBL	enables spooling (RDOS/DOS only)	BASIC CLI command
SPKILL	kills spooling (RDOS/DOS only)	BASIC CLI command
SPCLI	spooler command line interpeter	utility program and BASIC CLI command
START	starts a detached job (RDOS/DOS only)	BASIC CLI command
TAB	sets tab zones for output to terminal	command
TAB	moves cursor for output	expression used with PRINT statement/command
TCOPY	copies from tape to tape	BASIC CLI command
TERM	changes certain terminal key functions (RDOS/DOS only)	utility program and BASIC CLI command
TFER	copies a file between tape and disk (RDOS/DOS only)	BASIC CLI command
ULSPEED	sets ULM line speed	utility program and BASIC CLI command
VFU	pushes to the AOS program FCU.PR. Filename argument no longer required. Business BASIC ignores it (AOS, AOS/VS). Edits a format control file for a data channel line printer (RDOS/DOS).	BASIC CLI command

Table D-6 Device Control and Access Keywords

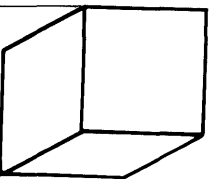
Keyword	Function	Type
ACL	sets/displays file access control list (AOS, AOS/VS only)	statement/command
AOS	executes AOS CLI or AOS CLI command (AOS, AOS/VS only)	BASIC CLI command
BYE	logs off BASIC (RDOS/DOS), terminates BASIC (AOS, AOS/VS)	statement/command and BASIC CLI command
CHATR	changes file attributes (RDOS/DOS only)	BASIC CLI command
CHLAT	changes link attributes (RDOS/DOS only)	BASIC CLI command
CLI	executes BASIC CLI program	utility program and BASIC CLI command
GETCM.SL	is used to write your own BASIC CLI commands	subroutine
LINK	links an alternate name to a file	BASIC CLI command
POP	terminates BASIC CLI and clears common area	BASIC CLI command
QUIT	terminates BASIC CLI but retains common area	BASIC CLI command
STMA	user system call	statement/command
UNLINK	removes a link entry (RDOS/DOS only)	BASIC CLI command

Table D-7 System Functions and Security Keywords

Keyword	Function	Type
DISK	displays remaining disk space	BASIC CLI command
FILES	displays filenames in current directory (AOS, AOS/VS only)	AOS CLI command
GDIR	displays current directory name	BASIC CLI command
GSDIR	displays the current system directory name (RDOS/DOS only)	BASIC CLI command
GSYS	displays the operating system name (RDOS/DOS only)	BASIC CLI command
GTOD	displays the time and date	BASIC CLI command
LIBRARY	displays filenames in library directory	utility program and BASIC CLI command
LIST	displays information for files in current directory	BASIC CLI command
LOCKS	displays your current locks (RDOS/DOS only)	utility program and BASIC CLI command
MDIR	displays master directory name (RDOS/DOS only)	BASIC CLI command
PD	program information displayer	utility program and BASIC CLI command
PED	display the system status	utility program and BASIC CLI command
PORTS	displays jobs on the system	utility program and BASIC CLI command
SCHANS	displays system channel assignments (RDOS/DOS only)	utility program and BASIC CLI command
SIZE	displays size of program	command
SIZE	displays current workspace allocations for a specific job (process)	utility
STAT	displays stats of all jobs (processes)	utility program and BASIC CLI command
SYS	returns system information	function
TPRINT	prints the tuning report (RDOS only)	BASIC CLI command
UCHANS	displays your channel assignments	utility program and BASIC CLI command

Table D-8 System Information Keywords

Appendix E
ASCII Character Sets

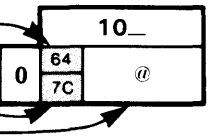


ASCII Character Set

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.

Legend:

Character code in decimal
EBCDIC equivalent hexadecimal code
Character



Octal	00_		01_		02_		03_		04_		05_		06_		07_	
0	0 00	NUL	8 16	BS (BACK- SPACE)	16 10	DLE ↑P	24 18	CAN ↑X	32 40	SPACE	40 4D	(	48 F0	0	56 F8	8
1	1 01	SOH ↑A	9 05	HT (TAB)	17 11	DC1 ↑Q	25 19	EM ↑Y	33 5A	!	41 5D	)	49 F1	1	57 F9	9
2	2 02	STX ↑B	10 15	NL (NEW LINE)	18 12	DC2 ↑R	26 3F	SUB ↑Z	34 7F	,, (QUOTE)	42 5C	*	50 F2	2	58 7A	:
3	3 03	ETX ↑C	11 0B	VT (VERT TAB)	19 13	DC3 ↑S	27 27	ESC (ESCAPE)	35 7B	#	43 4E	+	51 F3	3	59 5E	;
4	4 37	EOT ↑D	12 0C	FF (FORM FEED)	20 3C	DC4 ↑T	28 1C	FS ↑\	36 5B	$	44 6B	, (COMMA)	52 F4	4	60 4C	<
5	5 2D	ENQ ↑E	13 0D	RT (RETURN)	21 3D	NAK ↑U	29 1D	GS ↑]	37 6C	%	45 60	-	53 F5	5	61 7E	=
6	6 2E	ACK ↑F	14 0E	SO ↑N	22 32	SYN ↑V	30 1E	RS ↑↑	38 50	&	46 4B	. (PERIOD)	54 F6	6	62 6E	>
7	7 2F	BEL ↑G	15 0F	SI ↑O	23 26	ETB ↑W	31 1F	US ↑←	39 7D	' (APOS)	47 61	/	55 F7	7	63 6F	?

Octal	10_		11_		12_		13_		14_		15_		16_		17_	
0	64 7C	@	72 C8	H	80 D7	P	88 E7	X	96 79	` (GRAVE)	104 88	h	112 97	p	120 A7	x
1	65 C1	A	73 C9	I	81 D8	Q	89 E8	Y	97 81	a	105 89	i	113 98	q	121 A8	y
2	66 C2	B	74 D1	J	82 D9	R	90 E9	Z	98 82	b	106 91	j	114 99	r	122 A9	z
3	67 C3	C	75 D2	K	83 E2	S	91 8D	[	99 83	c	107 92	k	115 A2	s	123 C0	{
4	68 C4	D	76 D3	L	84 E3	T	92 E0	\	100 84	d	108 93	l	116 A3	t	124 4F	|
5	69 C5	E	77 D4	M	85 E4	U	93 9D	]	101 85	e	109 94	m	117 A4	u	125 D0	}
6	70 C6	F	78 D5	N	86 E5	V	94 5F	↑ or ^	102 86	f	110 95	n	118 A5	v	126 A1	~ (TILDE)
7	71 C7	G	79 D6	O	87 E6	W	95 6D	← or _	103 87	g	111 96	o	119 A6	w	127 07	DEL (RUBOUT)

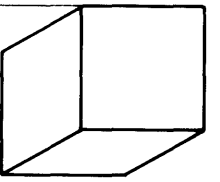
Character code in octal at top and left of charts.

↑ means CONTROL

DG-25256

Figure E-1 *ASCII Character Sets*

Appendix F
Error Messages



This appendix lists all the error messages that Business BASIC can generate. Under AOS, your system can also return exceptional condition messages described in the *AOS CLI User's Manual* and the *AOS Programmer's Manual*. The first number is the record number in the BASIC.ER file of error messages. The second number is the error code that BASIC returns in SYS(7). You may use either number in the ERM$ string function. You may add your own error messages at the end of the file (640 and above), using File Maintenance and table file ERRORS.TB.

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
0	0	Illegal channel number
1	1	Illegal character
2	2	Statement or command syntax is invalid
3	3	READ/DATA types inconsistent
4	4	SYSTEM (program is lost)
5	5	Statement number
6	6	Excessive number of variables (more than 348)
7	7	Statement may not be used as a command
8	8	Specification
9	9	Illegal reserved file name
10	10	Reserved file in use
11	11	Parenthesis
12	12	Illegal command
13	13	Line number
14	14	No more program memory
15	15	End of DATA
16	16	Arithmetic
17	17	Unassigned variable
18	18	GOSUB nesting
19	19	RETURN - no GOSUB
20	20	FOR nesting
21	21	FOR - no NEXT
22	22	NEXT - no FOR

Table F-1 Business BASIC Error Messages (continues)

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
23	23	No more data memory
24	24	No available channels
25	25	Option is not in this system
26	26	Program/data overflow
27	27	Illegal file number
28	28	New dimension exceeds old dimension
29	29	Expression
30	30	Illegal mode
31	31	Subscript
32	32	Undefined function
33	33	Function nesting
34	34	Function argument
35	35	Illegal format string
36	36	String size
37	37	User routine
38	38	Undimensioned string
39	39	Duplicate matrix
40	40	Matrices sizes
41	41	Matrix DIM
42	42	File already opened
43	43	Matrix not square
44	44	File unopened
45	45	Illegal record length
46	46	Input invalid
47	47	Wrong mode
48	48	Not a save file
49	49	No room for directory
50	50	Invalid operator command
51	51	User not on system
52	52	User in NOMSG state
53	53	Renumbering error(s)
54	54	Statement length
55	55	Block I/O error
56	56	Attempt to LOCK same area twice
57	57	Data base record is LOCKed
58	58	Incompatible save file
59	59	Zero STEP
60	60	Line too long
61	61	Missing right parenthesis in format
62	62	Save file is run only
63	63	Incorrect number of arguments
64	64	Incorrect argument type
65	65	Program swapping error
66	66	String argument in error
67	67	Error in index file or index parameters
68	68	Index file full
69	69	Excessive IPB errors
70	70	Invalid job/terminal number
71	71	Job/terminal already attached
72	72	Job is not waiting on input
73	73	Edit buffer is empty
75	75	Program may not be listed
76	76	Non-fatal system error
77	77	Illegal record number
79	79	Undefined RLS function code
80	80	Error executing ON ERR statement
81	81	Illegal record status
82	82	Data file full
83	83	Error from INFOS II
84	84	Not a data base file
85	85	Illegal command with optimized file
86	86	LFTABL$ string error
87	87	Missing ELSE or END IF
88	88	Extra ELSE or missing END IF
89	89	Illegal file type
90	90	Logical file does not exist
91	91	Attempt to issue LOCK/UNLOCK with RLS not running
92	92	Parameter range error

Table F-1 Business BASIC Error Messages

Record #	in BASIC.ER	Error Code Returned by SYS(7)
128	128	Invalid or out-of-range field
129	129	No key was given
130	130	Record does not exist
131	131	Record already exists
132	132	Validation error, key was changed
133	133	Function is invalid without a FIND function first
134	134	Control file is in error
135	135	Invalid format or page
136	136	No next record
137	137	Index for this record may be all messed up
138	138	Key could not be deleted
139	139	Request not completed
140	140	Invalid account/password
141	141	INPUT timed out
142	142	Device is assigned
143	143	Illegal function
144	144	File not found
146	146	Key already exists and duplicates not allowed
147	147	Insufficient space for logical file
148	148	File not on sector boundary
149	149	Record out of sequence
150	150	Illegal blocking factor
151	151	Illegal key length
152	152	Data dictionary does not exist
153	153	File types do not match
154	154	Wrong program

Table F-2 Utility Program Error Messages

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
256	0	Illegal channel number
257	-1	Illegal file name
258	-2	Illegal system command
259	-3	Illegal command for device
260	-4	Not a save file
261	-5	File already exists
262	-6	End of file
263	-7	File read protected
264	-8	File write protected
265	-9	File already exists
266	-10	File does not exist
267	-11	Permanent file
268	-12	File attribute protected
269	-13	File not open
270	-14	Fatal utility error
271	-15	Execute CLI.CM (no error)
272	-16	Invisible error code
273	-17	Channel already in use
274	-18	Line too long
275	-19	Attempt to restore a non-existent image
276	-20	Parity error
277	-21	Push depth exceeded
278	-22	Insufficient memory to execute program
279	-23	File space exhausted
280	-24	File data error
281	-25	Unit improperly selected
282	-26	Illegal starting address
283	-27	Attempt to read into system space
284	-28	File accessible by block I/O only
285	-29	Files on different directories
286	-30	Device not in system
287	-31	Illegal overlay number
288	-32	File not accessible by block I/O
289	-33	Invalid time or date
290	-34	Out of TCB's
291	-35	Signal to busy address
292	-36	File already squashed error
293	-37	Device already in system
294	-38	Insufficient contiguous blocks
295	-39	Simultaneous read or write to mux line
296	-40	Error in user task queue table
297	-41	No room for directory
298	-42	Illegal directory specifier
299	-43	Unknown directory specifier
300	-44	Partition too small

Table F-3 RDOS/DOS and Input/Output Error Messages (continues)

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
301	-45	Directory depth exceeded
302	-46	Directory in use
303	-47	Link depth exceeded
304	-48	File in use
305	-49	Task ID error
306	-50	Common size error
307	-51	Common usage error
308	-52	File position error
309	-53	Insufficient room in data channel map
310	-54	Directory not initialized
311	-55	No default directory
312	-56	Foreground already active
313	-57	Error in partition set
314	-58	Directory shared
315	-59	No room for UFT's
316	-60	Native operating system error - no RDOS equivalent
317	-61	Not a link entry
318	-62	Program not checkpointable
319	-63	SYS.DR error
320	-64	MAP.DR error
321	-65	Device timeout
322	-66	Entry not accessible by a link
323	-67	Not a source file
324	-68	Transmission terminated by receiver
325	-69	System deadlock
326	-70	I/O terminated by channel close
327	-71	Spool files active
328	-72	Task not found for abort
329	-73	Device previously opened
330	-74	System stack overflow
331	-75	No MCA receive request outstanding
332	-76	Attempt to release an open device
333	-77	.XMT or .IXMT messages must be non-zero
334	-78	'You can't do that'
335	-79	.TOVLD not loaded for queued overlay tasks
336	-80	Operator messages not SYSGEN'ed
337	-81	Disk format error
338	-82	Disk has invalid bad block table
339	-83	Insufficient space in bad block pool (core)
340	-84	Attempt to create a zero length contiguous file
341	-85	Program is not swappable

Table F-3 RDOS/DOS and Input/Output Error Messages (continues)

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
342	-86	Blank tape
343	-87	ALM line not ready
344	-88	Console interrupt received
345	-89	Read overrun error
346	-90	Read framing error
347	-91	Too many soft errors
348	-92	QTY buffer overflow
383	-127	Device is assigned, but not to you
384	-128	Illegal function
385	-129	Variable length blocks not allowed
386	-130	Rewrite mode not allowed for non-disk device
387	-131	Illegal function for device
388	-132	Open processing error
389	-133	Function does not process specified record format
390	-134	File already in use
391	-135	File currently opened for exclusive use
392	-136	File not open
393	-137	Peripheral device currently opened
394	-138	VL file processing error
395	-139	Unresolved resource conflict
396	-140	Rewrite inverted with changed data address
397	-141	File name already in use
398	-142	Block size exceeds window
399	-143	Virtual memory exhausted
400	-144	System translation table load error
401	-145	The system cannot open the file as requested
402	-146	Volume close error
403	-147	Insufficient space to open file
404	-148	Beyond logical bounds of file
405	-149	Translation specification error
406	-150	Volume does not exist
407	-151	Record is locked and 'hold' not requested
408	-152	Disk space exhausted
409	-153	Record beyond bounds of file
410	-154	Error while closing volume during internal volume switch
411	-155	Physical I/O error
412	-156	Residual disk error
413	-157	Disk or magnetic tape timeout

Table F-3 RDOS/DOS and Input/Output Error Messages (continues)

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
414	-158	Illegal access method
415	-159	Insufficient parameters for translation
416	-160	Special file open error in preopen
417	-161	Special file close error in preopen
418	-162	Internal special file close error
419	-163	RDOS open failure
420	-164	Volume already exists
421	-165	Zero length transfer request on disk device
422	-166	ISAM update conflict
423	-167	Index naming error
424	-168	Index not in data base index definition file
425	-169	File name too long
426	-170	No node space available
427	-171	Mag tape processing error
428	-172	System does not support device
429	-173	End of volume for output file
430	-174	End of volume for input file
431	-175	ISAM comparison error
432	-176	ISAM resolution error
433	-177	Illegal relative motion
434	-178	Invalid node address encountered internally
435	-179	Invalid entry address encountered internally
436	-180	Top level error
437	-181	Subindices not allowed
438	-182	Subindex not present
439	-183	(Sub)index boundary encountered
440	-184	Delete positioning error
441	-185	Multiple key write error
442	-186	Key too long or of zero length
443	-187	Invalid entry number encountered internally
444	-188	Neither 'keyed' nor 'relative motion' specified
445	-189	Key already exists and duplicates not allowed
446	-190	Key positioning error
447	-191	Illegal record length
448	-192	Not enough arguments
449	-193	Illegal attribute
450	-194	No debug address
451	-195	Command line too long
452	-196	No starting address
453	-197	Checksum error

Table F-3 RDOS/DOS and Input/Output Error Messages (continues)

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
454	-198	No source file specified
455	-199	Not a command
456	-200	Illegal block type
457	-201	No files match specifier
458	-202	Phase error
459	-203	Too many arguments
460	-204	Too many active devices
461	-205	Illegal numeric argument
462	-206	Fatal system utility error
463	-207	Illegal argument
464	-208	Improper or malicious input
465	-209	Too many level of indirect files
466	-210	Syntax error
467	-211	Bracket error
468	-212	Parenthesis error
469	-213	Unmatched <>
470	-214	Illegal nesting of <> and ()
471	-215	Illegal indirect filename
472	-216	Illegal nesting of () and {}
473	-217	Illegal variable
474	-218	Illegal text argument
475	-219	Text argument too long
512	-256	No data base record for index entry
513	-257	Node size too big for page
514	-258	Node size will not hold 3 entries
515	-259	Data base record is locked
516	-260	Subindex in use encountered during processing
517	-261	File and system versions incompatible
518	-262	Too many subindex links
519	-263	Subindex already present
520	-264	Subindex level limit exceeded
521	-265	Index entry with subindex may not be deleted
522	-266	Physical delete access must be 'keyed'
523	-267	Index entry is locked
524	-268	Write access must be 'keyed'
525	-269	Illegal label format on mag tape
526	-270	Illegal label specification
527	-271	Volume identifiers do not match
528	-272	File identifiers do not match (internal error)
529	-273	File sequence numbers do not match (internal error)
530	-274	File generation numbers do not match (internal error)

Table F-3 RDOS/DOS and Input/Output Error Messages (continues)

Record # in BASIC.ER	Error Code Returned by SYS(7)	Error Message
531	-275	File expiration date not yet reached
532	-276	Block count on trailer label disagrees with actual count
533	-277	Record formats do not match
534	-278	Incorrect file section number on header label.
535	-279	Excessive position levels
536	-280	System load size error
537	-281	Tape file not found
538	-282	Block size less than 8 bytes
539	-283	Record plus overhead is greater than block size
540	-284	Write is not at end-of-file for shared SAM update file
541	-285	Only one user may write to a shared SAM update file
542	-286	Spooling enabled on illegal device
543	-287	INFOS retreive key error
544	-288	Delete index positioning error
545	-289	Space management inconsistency
546	-290	Error searching current position table
547	-291	Tape mount cancelled by operator

Table F-3 RDOS/DOS and Input/Output Error Messages

NOTE:

These are the codes returned by SYS(31). Note that there is no positive record number associated with these errors.

Error Code Returned by SYS(31)	Error Message
0	Unknown message code 000000
-1	Illegal system command
-2	Channel not open
-3	Channel already open
-4	Shared I/O request not map slot aligned
-5	Insufficient memory available
-6	Illegal starting address
-7	Illegal overlay number
-8	Illegal set time argument
-9	No task control block available
-10	?XMT to address already in use
-11	Error in user task queue table
-12	Task I.D. error
-13	Data channel map is full
-14	System call parameter address error
-15	Task not found for abort
-16	Insufficient room in buffer
-17	File space exhausted
-18	User stack fault
-19	Directory does not exist
-20	Illegal filename character
-21	File does not exist
-22	File name already exists
-23	Non-directory argument in pathname
-24	End of file
-25	Directory delete error
-26	Write access denied
-27	Read access denied
-28	Append and/or write access denied
-29	No free channels
-30	Release of non-active shared slot
-31	Illegal process priority
-32	Illegal maximum process size
-33	Illegal process type
-34	Console device specification error
-35	Out of swap file room
-36	Device already in system
-37	Illegal device code
-38	Error on shared partition set
-39	Error on remap call
-40	Illegal ghost gate call
-41	More than 64 processes
-42	IPC message longer than buffer
-43	Invalid port number
-44	No matching send
-45	No outstanding receive

Table F-4 AOS and AOS/VS Input/Output Errors (continues)

Error Code Returned by SYS(31)	Error Message
-46	Illegal origin port
-47	Illegal destination port
-48	Invalid shared library reference
-49	Illegal record length specified
-50	Attempt to release console device
-51	Device already in use
-52	Attempt to release unassigned device
-53	Attempt to close unopen channel/device
-54	I/O terminated by close
-55	Line too long
-56	Parity error
-57	Resident process tried to create son and block
-58	Not a directory
-59	Shared I/O request not to shared area
-60	Too many subordinate processes
-61	File read error
-62	Device timeout
-63	Wrong type I/O for open type
-64	Filename too long
-65	Positioning before beginning of file
-66	Caller not privileged for this action
-67	Simultaneous requests on same channel
-68	Illegal file type
-69	Insufficient room in directory
-70	Illegal open
-71	Attempt to access process not in hierarchy
-72	Attempt to block unblockable process
-73	Invalid system call parameter
-74	Attempt to start multiple ghosts
-75	Channel in use
-76	Not enough contiguous disk blocks
-77	Stack overflow
-78	Inconsistent bit map data
-79	Illegal block size for device
-80	Attempt to ?XMT illegal message
-81	Physical unit failure
-82	Physical write lock
-83	Physical unit offline
-84	Illegal open option for file type
-85	Too many or too few device names
-86	Disk and file system revision numbers don't match
-87	Inconsistent device information block (DIB) data
-88	Inconsistent logical disk
-89	Incomplete logical disk
-90	Illegal device name type
-91	Illogical process address space definition
-92	Logical disk in use, cannot release

Table F-4 AOS and AOS/VS Input/Output Errors (continues)

Error Code Returned by SYS(31)	Error Message
-93	Too many directories in search list
-94	Can't get IPC data from father
-95	Illegal library number given
-96	Illegal record format
-97	Too many or too few arguments to PMGR
-98	Illegal ?GTMES parameters
-99	Illegal CLI message
-100	Message receive disabled
-101	Not a console device
-102	Attempt to exceed maximum index level
-103	Illegal channel
-104	No receiver waiting
-105	Short receive request
-106	Transmitter inoperative
-107	Illegal user name
-108	Illegal link number
-109	Disk positioning error
-110	Message text longer than specified
-111	Short transmission
-112	Illogical histogram call
-113	Illegal retry value
-114	Assign error-already your device
-115	Mag tape request past logical end of tape
-116	Packet specifies stack too small
-117	Packet would create too many tasks
-118	Spooler open retry count exceeded
-119	Illegal ACL
-120	?STMAP buffer contains invalid/write-protected page
-121	Partner process has not opened IPC file
-122	FPU hardware not installed
-123	Illegal process name
-124	Process name already in use
-125	Disconnect error on modem
-126	Process must block to pass generic files
-127	System not installed
-128	Maximum directory tree depth exceeded
-129	Releasing out-of-use overlay
-130	Resource deadlock
-131	File is open, can't exclusively open
-132	File is exclusively opened, can't open
-133	Initialization privilege denied
-134	Multiple ?IMSG calls to same DCT
-135	Illegal link
-136	Illegal dump format
-137	Exec not available
-138	Exec request function unknown
-139	Exec's process sub-tree only

Table F-4 AOS and AOS/VS Input/Output Errors (continues)

Error Code Returned by SYS(31)	Error Message
-140	Request refused by system operator
-141	Can not dismount, was not mounted
-142	Switch or argument value greater than 65535
-143	Input file does not exist
-144	Output file does not exist
-145	List file does not exist
-146	Data file does not exist
-147	Recursive generic file open failure
-148	No message waiting
-149	User data area (UDA) does not exist
-150	Illegal device type from AOSGEN
-151	AOS restart of system call
-152	Fatal user runtime error
-153	User commercial stack fault
-154	User floating point stack fault
-155	User data area (UDA) already exists
-156	Illegal screen-edit request to PMGR
-157	"?SEND" destination console held by "^S"
-158	Data overrun error
-159	Control point directory max size exceeded
-160	System or bootstrap disk not part of master logical disk
-161	Universal system, you can't do that
-162	Execute access denied
-163	Can't initialize LD, run fixup over it
-164	File access denied
-165	Directory access denied
-166	Attempt to define greater than one INFOS process
-167	INFOS II process not running
-168	Attempt to issue MCA request while direct I/O in progress
-169	Attempt to issue MCA direct I/O with req. outstanding
-170	Last task was killed
-171	Resource load or release failure
-172	Zero length filename specified
-173	Buffer overflow
-174	Transmission failure (too many NAKs)
-175	Transmission failure (timeouts)
-176	Disconnect occurred on sync line
-177	EOT character received
-178	Possible lost data on HASP line
-179	Sync DCU inoperative (I/O aborted)
-180	Conversational reply received
-181	End of polling list reached without a response
-182	Illegal relative terminal number
-183	Reverse interrupt response received
-184	Illegal line number specified

Table F-4 AOS and AOS/VS Input/Output Errors (continues)

Error Code Returned by SYS(31)	Error Message
-185	Not enough space for polling list
-186	Contention situation while bidding
-187	Out-of-sequence gen entry during sinit
-188	Request to a non-synchronous line
-189	Not enough memory for communications buffer
-190	Line already enabled when ?SEBL call issued
-191	Line already disabled when ?SDBL call issued
-192	I/O request on a disabled line
-193	Line in session on initial I/O request
-194	Line not in session on continue I/O request
-195	Buffer byte count larger than system buffer
-196	BID error (too many NAKS)
-197	Wait-a-bit received (HASP line only)
-198	User buffer byte pointer invalid
-199	Retry count exceeded
-200	ETX code received
-201	Input status format error
-202	Failure to connect error
-203	Uninterpretable response received
-204	Enq received
-205	Block check error
-206	Initialization parameter error
-207	Transmitter failure error
-208	Line not multidrop
-209	Not a control station
-210	Polling list not defined
-211	Inconsistent tab format
-212	Cannot delete permanent file
-213	System call abort
-214	Unknown message code 000326
-215	Unreadable tape label
-216	Incorrect labeled tape volume mounted
-217	Incorrect labeled tape file set
-218	Incorrect labeled tape file section number
-219	Incorrect labeled tape file generation number
-220	Incorrect labeled tape file version number
-221	No operator available
-222	Unknown tape label format
-223	Unknown message code 000337
-224	Unknown message code 000340
-225	Unknown message code 000341
-226	Memory release error
-227	Illegal translation parameter
-228	No such argument
-229	Not in CLI format
-230	Illegal bias factor
-231	CPU time limit exceeded

Table F-4 AOS and AOS/VS Input/Output Errors (continues)

Error Code Returned by SYS(31)	Error Message
-232	Error in setting max CPU limit
-233	File element size not multiple of 4
-234	WACK response received
-235	Process is not a server
-236	Connection does not exist
-237	Connection table full
-238	Directory in use-cannot delete
-239	Attempt to grow a shared file
-240	Illegal directory name specification
-241	Network not available
-242	Host already exists
-243	Illegal host specification
-244	Host does not exist
-245	Cannot rename host entries
-246	Empty mailbox on ?RECNW
-247	Unable to access network in this manner
-248	Attempt to create multiple local hosts
-249	Task not awaiting ?IWKUP
-250	Illegal remote ?PROC parameter(s)
-251	Illegal host name
-252	Not proper for a virtual circuit
-253	Invalid HDLC window size
-254	Invalid HDLC frame size
-255	HDLC send active
-256	Invalid HDLC call type
-257	Remote is disconnecting
-258	Local received invalid response
-259	Local received CMDR
-260	Local is in can't-send condition
-261	Local is disconnecting
-262	Local was reset
-263	HDLC buffer overflow
-264	HDLC receive active
-265	HDLC link initialization failed
-266	Local received invalid command
-267	Non-HDLC enable attempted
-268	Interrupt wait task already defined
-269	Sync map slot error
-270	Sync buffer error
-271	Sync DCU inoperative
-272	Error opening SLDCU.PR
-273	Error reading SLDCU.PR
-274	Error closing SLDCU.PR
-275	Error getting memory
-276	Unknown sync error
-277	Connection has been broken
-278	HDLC call attempted with no DCU200
-279	Cannot connect to self
-280	No connection

Table F-4 AOS and AOS/VS Input/Output Errors (continues)

Error Code Returned by SYS(31)	Error Message
-281	Tape controller does not support this density mode
-282	Indecipherable tape density
-283	File/tape density mismatch
-284	Illegal labelled tape level
-285	Illegal labelled tape character
-286	Incorrect labelled tape sequence number
-287	Incorrect labelled tape block count
-288	Volid too long or null
-289	Owner ID too long
-290	User labels too long or too many
-291	Record length exceeds block length
-292	AP already busy
-293	AP does not exist
-294	AP WCS is not loaded
-295	Attempt to release a non-AP page
-296	Attempt to release an AP page
-297	Only one file on ANSI level 1 lmt
-298	Cannot delete unexpired file on LMT
-299	Process console does not exist
-300	Termination of histogram target process
-301	Error detected while histogram in progress
-302	Invalid IPC message
-303	Multiple users of file, cannot truncate
-304	Shared file, cannot truncate
-305	File being truncated, cannot open
-306	Framing error
-307	Internal directory inconsistency
-308	Commercial fault
-309	Fixed point fault
-310	Floating point fault
-311	Can't select even parity for 9 track drives
-312	Too many tape retries
-313	No statistics available for specified device
-314	No statistics available for specified unit
-315	Edit program space exceeded
-316	Edit program too large
-317	Edit program version incompatibility
-318	Editread status error
-319	Illegal exec string paramater
-320	Bad internal exec IPC
-321	Unknown message code 000501
-322	Unknown message code 000502
-323	Unknown message code 000503
-324	Unknown message code 000504
-325	Unknown message code 000505
-326	Unknown message code 000506
-327	Unknown message code 000507
-328	Unknown message code 000510
-329	Unknown message code 000511

Table F-4 AOS and AOS/VS Input/Output Errors (continues)

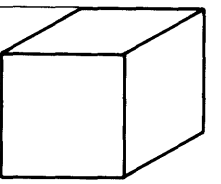
Error Code Returned by SYS(31)	Error Message
-330	Unknown message code 000512
-331	Unknown message code 000513
-332	Unknown message code 000514
-333	Unknown message code 000515
-334	Unknown message code 000516
-335	Unknown message code 000517
-336	Unknown message code 000520
-337	Unknown message code 000521
-338	Unknown message code 000522
-339	Unknown message code 000523
-340	Unknown message code 000524
-341	Unknown message code 000525
-342	Unknown message code 000526
-343	Unknown message code 000527
-344	Unknown message code 000530
-345	Unknown message code 000531
-346	Unknown message code 000532
-347	Unknown message code 000533
-348	Unknown message code 000534
-349	Unknown message code 000535
-350	Unknown message code 000536

Table F-4 AOS and AOS/VS Input/Output Errors

Error Code	Error Message
-192	Not enough arguments
-193	Illegal attribute
-194	No debug address
-195	Command line too long
-196	No starting address
-197	Checksum error
-198	No source file specified
-199	Not a command
-200	Illegal block type
-201	No files match specifier
-202	Phase error
-203	Too many arguments
-204	Too many active devices
-205	Illegal numeric argument
-206	Fatal system utility error
-207	Illegal argument
-208	Improper or malicious input
-209	Too many levels of indirect files
-210	Syntax error
-211	Bracket error
-212	Paren error
-213	Unmatched < >
-214	Illegal nesting of < > and ()
-215	Illegal indirect filename
-216	Illegal nesting of () and {}

Table F-5 Business BASIC CLI Error Messages

Related Documents



Basic BASIC **069-000003**
This manual introduces the BASIC computer language and covers its elementary commands and statements. It is written for the novice who has no previous knowledge of BASIC.

A Guide to Using Business Basic **069-000028**
(AOS/VS, AOS, RDOS, DOS)
Introduces the reader to Business BASIC and describes its major features. Contains overview information on certain Business BASIC utilities, such as the Database Generator (DBGEN), File Maintenance, and Screen maintenance.

Business BASIC Commands, Statements, and Functions **093-705005**
(AOS/VS, AOS, RDOS, DOS)
An alphabetical directory of Business BASIC statements, commands, and functions. it is intended to be used as a reference manual for programmers.

Business BASIC Subroutines, Utilities, and BASIC CLI **093-705006**
(AOS/VS, AOS, RDOS, DOS)
An alphabetical directory of Business BASIC subroutines, utilities, and the BASIC CLI commands. It is intended to be used as a reference manual for programmers.

Business BASIC System Management **093-705007**
(AOS/VS, AOS, RDOS, DOS)
Describes how to load and generate Business BASIC on AOS/VS, AOS, RDOS, and DOS. It is intended for the system manager or system operator.

Business BASIC (AOS/RDOS/DOS) Reference Card **069-705010**
A pocket-sized reference card for Business BASIC programmers on AOS/VS, AOS, RDOS or DOS.

DASHER® D2 File Maintenance **093-000212**
and Screen Maintenance **Template**

DASHER® D200 File Maintenannce **093-000265**

DASHER® D100 CFM and CSM Template **093-000329**

DASHER® D2 CFM and CSM Template **093-000330**

DASHER® D3 CFM and CSM Template **093-000331**

DASHER® D200 CFM and CSM Template **093-000332**

BusiGEN™ User Guide **069-705011**
Describes how to use the BusiGEN program generator to design, code, and document interactive Business BASIC data entry, file maintenance, data inquiry, and report-writer programs.

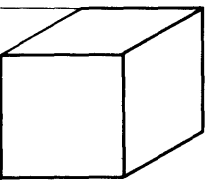
BusiPEN™ User Guide **069-700007**
Describes how to use the BusiPEN graphics generator to create bar, line, and pie charts.

BusiTEXT™ User Guide **069-705004**
Describes how to use the BusiTEXT word processing system.

Business BASIC Report Writer User's Manual **093-000333**
Describes how to use the Business BASIC Report Writer software to design report writer programs.

Business BASIC Data Dictionary User's Manual **093-000334**
Describes how to use the Business BASIC Data Dictionary software to create, maintain, and access one or more data dictionaries.

Index



reader comment form

Business BASIC Technical Concepts *093-705004-00*

Your comments will help us improve the quality of this publication. They will be carefully reviewed by the writers. Please refer to page numbers if appropriate.

DID YOU FIND THE MATERIAL:

	YES	NO		YES	NO
• Useful?	☐	☐	• Well illustrated?	☐	☐
• Complete?	☐	☐	• Well written?	☐	☐
• Accurate?	☐	☐	• Easy to read?	☐	☐
• Well organized?	☐	☐	• Easy to understand?	☐	☐

COMMENTS:

_____

_____

_____

_____

_____

_____

_____

_____

HOW DID YOU USE THIS PUBLICATION?

- ☐ As an introduction to the subject
- ☐ For information about operating procedures
- ☐ To instruct in a class
- ☐ As a student in a class
- ☐ As a reference manual
- ☐ Other *(please explain)*:

Name _____ **Title** _____

Firm _____ **Date** _____

Street _____ **State** _____

City _____ **Zip** _____

First fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 WESTBORO, MASS 01580

POSTAGE WILL BE PAID BY ADDRESSEE:

DataGeneral

ATTN: Small Business Systems Documentation
MS F213
Westboro, Ma. 01580
USA

Second fold

CUT ALONG DOTTED LINE

THIS IS THE SPINE FOR YOUR NOTEBOOK COVER

CUT ON DOTTED LINE

order number 069-705023
for matching cover insert.

order number 069-705022
for 1 inch black ring binder
to hold all inserts.

093-705004-00

A Small Business Systems Publication

DATA GENERAL CORPORATION, Westboro, Massachusetts 01580