DataGeneral

Customer Documentation

Programming with Business BASIC

Programming with Business BASIC

093-000480-00

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000480

Printed in the United States of America
Revision 00, July 1986

NOTICE

Programming with Business BASIC
093-000480
Revision 00, July 1986
(Business BASIC Rev. 4.0 (AOS/VS, AOS)
Business BASIC Rev. 8.0 (RDOS))

Preface

Scope

Data General's Business BASIC is a powerful, interactive programming language that runs on the following operating systems: mapped ECLIPSE® RDOS, DG/RDOS, AOS, AOS/VS, and AOS/WS. Because there is no significant difference to the programmer, this guide uses the term "AOS" to refer to AOS, AOS/WS, and AOS/VS and the term "RDOS" to refer to RDOS and DG/RDOS except where AOS and/or RDOS differ from their companion operating systems.

This manual is for experienced programmers who have not used Business BASIC. The purpose of the manual is to acquaint these programmers with Business BASIC operations and programming procedures. The manual provides an overview of what commands, functions, subroutines, and utilities are available to help programmers. It is not intended to provide detailed instructions on how these features work. Explanations of these Business BASIC features are contained in the Business BASIC reference manuals. This manual also discusses the file and database structures supported by Business BASIC.

Document Set

Programming with Business BASIC is part of a four-manual set that describes the language, its utilities and subroutines, and how the system is set up. The other manuals in this set are:

- *Business BASIC Reference Manual for Commands, Statements, and Functions*
- *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI*
- *Business BASIC System Manager's Guide*

The titles and ordering numbers for these documents are listed in "Related Documents" at the end of this manual.

Organization of This Manual

The manual is divided into eight chapters, a glossary, and three appendixes.

Chapter 1 presents an overview of the Business BASIC software package and information on the Business BASIC modes of operation.

Chapter 2 discusses program development, execution, debugging, and documenting.

Chapter 3 contains information on Business BASIC variables, expressions, and arithmetic operations.

Chapter 4 discusses Business BASIC subroutines, user-written subroutines, assembly language subroutines, and utilities.

Chapter 5 presents an overview of the Business BASIC file structure and its input and output operations.

Chapter 6 contains information about the two Business BASIC database structures.

Chapter 7 deals with the INFOS II® system files (AOS only).

Chapter 8 has information on transporting Business BASIC files between operating systems.

The glossary contains definitions of terms used by Business BASIC.

Appendix A lists the Business BASIC keywords, subroutines, and utilities.

Appendix B has an ASCII character set chart and an eight-bit character set chart.

Appendix C contains example programs.

Typographical Conventions

This manual uses certain conventions and typefaces to represent the various elements of the Business BASIC language syntax.

Information that is displayed in this typeface indicates a general format for the Business BASIC language syntax.

```
Information that is displayed in this typeface indicates a syntax structure that is
exactly as it would appear on your terminal
```

This manual uses the term "IKEY" to refer to the interrupt or escape key sequence your Business BASIC system uses. In many cases, this is the ESCAPE key; however, some systems use control/key combinations. Check with your system manager to see what your system uses.

Also, this manual uses the term "new line key" to refer to the key you press to terminate a line. On AOS systems you use the NEW LINE key, and on RDOS systems you use the CR (carriage return) key.

The other conventions used in this manual are described below.

UPPERCASE Indicates a Business BASIC subroutine, utility, or command.

lowercase Indicates a generic term representing a complete syntactical entry to be supplied by the programmer.

End of Preface

Contents

Chapter 1 - Introduction to Business BASIC

Commands, Statements, and Functions 1-2
Business BASIC Utility Programs 1-3
Business BASIC Subroutines 1-3
BASIC CLI 1-3
Business BASIC Operational Modes 1-4
File Structures 1-5
Logging On and Off Business BASIC 1-6

Chapter 2 - Program Development

Writing Business BASIC Programs 2-1
    Adding Program Statements 2-4
    Business BASIC Programming Features 2-4
Modifying Your Program 2-6
    Keyboard Editing Commands 2-7
    Saving Programs 2-10
Executing Business BASIC Programs 2-11
Continuing Execution of a Program 2-13
Interrupting and Debugging Programs 2-14
    Debugging Aids 2-16
    Handling Interrupts from within Programs 2-16
      The ON IKEY Statement 2-17
      The ON ERR Statement 2-17
Documenting and Storing Programs 2-18
    SAVE Files 2-19
    Listing Files 2-19

Chapter 3 - Numeric, Array, and String Variables

Variables 3-1
Numeric Data 3-2
    Numeric Variables 3-3
    Precision 3-3
      Storage of Numeric Variables 3-3
      Data Transfer of Numeric Variables 3-3

Arrays 3-4
Creating Arrays 3-5
Default Array Dimensions 3-5
Accessing Array Elements 3-5
Changing Array Dimensions 3-6
Assigning Values to Numeric Elements 3-6
Numeric Expressions 3-7
Arithmetic Operators 3-8
Relational Operators 3-9
Boolean Logic Operators 3-9
Handling Decimals 3-11
Predefined Numeric Instructions 3-13
Character Data 3-14
String Literals 3-15
String Variables 3-15
Accessing Strings 3-16
Using Strings in Expressions 3-17
Assigning Values to Strings 3-18
Concatenating Strings 3-19
String Functions 3-19
Using Variables to Transfer Data 3-20
Numeric/String Conversions 3-21

Chapter 4 - Subroutines and Utilities

Subroutines 4-1
Business BASIC Subroutines 4-1
Using Subroutines 4-2
Writing Business BASIC Subroutines 4-3
Errors with Subroutines 4-4
Subroutine Example 4-4
Assembly Language Subroutines 4-6
Utilities 4-6
Using Utilities 4-6
Common Area 4-7

Chapter 5 - File Overview

Operating System Files 5-1
RDOS Files 5-2
Sequential File Organization 5-2
Random File Organization 5-2
Contiguous File Organization 5-5
AOS Files 5-6
Business BASIC File System 5-6
Filename Conventions 5-8
Creating Simple Disk Files 5-9
File Access 5-9
File Types 5-11

Simple Disk Files 5-11
Linked-Available-Record Files 5-11
Index Files 5-12
Logical Files and Subfiles 5-19

Chapter 6 - Database Structures in Business BASIC

Logical File Database Structure 6-2
Creating a Logical File Database 6-3
Logical Files 6-3
Volume Label File Format 6-3
Logical File Table (LFTABL$) 6-4
Logical File Input and Output 6-5
PARAM File Database Structure 6-6
Setting up a PARAM Database 6-6
The PARAM File 6-7
C1 (File Characteristics) Array 6-8
Building a C1 Array 6-9
Modifying a Record in the C1 Array 6-11
Positioning to a Record 6-11
Writing a Record in the PARAM Structure 6-12
Deleting a Record in the PARAM Structure 6-12
Input and Output with the PARAM Database 6-13
Converting from a PARAM Database to a Logical Database 6-14
Comparing Databases 6-14

Chapter 7 - The INFOS® II File System (AOS Only)

Introduction to INFOS II 7-1
Argument Pairs 7-2
Channel Strings 7-3
Creating an INFOS II File 7-4
Accessing INFOS II Files 7-5
Error Handling 7-6

Chapter 8 - Transporting Programs between RDOS and AOS

Transferring Files 8-1
Moving Files from RDOS to AOS 8-2
Moving Text Files to AOS 8-2
Moving Logical Database Files to AOS 8-2
Moving SAVE Files to AOS 8-2
Moving Files from AOS to RDOS 8-3
Moving Text Files to RDOS 8-3
Moving Logical Database Files to RDOS 8-3
Moving SAVE Files to RDOS 8-3
Operating System Differences 8-3

Glossary

Appendix A - Subroutine, Utility, and Keyword Summary

Appendix B - ASCII Tables

Appendix C - Example Programs

Setting Up a Logical File Database C-1
Setting Up a PARAM File Database C-4
Comparing Logical, PARAM Code C-10
Enlarging a Logical Database C-11

Related Documents

Illustrations

Figure

1-1	Components of the Business BASIC Software Package	1-1
2-1	Business BASIC Program Statement Syntax	2-3
2-2	Flow of Program Control with SWAP Command	2-12
2-3	Flow of Program Control with CHAIN Command	2-12
3-1	One- and Two-Dimensional Arrays	3-4
5-1	Format of an RDOS Sequential File	5-3
5-2	Format of an RDOS Random File	5-4
5-3	Format of an RDOS Contiguous File	5-5
5-4	Stages in AOS File Growth	5-7

Tables

Table

2-1	Keyboard Editing Commands	2-8
2-2	Program Execution Commands	2-15
2-3	Commands to Save Programs	2-20
3-1	Precedence of Arithmetic Operations	3-8
3-2	Relational Operators	3-9
3-3	Hierarchy of Operators in Business BASIC	3-11
3-4	Numeric Functions and Statements	3-14
3-5	References to Strings	3-16
3-6	Uses of String Expressions	3-17
3-7	Assigning Characters to String Locations	3-18
3-8	String Functions and Statements	3-20
5-1	Filename Extensions	5-8
5-2	File Input and Output Commands	5-10
5-3	Contents of Record 0 of a Linked-Available-Record File	5-12
5-4	Active Data Record in a Linked-Available-Record File	5-12
5-5	Deleted Data Record in a Linked-Available-Record File	5-12
5-6	Contents of Block 0 of an Index File	5-14
5-7	Format of a Block Containing Keys for an Index File	5-15
6-1	Contents of a Volume Label File Record	6-3
6-2	LFU Command Summary	6-4
6-3	Contents of an LFTABL$ Record	6-5
6-4	I/O Commands Used with Logical Files	6-6
6-5	Record 0 of the PARAM File	6-8
6-6	Contents of a PARAM File Record	6-8
6-7	Column Contents of the C1 Array	6-9
6-8	I/O Commands Used with PARAM Database Files	6-13
6-9	Logical, PARAM Database Features	6-15
7-1	Business BASIC INFOS® II Statements	7-2
8-1	Methods of Moving Files between Operating Systems	8-1
A-1	Business BASIC Subroutines	A-1
A-2	Business BASIC Utilities	A-4
A-3	BASIC CLI Commands	A-6
A-4	Business BASIC Commands and Statements	A-8
A-5	Business BASIC Functions	A-12

A-6 Boolean Logic Operators A-13

B-1 Standard ASCII Character Set B-1
B-2 DG International Symbols (8-bit ASCII Character Set) B-3

Chapter 1
Introduction to Business BASIC

Business BASIC is an interactive programming language that contains many standard BASIC commands, statements, and functions in addition to specialized statements and functions for handling file access, controlling the format of data, and performing system tasks. Business BASIC supports simple file structures and database file structures. These file structures include ISAM files, logical files, a subfile/master file system, and an interface to Data General's INFOS II® file system.

This chapter provides an overview of the structure and composition of the Business BASIC software package. It also presents some general information on Business BASIC.

The major components of the Business BASIC package are shown in Figure 1-1.

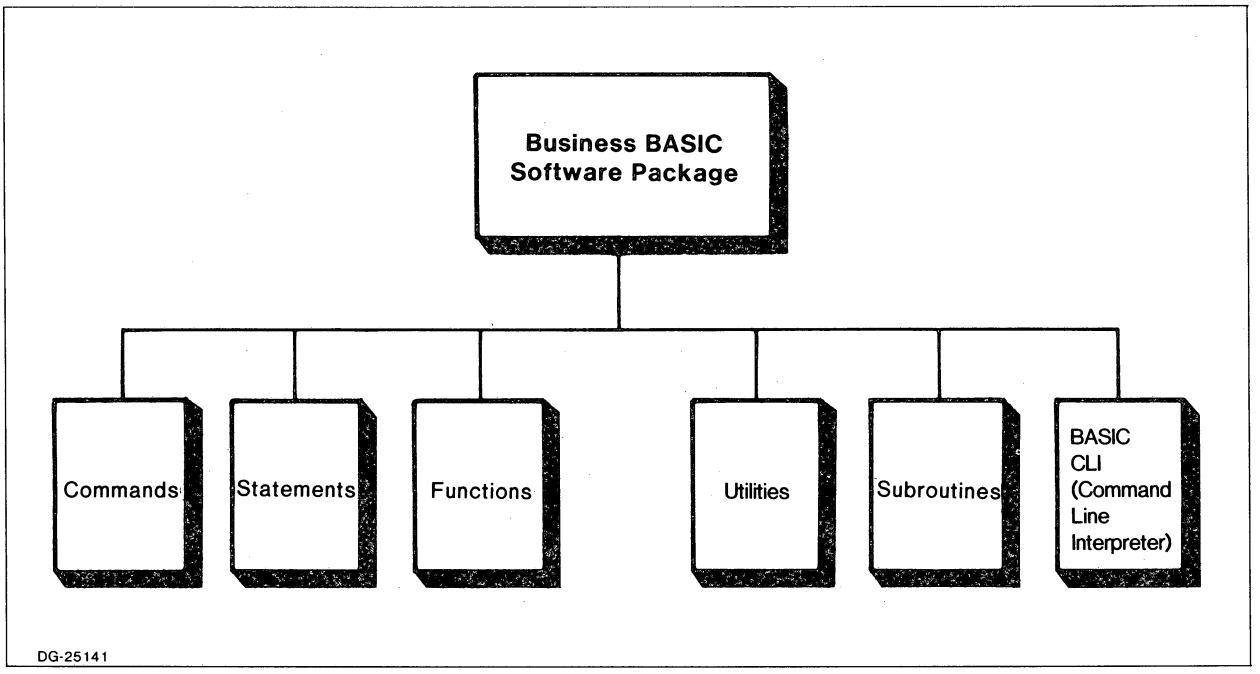

Figure 1-1. Components of the Business BASIC Software Package

The boxes in Figure 1-1 represent tools that you can use when you program with Business BASIC. These tools can be summarized as follows:

1) The three boxes at the left (Commands, Statements, Functions) make up the Business BASIC language.

2) The next two boxes represent the utility programs and subroutines provided by Business BASIC. The utilities and subroutines perform a variety of tasks to help you use Business BASIC.

3) The BASIC Command Line Interpreter (CLI) simulates the RDOS CLI. This lets you perform operating system functions without leaving Business BASIC.

Commands, Statements, and Functions

The commands, statements, and functions that make up the Business BASIC language are the fundamental tools used in designing and developing application programs.

By definition, a command is an instruction that is entered without a line number and is executed immediately.

A statement is preceded by a line number and is not executed until a command such as RUN is entered.

A function can be either a numeric or string expression. It is used as an expression within statements or commands.

To enter a statement, command, or function, type in the information and then press the new line key.

In most cases, statements and functions use arguments. These can include variables, numeric and string assignments, messages to print, and subroutine destinations.

Business BASIC supports three types of variables — numeric, array, and string.

You can have numeric constants and numeric variables. Since Business BASIC is an integer language, you cannot assign a fractional number to a variable. There are no default assignments for numeric variables.

Arrays, like numeric variables, accept only integer values. Business BASIC does not support string arrays. You can have one-dimensional or two-dimensional arrays.

Strings contain character data. You can use a string constant or assign a string to a string variable. You must dimension a string variable before you can use it.

More information on variables is contained in chapter 3.

Appendix A contains a summary of the commands, statements, and functions that are included in your Business BASIC software package. For more information on these features, see the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

Business BASIC Utility Programs

Utilities are Business BASIC programs that perform data processing functions, such as formatting screens and maintaining your database.

Most utilities are stand-alone programs; they can be executed using the RUN, CHAIN, or SWAP commands or through the BASIC CLI by entering the utility name preceded by an exclamation point (!). Some utilities, however, have restricted execution modes.

Utilities are discussed in chapter 4. A list of the utilities provided with your software package is in appendix A. For explanations of how individual utilities work, see the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Business BASIC Subroutines

The Business BASIC software package contains pre-written subroutines. The subroutines are specialized portions of Business BASIC code. Their modular design makes them easy to incorporate in application programs. The Business BASIC subroutines help you meet the processing needs of your programs and reduce the amount of coding you need to do.

The subroutines included in the software package reside in the Business BASIC library directory. Their filenames have .SL extensions to distinguish them from utility programs and other files. You execute a subroutine from within a Business BASIC program with a GOSUB line number statement. The line number is the entry point to the subroutine.

You can also write your own subroutines and place them in the library directory so that you can reuse them. In addition, Business BASIC provides an interface that allows you to use assembly language subroutines.

Subroutines are discussed in chapter 4. A list of the subroutines provided with your software package is in appendix A. Explanations of individual subroutines are contained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

BASIC CLI

Business BASIC also has its own Command Line Interpreter (CLI). The BASIC CLI simulates the RDOS CLI without forcing you to leave Business BASIC. This means that you do not need to exit from Business BASIC to perform processing activities such as creating and deleting files, moving files between directories, and printing files.

You start the BASIC CLI by entering RUN CLI, CHAIN CLI, SWAP CLI, or !CLI. An exclamation point (!) prompt indicates that you are in the BASIC CLI.

There are two ways to execute a BASIC CLI command:

- Start the BASIC CLI (which puts you in BASIC CLI mode) and then enter the command.
- Enter the command preceded by an exclamation point at the asterisk prompt:

```
* !command
```

You can also execute programs and utilities through the BASIC CLI by using one of these two methods. The BASIC CLI performs a SWAP to the file specified when it does not recognize a command.

To leave the BASIC CLI, use the POP command or the QUIT command, both of which return you to your previous level, or enter BYE, which logs you off Business BASIC.

Appendix A contains a list of the BASIC CLI commands. For more information on the BASIC CLI, see the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Business BASIC Operational Modes

To execute Business BASIC programs, you enter a RUN, SWAP, or CHAIN command followed by the appropriate argument. You can execute programs from keyboard mode, BASIC CLI mode, or program mode.

Keyboard mode is indicated by an asterisk (*) prompt while BASIC CLI mode is indicated by an exclamation point (!) prompt. You can execute a program using BASIC CLI mode while you are in keyboard mode by entering the program name preceded by an exclamation point (i.e., !program-name) or you can execute the BASIC CLI and then type in the program name. Program mode occurs when you execute a program and then use it to perform tasks, such as executing another program. You can enter program mode by executing the program from keyboard mode or from BASIC CLI mode. The primary operational mode when you are working in Business BASIC is keyboard mode. You automatically enter keyboard mode when you log on to Business BASIC. While in keyboard mode, you can execute commands or you can create and run Business BASIC programs (unless you are working on a run-only Business BASIC system).

Everything you type while in keyboard mode goes into an assigned buffer area called working storage. This is the area of the computer's memory that holds your program and data. Anytime you ENTER or LOAD a program, it is stored in working storage in binary format.

When a program is in working storage, you can execute it by entering the RUN command with no arguments or with a line number. You can also execute a program by entering RUN "program-name. In addition, you can SWAP "program-name, CHAIN "program-name, or CON (continue) a program.

File Structures

Business BASIC supports the following categories of files:

- Operating system files.
- Business BASIC files (i.e., operating system files containing an embedded Business BASIC structure).
- INFOS II files (i.e., operating system files containing an embedded INFOS II structure).

The operating system files are disk files that are physically organized in one of four ways based on how you created the files and which operating system you have (RDOS allows three forms of internal structure, while AOS permits only one).

Under RDOS, each operating system file is organized in one of the following ways:

- Sequential. RDOS maintains a series of pointers to each block of file information. The blocks do not have to be adjacent. These files permit only beginning-to-end input and output access.
- Random. RDOS maintains an index that contains pointers to the data blocks in a random file. The blocks do not have to be adjacent. These files permit direct access of data.
- Contiguous. These files consist of a fixed number of disk blocks that are physically adjacent. These files permit direct access of data.

Under AOS, all operating system files are built from 512-byte disk blocks. AOS then uses a hierarchical index to connect the disk blocks within files.

With Business BASIC running, you can access any of these files, except the RDOS sequential files, by using direct random access or sequential access. You must use sequential access with the RDOS sequential files.

Business BASIC supports two additional file structures to increase your flexibility in working with data. These files are operating system files containing an embedded Business BASIC structure that tells the BASIC interpreter how the files are organized. The two Business BASIC file types are:

- Linked-available-record files. These files use dynamic record allocation and allow you to access the records directly.
- Index files. Business BASIC uses the indexed sequential access method (ISAM) of working with index files.

On AOS systems, Business BASIC also provides an interface to the INFOS II file management system. The INFOS II files are operating system files that contain an embedded INFOS II structure. The INFOS II file system provides data handling capabilities that let you create, maintain, and use many types of databases in batch and multiterminal environments.

In addition to the file structures, Business BASIC supports two database structures for working with files — the logical file database structure and the PARAM file database structure. These structures increase the number of files you can open within a program by allowing you to use subsections of a physical file. These subsections are called logical files in the logical structure and subfiles in the PARAM structure. Since the operating system does not recognize subsections of files, the database structures catalog the subsections so that the Business BASIC system can use them.

Logging On and Off Business BASIC

There are several ways to execute Business BASIC. The method you use is determined by your operating system and the execution procedure set up by your system manager. This section provides only a general overview of logging on and off Business BASIC. More information on the logon procedures is contained in the *Business BASIC System Manager's Guide.* Consult with your system manager to learn your system's Business BASIC logon procedure.

Under AOS, there are several ways to log on to Business BASIC. Logging on to Business BASIC can be a two-step procedure where first you log on to the operating system, then you execute Business BASIC. You can also have your AOS profile set up so that Business BASIC is automatically executed when you log on to AOS. Once executed, Business BASIC displays information about your account and then an asterisk prompt (for keyboard mode), indicating you can start working.

To leave Business BASIC on an AOS system, enter BYE. Generally, this returns you to the AOS CLI. If you logged on with a CHAIN command from the AOS CLI, then BYE logs you off the AOS system as well as the Business BASIC system.

Under RDOS, logging on to Business BASIC involves several steps. When you are on an RDOS system that is running Business BASIC, the banner

```
DGC BBASIC X.XX
```

is on your screen. This indicates you can start the Business BASIC logon procedures. First, press the escape key. Business BASIC then prompts you for information such as your terminal type, your account, and your password. (This procedure is described in the *Business BASIC System Manager's Guide.*) After you respond correctly to the prompts, the asterisk prompt appears indicating you are in keyboard mode, and you can begin working in Business BASIC.

To leave Business BASIC on an RDOS system, enter BYE.

Under both AOS and RDOS, you are automatically logged off Business BASIC if you enter BYE while in the BASIC CLI.

End of Chapter

Chapter 2
Program Development

This chapter uses examples to show you the Business BASIC program development phases. The examples are simple so that you can begin working in Business BASIC quickly and at the same time extrapolate the information you need to develop application programs. To perform these examples, you need to be running Business BASIC and, unless otherwise specified, in keyboard mode (indicated by the asterisk prompt).

Programming with Business BASIC consists of four steps:

- Writing the program.
- Executing the program.
- Interrupting and debugging the program.
- Documenting and storing the program.

These steps are discussed in this chapter.

Writing Business BASIC Programs

Because Business BASIC is an interpreter, not a compiler, executing a single program generally requires two main steps:

- Getting the program into working storage.
- Entering the command, such as RUN, that begins program execution.

The program can be an existing program that is stored elsewhere (including one written in an editor) or one that you created by typing in program statements while in keyboard mode. Each line you type in while in keyboard mode is stored in working storage and added to the lines already there.

Before you write a program in keyboard mode, enter the NEW command. This clears working storage.

To create program statements, type in both the statement line number and the statement contents; then press the new line key. Each time you type in a program statement, the Business BASIC interpreter checks the syntax of the line and reports any errors. If there are none, the system adds the statement to the others in working storage.

The following is a four-line Business BASIC program that multiplies 3 by 300. Type in the program and execute it by entering the command RUN. Business BASIC displays the result on your screen.

```
* NEW
* 10 LET A = 300
* 20 LET B = 3
* 30 PRINT A*B
* 40 END
* RUN
 900

*
```

You do not have to use an END statement in Business BASIC programs. Normal program execution halts when Business BASIC encounters either the last line of code or an END/STOP statement. However, since Business BASIC executes all the program statements in working storage, using an END statement prevents the system from executing lines left in working storage by an earlier program. (You can also avoid this problem by clearing working storage with a NEW command before placing your program in it.)

The program remains in working storage until you replace it with another program, clear working storage with the NEW command, or log off Business BASIC.

Figure 2-1 illustrates the general code syntax you've used in your program. This syntax is the same for almost all lines of Business BASIC code and consists of the following:

1) XXXX (LINE NUMBER). Start each line of code with a line number in the range 1 to 9999. Business BASIC left-pads all line numbers containing fewer than four digits with zeroes when you LIST the program; thus, a line number you enter as 1 becomes 0001 when you LIST the program.

2) KEYWORD. Keywords are the instructions that tell the program to perform an action. The program you just executed contained four keywords — two LET statements, one PRINT statement, and an END statement.

3) ARGUMENTS. Arguments (also known as parameters) often follow statements. Arguments can include variables, numeric/string assignments, messages to print, and subroutine destinations. For example, line 10 of the previous program contained the argument A=300.

Working storage holds the values you assigned to variables as well as the program. At this point the variable A has a value of 300, which was assigned when the program was executed. To check this, enter PRINT A; the value 300 appears next to the A:

```
* PRINT A 300

*
```

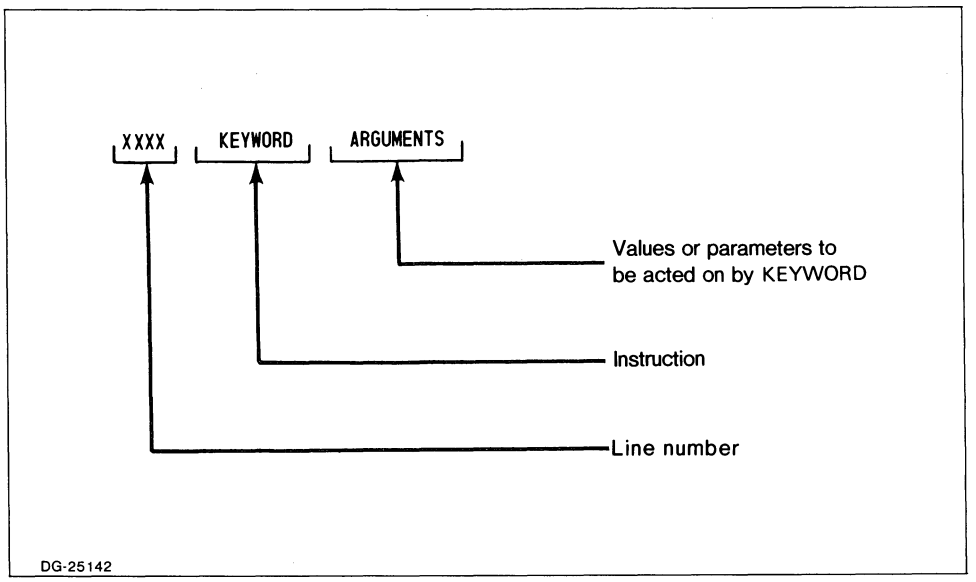

Figure 2-1. Business BASIC Program Statement Syntax

To see the program contents of working storage, use the LIST command:

```
* LIST
0010 LET A=300
0020 LET B=3
0030 PRINT A*B
0040 END

*
```

You could have entered program statements 10 and 20 as:

```
* 10 A = 300
* 20 B = 3
```

because the Business BASIC interpreter inserts optional keywords, such as LET and removes extra spaces (A = 300). Business BASIC also left-pads any line numbers that are part of your program statements (i.e., GOTO 10 becomes GOTO 0010) and, in some cases, inserts spaces. Any omitted keywords, zeroes, or spaces are displayed when you LIST the program. These characters are also included when Business BASIC calculates the length of a line. The LIST command permits up to 132 characters in a line of code. After 132 characters, LIST truncates the display but not the internal code, which can be longer. However, if you use the LIST command to store a program on disk, it stores only the displayed lines, including the ones it truncated, not the internal lines. ENTER also allows only 132 characters in a line. If you ENTER a program with longer lines, you receive the message: `ERROR 18 -- LINE TOO LONG`.

If you need to check a specific program line, you can LIST just that line:

```
* LIST 30
0030 PRINT A*B

*
```

You can also use LIST to display a range of line numbers. LIST is explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

Adding Program Statements

You can add a statement to a program in working storage by typing in the new statement. The system adds the statement sequentially. Be sure to give the new statement a unique line number; otherwise, it overwrites any statement with the same line number.

In the program you now have in working storage, you can add a statement to change the value of A (currently 300). The value, however, does not change until you execute the program again.

```
* 35 A = A*B
* LIST
0010 LET A=300
0020 LET B=3
0030 PRINT A*B
0035 LET A=A*B
0040 END

* PRINT A 300

* RUN
 900

* PRINT A 900

*
```

Business BASIC Programming Features

When a program is run, Business BASIC executes the statement with the lowest line number and then proceeds to the next higher number, unless the statement directs the system elsewhere — as GOTO and GOSUB statements do.

The GOTO statement transfers control to a specific statement. The GOSUB/RETURN statements transfer program control to a segment of code (a subroutine) and then return you to the statement following the subroutine call. These statements, especially the GOSUB/RETURN statements, can increase the modular structure of your program, making it easier to debug. To make your program modules easy to identify, you can include a REM statement (remark statement) to describe them.

Business BASIC also provides flow-control constructions such as the FOR/NEXT and IF statements. The FOR/NEXT statements allow program looping with the termination test occurring at the top of the loop. The IF statement provides your program with decision-making capability by transferring program control based on the value of an expression or the logical answer to a relational expression.

In addition, Business BASIC is structured so that you can use its input/output capabilities to make your programs interactive. You have already used the PRINT statement to display information at your terminal. The INPUT statement lets you enter data from the terminal. (Use the INPUT FILE statement to enter data from a file.) When you use the INPUT statement, you can supersede its question mark (?) prompt by entering INPUT with a string.

To see a simple example of how the preceding statements can apply to a program, modify the previous example by entering the following statements:

```
* 5 LET X = 0
* 10 INPUT "Enter a number: ",A
* 36 IF A > 250 THEN GOTO 40 ELSE GOSUB 50
* 37 GOTO 30
* 50 REM *** INCREASE THE VALUE OF A
* 60 FOR J=1 TO 10
* 70    LET A = A + J
* 80    PRINT "This is the value of A: ", A
* 90 NEXT J
* 100 LET X = X + 1
* 110 PRINT "The number of subroutine calls is: ", X
* 120 RETURN
```

Your program now looks like this:

```
* LIST
0005 LET X=0
0010 INPUT "Enter a number: ",A
0020 LET B=3
0030 PRINT A*B
0035 LET A=A*B
0036 IF A>250 THEN GOTO 0040 ELSE GOSUB 0050 : *** INCREASE THE VALUE OF A
0037 GOTO 0030
0040 END
0050 REM *** INCREASE THE VALUE OF A
0060 FOR J=1 TO 10
0070    LET A=A+J
0080    PRINT "This is the value of A: ",A
0090 NEXT J
0100 LET X=X+1
0110 PRINT "The number of subroutine calls is: ",X
0120 RETURN

*
```

This program uses the INPUT statement to display a prompt and get a value, the IF statement to decide where to transfer program control, the GOTO and GOSUB statements to transfer program control, the REM statement to identify the subroutine, and the FOR/NEXT statements to increase the value of A. When you LIST the program, Business BASIC pads all the line number references to four digits, appends the text of the REM statement to the subroutine call, and sets the space indentations in the FOR/NEXT loop.

To get the line numbers back in a balanced number sequence, use the RENUMBER command. If you want to renumber only a section of your code, you can use the RENUM utility.

FOR/NEXT, INPUT, GOTO, GOSUB/RETURN, LIST, REM, and RENUMBER are explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.* RENUM is explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Modifying Your Program

To modify a program in working storage, you can:

- Replace a statement by typing in a new statement with the same line number or add a new statement by giving it a unique line number.
- Delete statements by using the ERASE command, by typing in a line number without any information following it, or by typing in either a range of line numbers (10,20) or a line number followed by a comma (20,), which removes all the lines from that number through the end of the program.
- Bring in statements from a listing file with the ENTER command.
- Use Business BASIC keyboard editing commands to change statements.
- On AOS systems, use the AOS SCREENEDIT control characters to change statements.

One way to change a program statement is to retype the line. Whenever Business BASIC encounters duplicate line numbers, it overwrites the existing statement with the new statement. This also happens when you ENTER a program or subroutine into working storage and a program is already there. As Business BASIC merges the new program statements with the current program, it checks the line numbers of both pieces of code. Each time it finds duplicate line numbers, it replaces the existing statement with the statement being brought into working storage.

You can also delete program statements. The ERASE command deletes a range of statements. Another way to delete a range of statements is by typing in the starting line number and the ending line number separated by a comma (i.e., 15, 45). To delete all the lines from one line number to the end of your program, type in the line number followed by a comma. If you want to delete a single program statement, type in its line number and press the new line key. Thus, by typing in 80 by itself, you could remove line 80 of your current program:

```
* LIST 80
0080            PRINT "This is the value of A: ",A

* 80
* LIST
0005 LET X=0
```

```
0010 INPUT "Enter a number: ",A
0020 LET B=3
0030 PRINT A*B
0035 LET A=A*B
0036 IF A>250 THEN GOTO 0040 ELSE GOSUB 0050 : *** INCREASE THE VALUE OF A
0037 GOTO 0030
0040 END
0050 REM *** INCREASE THE VALUE OF A
0060 FOR J=1 TO 10
0070   LET A=A+J
0090 NEXT J
0100 LET X=X+1
0110 PRINT "The number of subroutine calls is: ",X
0120 RETURN

*
```

Business BASIC does not automatically renumber your program when you delete lines.

Business BASIC provides several methods for changing an incorrect statement line. If you type a character and then want to erase it, use the DEL key. To erase an entire line, press CTRL-X on RDOS systems and AOS systems generated for terminal type 6. On other AOS systems, press CTRL-U. This erases only the line you are typing in; it does not affect an existing program line. To tell the system to ignore the line you are typing, press the IKEY.

Keyboard Editing Commands

To modify the contents of program statements, you can use the Business BASIC keyboard editing commands (the dot editor) or, if you are on an AOS system generated for either terminal type 8 or 9, the AOS SCREENEDIT control characters. This section deals with the keyboard editing commands; the AOS SCREENEDIT control characters are discussed in appendix F of the *Business BASIC System Manager's Guide* and in the *Command Line Interpreter (CLI) User's Manual (AOS and AOS/VS).*

The keyboard editing commands work with program statements in working storage. To use these commands, the statement you want to modify must be in the edit buffer, a special buffer that holds one line from working storage at a time. Lines are placed in the edit buffer in one of three ways:

- Any statement you enter that causes an error is automatically placed in the edit buffer by the Business BASIC interpreter.
- The last line displayed following a LIST command remains in the edit buffer.
- Any line specified by the LIST line-number command remains in the edit buffer.

The line stays in the edit buffer until you replace it with a new line or use a command to move it to working storage.

Most of the editing commands echo the edited line on your terminal but do not change the actual line in working storage. This lets you make additional changes to the line. When the line is correct, you must move it to working storage with the . (period) command. This empties the edit buffer. The only editing command that does not work this way is the .C command, which always places the modified line in working storage and leaves the edit buffer empty.

Several of these commands use delimiters to separate the command from the text. The delimiter is always the first character to follow the command and must be used consistently in each command line. When a command requires more than one delimiter, the delimiter must be the same each place it is used in that command line. Unless otherwise specified, the delimiter can be anything that is not part of the string of text. Table 2-1 contains a summary of the keyboard editing commands.

Command	Function
.(period)	Sends the line in the edit buffer to working storage.
.A string	Appends string to the line in the edit buffer. A space is frequently used as a delimiter with this command.
.C/string1/string2[/G]	Changes the first occurrence of string1 to string2. If you include the G switch, .C changes all occurrences of string1 to string2 in the edit buffer. You cannot use a space as delimiter with the .C command. The .C command passes the line to working storage.
.E/string1/string2[/G]	Changes the first occurrence of string1 to string2. If you include the G switch, .E changes all occurrences of string1 to string2 in the edit buffer. You cannot use a space as a delimiter with the .E command.
.I string	Changes the entire line in the edit buffer to string. A space is frequently used as a delimiter with this command.
.P	Displays the contents of the edit buffer.

Table 2-1. Keyboard Editing Commands

The . (period) command moves the modified program statement from the edit buffer to working storage, leaving the edit buffer empty. Use this command in conjunction with the commands that modify lines (except for .C) to place the corrected line in working storage. If you do not use the . command, no changes are made to the actual program statement.

The .P command displays the line in the edit buffer; it does not affect the contents of the edit buffer or of working storage. To place a line in the edit buffer and then check it, type in the following two commands:

```
* LIST 20
0020 LET B=3

* .P
0020 LET B=3

*
```

The .E and .C commands change text in a program statement. The existing text (string1) and the new text (string2) are set off by delimiters. Both .E and .C accept any character as a delimiter except the space character.

The .E and .C commands differ in that .C places the changed line in working storage, while the .E command leaves the revised line in the edit buffer. With the .E command, the actual program statement is not changed unless you use the . command to move the new line to working storage. If you use the .C command, the Business BASIC interpreter checks the syntax of the line being passed to working storage. If the edited line causes an error, the system displays an error message and returns the line to the edit buffer.

The command formats are:

.E/string1/string2[/G]

.C/string1/string2[/G]

Without the G switch, both .E and .C change only the first occurrence of string1 to string2. When the G switch is used, all occurrences of string1 are changed to string2.

To see how .E and .C work, LIST line 50. This places it in the edit buffer.

```
* LIST 50
0050 REM *** INCREASE THE VALUE OF A

*
```

Now use the .E command to change the line:

```
* .E/INCREASE/MODIFY
0050 REM *** MODIFY THE VALUE OF A

*
```

Use the .P command to view the contents of the edit buffer and then the . (period) command to move the line to working storage.

```
* .P
0050 REM *** MODIFY THE VALUE OF A

* .
0050 REM *** MODIFY THE VALUE OF A

* LIST 50
0050 REM *** MODIFY THE VALUE OF A

*
```

If you had used the .C command instead of the .E command, you would have received the error message — `ERROR 73 - Edit buffer is empty` — when you used the .P command.

The keyboard commands .A and .I change lines instead of strings in a line. The .A command appends a string to the end of a line, and the .I command replaces the existing line with the line you type in. Both commands leave the new

line in the edit buffer instead of placing it in working storage. Their formats are:

.A string

.I string

To change line 20 of your sample program by appending +A to it, use the .A command:

```
* LIST 20
0020 LET B=3

* .A +A
0020 LET B=3+A

* .
0020 LET B=3+A

*
```

With the .I command, you type in a string that is identical to the way you want your program statement to look, including the line number (the edit buffer treats line numbers as characters in the line). The .I command does not echo the modified line on your terminal.

```
* LIST 20
0020 LET B=3+A

* .I 0020 PRINT B*A

* .
0020 PRINT B*A

*
```

In addition to the keyboard editing commands, Business BASIC has an EDIT utility that you can use to create or edit a text file. To conserve memory, EDIT uses a disk-resident buffer to hold the lines being modified. The EDIT utility is explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Saving Programs

Use either the LIST or SAVE command to save the contents of working storage. LIST by itself displays the contents of working storage at your terminal; LIST with a filename preceded by a quotation mark creates a listing file (text file) that contains the program statements in working storage and is stored on disk. SAVE places the contents of working storage in a SAVE file, which is stored on disk in binary format. The formats for these commands are:

* LIST "filename

* SAVE "filename

Use the ENTER statement/command to bring a listing file into working storage. Use the LOAD command to bring a SAVE file into working storage. (You can also execute a SAVE file directly typing in RUN "filename, CHAIN "filename, SWAP "filename, or !filename.) Before you bring any program into working storage, clear working storage by executing a NEW command.

```
* NEW                (Clear working storage)
* ENTER "filename    (This is a LISTed file)
* LOAD "filename     (This is a SAVEd file)
```

To save the sample program you've been working with, type in the following command:

```
* SAVE "SAMPLE
```

The file SAMPLE now exists in your directory as a SAVE file.

When you create a program in working storage or when a program completes its execution, it stays in working storage until one of the following is executed:

- A BYE statement/command to log the user off Business BASIC.
- A NEW statement/command to clear working storage.
- A RUN command to execute another program.
- A CHAIN statement/command to execute another program.
- A LOAD command to bring a new program into working storage.

If you use a SWAP command to execute another program, the system saves the current contents of working storage, brings the new program into working storage, executes the new program, clears the new program from working storage, and places the old contents back in working storage.

RUN, CHAIN, and LOAD clear working storage before bringing in the new program.

Executing Business BASIC Programs

When you create, ENTER, or LOAD a program into working storage, you can execute it by typing in:

```
* RUN
```

The system clears all values currently assigned to variables and starts executing the program beginning with the lowest line number. The system stops executing the program if it encounters a STOP/END statement or the last statement in the program; it also stops executing the program if an error occurs or if you press the IKEY. If you resume execution of a program by entering RUN line-number or the command CON (continue), the system continues executing the program without clearing the values from the variables.

Other ways to execute programs include typing in SWAP "filename, CHAIN "filename, and !filename (which uses the BASIC CLI). Also, you can execute a program by typing in "filename without a command in front of it while in keyboard mode; this causes Business BASIC to SWAP to that file. In addition, your system manager can set up your system so that it automatically executes a program when you log on to Business BASIC.

Both SWAP and CHAIN can be executed in keyboard mode. Generally, however, they are program statements, and they are used by a program running in working storage to execute a program stored on disk. SWAP returns control to the program in working storage, but CHAIN does not. Figures 2-2 and 2-3 show the flow of program control resulting from SWAP and CHAIN.

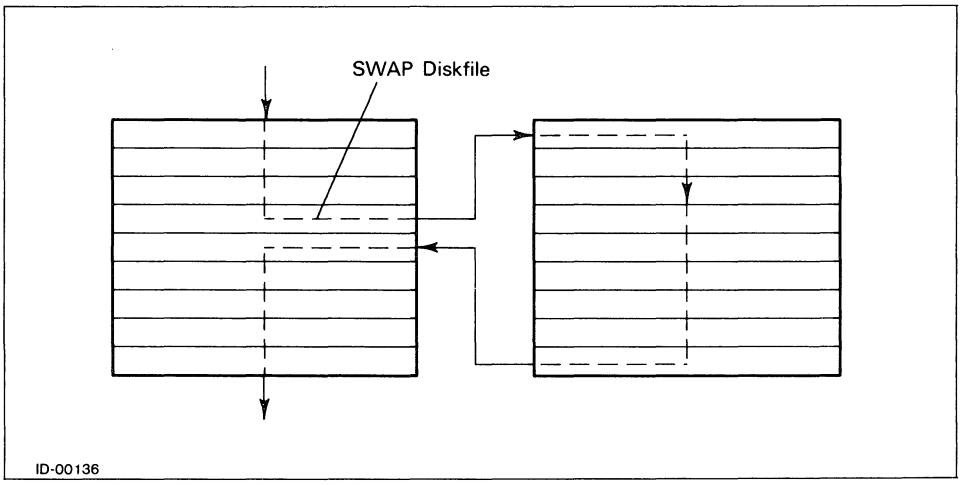

Figure 2-2. Flow of Program Control with SWAP Command

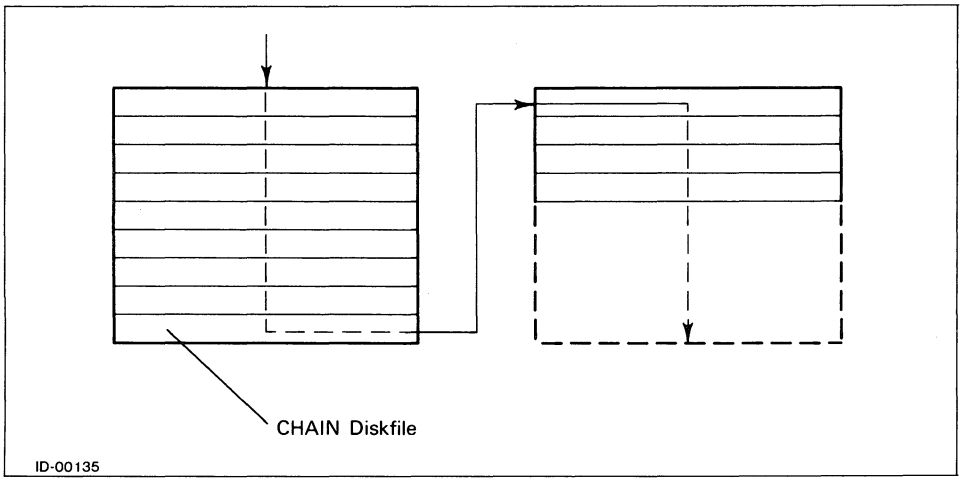

Figure 2-3. Flow of Program Control with CHAIN Command

Continuing Execution of a Program

Business BASIC lets you bring a program into working storage and execute it from the point where it last stopped. Do this by combining the SWAP or CHAIN command with the CON command — SWAP THEN CON or CHAIN THEN CON. These versions of SWAP and CHAIN, like the RUN command with a line number, do not clear values from variables when they continue execution of the target program.

Enter the following program:

```
* NEW
* 10 LET X=4
* 20 PRINT X
* 30 LET X=100
* 40 STOP
* 50 PRINT "X= ";X
* 60 END
*
```

RUN the program. It stops at line 40:

```
* RUN
4

STOP AT 0040
*
```

Now SAVE the program as "PROG2".

```
* SAVE "PROG2
*
```

If you use CON to continue executing PROG2, the program displays X with a value of 100, which was assigned in line 30, and then ends. Instead of continuing PROG2, type in the following program that SWAPs to PROG2.

```
* NEW
* 10 LET X=2
* 20 SWAP "PROG2 THEN CON
* 30 PRINT "BACK ALREADY!"
* 40 PRINT "X now = ";X
* 50 END

*
```

When executed, the new program assigns the value 2 to X and SWAPs to PROG2. PROG2 displays the value of X:

```
* RUN

X= 100
BACK ALREADY!
X now = 2

*
```

In this example, the SWAP THEN CON statement:

1) Stores the working storage (calling) program and its values, including the value 2 for X, in a swapping file.

2) Brings PROG2 into working storage from disk.

3) Resumes execution of PROG2 at the point where the program stopped before it was SAVEd. In this case, PROG2 stopped at line 40 and was SAVEd so execution resumes with line 50. In addition, the variables in PROG2 retain the values assigned to them during the last execution.

4) Returns the calling program to working storage when PROG2 ends. The values assigned in the calling program are retained — they are not affected by any action PROG2 took.

Therefore, the first time X is printed (by line 50 in PROG2), it equals 100, the value assigned to it the last time PROG2 ran. The second time X is printed it equals 2, the value assigned to it in the calling program.

To pass values for variables directly between SWAPping programs, you can use the BLOCK READ and BLOCK WRITE statements. These statements transfer data through the common area, a 512-byte memory location used for storing information (see chapter 4). To transfer single-word values, use STMA 1. The *Business BASIC Reference Manual for Commands, Statements, and Functions* explains the BLOCK READ and BLOCK WRITE statements and STMA 1.

Table 2-2 illustrates the differences between the program execution commands.

Interrupting and Debugging Programs

You can interrupt Business BASIC programs by pressing the IKEY. Normally, when an interrupt occurs during program execution, the program halts and your terminal returns to keyboard mode. From keyboard mode, you can check the values assigned to your program variables. You can also assign new values to variables by entering a keyboard assignment command, such as LET X=2. Then you can execute the program from a line number to see what the program does with new values.

The CON command is a useful debugging aid. If your program stops for any reason (a STOP statement, an error, or an interrupt), you can use the CON command to continue execution at the next higher line. CON does not affect either the value of the variables or the status of files (i.e., where the file pointer is or whether the files are open or closed).

Command Format	Usage	Site of Executed Program	Program Kept in Working Storage after Execution	Variable Values Retained 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 ...	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 2-2. Program Execution Commands

The following program causes an error because Z has not been assigned a value:

```
* NEW
* 10 LET X=2
* 20 LET Y=4
* 30 PRINT Z
* 40 PRINT "Z TIMES 10 EQUALS:";Z*10
* 50 STOP
* RUN

ERROR 17 AT 0030--UNASSIGNED VARIABLE
*
```

To get around the error temporarily, assign a value to Z from the keyboard and enter the CON command:

```
* Z=X+Y
* CON
```

```
 Z TIMES 10 EQUALS: 60
STOP AT 0050
*
```

CON started execution at line 40, the line after the one that caused the error. The program accepted the value you assigned Z in working storage and used it to calculate the value of Z times 10. However, if you RUN the program again, you discover:

```
* RUN

ERROR 17 AT 0030--UNASSIGNED VARIABLE
*
```

The error still exists in the program. This is because RUN clears all values for variables, including the keyboard assignment for Z. To correct the program, type in a line 25, which assigns a value to Z within the program, and then RUN the program from line 25.

```
* 25 LET Z=X+Y
* RUN 25
  6
Z TIMES 10 EQUALS: 60

STOP AT 0050
*
```

Debugging Aids

Business BASIC provides several utilities to help you debug programs. These include:

- PD, which provides information about a SAVE file or the program in working storage.
- RNAM, which renames program variables in a SAVE file.
- SIZE, which displays the working storage space allocations.
- VAR, which lists the variables in a SAVE file or a program in working storage.

These utilities are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Handling Interrupts from within Programs

Business BASIC also has two program statements to help you debug your program — the ON IKEY statement and the ON ERR statement. If Business BASIC processes one of these statements before encountering an interrupt or an error, then the statement directs the action the system takes when an interrupt or error occurs.

The ON IKEY Statement

With the ON IKEY statement, you can trap any interrupts in your program. An ON IKEY statement has the following form:

```
ON IKEY THEN statement
```

where statement is a valid Business BASIC statement excluding FOR, NEXT, DATA, END, REM, and DEF. The way ON IKEY works is:

1) Place the ON IKEY statement in your program.

2) Business BASIC processes the ON IKEY statement.

3) After that, any time Business BASIC encounters an interrupt, it executes the THEN statement portion of ON IKEY.

You can suspend the interrupt condition by executing an STMA 6,5. This disables interrupts so that ON IKEY neither traps an interrupt nor halts the program. If an interrupt occurs after STMA 6,5 has been set, Business BASIC sets SYS(26) to 1. This lets you to see that an interrupt occurred. You can restore interrupt handling by using STMA 7,5 to re-enable the ON IKEY statement.

When an interrupt occurs before an ON IKEY statement has been executed, program execution halts and Business BASIC returns you to keyboard mode.

ON IKEY THEN INT cancels the previous ON IKEY statement and returns interrupt handling to Business BASIC.

If you interrupt an input/output statement, you risk losing data. You can continue the program, but CON starts execution only at the next statement and does not resume interrupted input/output operations.

The ON ERR Statement

The ON ERR statement traps errors in your program the same way ON IKEY traps interrupts. An ON ERR statement has the following form:

```
ON ERR THEN statement
```

where statement is a valid Business BASIC statement excluding FOR, NEXT, DATA, END, REM, and DEF. The way ON ERR works is:

1) Place the ON ERR statement in your program.

2) Business BASIC processes the ON ERR statement.

3) After that, any time Business BASIC encounters an error, it executes the THEN portion of the ON ERR statement and sets SYS(7) and SYS(31) to the error code of the appropriate error message.

   A positive error code indicates a Business BASIC error; a negative error code indicates an operating system input/output error. Both SYS(7) and SYS(31) contain the same number when a Business BASIC error occurs. If an operating system error occurs, SYS(7) holds the code for the RDOS error message while SYS(31) holds the code for the equivalent native operating system error message. On an AOS system, SYS(31) contains the code for an AOS error message; on an RDOS system, SYS(31) contains the code for an RDOS error message (i.e., the same code that SYS(7) contains). You can use SYS(7) with the ERM$ function or SYS(31) with the AERM$ function to retrieve the error message. However, if you are on AOS and SYS(7) contains -60, you must use SYS(31) with AERM$ to get the appropriate AOS error message.

If an error occurs before the ON ERR statement has been executed, Business BASIC halts program execution, displays an error message, and returns you to keyboard mode. In both cases Business BASIC sets SYS(7) and SYS(31) to the error code of the appropriate error message.

ON ERR THEN INT cancels the previous ON ERR statement and returns error handling to Business BASIC.

ON ERR, ON IKEY, SYS, ERM$, and AERM$ are explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

Documenting and Storing Programs

Business BASIC stores a program as either a SAVE file or listing file, based on how you place the file on disk. If you create the file in working storage and SAVE it, you have a SAVE file; if you LIST it to disk or write it in an editor, you have a listing file.

The type of file you have and where you created the file (working storage or an editor) affects the internal documentation of your program. Documentation in Business BASIC programs takes two main forms — colon comments, which are in-line comments, and REM statements.

You can include REM comments in any kind of file; however, colon comments go in listing files only. To include colon comments in a file, you can either:

- Write the file in an editor and type in the colon comments.
- Add colon comments to an existing listing file by using the EDIT utility.

You cannot place colon comments in files in working storage because everything there is in binary format. Even when you ENTER a listing file, Business BASIC converts it from ASCII format to binary format and strips out the colon comments. Thus, if you use the LIST command to display an ASCII file that contained colon comments or if you type in a program statement in work-

ing storage with a colon comment, none appears on your screen. A line you type in as:

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

appears on your screen when you LIST the line as:

```
* LIST
0010 LET A=12

*
```

To see a listing file's colon comments, you must either look at the file in an editor or use the BASIC CLI command TYPE to display the file at your terminal. To avoid losing those comments, do not use the REPLACE command to store the file on disk. (REPLACE deletes the file on disk and replaces it with the file in working storage, which is in binary format.) To make changes to a listing file, use an editor or the EDIT utility.

SAVE Files

The SAVE file is a program that you have stored on disk using either the SAVE or REPLACE statement/command (i.e., SAVE "filename or REPLACE "filename). This file is stored in binary Business BASIC SAVE file format. In this format, each keyword is assigned a number so that the number, not the larger alphanumeric word, is stored. Using this format reduces the amount of space a file takes up on disk, but it prevents you from using an editor to modify the file.

To preserve internal program documentation, use the REM statement or keep a copy of the program in a listing file.

Listing Files

A listing file is an ASCII format file. To get a listing file, either write the program in an editor or create it in working storage and then use the command LIST "filename to store the file on disk. Using LIST "filename always produces a listing file (i.e., an ASCII text file that contains a program and its REM comments).

You can create an ASCII version of a SAVE file by LOADing the file into working storage and then using the LIST command to store the file (with another filename) on disk. You can only LIST a file to disk when there is no file named filename on disk.

Table 2-3 summarizes the differences between the LIST, SAVE, and REPLACE commands.

Command	Output	Access Command	Effect
LIST	ASCII	ENTER	Creates an ASCII format file; the file cannot exist prior to being LISTed. Does not preserve variable values or last line number executed. Allows you to display results of typing and editing in sequential line number order and to use keyboard editing commands. Retains REM comments but not colon comments.
SAVE	BASIC SAVE FILE FORMAT	LOAD, RUN, CHAIN, SWAP	Creates a program in Business BASIC SAVE file format; the file cannot exist prior to being SAVEd. Preserves variable values and last line number executed. Retains REM comments but not colon comments.
REPLACE	BASIC SAVE FILE FORMAT	LOAD, RUN, CHAIN, SWAP	Creates a program in Business BASIC SAVE file format or overwrites a disk file with the contents of working storage; Preserves variable values and last line number executed. Retains REM comments but not colon comments.

Table 2-3. Commands to Save Programs

End of Chapter

Chapter 3
Numeric, Array, and String Variables

This chapter deals with Business BASIC variables and how they can be used in your programs. In addition, it describes Business BASIC arithmetic, the use of expressions in your programs, and string functions.

Variables

Business BASIC uses three types of variables: numeric, array, and string. Numeric values have no default values; you must assign a value to them before you use them or you will get an error message. Array elements have a default value of 0 while strings have a default value of the null string, which has a length of 0.

Normal variable names can be up to six characters long. The first character must always be a letter; however, the following characters can be letters or numbers.

In certain cases, a variable name can have up to seven characters. Then the last character is a special character that provides additional information about the variable. A special character must always be the last character in a variable name; when you are using a special character, the variable name can be two to seven characters long. The special characters are:

- $ to indicate a string variable.
- % to indicate a numeric variable that only allows 2 bytes of data to be transferred with READ/WRITE FILE statements or PACK/UNPACK statements.
- # to indicate a numeric variable that only allows 6 bytes of data to be transferred with READ/WRITE FILE statements or PACK/UNPACK statements.

Another restriction on variable names is that they cannot be keywords. The file APERM.PS in the library directory of your Business BASIC system contains a list of keywords. Here are some examples of legal and illegal variable names:

Legal	Illegal	Reason
A	1	Variable names must begin with a letter.
A303	1AB	Variable names must begin with a letter.
Z6B8	Z2345678	Variable names cannot be longer than six characters (seven if you end the name with a special character). Business BASIC truncates variable names that are too long instead of giving you an error message and uses the truncated name for the variable.
WAGE	$WAGE	Variable names must begin with a letter. In addition, special characters can be used only as the last character in the variable name.
WAGE$	#WAGE	Variable names must begin with a letter. In addition, special characters can be used only as the last character in the variable name.
WAGE#	A$%#	Special characters can be used only as the last character in a variable name. All other characters in a variable name must be either letters or numbers.

You can assign a value to a variable or change the value of a variable with the following statements:

- READ/DATA
- READ FILE
- PACK
- UNPACK
- LET
- INPUT/INPUT USING
- TINPUT

Numeric Data

Numeric data is limited to integers; however, Business BASIC provides formatting techniques (such as PRINT USING) that allow you to maintain numeric precision and print numbers with decimal points. In Business BASIC, numeric data includes numeric constants, numeric variables, and arrays.

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

```
59
-771083
+941
```

Numeric Variables

A numeric variable is a data item that has a numeric value assigned to it during program execution. Examples of numeric variables include:

```
A
A3
A#
NUM%
NUM1
OUTPUT
```

Precision

Precision refers to the number of bytes used to store a numeric variable or array element. With Business BASIC you can have either a double precision or triple precision system. The precision of your system is determined by your system manager when Business BASIC is generated.

Storage of Numeric Variables

There is a distinction between storage precision and data transfer precision. Only double and triple precision are available for storage, while single, double, and triple precision are available for data transfer. Double precision numeric variables store numbers using 4 bytes and can range in value from -2,147,483,648 to +2,147,483,647. Triple precision numeric variables store numbers using 6 bytes and can range in value from -140,737,488,355,328 to +140,737,488,355,327.

Data Transfer of Numeric Variables

Business BASIC supports three forms for transferring numeric information to and from a file:

- 2 bytes of data. Variables used for transferring 2 bytes of data are indicated by a percent sign (%) at the end of the variable name. This is signed data with a value range of -32768 to 32767.
- 4 bytes of data. Variables used for transferring 4 bytes of data have no special character at the end of the variable name.
- 6 bytes of data. Variables used for transferring 6 bytes of data are indicated by a pound sign (#) at the end of the variable name.

This means that for a READ/WRITE FILE statement or a PACK/UNPACK statement, 2 bytes are read in or written out for each numeric variable whose name ends with a percent sign; 4 bytes for each numeric variable that has no special character in its name; and 6 bytes for each numeric variable whose name ends with a pound sign. Variables that transfer 6 bytes of data cannot be used in a double precision system, but you can use variables that transfer 4 bytes of data on a triple precision system. Variables that transfer 2 bytes of data can be used on both systems.

For example, the statement:

```
* 0010 WRITE FILE (0), VAR#
```

transfers 6 bytes (triple precision). If the variable name had been simply VAR, it would transfer 4 bytes, while with the variable name VAR%, it would transfer 2 bytes.

Arrays

An array is an ordered set of integer values. Business BASIC does not support string arrays. Each member of the set is an array element. BASIC stores each array element according to the precision of the system. On double precision systems, an array element holds a 4-byte value; on triple precision systems, an array element holds a 6-byte value. The only restriction on the number of array elements you can have is the amount of available memory.

Arrays can have one or two dimensions. Elements of one-dimensional arrays form one row; elements of two-dimensional arrays form several rows and columns. Indexing for arrays is zero-based; thus, arrays always start at element 0.

Figure 3-1 shows how one- and two-dimensional arrays are set up.

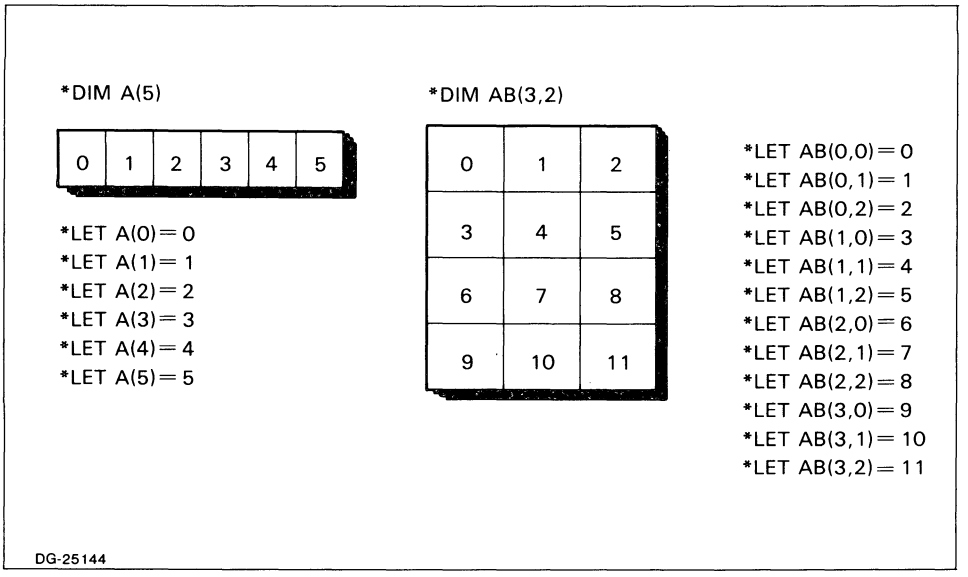

Figure 3-1. One- and Two-Dimensional Arrays

Creating Arrays

You create an array by using the DIM statement to specify the number of elements in the array. For example:

```
* 10 DIM A(5),B(2,6)
```

dimensions array A as a one-dimensional array with six elements and array B as a two-dimensional array with three rows and seven columns that holds 21 elements.

Default Array Dimensions

If you use an array without dimensioning it, Business BASIC assigns default dimensions to it. The default dimensions for undeclared arrays are:

- One-dimensional arrays — 11 elements (column positions 0 to 10).
- Two-dimensional array — 121 elements (rows 0 to 10, columns 0 to 10).

For example, if you enter the following commands:

```
* NEW
* LET C(3)=7
```

you create array C with 11 elements. This is the same as if you had entered DIM C(10). If you enter these statements:

```
* NEW
* LET D(3,2)=6
```

you create a 121-element array D with a default dimension of (10,10).

You can conserve space by using a dimension statement to declare explicitly an array requiring fewer than the default number of elements.

Accessing Array Elements

You refer to an array element by specifying the array's name followed by one or two subscripts enclosed in parentheses or brackets. Subscripts can be numbers, variables, or expressions that identify a specific row and column position of an element. A subscript must evaluate to a number between zero and the value declared in the array's dimension statement.

One-dimensional array elements are identified by a single subscript. For example, elements of an array declared as DIM B(5) are referred to as:

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

A two-dimensional array is referred to by two subscripts separated by a comma. If you use only one subscript to refer to an element in a two-dimensional array, the system uses a default of 0 for the column subscript. For example, A(1,2) refers to row 1, column 2, but A(1) refers to row 1, column 0. Some elements of an array declared as DIM E(24,5) are :

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

You get an error message if O, I-3, or J*K evaluates to a number outside the array range.

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

Changing Array Dimensions

You can change the dimensions of a declared array by issuing a DIM command. This lets you redimension an array to the same number of elements or fewer elements; you cannot make an array larger than its original dimension. For example, you can redimension an array declared as B(2,3) to B(1,2) or B(3,2), but not to B(3,3) since that array would be greater than the original array.

Redimensioning alters the way you refer to array elements. It does not change the values in the array or the amount of storage the array occupies. If you make an array smaller, you do not free unused memory locations; you just make it impossible to refer to them. If you change the one-dimensional array E from 11 elements to 10:

```
* DIM E(10)
* DIM E(9)
```

the element E(10) cannot be accessed. In addition, references to subscripts outside the newly defined range of subscripts cause an error.

Assigning Values to Numeric Elements

You can assign values to array elements and other numeric variables by using one of the Business BASIC statements — LET, READ/DATA, READ FILE, INPUT, INPUT USING, TINPUT, PACK, and UNPACK. The following example uses LET statements to assign values to elements in the arrays A and AB.

```
* 0010 DIM A(8)
* 0020 LET A(4)=5
* 0030 LET AB(4,4)=6
* 0040 LET AB(2,1)=7
```

This example uses the DIM statement to explicitly dimension array A as (8), but it implicitly dimensions array AB as (10,10), the default dimension for two-

dimensional arrays. The LET statement operates the same regardless of how the array is dimensioned.

This example uses the READ command to assign values to the variables X and Y as well as array element (1,3). The values are obtained from line 80, which has the data command. You can also use the INPUT, INPUT USING, and TINPUT statements to assign values to these variables.

```
* 0010 DIM ARA(12,12)
* 0020 READ X,Y
* 0030 READ ARA(1,3)
* 0040 PRINT "X="X
* 0050 PRINT "Y=";Y
* 0060 PRINT "ARA(1,3)=";ARA(1,3)
* 0070 STOP

* 0080 DATA 32,46,1400
* RUN
X=32
Y=46
ARA(1,3)=1400

STOP AT 0070
*
```

Use different variable names when you use numeric variables and arrays in the same program. Business BASIC is structured so that you can refer to a simple variable using zero subscripts. This means that when you have a variable and an array with the same name, the simple variable and element 0 of the array have the same value.

```
* 10 DIM X(5)
* 20 X=9
* 30 X(0)=3
* 40 PRINT "The variable X is ",X
* 50 PRINT "Element 0 of array X is ",X(0)
* 60 END
* RUN
The variable X is 3
Element 0 of array X is 3

*
```

The value in both X and in X(0) is always the value that was assigned most recently.

Numeric Expressions

A numeric expression is any combination of numbers, numeric variables, array elements, and numeric functions linked together by arithmetic operators and parentheses. Business BASIC supports expressions that use arithmetic operators, relational operators, and Boolean logic operators.

Arithmetic Operators

You can use arithmetic operators in any numeric expression to add, subtract, divide, multiply, and perform exponentiation. Business BASIC evaluates numeric expressions according to the precedence shown in Table 3-1.

Precedence	Operator	Action That Is Performed	Example
1	( )	Any expression in parentheses; the innermost expression in nested parentheses.	(5−(1+1))
2	^	Exponents	A^B
3	+,−	Unary plus, Unary minus.	A−(+B) A+(−B)
4	*,/	Multiplication, Division.	A*B A/B
5	+,−	Addition, Subtraction.	A+B A−B

Table 3-1. Precedence of Arithmetic Operations

When two operators have equal precedence, Business BASIC evaluates them from left to right. If you enter the statement:

```
* LET X=Z+(-A)+B*C^D
```

Business BASIC calculates X in the following order:

1) A is negated.

2) C is raised to the power of D.

3) B is multiplied by the result of step 2.

4) The result of step 1 is added to Z.

5) The result of step 3 is added to the result of step 4.

Using parentheses changes the order of numeric operations. For example, when you enter the statement:

```
* LET X=Z-((A+B)*C)^D
```

Business BASIC evaluates X as follows:

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.

Relational Operators

Business BASIC expressions accept relational operators. These operators can be used wherever an expression is valid. All relational operations are evaluated as true or false and reduced to a value of 1 (true) or 0 (false). Business BASIC executes an action based on how the relational operator evaluates. These operators are often used in decision-making statements, such as the IF statement. Using these operators can reduce the amount of code you need to perform various functions. Table 3-2 summarizes the relational operators.

Operator	Meaning	Example
=	Equals	A=B
<	Less than	A<B
<=	Less than or equal to	A<=B
>	Greater than	A>B
>=	Greater than or equal to	A>=B
<>	Not equal	A<>B

Table 3-2. Relational Operators

The following program lines use relational operators as part of an assignment statement:

```
* 40 LET A=(X>10)
* 50 LET B=10*(A$>B$)+20*(A$=B$)+30*(A$<B$)
```

Line 40 results in A=1 if the variable X is greater than 10, otherwise A=0. Carrying this one step further, line 50 results in B=10 if only A$>B$ is true, B=20 if only A$=B$ is true, and B=30 if only A$<B$ is true.

You can also use relational operators to determine what action your program takes.

```
* 0010 IF A^B>=32767 THEN GOTO 0300
* 0020 IF ABS(X*Y)<>A THEN GOSUB 0900
```

When the expression A^B>= 32767 evaluates to true, program control goes to line 300. Line 20 is executed only when the expression A^B>=32767 evaluates to false. In line 20, program control goes to the subroutine that starts at line 900 when the expression ABS(X*Y) <> A evaluates to true; otherwise, the next sequential statement is executed.

Boolean Logic Operators

Business BASIC contains three Boolean logic operators: AND, OR, and NOT. (Business BASIC also has the functions AND and OR, which work with binary expressions. The system determines whether you are using the Boolean operator or the function based on the placement of AND or OR in the statement.) The Boolean operators can be used anywhere an expression is valid.

Before the Boolean operator is executed, the expressions to be tested are evaluated as true or false, and the operands are reduced to 1 or 0. In Boolean evaluation, a value of 0 is considered false, and any other value is considered true. Thus, when you use Boolean operators,

- 3 AND 0 is the same as 1 AND 0 (TRUE and FALSE)
- -1 OR 0 is the same as 1 OR 0 (TRUE or FALSE)
- -1 AND 3 is the same as 1 AND 1 (TRUE and TRUE)

The Boolean operators work in the following way:

- NOT reverses the logical value of an expression (i.e., if an expression evaluates to true, NOT causes it to evaluate to false).
- AND evaluates to true only if the two expressions it is used with both evaluate to true; if either expression is false, then AND evaluates to false.
- OR evaluates to true if either or both of the two expressions it is used with evaluate to true; OR evaluates to false only if both expressions are false.

You can use the Boolean operators in a variety of statements. They are often used in connection with decision-making statements, such as IF. For example, with the statement:

```
* 0010 IF A>1 AND B=2 THEN GOTO 0100
```

Business BASIC checks to see whether the expression A>1 is true or false. The expression is true when A is greater than 1 and false when A is less than or equal to 1. Next, Business BASIC evaluates the expression B=2. This expression is true only when B has a value of 2. If both expressions evaluate to true, Business BASIC executes the GOTO 100. If either expression or both expressions evaluate to false, Business BASIC executes the next sequential statement.

You can also combine Boolean operators in a statement.

```
* 0020 IF C AND NOT D GOSUB 0200
```

This statement is set up so that expression C evaluates to false only when C equals 0; otherwise, it is considered true. The expression NOT D evaluates to true only when D equals 0. This is because NOT changes the value of 0 (false) to non-zero (true). The GOSUB 200 is executed only when C is a non-zero value and D is 0.

The following statement uses NOT to evaluate an OR expression:

```
0030 IF NOT (E OR F) GOTO 0100
```

Control goes to line 100 when both E and F equal 0, thus making the OR expression evaluate to false.

You can also use Boolean operators in assignment statements.

```
0040 LET G=NOT (H AND I)
```

This statement assigns G a value of 0 when both H and I are non-zero and a value of 1 when either or both H and I equal 0.

The next example uses an OR expression to determine what value to print.

```
0050 PRINT J OR K
```

Business BASIC prints 0 when both J and K equal 0; otherwise, if either or both variables are non-zero, Business BASIC prints 1.

Table 3-3 summarizes the precedence in Business BASIC of Boolean operators, relational operators, and arithmetic operators.

Precedence	Operator
1	Parentheses
2	Exponentiation
3	Unary plus, Unary minus, NOT
4	Multiplication, Division
5	Addition, Subtraction
6	Not equal, Greater than, Greater than or equal to, Equal, Less than or equal to, Less than
7	AND
8	OR

Table 3-3. Hierarchy of Operators in Business BASIC

You need to use parentheses with Boolean operators only when you are using a Boolean operator to evaluate an expression involving another Boolean operator or when you want to make the statement more readable. This means the statement `LET A=(B=C)`, which sets A equal to 1 if B equals C and A equal to 0 if B does not equal C, can also be expressed without parentheses: `LET A=B=C`. This is not equivalent to the multiple assignment syntax: `LET A,B=C`.

Handling Decimals

All numbers in Business BASIC are 4-byte or 6-byte integers, depending on whether your system uses double precision or triple precision. Use the following procedures when you are working with decimals:

- To add decimals, multiply by an appropriate power of 10.
- To dispose of 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 must have as many decimal places as are needed in the quotient.

The following is an example of adding and dropping decimals:

```
:
: This program adds two numbers X and Y and prints the result with
: two decimal places and again with one decimal place.
:
: X=.73 and Y=47.6
:
0010 DATA 73,476,73,476
0020 READ X,Y
0030 LET X=X+Y*10                    :for two decimal places
0040 PRINT USING "E6.2,T10,Z",X
0050 READ X,Y
0060 LET Y=(X+(5*SGN(X)))/10+Y       :for one 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
:
: When this program is run the following appears on the screen:
: 48.33    48.3
:
```

The next example involves multiplying with decimals:

```
:
: This program multiplies two numbers X and Y and prints the
: result with one decimal place and again with two decimal places.
:
: 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
:
: When this program is run the following is shown on the screen:
: 34.7     34.75
:
```

The next code segment illustrates dividing with decimals:

```
:
: This program divides two numbers X and Y. First X is divided by Y;
: then Y is divided by X. The result is always shown with one decimal
: place.
:
: X=84.73 and Y=47.6
: X/Y=1.78
: Y/X=.561
: Z will always have 1 decimal place
:
0010 LET X=8473
0020 LET Y=476
0030 LET Z=X/Y                                    : X/Y unrounded
0040 PRINT USING "E6.1,T10,Z",Z
0050 LET Z=(((X*10)/Y)+SGN(X)*SGN(Y)*5)/10        : X/Y rounded
0060 PRINT USING "E6.1,T10,Z",Z
0070 LET Z=Y*100/X                                : Y/X unrounded
0080 PRINT USING "E6.1,T10,Z",Z
0090 LET Z=(((Y*1000)/X)+(SGN(X)*SGN(Y))*5)/10    : Y/X rounded
0100 PRINT USING "E6.1",Z
:
: When this program is run the following is shown on the screen:
: 1.7      1.8       0.5       0.6
:
: Note that in the unrounded examples the answer is not accurate
: beyond the integer portion. If accuracy is important, then
: rounding is required.
:
```

Predefined Numeric Instructions

Business BASIC provides several functions and one statement that deal with numeric computations. These are summarized in Table 3-4. You can use these features in any Business BASIC statement that allows numeric expressions. They are explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

Format	Usage	Description
ABS	Function	Computes the absolute value of a numeric expression.
AND	Function	Sets bits based on the result of the logical AND of two expressions.
ASC	Function	Returns the ASCII value for a string.
CHR$	Function	Places the binary value of a number into a string.
DEF	Function	Creates user-defined functions.
INT	Function	Truncates a number to make it an integer.
LEN	Function	Finds the current length of a string.
MAX	Function	Finds the larger of two numeric expressions.
MIN	Function	Finds the smaller of two numeric expressions.
MOD	Function	Finds the remainder after dividing two numeric expressions.
OR	Function	Sets bits based on the result of a logical inclusive OR of two expressions.
POS	Function	Determines the position of substring in a string.
RND	Function	Produces a random number.
SGN	Function	Determines the sign of a numeric expression.
SHFT	Function	Shifts bits left or right.
SQR	Function	Computes the square root of a numeric expression.
VAL	Function	Converts a string to a number.
VALUE	Statement	Converts a string to a number.

Table 3-4. Numeric Functions and Statements

Character Data

Business BASIC uses strings to handle character data. A string is a combination of characters. It can include letters, digits, spaces, special characters, and sometimes binary values. You can have a string variable or a string literal (also called a string constant).

Anytime you use a string variable, you must dimension it first. Business BASIC does not provide default dimensioning for strings as it does for arrays. After you dimension a string, it has a default value of the null string until you assign it a value.

String Literals

A string literal is a combination of characters that use quotation marks as a delimiter. For example:

```
"Data General Corporation"
```

You can include a special character or control codes in string literals. Do this by using the character's ASCII decimal equivalent value in angle brackets in the string. Use the form <n>, where n is a number from 0 to 255. The angle brackets do not appear when the string is displayed. When you type in:

```
* PRINT "At the sound of the tone <7>"
```

the string `At the sound of the tone` is displayed and the terminal bell rings (7 is the ASCII code for bell).

Just as you can assign numeric constants to numeric variables, you can also assign string literals to string variables. To do this, use one of the Business BASIC assignment statements, such as LET or READ/DATA. In addition, string literals are often used with INPUT statements as user prompts. This string assignment includes a string literal as a prompt as well as a value for a string variable:

```
* 10 DIM A$(20),B$(20)
* 20 INPUT "Enter your string: ", B$
* 30 LET A$="This is a string: <10>"
* 40 PRINT A$,B$
* 50 END
* RUN
Enter your string: Good morning.
This is a string:
       Good morning.

*
```

The string literal in line 30 also included the ASCII code for a new line. Thus, line 40 printed the value of the string variable and then moved the cursor to the next line before printing the value of B$.

You can also use string literals with PRINT:

```
* PRINT "THIS IS A STRING LITERAL"        THIS IS A STRING LITERAL
*
```

String Variables

String variables are data items that have string values assigned to them. Business BASIC requires that the names of string variables end in dollar signs (i.e., A$). String variables use 1 byte to hold the ASCII code for each character of the string. Unlike arrays, string variables start with an index of 1 (i.e., positions 1-8 instead of 0-7).

Since Business BASIC does not provide a default dimension for strings, you must use the DIM statement to allocate the maximum number of characters (bytes) for the string variable before you assign a string value to it.

```
* DIM A$(25),B3$(215)
```

This statement dimensions two string variables: A$ has a maximum of 25 bytes (1-25) and B3$ has a maximum of 215 bytes (1-215). You can redimension a string; however, the new maximum length must be less than or equal to the original maximum length.

You can dimension both arrays and strings in a single line of code:

```
* 0010 DIM ARRAY(5,6),C(20),STRNG$(30)
```

Line 10 dimensions ARRAY as a 42-element (6 by 7), two-dimensional array; C as a 21-element, one-dimensional array; and STRNG$ as a 30-character string variable. To change these dimensions, enter:

```
* 0020 DIM ARRAY(6,5),C(2,6),STRNG$(28)
```

This statement redimensions ARRAY to a 7 by 6, two-dimensional array (still 42 elements), redimensions C to a 3 by 7, two-dimensional array (still 21 elements), and redimensions STRNG$ to a maximum of 28 characters.

Accessing Strings

Business BASIC allows you to access entire strings and subsections of the strings (called substrings). You refer to the complete string by specifying the name of the string variable. To access a substring, specify the string variable's name followed by a subscript. Subscripts indicate the character locations of the substring within the string. Use only one subscript if your substring continues to the end of the string. Use two subscripts if your substring ends before the main string. The first subscript specifies the starting character position of the substring and the second one specifies the ending character position.

A subscript can be a number, a numeric variable, or an expression that evaluates to a number between 0 and the value declared in the string's dimension statement. The subscript 0 refers to the position that follows the last character of the string (use this to add a substring to the end of a string). Table 3-5 illustrates different ways to refer to a string.

Reference	What It Specifies
A$	Entire string.
A$(2)	Second through last character.
A$(R)	Rth through last character, where R is a number in the range 1 to the maximum length of the string.
A$(3,7)	Third through seventh character.
A$(I,J)	Ith through Jth character, where I and J are both numbers in the range 1 to the maximum length of the string and where I is less than or equal to J. If I is greater than J, this is a null string.
A$(0)	The position immediately following the last character in A$ (a string can contain fewer than the maximum number of characters allowed). This is equivalent to A$(LEN(A$)+1).

Table 3-5. References to Strings

Using Strings in Expressions

You can use string expressions (string constants, string variables, and subscripted string varibles) in LET, PRINT, INPUT, and READ statements, and in relational expressions in IF statements. Table 3-6 contains examples of how string expressions are used.

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 program control to line 10 if the Ith through the last character of A$ is the same as the Jth through the last character of B$.
INPUT C$, D$(1,1)	Lets you enter a string literal for C$ and a single character for D$.

Table 3-6. Uses of String Expressions

Business BASIC does not support string arrays. You can create the equivalent of a string array, though, by dimensioning a string that is large enough to hold all the data that would normally go in the string array and then loading the data into the string. You can use the following algorithm to simulate a string array with N items, each having a length of SIZE characters:

1) Dimension a string to N*SIZE characters:

   * DIM STRING$[N*SIZE]

2) Access the Ith array item as:

   * STRING$[(I-1)*SIZE+1,(I-1)*SIZE+SIZE]

   or equivalently as:

   * STRING$[(I-1)*SIZE+1, I*SIZE]

You can use a similar procedure to create a multidimensional string array. The following example translates a month's numeric code into its three-letter abbreviation:

```
* LIST
0010 DIM MONTHS$[36],OUT$[30]
0020 LET MONTHS$="JanFebMarAprMayJunJulAugSepOctNovDec"
0030 INPUT "Enter a number between 1 and 12: ",I
0040 IF I>12 OR I<1 THEN GOTO 0100
0050 LET OUT$=MONTHS$[(I-1)*3+1,(I-1)*3+3]," represents month number ",I
0060 PRINT OUT$
0070 PRINT
0080 GOTO 0030
0100 END

*
```

Assigning Values to Strings

Use Business BASIC assignment statements to assign values to entire strings or to store characters into different locations of a string.

With the LET and READ statements, you can use:

- String variables
- Subscripted string variables
- String functions
- The concatenation operator (comma)

The PRINT, INPUT, PACK, and UNPACK statements work with all variable forms except string functions. (A string function is a built-in Business BASIC function that evaluates to a string value.)

There are two restrictions on string assignments:

- You cannot assign more characters to a string than the string was dimensioned to hold. When you try to assign too many characters, Business BASIC truncates the data to fit the string. You do not receive an error message.
- Strings must be filled beginning from position 1. For example, you cannot assign A$(3,5)="ABC" if A$(1,2) are empty.

Table 3-7 contains examples of string assignments.

Assignment	Action
LET A$=B$	Replaces the contents of A$ with the contents of B$.
LET A$=""	Sets the length of A$ to 0.
LET A$=A$,B$	Appends the contents of B$ to the current contents of A$.
LET A$(0)=B$	Appends the contents of B$ to the current contents of A$.
LET A$=FILL$(0)	Fills A$ to its dimensioned length with nulls.
LET A$=B$,A$	This produces unpredictable results because A$ has been changed by the time it is to be appended.
LET A$=4+5	Replaces the contents of A$ with the string constant 9.
LET A$=12345	Replaces the contents of A$ with the string of digits 12345, not the number 12345.
LET A$=CHR$(12,4)	Replaces the contents of A$ with a 4-byte string holding the binary value of the number 12.

Table 3-7. Assigning Characters to String Locations

Concatenating Strings

You can concatenate a string in an assignment statement by separating the string expressions with commas. In the following program, commas are used to concatenate strings A$ and B$ when string C$ is formed:

```
* LIST
0010 DIM A$(16),B$(23),C$(50)
0020 LET A$= "THE YEAR IS 19XX"
0030 LET B$= "THE MONTH IS XXXXXXXXXX"
0040 LET C$=A$(1,14), "86; ",B$(1,13), "JUNE"
0050 PRINT C$
0060 END
* RUN
THE YEAR IS 1986; THE MONTH IS JUNE

*
```

String Functions

Business BASIC supplies several functions and statements that provide additional string assignment capabilities. The functions can be used as assignment statements or commands. Table 3-8 lists these features. They are explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

Business BASIC also provides two string functions that deal with error messages:

- AERM$, used with SYS(31).
- ERM$, used with SYS(7).

When a Business BASIC error occurs, both functions return the same Business BASIC error message. When an operating system input/output error occurs, ERM$ used with SYS(7) returns the RDOS error message while AERM$ used with SYS(31) returns the equivalent native operating system error message. On an AOS system, this is an AOS error message; on an RDOS system, this is an RDOS error message (i.e., this message is the same as the one that ERM$ used with SYS(7) returns). If you are on an AOS system and SYS(7)equals -60, indicating there is no equivalent RDOS error message, you must use AERM$ with SYS(31) to get the appropriate AOS error message.

Format	Usage	Description
ASC	Function	Return the ASCII value of a string.
CHR$	Function	Place the binary value of a number in a string.
CRM$	Function	Cram (store) 3 bytes of a string into 2 bytes.
EXTRACT	Statement	Extract the next field from a string.
FILL$	Function	Finds the current length of a string.
PACK	Statement	Encode string and numeric information into a string variable known as a record string.
POS	Function	Determine the position of a substring in a string.
SCANUNTIL	Statement	Scan a string until characters in a substring are found.
SCANWHILE	Statement	Scan a string while characters match those in a substring.
STRPOS	Statement	Find the starting position of a substring in a string.
TRUN$	Function	Truncate a string.
UCM$	Function	Expand a crammed string so that 2 bytes are again stored in 3 bytes.
UNPACK	Statement	Take information from a record string containing binary information and place it in separate variables.
VAL	Function	Convert a string of digits to a number.
VALUE	Statement	Convert a string of digits to a number.

Table 3-8. String Functions and Statements

Using Variables to Transfer Data

File access statements (READ FILE, LREAD FILE, WRITE FILE, LWRITE FILE, BLOCK READ FILE, BLOCK WRITE FILE, INPUT FILE, and PRINT FILE) use variables to transfer data to and from files. BLOCK READ and BLOCK WRITE use variables to transfer data to and from the common area (a 512-byte memory location used for storing data that is being tranferred between programs). These methods for transferring variables apply to string variables as well as to numeric variables and arrays.

The size of the variables you supply in the arguments determines the number of bytes transferred. String variables transfer their maximum (dimensioned) length even when they are only partially filled with data (empty string bytes transfer as null bytes). Substrings transfer the number of bytes specified in their subscripts. Both substrings and entire strings transfer one byte per character.

Numeric/String Conversions

There are three Business BASIC functions and three statements that convert numeric data to string data and vice versa. These are:

- CHR$, which puts the binary value of a number into a string.
- ASC, which returns the ASCII value of a string.
- PACK, which encodes string and numeric information into a single string variable, known as a record string.
- UNPACK, which decodes string and numeric information from a record string.
- VAL, which converts a string of digits to a number.
- VALUE, which converts a string of digits to a number.

These functions are explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

End of Chapter

Chapter 4
Subroutines and Utilities

The Business BASIC software package contains subroutines and utilities to aid you in programming. A subroutine is a segment of code that is designed to perform a specific function from within a program. A utility is a Business BASIC program that performs a specialized task.

This chapter discusses the subroutines and utilities that come with the Business BASIC package. In addition, it covers writing your own subroutines and utilities, modifying existing ones, and creating assembly language subroutines.

Subroutines

Subroutines fall into three categories:

Pre-written subroutines.	These are the subroutines that come with the Business BASIC software package. They are located in the library directory. Use the ENTER command to place them in working storage and merge them with your program.
User-created Business BASIC subroutines.	These are the subroutines that you write for your own specialized processing requirements. You can store them in the library directory and then access them repeatedly just as you access the Business BASIC-supplied subroutines.
Assembly language subroutines.	These are subroutines written in assembly language. To use them with your programs, you must add them (or have your system manager add them) to your Business BASIC system when the system is generated.

Business BASIC Subroutines

Business BASIC features a number of pre-written subroutines that can be used within application programs. These subroutines have .SL extensions on their filenames to distinguish them from utility programs and other files.

The Business BASIC subroutines, their entry points, and the line numbers they occupy are listed in appendix A. These subroutines are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Using Subroutines

There are two general procedures for using subroutines:

- Write them as elements of a specific program and type in the program and its subroutines at the same time.
- Write them as individual code segments, test them, and then store them in the library directory until you need them. You can add them to your program later.

The second method helps you debug your program because you have program elements that you know work. Error tracing is more difficult when you debug a complete Business BASIC program at one time.

To use an existing subroutine, it must be merged with a program. Thus, you need to:

1) Place the main program in working storage (either by creating it there or by loading it).
2) Use the ENTER command to load the subroutine into working storage and merge it with the main program.
3) Use the SAVE command to store the now-complete program. This makes the subroutine a permanent part of the program so that you no longer need to enter the subroutine each time you run the program.

Instead of performing the last two steps, you can include an ENTER subroutine-name statement in your program. Then the subroutine is automatically added each time you RUN the program. You no longer need to SAVE the program to retain the subroutine. The disadvantage to this method is that adding subroutines at runtime slows down program execution. The advantage is that, since the pre-written subroutines can change each time a new revision of Business BASIC is released, you will always have the most up-to-date version of the subroutine.

If you ENTER the Business BASIC-supplied subroutines into your program once and then SAVE the program, you should keep track of which programs use which subroutines. This aids you in replacing subroutines that have been updated when you get a new revision of Business BASIC. Another way to maintain up-to-date subroutines is to keep a copy of your program with statements that ENTER your subroutines. Place a STOP statement after the ENTER statements. Only run this version of your program when you get a new revision of Business BASIC. Then delete the ENTER statements through the STOP statement and SAVE the program with the new subroutines in it.

Before you ENTER a file into working storage, check its line numbers to see if they are identical to the line numbers of the program already there. If the line numbers are the same, Business BASIC replaces the existing program state-

ments with the program statements that are being ENTERed. You can use the Business BASIC CLI TYPE command to check the line numbers of a subroutine. For example:

```
* !TYPE subroutine-name.SL
```

The TYPE command displays the subroutine at your terminal. It also displays the colon comments that are included with Business BASIC subroutines to explain how they work. These comments are stripped from the subroutine when it is ENTERed into working storage. You can also use the BASIC CLI PRINT command to print the subroutines with their comments. In addition, two other BASIC CLI commands work together to create a file containing only program comments. These are BLDCOM and PRTCOM. The BASIC CLI commands are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

To execute a subroutine from within a program, use the GOSUB statement or the ON GOSUB statement. Your program can contain several GOSUB statements to the same subroutine. Each subroutine has RETURN statements that return control to the statement immediately following the GOSUB statement.

Writing Business BASIC Subroutines

Business BASIC lets you write your own subroutines or modify the subroutines supplied with the software package. The following list contains guidelines for writing or changing subroutines:

- Always include a RETURN statement. This returns program control to the line after the GOSUB statement. You can have more than one RETURN statement in your subroutine. This enables you to use program logic that has the subroutine end at different places, according to which conditions are met.
- Start your subroutine with a REM statement that describes the subroutine. When a program is LISTed, Business BASIC displays REM comments from the entry point of the subroutine next to the GOSUB statement calling the subroutine. This helps you follow the flow of your program.
- Select line numbers for your subroutine that are outside the range normally used in your programs. This makes it easier to merge your subroutine with other programs by reducing the chance that the subroutine line numbers will overlap with the program line numbers. The subroutines supplied with your Business BASIC software package all use high line numbers since most application programs start with low line numbers. Appendix A lists the line numbers used by these subroutines.
- Use the .SL extension on the filename to identify it as a subroutine. Your subroutine will work without this extension; however, this is the naming convention used with the subroutines supplied by Business BASIC.
- If you want to use your subroutine with other programs, store it in the library directory. RDOS users have access to the library directory without having to specify the directory's name. AOS users need to include the

pathname to the library in the subroutine name when they LIST the subroutine to disk. This means AOS users need to have the pathname to the library directory in their search lists.

Business BASIC lets you nest subroutines up to a depth of eight. Nesting occurs when one subroutine calls another subroutine. If you nest more than eight subroutines or disrupt the logic of a nested subroutine, an error occurs. The GOSUB/RETURN stack can be reset by using the STMA 8 statement/command.

To add a subroutine to your program, follow the steps listed in the previous section, "Using Subroutines."

Errors with Subroutines

When you use subroutines in a program, certain errors can occur; in particular, `ERROR 13 -- LINE NUMBER` and `ERROR 19 -- RETURN - NO GOSUB`.

If a GOSUB statement is used when the subroutine has not been ENTERed, ERROR 13 occurs. The GOSUB refers to a line number that doesn't exist in working storage. This can be prevented by saving the program once its subroutines have been added.

ERROR 19 occurs whenever BASIC executes a subroutine that was not called by a GOSUB statement. This can happen if you write a program or ENTER a program into working storage without issuing a NEW command to clear working storage. A subroutine could have been left in working storage by a previous program. Then, since Business BASIC executes all working storage program statements sequentially, it also executes the subroutine. You can avoid this error by clearing working storage before writing programs and including an END statement in all programs. Both the END and STOP statements halt program execution.

Errors also occur when you do not provide proper values for variables or proper variable names for the subroutine variables. Each subroutine requires specific input variables and returns specific output variables.

You must also supply the proper entry point to the subroutine. Most subroutines have more than one entry point.

Subroutine Example

The following program, GET, uses the Business BASIC supplied subroutine GETCM.SL to create a BASIC CLI command. This program returns the combined values of the switches /B and /C. To use the subroutine, you have to dimension T9$ (GETCM.SL's input variable) and X$ (GETCM.SL's output variable) in your main program. GET has one GOSUB 7550 to initialize the GETCM.SL values. Then, in line 0060, GET loops back to the GOSUB 7500 until S (another of GETCM.SL's output variables) returns with a -1. In line 0050, GET prints the BASIC CLI command and the value of S.

```
* LIST
0010 DIM T9$[512],X$[24]
```

```
0020 GOSUB 7550
0030 GOSUB 7500
0040 IF S=-1 THEN STOP
0050 PRINT X$, TAB(35),S
0060 GOTO 0030

* ENTER "GETCM.SL
* SAVE "GET
* LIST
0010 DIM T9$[512],X$[24]
0020 GOSUB 7550
0030 GOSUB 7500
0040 IF S=-1 THEN STOP
0050 PRINT X$, TAB(35),S
0060 GOTO 0030
7500 REM GETCM.SL
7505 IF T9$[Q9,Q9]= "<255>" THEN GOTO 7540
7510 LET X$=TRUN$(T9$[Q9])
7512 IF X$=" " THEN GOTO 7540
7515 LET Q9=Q9+LEN(X$)
7520 IF Q9>508 THEN STOP
7525 UNPACK L ,T9$[Q9+1],S
7530 LET Q9=Q9+5
7535 RETURN
7540 LET S=-1
7545 RETURN
7550 REM \ INITCM
7552 LET Q9=1
7554 BLOCK READ T9$
7556 RETURN
7559 REM * END GETCM.SL

* !GET/B/C

GET                                                    1610612736

STOP AT 40
```

Since you ENTERed GETCM.SL into the program in working storage and then SAVEd the complete program, you don't have to ENTER the subroutine each time you execute the program. However, when you ENTERed the subroutine, Business BASIC stripped out the colon comments explaining GETCM.SL. To see those, you need to !TYPE the subroutine. (Note: Instead of saving GET with GETCM.SL in it, you could include the statement 0005 ENTER GETCM.SL in your program. This would merge GETCM.SL into your program each time you executed GET.)

For more information on GETCM.SL and other Business BASIC subroutines, see the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.* In addition, the entries in the manual for the screen utilities CSM and SM both contain examples of programs that use subroutines.

Assembly Language Subroutines

You can also use Business BASIC to call assembly language subroutines. Assembly language programs must be put into the Business BASIC interpreter at the time your system is generated.

Assembly language subroutines usually perform tasks that Business BASIC can't do and, as a result, should be executed judiciously. Improper use of assembly language subroutines, system calls, or task calls can crash the system.

More information on assembly language subroutines is included in the *Business BASIC System Manager's Guide.* Information on calling these subroutines is in the *Business BASIC Reference Manual for Commands, Statements, and Functions* under the UCALL entry.

Utilities

Utility programs written in Business BASIC are included in your software package. These programs are designed to help with file processing tasks and maintaining databases. Appendix A lists the utilities supplied with your Business BASIC software package. These utilities are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Using Utilities

Business BASIC provides two types of utilities — those that can be executed in a variety of ways and those that must be executed by using the SWAP command.

In general, you can execute the first kind of utility by entering RUN "utility-name, CHAIN "utility-name, or SWAP "utility-name. These utilities can also be executed through the BASIC CLI by entering either the utility name while in the BASIC CLI or !utility-name while in keyboard mode. When you use the BASIC CLI, it executes the utility by performing a SWAP to the program named in addition to closing any open files.

Some utilities are run-only. Do not use any of the three modes of executing the SWAP command (SWAP "utility-name, !utility-name, or "utility-name) with these utilities. For example, you should only execute the DBGEN utility by entering:

```
* RUN DBGEN
```

The second kind of utility must be run in conjunction with another program. Information is passed to it, and it returns the results through the common area.

When utilities return information to the calling program or use the common area to pass information to programs, they need to be executed with a SWAP. They do not work properly unless they can read the common area with a BLOCK READ statement and interpret its data. If you CHAIN to one of

these utilities, the information passed from the utility to the common area must be retrieved using keyboard mode commands. The following is a list of the SWAP-only utilities:

- FILESORT
- IBUILD
- OPEN
- QFILESORT
- SIZE
- TBUILD
- XBUILD

You also need to SWAP to a utility if that utility works with open files. This is the case with the LOCKS utility. LOCKS, like some other utilities, is structured internally so that it checks to see how it was executed. If it was not executed by a SWAP command, LOCKS issues a NEW command to close all open files. You cannot execute these utilities through the BASIC CLI because the BASIC CLI issues a CLOSE command to close all open files.

Common Area

The common area is a 512-byte memory location used to store information and pass it from one program to a subsequently running program. You can access the common area directly by using the GETCM.SL subroutine or the BLOCK READ and BLOCK WRITE commands.

Each user has only one common area. Data cannot be sent into another user's common area. Use the BLOCK WRITE statement/command to put data into the common area and the BLOCK READ statement/command to retrieve that data.

With BLOCK WRITE and BLOCK READ, you can use a string to hold the information going into or coming from the common area. The string must be dimensioned to at least 512 bytes because BLOCK WRITE and BLOCK READ each transfer a 512-byte block.

Each SWAP-only utility uses a string to access the common area. This string must be built according to the instructions given with that utility. When a string of information from a utility is retrieved from the common area, you can use the UNPACK statement/command or the ASC function to extract the binary values from the string.

Some utilities, like OPEN, require literal filenames and numbers in the string. Others, like FILESORT, require binary values in the string. Use the CHR$ function to put a binary value in a string.

You can also use an array with BLOCK READ and BLOCK WRITE for passing data to and from the common area; again, the array must be able to hold at least 512 bytes. With a triple precision system, each array element holds 6 bytes. Therefore, the array used for transferring information to and from the common area must have at least 86 elements (516 bytes), since 85 elements use only 510 bytes. When this information is retrieved from the common area, the last 2 bytes will contain meaningless information.

In a double precision system, each array element holds 4 bytes; therefore, the array used for passing information to and from the common area must have at least 128 elements (512 bytes).

End of Chapter

Chapter 5
File Overview

Business BASIC supports the following categories of files:

- Operating system files.
- Business BASIC files (i.e., operating system files containing an embedded Business BASIC structure).
- INFOS II files (i.e., operating system files containing an embedded INFOS II structure).

Business BASIC places restrictions on naming files. Any file, however, that follows the Business BASIC filename conventions can be used by Business BASIC programs and can be accessed using one of these three methods:

- Sequential access.
- Direct or random access.
- Indexed sequential access.

This chapter provides an overview of files and file handling under Business BASIC. The chapter discusses:

- Operating system files.
- Filename conventions required by Business BASIC.
- File access.
- Business BASIC system files.

Information on INFOS II files is contained in chapter 7.

The information in this chapter is general; in some cases you will need to consult the Business BASIC reference manuals to determine which commands, utilities, and subroutines work with your Business BASIC system. Business BASIC supports two database structures, and some commands, utilities, and subroutines only work with one of the structures. Information on file structures and file access methods specific to these database structures is in chapter 6.

Operating System Files

Operating system files are disk files. They differ according to their internal organization; i.e., how the operating system stores them on disk. RDOS systems support three internal file formats; AOS systems support one internal file format.

RDOS Files

Under RDOS, each operating system file is organized in one of the following ways:

- Sequential.
- Random.
- Contiguous.

The next three sections describe these methods of file organization.

Sequential File Organization

Sequential files use a flexible file structure but a very rigid access method.

In a sequential file, the system maintains a series of pointers to each block of file information. RDOS stores information on the disk in blocks of 512 bytes. The last 2 bytes of each block contain a pointer to the next block of data in the file. These blocks can be anywhere on the disk; they do not have to be adjacent blocks.

To build a sequential file, the system appropriates the next available disk block when it needs space and constructs a pointer to that block. Figure 5-1 illustrates how a sequentially organized file looks on the disk.

The disadvantage to sequential files is that they permit only beginning-to-end input and output access. After processing any given block of a sequential file, the system can step either to the previous block or to the next block in the series. For example, to access a record in the tenth block of a sequential file after accessing a record in the first block, the system must read all the intervening blocks. This makes data access for sequential files more time-consuming than data access for either random or contiguous files. The advantage to sequential files is that, since they appropriate space as it is needed, you can always append records to a file. Also, you can have variable length records.

Random File Organization

Random file organization provides a flexible file structure and easily accessible data.

With random organization, RDOS maintains an index file that contains pointers to data blocks in the random file. To find a record, the operating system gets the location of the block holding the record from the index file. The system then accesses the block directly, resulting in faster access than a sequential file provides.

Like a sequential file, a random file allocates space as it needs it, so you can always append records to it. However, maintaining the index file means that random files require more disk space than sequential files.

Figure 5-2 shows how a random file works.

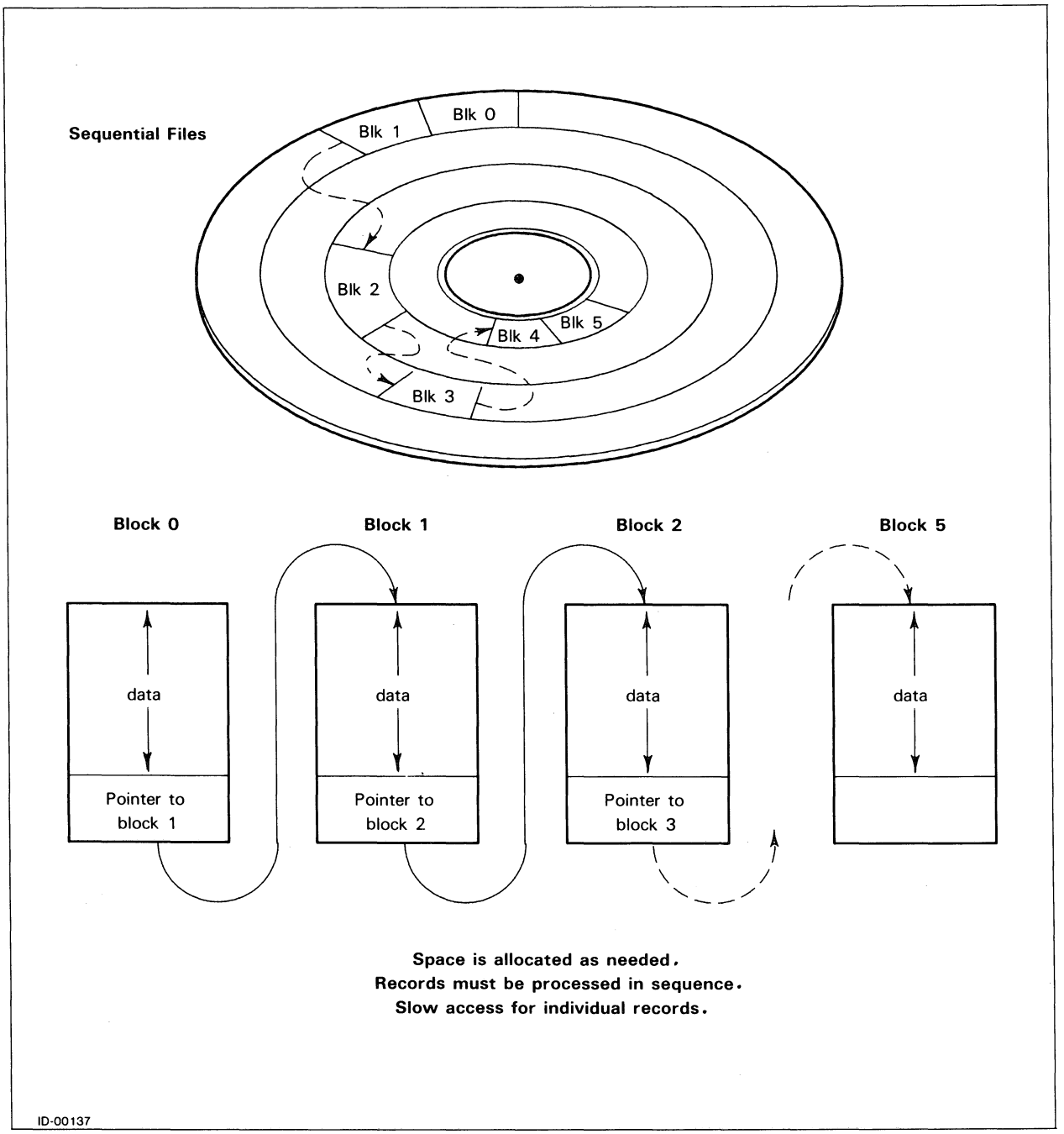

Figure 5-1. Format of an RDOS Sequential File

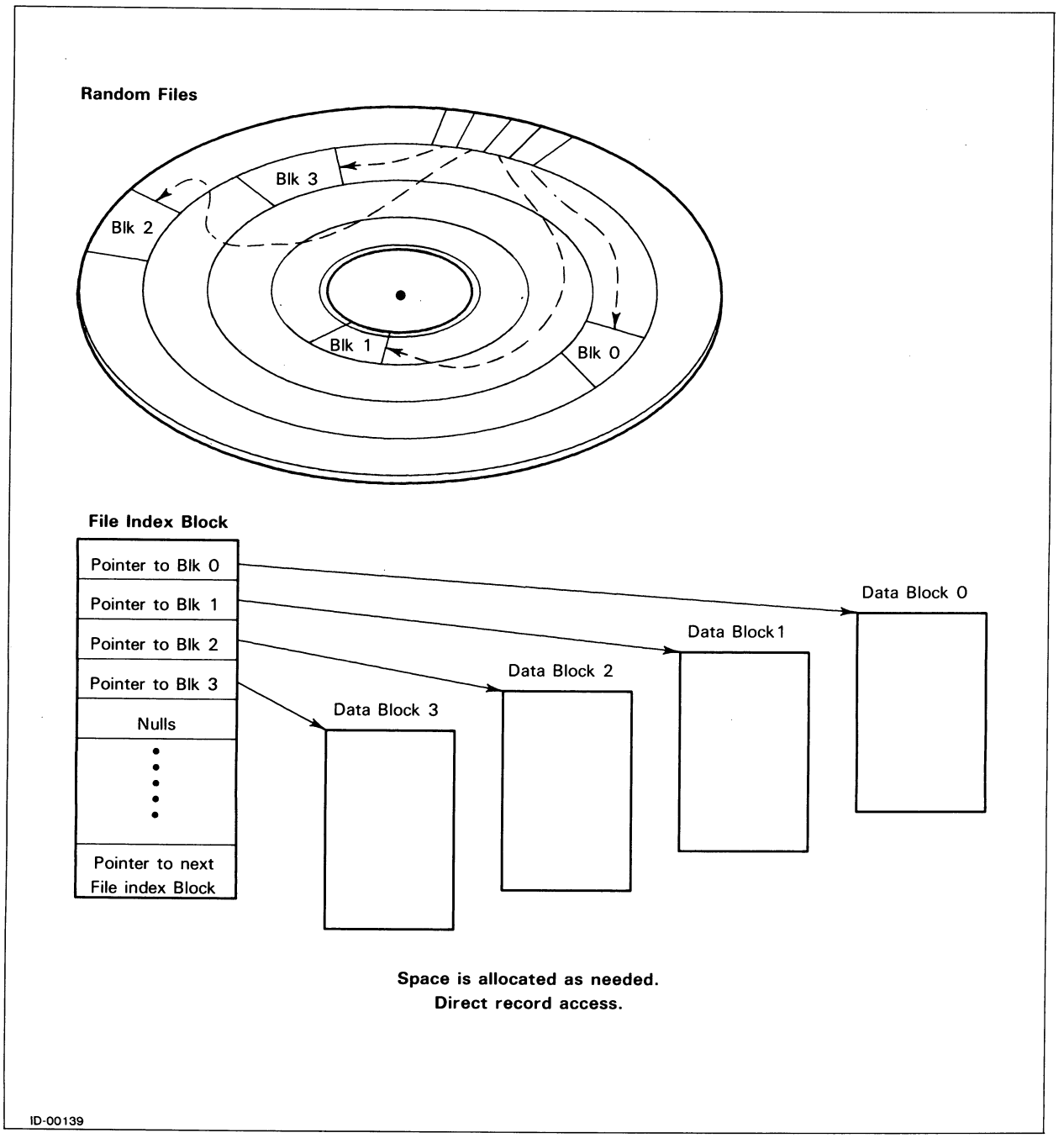

Figure 5-2. Format of an RDOS Random File

Contiguous File Organization

Contiguous files have a rigid structure, but they provide the fastest access to data of the three RDOS file formats.

Contiguous files consist of a fixed number of disk blocks that are physically adjacent on the disk. This allows the operating system to calculate the location of a record and access it directly. Figure 5-3 provides an example of contiguous file organization.

The disadvantage to contiguous files is that you must allocate all the disk space that the file will require when you create the file. These files occupy a fixed

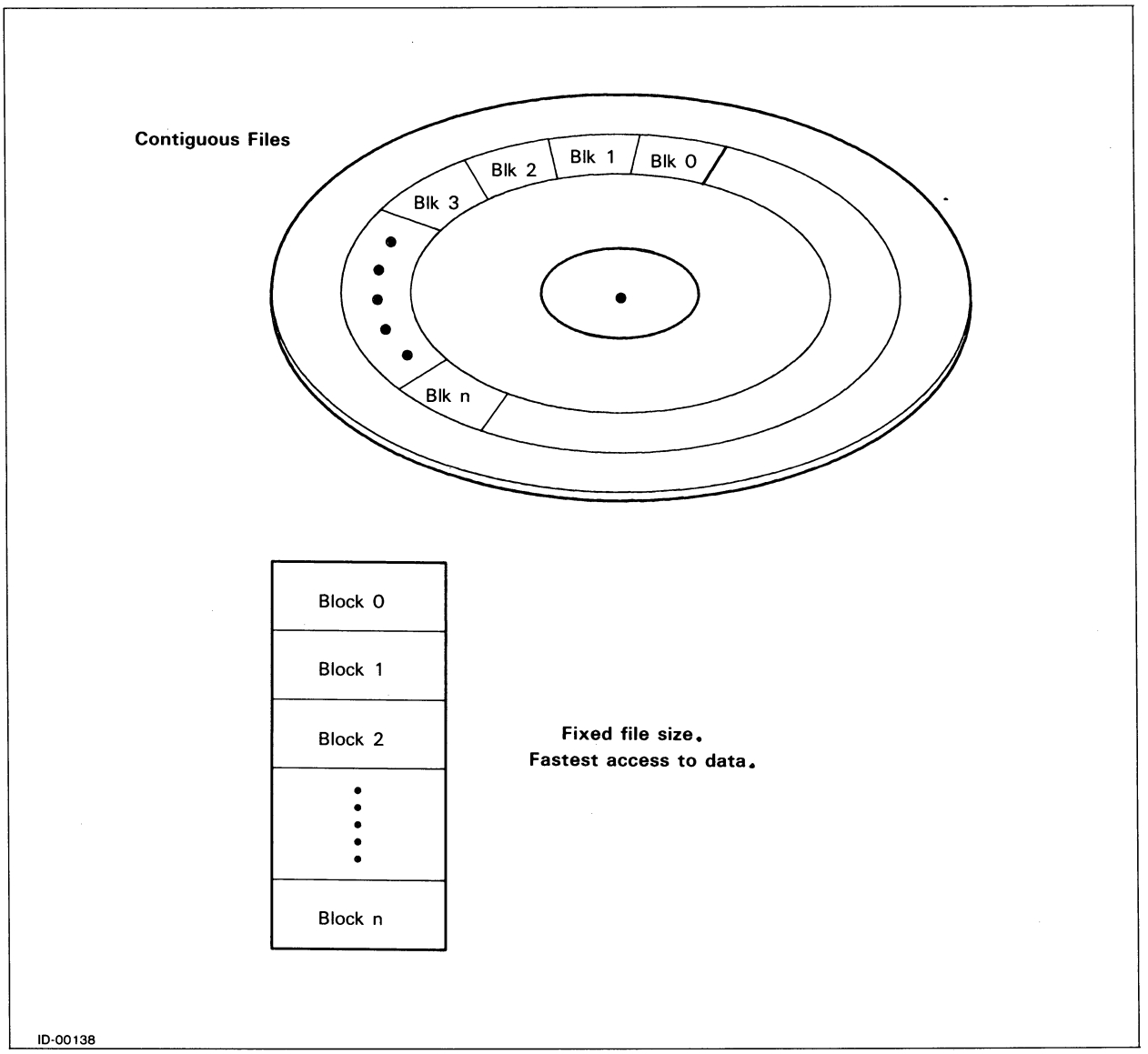

Figure 5-3. Format of an RDOS Contiguous File

series of disk blocks, so you cannot expand or shrink the files. In addition, a contiguous file can be created only when the required number of contiguous disk blocks is available.

AOS Files

AOS builds a disk file from one or more 512-byte disk blocks. The system uses a hierarchical index to connect the disk blocks within files. These files are similar to the RDOS random organization files, except that AOS uses a file element size.

The file element is the basic unit of storage. It consists of one or more contiguous blocks (blocks with sequential physical addresses). You can specify the file element size when you create a file using the BASIC CLI command CCONT, or you can use the system default value. On AOS/VS systems, the default file element size is four; you can change this default value when you generate your system. On AOS systems, the default value is one. 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.

If you create an AOS file with an element size of two, the file initially contains two contiguous blocks (see figure 5-4). If file storage requirements exceed the two blocks, the system allocates an index block that contains the addresses of each of the file elements. The index element provides no data storage for itself. Each index element is one disk block (not one file element size) in length.

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

The system handles the storage and retrieval of information; however, you should consider the trade-offs of changing the file element sizes when you design your 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.

Business BASIC File System

The Business BASIC file system requires you to follow certain conventions regarding filenames. If you follow these conventions, Business BASIC lets you create simple disk files and access them. It also lets you create and access file types that are specific to Business BASIC.

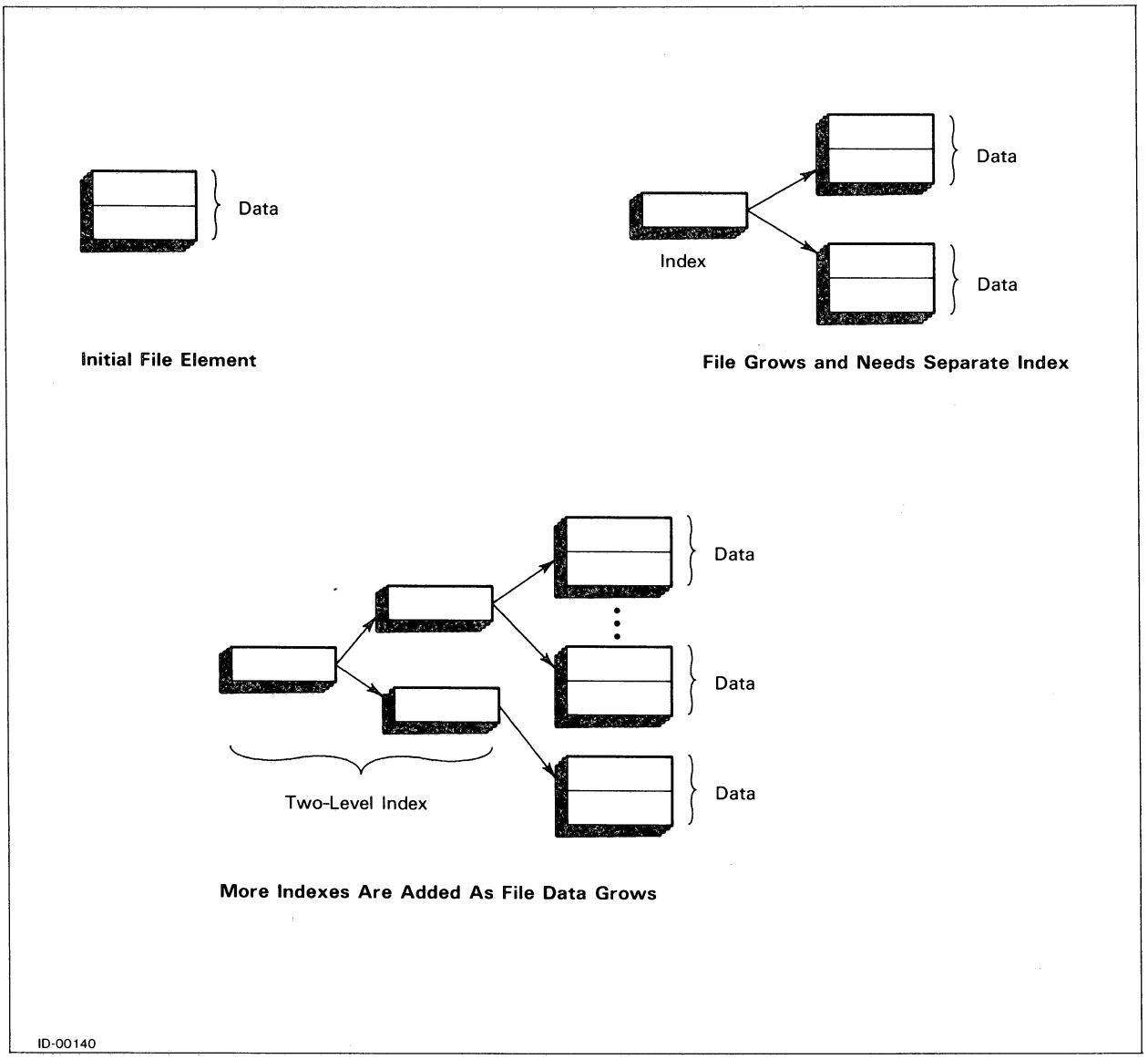

Figure 5-4. Stages in AOS File Growth

Filename Conventions

The Business BASIC filename conventions were originally imposed under RDOS, which was developed prior to AOS. Under RDOS, the filename conventions are:

- The filename must not exceed 10 characters in length. The filename and its extension must not exceed 13 characters. A filename extension consists of a period (.) and up to two alphanumeric characters. RDOS Business BASIC truncates filenames that exceed 13 characters.
- The filename can consist of uppercase and lowercase letters (these are equivalent under RDOS), numbers, and a dollar sign ($). The period can be used only to indicate a filename extension.

Under AOS, the filename conventions are:

- Filenames have a maximum length of 31 characters. However, many Business BASIC utilities restrict you to 10-character filenames followed by an extension consisting of a period and up to two characters. This is to maintain compatibility with RDOS.
- Filenames can consist of uppercase and lowercase letters (these are equivalent under AOS), numbers, period (.), dollar sign ($), question mark (?), and underscore (_).

You can use filename extensions to help organize your files. For example, listing files are often indicated by using a .LS extension. In addition, your Business BASIC software package follows certain conventions in using extensions. Table 5-1 lists these extensions and their meanings.

Extension	Meaning
.BA	A utility source file listing that has comments. The .BA files are contained in the $DOC directory, which was supplied with your Business BASIC software.
.DB	A database file in the logical file database structure. Business BASIC appends this extension to the filename you supply.
.LS	A listing file.
.SF	A file used with the Business BASIC Data Dictionary.
.SL	A Business BASIC subroutine file in ASCII format.
.Sn	A screen file. This is a file used with the Conversational Screen Maintenance (CSM) utility or the Screen Management (SM) utility. The n represents the terminal type.
.SP	A commented utility source file module that is accessed by one or more .BA files. .SP files are contained in the $DOC directory, which was supplied with your Business BASIC software.
.TB	A table file. Used with the File Maintenance (FM) utility.
.VL	A volume label file in the logical file database structure. Business BASIC appends this extension to the filename you supply.

Table 5-1. Filename Extensions

Creating Simple Disk Files

There are four ways to create simple disk files with Business BASIC:

- The BASIC CLI command CCONT.
- The BASIC CLI command CRAND.
- The BASIC CLI command CREATE.
- The OPEN FILE command.

Under RDOS, the CCONT command sets up a contiguous file; the CRAND command, a random file; and the CREATE command, a sequential file. These commands set up random files under AOS.

The OPEN FILE command creates files in your directory only if you open the file in mode 0, 1, or 2. These modes also specify the type file you have.

The BASIC CLI commands are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.* The OPEN FILE command is discussed in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

File Access

Business BASIC file access involves several steps. You need to open the files you are going to use as well as set up the variables you will use in transferring information between files. With numeric variables, the amount of data you can transfer depends upon the variable's precision, while with string variables, the amount is determined by the string's dimension.

When you open a file, you must associate its name with a channel number and specify an access mode.

The channel number represents the line of communication from your working storage area to the data file or device. Channel numbers can range from 0 to 16; however, channel 16 always refers to the terminal. A channel number can be associated with only one file at a time. The channel number cannot be used again until the file is closed with a CLOSE FILE statement. Once a file has a channel number, you use the channel number instead of the filename when reading from or writing to a file. Any file assigned to channel 16 reads data from and writes data to the terminal.

The file access mode is determined in part by the file organization (i.e., sequential, random, or contiguous). You specify a file access mode with the OPEN FILE statement. When a data file uses sequential file organization, you must open it in sequential access mode. You can open files using random or contiguous organization in either random or sequential access mode.

To access data in the file, use the file pointer. It moves each time you access a record. At the end of the operation, the file pointer points to the byte immediately after the last byte that was read or written. (You can use the GPOS, or get position, function to check the position of the file pointer.)

With files opened in random mode, use the POSITION FILE statement to move the file pointer. POSITION FILE lets you go directly to the record you want. This statement does not work with sequentially opened files, since sequential access mode does not allow direct access of records.

To access a record directly, use the record's relative position to place the file pointer. With fixed-length records, you calculate the record's relative position as:

```
record number * record length
```

To use the sequential access method with data files that are randomly or contiguously organized, position the file pointer to byte 0 of the file. Then process the records according to their order in the file. The file pointer moves with each read or write, so it is automatically positioned and ready to access the next record. You can also position the file pointer to any record in the file and read the file sequentially from that point.

With sequential files, you can use the end of file function (EOF) to determine when the data has ended.

Since random access mode lets you access files faster, it is usually used for files containing records that require frequent updating, such as a store billing program. Sequential mode is used for files to which new records are constantly being added; for example, a list of company products.

Table 5-2 lists the commands provided for file input and output. These commands are explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

Keyword	Purpose
BLOCK READ FILE	Get data from a file in multiples of 512 bytes.
BLOCK WRITE FILE	Place data in a file in multiples of 512 bytes.
CLOSE	Close all opened files.
CLOSE FILE	Close a specific file.
EOF	Check for end of file.
GPOS	Determine the current position of the file pointer.
INPUT FILE	Get ASCII data from a file.
INPUT FILE USING	Get ASCII data from a file and allow an error and a terminator trap.
LOPEN FILE	Open a database file.
LREAD FILE	Get binary data from a record.
LWRITE FILE	Place binary data in a record.
OPEN FILE	Open a file.
POSITION FILE	Position the file pointer.
PRINT FILE	Send ASCII data to a file, terminal, or device.
PRINT FILE USING	Send ASCII data according to a format to a file, terminal, or device.
READ FILE	Get binary data from a file.
WRITE FILE	Output binary data to a file.

Table 5-2. File Input and Output Commands

File Types

Since Business BASIC supports operating system files and files specific to the Business BASIC system, files used on a Business BASIC system fall into one of three file types:

- Simple disk files (including direct random format data files).
- Linked-available-record format data files.
- Index files.

These file types give you the flexibility to select a method of data file organization that is appropriate for your needs — from simple to complex. With the exception of RDOS sequential files, you can use any access method with these files. The last two categories (linked-available-record files and index files) both contain embedded Business BASIC structure that tells the BASIC interpreter how the file is organized.

Simple Disk Files

Simple disk files are flat files or operating system files. These files consist of a stream of bytes that you can access at any point. The internal organization of the files can be sequential, random, or contiguous. You must access RDOS sequentially organized files using sequential access; however, you can use sequential or direct random access with files that are organized randomly or contiguously.

If you are using direct random access with your files, then you handle all record assignments. These files must contain fixed-length records. You can position directly to a specific record.

Linked-Available-Record Files

The linked-available-record format uses dynamic record allocation, which allows a file to reuse space left by deleted records. Each deleted record contains a pointer to the next available record in the file. Use either the LFU utility or the INITFILE utility to create linked-available-record files.

The first record (record 0) in this type of file is reserved by Business BASIC and holds information on the next available record. The rest of the records contain user data.

Business BASIC uses the first 2 bytes of each record to store information on whether a record is active or deleted. Then, for each active record after record 0, data is stored in bytes 2 through the end of the record. In a deleted record, bytes 2-5 point to the next record available to receive data, thus creating the deleted-record chain.

Record 0 is always the same size as the other records in the file. The minimum record size for a linked-available-record file is 6 bytes. If the record size is less than 6 bytes, the information normally contained in bytes 2-5 of record 0 will be corrupt. Table 5-3 describes the contents of record 0.

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.

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 5-3. Contents of Record 0 of a Linked-Available-Record File

The format of an active data record is shown in Table 5-4, and the format of a deleted data record is shown in Table 5-5.

Bytes	Description
0-1	Status flag (greater than 0 when the record is active).
2-end	User data.

Table 5-4. An Active Data Record in a Linked-Available-Record File

Bytes	Description
0-1	Status flag (usually 0, though it can also be less than 0).
2-5	Record number of next deleted record in deleted chain or -1 if last value in chain.
6-end	Unused; it contains old data from when record was active.

Table 5-5. A Deleted Data Record in a Linked-Available-Record File

If the deleted-record chain for your linked-available-record file is destroyed, you can rebuild it using either the LRELINK utility or the RELINK utility. You should also run one of these utilities if a system crash occurs while a linked-available-record format file is being updated. These utilities are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

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; instead, it contains keys that point to entries in a data file.

Business BASIC uses the indexed sequential access method (ISAM) of working with index files. Usually, an index file and its corresponding data file are re-

ferred to collectively as an ISAM file. In Business BASIC, an ISAM file consists of a data file and one or more index files. (The DBGEN and FM utilities limit you to three index files per data file.)

Some additional characteristics of index files include:

- More than one index file can point to the same data file.
- Duplicate key values are permitted, but only when you specify that they are allowed. These are keys with identical values that point to different data file entries.
- The key values are maintained in sorted order, thus allowing you to read a data file sequentially without going through the time-consuming task of sorting the data file.
- The maximum key size is 122 bytes.
- The maximum index file size is 65535 blocks (0-65534).
- Unlike data files, index files are organized internally into fixed-length blocks, not records.

Creating Index Files

There are three major steps to setting up an index:

1) Calculate the index file parameters.

2) Create and initialize the index.

3) Build the index by adding keys.

Business BASIC provides several utilities and subroutines for you to use in performing these steps.

INDEXCALC computes and prints information for the index, including the maximum number of keys per index block, the number of keys per block with the user-specified blocking factor, the number of blocks at level 0 (one, two, etc. for as many levels as needed), the number of blocks (either 512 or 2048 bytes per block), the number of sectors in the index, and the number of sectors in the associated data file.

The LFU and INITFILE utilities create and initialize an index file or a linked-available-record file by writing the first record with header information.

IBUILD creates an index file from a sorted data file, a sorted tag file, or an index file.

TBUILD creates a temporary (tag) file from a linked-available-record data file or an index file. A tag file is sorted faster than a data file. The final index file is maintained in sorted order.

XBUILD creates an index file with a blocking factor of 50%. Use INDEXBLD or IBUILD to create an index file with a different blocking factor.

LINDEXBLD and INDEXBLD create an index file from an unsorted data file, tag file, or another index file, or re-create an index file. (Each of these programs use IBUILD, XBUILD, and TBUILD, depending on the options you select when you execute the program.)

IREBLD rebuilds an index file.

LINITINDEX.SL and INITINDEX.SL initialize an index file but do not initialize a data file. If you use these subroutines with existing index files, the index files are initialized and any data that had been in them is lost.

These utilities and subroutines are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Index File Formats — Index files differ from data files in their internal makeup. Instead of being organized in records, they contain fixed-length blocks of 512 bytes (RDOS and AOS) or 2048 bytes (AOS only) each. You specify the index block size when you use the LFU utility or the INITFILE utility to create the index.

The first block, block 0, is reserved as a header block that describes the index file. Business BASIC sets up this block when you create the index. Table 5-6 describes the format of block 0. The remaining index file blocks contain key and pointer information. The format for these blocks is illustrated in Table 5-7.

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. These determine whether the index allows 512-byte or 2048-byte blocks and whether duplicate keys are allowed. The flags can have these meanings: 0 512-byte blocks, no duplicates 1 512-byte blocks, duplicates allowed 2 2048-byte blocks, no duplicates 3 2048-byte blocks, duplicates allowed
14-end	Reserved.

Table 5-6. Contents of Block 0 of an Index File

Bytes	Description
0-1	Number of entries in the block.
2-3	Number of the next block in sequence.
If this is a 512-byte block index then:	
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.
If this is a 2048-byte block index then:	
4-5	Reserved
6-7	Update counter
8-end	Key entries. These consist of a key and a record pointer to the data file, if duplicates are not allowed. If duplicates are allowed, the key entry consists of a key, an occurrence number, and a record pointer to the data file. Keys are fixed length in size and must be an even number of bytes. The occurrence number requires 2 bytes and is used to distinguish between duplicate keys. Record pointers require 4 bytes and must be a positive signed number.

Table 5-7. Format of a Block Containing Keys for an Index File

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. A block always contains at least one available entry. If an addition uses the last available entry, the block is divided into two new blocks. When a block with an odd number of keys is split, the extra key is in the original block.

If you add keys using KADD and the block needs to split, the blocking factor is automatically 50 percent.

Using Index Files

Business BASIC provides four K commands for working with existing index files:

- KADD adds a key.
- KFIND finds a key.
- KNEXT finds the next key.
- KDEL deletes a key.

These statements use block 0 of an index file.

KADD adds a key entry (string and record pointer) to the index file described in a 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 attempt to add one, KADD returns the record pointer with a value of 0 and does not add the new key.

KFIND searches the index file 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 the first key with a value 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 locates the key ABCA. If KFIND cannot find a key value equal to or greater than the key you supply, it returns a value of 0 for the record pointer.

KNEXT locates the next key in sequence. You must execute a KFIND before your first KNEXT. After that, just execute KNEXT. 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. For example, use KFIND with a null key to return the first key in the index. Then use KNEXT repeatedly to reach the key you want. You can also use KNEXT to find all occurrences of duplicate keys.

KDEL deletes keys from an index file. If KDEL does not find a match for the key you entered, it returns 0 for the record pointer. If it does find a key, it returns the record pointer of the deleted key.

These commands are described in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

When you add keys to and delete keys from an index file, the multi-leveled ISAM key structure dynamically expands, but it does not dynamically contract. Business BASIC reuses space freed by deleted keys; however, the index file can become full, even when there appears to be room for the key entries. This occurs when a large number of keys are deleted and then replaced by new keys with a different range of values. When this happens, rebuild the index file. Several Business BASIC utilities rebuild index files. These utilities accept an unsorted data file, a tag file, or another index as input files for the new index.

When a data file is used as the input file, the location of a key field must be the same in each data record, and there can only be one key per record per index. This is also true when building tag files from a data file.

If the data file contains multiple record types with a separate index for each type, then the data file cannot be used as an input file with the ISAM utilities In this case, use the old index file as the input file unless the structure of the old index file is corrupt. If the structure is corrupt, you need to write a program to rebuild the index.

You also need to write your own program if you want an index that contains several keys for each data record. INDEXBLD, for example, asks for the number of fields in the key and the starting byte locations within the data record for each field in the key. It writes one key in the index for each record, regardless of type, and it uses the same locations within each data record. FM is the

only Business BASIC utility that supports multiple record types in one data file with separate indexes for each type.

Two utilities that you can use once you have built an index are INDEXPRT and INDEXVRFY. These help you check an existing index.

Index File Example

The following example sets up a small database that uses an index file. The data file is EMPDATA and the index file is EMPINDX. To set up EMPINDX and EMPDATA, first run INDEXCALC to determine information needed to create the index file, including the number of keys per index block and the number of blocks in the index, and then to determine the number of sectors needed by EMPDATA. After you get this information, run INITFILE to create these two files.

The dialog for INDEXCALC, which calculates numbers for EMPINDX and EMPDATA files, is:

```
* RUN "INDEXCALC
INDEXCALC VERSION X.XX

BYTES PER KEY : 4
BYTES PER DATA RECORD : 25
MAXIMUM NUMBER OF DATA RECORDS : 100
INDEX BLOCKING FACTOR (% PERCENT) [50]: 50
INDEX BLOCK SIZE (512 or 2048) [512]: 512
DUPLICATE KEYS ALLOWED? (Y or N) [N]: N

63 MAXIMUM KEYS PER INDEX BLOCK
32 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
4 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
6 BLOCKS ( 512 bytes each) IN INDEX
6 SECTORS IN INDEX
5 SECTORS IN DATA FILE

CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y or N) [N]: N
```

The dialogue for INITFILE, which creates and initializes EMPINDX and EMPDATA, is:

```
* RUN "INITFILE
INDEX (0), DATA (1), STOP(2) [0]: 0
SUB FILE NAME EMPINDX
FILE NOT IN PARAM FILE!
DO YOU WISH TO ADD? (Y OR N) [Y]: N
MASTER FILE NAME: EMPINDX
INDEX BLOCK SIZE (512 or 2048) [512]: 512
BYTE OFFSET TO SUB FILE: 0
MAXIMUM NUMBER OF INDEX BLOCKS: 6
BYTES PER KEY: 4
BLOCKING FACTOR (% PERCENT) [50]: 50
```

```
DUPLICATE KEYS ALLOWED? (Y OR N) [N]: N
INDEX (0), DATA (1), STOP(2) [0]: 1
SUB FILE NAME EMPDATA
FILE NOT IN PARAM FILE!
DO YOU WISH TO ADD? (Y OR N) [Y]: N
MASTER FILE NAME: EMPDATA
BYTE OFFSET TO SUB FILE: 0
BYTES PER DATA RECORD: 25
MAXIMUM NUMBER OF DATA RECORDS: 100
SHOULD FILE BE NULL FILLED: N
INDEX (0), DATA (1), STOP(2) [0]: 2
```

The following program uses the two files you have set up to add information on employees at Widget Supply Co. As data is added to EMPDATA, keys are added to EMPINDX.

```
* LIST
0010 DIM NAME$[25],X$[512],KEY$[4],BUF$[544],DESC$[18],C1[1,3]
0020 ON ERR THEN GOTO 9900
0030 RSIZE=25 \ R1=0 \ IFILE=1 \ DFILE=2
0499 REM * Control module
0500 GOSUB 1000 : * Open Files
0600 GOSUB 2000 : * Input data & write data records & keys
0999 END
1000 REM * Open Files
1010 CLOSE
1020 OPEN FILE(IFILE,5), "EMPINDX"
1030 OPEN FILE(DFILE,5), "EMPDATA"
1040 DESC$=CHR$(IFILE,2),CHR$(0,4),CHR$(0,2), "EMPINDX",FILL$(0)
1190 RETURN
2000 REM * Input data & write data records & keys
2010 R1=R 1+1
2020 IF R1>100 THEN 2900
2030 GOSUB 6000 : * Input screen
2040 INPUT USING "",@(10,41),@(-10,6),EMPNO
2050 IF EMPNO=0 THEN GOTO 2900 : * Exit
2060 INPUT USING "",@(12,41),NAME$
2070 LET KEY$=CHR$(EMPNO,4)
2080 POSITION FILE(DFILE,RSIZE*R1)
2090 WRITE FILE[DFILE],NAME$
2100 KADD DESC$,BUF$,KEY$,R1
2110 REM * Force error if any problem on KADD
2120 IF R1<=0 THEN STMA 19,67
2130 GOTO 2000 : * Input data & write data records & keys
2900 REM * Exit
2910 PRINT @(22,1)
2950 RETURN
6000 REM * Input screen
6010 PRINT @(-30);@(1,18);"W I D G E T  S U P P L Y  C O M P A N Y"
6020 PRINT @(3,28);"Employee Information"
6030 PRINT @(10,2);"Employee Number:"
```

```
6040 PRINT @(12,2):"Name:"
6090 RETURN
9900 REM * Error handler
9910 PRINT "<7> ** Error at line";SYS(20);"-";
9920 IF SYS(7)<>-60 THEN LET X$=ERM$(SYS(7)) ELSE X$=AERM$(SYS(31))
9930 PRINT X$
9940 END
```

Logical Files and Subfiles

Files in the Business BASIC file structure can be physical (disk) files or logical subsections of a physical file. These subsections are called logical files or subfiles. The distinction between the two terms is that logical files are used with the Business BASIC logical database structure and subfiles are used with the PARAM database structure (database structures are discussed in chapter 6).

The advantage to using logical files and subfiles is that by dividing a physical file into subsections, Business BASIC lets you simultaneously open an unlimited number of files in the same program. Business BASIC restricts the number of physical files you can open in one program to 16. Each logical file or subfile has a fixed size and begins where the previous one ends.

Use the LFU utility or the INITFILE utility to create these files.

End of Chapter

Chapter 6
Database Structures in Business BASIC

Business BASIC supports two database structures — the logical file database structure and the PARAM file database structure. These structures increase the number of files you can access from one program by letting you set up files that are subsections of a physical (disk) file. The subsections are called logical files or subfiles. Business BASIC permits you to open only 16 physical files (channels 0-15) simultaneously in a BASIC program; however, you can open an unlimited number of logical files or subfiles simultaneously in a program. Since the operating system does not recognize logical files or subfiles, the database structures catalog the logical files and the subfiles by noting their names, their locations within the physical file, the size of their records, and the maximum number of records they contain. You use this information to access these files. To distinguish between the two database structures, this manual uses the terms "database file" and "logical file" only with the logical structure and "master file" and "subfile" only with the PARAM structure.

While the logical structure and the PARAM structure both perform the same function, not all Business BASIC features work with both structures (i.e., some work only with the logical structure while others work only with the PARAM structure). Both structures, however, support data and index files. The data files can use either direct-access format or linked-available-record format while the index files provide Indexed Sequential Access Method (ISAM) for data records.

You can create files that are not in either database structure, but not all Business BASIC features will work with these files.

This chapter describes the two Business BASIC database structures and how to set up and use files with them. The logical file database structure is more recent than the PARAM structure, and some of its features are more efficient and easier to use than the PARAM features. For example, the logical structure computes the byte offset for each record in a logical file; the PARAM structure does not. If you are preparing to set up a database, use the logical database structure. If you are already using the PARAM structure, you should consider whether it's feasible for you to use the PARAMCON utility to convert your database to a logical database. Switching database structures requires some program modifications, such as using the LREAD statement instead of the READ statement.

Logical File Database Structure

The logical file database structure consists of a file set made up of a database file (indicated by a .DB extension) and a volume label file (indicated by a .VL extension). Both the .DB file and the .VL file are physical files. The .DB file contains the actual data. The .VL file contains information on each logical file in the .DB file and maps each logical file to the .DB file. Logical files appear on disk as links to the volume label file.

If you had a simple logical file database named CUST that contained a data file with two indexes, you would have on disk the following two physical files:

```
CUST.DB     (the database file)
CUST.VL     (the volume label file)
```

and the following three logical files:

```
CUSTOMER    (the data file)
CUSTI1      (an index file)
CUSTI2      (an index file)
```

Each of the three logical files is linked to the volume label file, CUST.VL. When you execute a program using these files, the information in the volume label file is copied into the logical file table string (LFTABL$). Both the volume label file and LFTABL$ are discussed later in this chapter.

Creating a Logical File Database

Setting up a database involves the following steps:

1) Design your database so that you know the record size and the number of records you want in the data file.

2) Execute the INDEXCALC utility to determine the number of keys per index block and the number of blocks (sectors) needed for the index file and the number of sectors needed for the data file.

3) Create the logical and physical files using the Logical File Utility (LFU).

To use the information in the database, you need to:

1) Dimension the string variable LFTABL$ and fill it with nulls to a length of at least 26 times the highest logical file number to be used.

2) Use the LOPEN statement to open your logical files.

3) Access the files by using the input/output statements listed in Table 6-4.

An example of setting up a database and a program that uses that database is in appendix C.

Logical Files

You can define your logical files as one of three types:

D Direct random file. The user handles all record assignments.

L Linked-available-record file. Record allocation assignments are made dynamically by Business BASIC.

I Index file. These files are maintained via the ISAM statements.

Logical data files are allocated in 512-byte increments. Index files are allocated in either 512-byte or 2048-byte increments. The increment used depends on the index block size you enter when you LFU LCREATE the index. Only AOS supports 2048-byte index blocks.

Volume Label File Format

The volume label file contains records that describe the size and location of the logical files in the database file. Each record is 32 bytes long. Table 6-1 describes the contents of a volume label file record.

Bytes	Description
0-9	Name of the logical file.
10-12	Starting sector number of a logical file in the database file. The formula for calculating the starting sector is:
	Maximum starting sector number + ((Last valid record # + 1) * record length + 511) / 512 <= 4194303
13-14	Record length in bytes.
15	File type (D, L, I).
16-19	Last valid record number of logical file.
20-21	Revision mark.
22-31	Unused.

Table 6-1. Contents of a Volume Label File Record

You use the LFU utility to maintain the volume label file. When you create logical files with LFU, the utility checks the information in the volume label file to determine what file space is available. If you delete a logical file using LFU, the utility places *DEL in the field on the volume label file where the logical name should appear. LFU reuses this space if another logical file of the same size is created.

LFU consists of a series of commands that let you perform these maintenance tasks. The commands are listed in Table 6-2 and are explained under the LFU entry in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Command	Function
LCREATE	Creates a logical file of type D, L, or I.
LDELETE	Deletes a logical file.
LINIT	Initializes a type D, L, or I logical file.
LLIST	Displays the type, location in the database file, size in blocks (sectors), record length, last valid record number, and size in bytes of the logical file.
LRENAME	Renames a logical file.
PCREATE	Creates physical database and volume label files associated with a logical database file set.
PDELETE	Deletes the database and volume label file.
PLIST	Displays information on the logical files within a database file.
PRENAME	Changes the names of the database and volume label files.
STOP	Terminates LFU.

Table 6-2. LFU Command Summary

Logical File Table (LFTABL$)

The logical file table string (LFTABL$) is a string variable you dimension that stores the definitions of all the logical files opened with the LOPEN statement. The information in LFTABL$ is used by the logical input/output statements.

LFTABL$ consists of a series of 26-byte records, where each record holds information on a logical file. The LOPEN statement places the logical file definitions in LFTABL$. LOPEN gets the logical file characteristics from the volume label file.

Since LOPEN places information in LFTABL$, you must dimension LFTABL$ and fill it with nulls to a length at least 26 times the highest logical file number before you use an LOPEN FILE statement in your program. Once a file has an entry in LFTABL$, you can refer to it with the input/output statements listed in Table 6-4.

The LFDATA.SL subroutine enables you to access information in LFTABL$; however, you should not change this information. Changing data in LFTABL$ can cause the logical input/output statements to perform incorrectly.

Table 6-3 describes the contents of an LFTABL$ record.

Bytes	Description
1-2	Channel number on which the file was opened with LOPEN.
3-6	Starting byte.
7-8	Flags. These are set when you LOPEN the file.
9-18	Name of the logical file.
19-20	Record length in bytes of the logical file.
21-24	Last valid record number of the logical file.
25	Record type (D, L, or I).
26	Reserved.

Table 6-3. Contents of an LFTABL$ Record

Logical File Input and Output

Business BASIC provides several commands (i.e., commands, statements, and subroutines) that perform input and output operations on logical files. Some of the commands are tailored to the logical file database while others can be used with both databases. Commands that work only with the logical file database require you to open the file using the LOPEN statement.

To perform input and output operations in your program:

- First OPEN all the files that are not part of the logical database structure.
- Then LOPEN the files in the logical database structure.

You open files in this order because the OPEN statement makes you assign a channel number to the file. With the LOPEN statement, the system assigns the channel number, and you do not have access to which channels are free to be used with the OPEN statement. If you try to OPEN a file on a channel that is in use, an error occurs.

Table 6-4 lists the input and output commands that you can use with logical files. These commands are explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

Command	Action
DELREC	Delete a logical record in a linked-available-record file. (Logical database only)
GETLAST.SL	Retrieve the number of active records and the highest record in use in a linked-available-record file. (Logical database only)
GETREC	Allocate a logical record from a linked-available-record file. (Logical database only)
KADD	Add a key entry to an index file.
KDEL	Delete a key entry from an index file.
KFIND	Find a key entry in an index file.
KNEXT	Return the next key entry in an index file.
LOCK/UNLOCK	Synchronize the updating of files that are shared between programs.
LOPEN FILE	Open and/or define a logical file. (Logical database only)
LREAD FILE	Read a logical record. (Logical database only)
LWRITE FILE	Write a logical record. (Logical database only)

Table 6-4. I/O Commands Used with Logical Files

PARAM File Database Structure

The PARAM file database structure consists of a file set that includes a PARAM file and a master file. Both files are physical files. The PARAM file contains information on the subfiles, which are logical subsections of the master file.

The PARAM structure permits three types of subfiles:

- Direct random files. The user handles all record assignments. You cannot create this type of file using INITFILE.
- Linked-available-record files. Record allocation assignments are made dynamically by Business BASIC.
- Index files. These files are maintained via the ISAM statements.

Setting up a PARAM Database

To set up a PARAM database structure, perform the following steps:

1) Design your database so that you know the record size and the number of records you want in the data file.

2) Run INDEXCALC to determine the number of keys per index block, the number of blocks (sectors) needed in the index file, and the number of sectors needed for the data file.

3) Create the PARAM file. (You only need to create the PARAM file once.)

4) Run INITFILE to initialize the files and enter the necessary information in the PARAM file.

5) Use the input/output commands in Table 6-8 to access your database.

An example of setting up a PARAM database and a program that uses it is in appendix C.

The PARAM File

The PARAM file consists of records that specify the size and location of subfiles. After you set up the PARAM file, you use the OPEN utility to extract the subfile information and place it in the C1 (file characteristics) array. Your program then uses the information in the C1 array to locate records in the subfile.

Each record in the PARAM file is 42 bytes long and contains information on only one subfile. Record number 0 is reserved for information describing PARAM itself.

The Business BASIC software package includes an empty PARAM file. This file is in the library directory and contains 10 records, three of which are blank. To increase the number of records available, change the record limit in record 0 (see Table 6-5). You can use one PARAM file for all programs, or you can have as many as one PARAM file per directory.

On AOS systems, when you use more than one PARAM file, include the directory containing the PARAM file you need on your search list. Also, be sure that your program uses the correct PARAM; using the wrong PARAM file can destroy parts of your database.

To set up a PARAM file, follow these steps:

- Use a BASIC CLI command such as CCONT or CRAND to create the PARAM file.
- Initialize record 0 of the PARAM file using the File Maintenance (FM) utility. (Table 6-5 contains a description of record 0 and appendix C discusses using FM to set up record 0.)
- Use the interactive utility INITFILE to add information describing each physical file and subfile.

In addition to INITFILE, you can also add entries to the PARAM file by using:

- The FM utility.
- A user-written utility.

CCONT, CRAND, FM, and INITFILE are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Table 6-5 describes record 0 of a PARAM file, and Table 6-6 describes the record structure for the rest of the PARAM file.

Bytes	Contents
0-1	1; this is the active record status indicator.
2-11	PARAM (the rest of the string is filled with nulls).
12-21	PARAM (the rest of the string is filled with nulls).
22-25	0; this is the beginning byte of the PARAM file.
26-27	42; this is the length of each record in the PARAM file.
28-31	Maximum number of records to be kept in the PARAM file.
32-35	Highest record number in use. (When records are added in the PARAM file, this number should be incremented. However, this number should not be decremented when a record is deleted. If you are adding records directly to the PARAM file or if you are using FM to add records, you must increment this number yourself; it is not done automatically in those cases.)
36-41	Unused.

Table 6-5. Record 0 of the PARAM File

Bytes	Description
0-1	Status indicator; must equal 1.
2-11	Name of the subfile or physical file.
12-21	Name of the physical file that holds the subfile.
22-25	Byte pointer to the start of the subfile or physical file.
26-27	Record length of the subfile or physical file.
28-31	Last record number of the subfile or physical file.
32-35	Number of the last record containing data.
36-41	Unused.

Table 6-6. Contents of a PARAM File Record

C1 (File Characteristics) Array

The C1 or file characteristics array is set up in your program and contains information about your master files and subfiles. This information is used by the subroutines GETREC.SL, DELREC.SL, and POSFL.SL to compute the position of records within a linked-available-record file.

The C1 array is a two-dimensional array with four columns (0, 1, 2, 3) and n rows, where n is the number of subfiles used in your program. Remember that arrays are zero-based; thus, if you have n files, the maximum row number you have is n-1 (i.e., 0 to n-1). Table 6-7 explains how the columns in the C1 array are used.

Column	Description
0	Contains the number of the channel you used to open a master file. (The channel number is the number you associate with a file for all file access.)
1	Contains the byte offset relative to 0 to the beginning of the subfile within the master file. (The master file always starts at byte 0.)
2	Contains the file size (i.e., the number of records in the file).
3	Contains the record size (i.e., the bytes per record).

Table 6-7. Column Contents of the C1 Array

Building a C1 Array

You dimension the C1 array from within your program. There are three ways to build the C1 array and add information to it:

- The OPEN utility.
- The FINDFILE.SL subroutine.
- The LET statement.

Both OPEN and FINDFILE.SL use the PARAM file to get the information necessary to build a C1 array. With the LET statement, you assign values to the array.

In the example that follows, the record size, channel number, or file size of any file can be changed by changing the values in the C1 array. This example uses the LET statement to build the C1 array.

```
0010 DIM C1(2,3)                 :Dimension C1 to be 3*4 array.
0020 LET C1(0,0)=2               :File 0 (EMPDATA) opened on channel 2,
0030 LET C1(0,1)=0               :and begins at byte 0 of EMPLOYEE,
0040 LET C1(0,2)=400             :and contains a maximum of 400 records,
0050 LET C1(0,3)=128             :with 128 bytes in each record.
0060 LET C1(1,0)=2               :File 1 (EMPIX) also opened on channel 2
                                 :because EMPDATA and EMPIX are in the
                                 :same physical file EMPLOYEE, opened on
                                 :channel 2.
0070 LET C1(1,1)=51712           :EMPIX begins at byte 51712 (block 101),
0080 LET C1(1,2)=15              :and contains 15 index blocks
0090 LET C1(1,3)=512             :with 512 bytes in each block.
0100 LET C1(2,0)=3               :File 2 (FEDTAX) is opened on channel 3,
0110 LET C1(2,1)=0               :starts at byte 0 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 -- TIME TO OPEN FILES
0150 LET C%=C1(0,0)              :Channel number now in C%.
0160 OPEN FILE(C%,0),"EMPLOYEE"  :This will allow access to both EMPDATA
                                 :and EMPIX.
0170 LET C2%=C1(2,0)             :Channel number of FEDTAX now in C2%
0180 OPEN FILE(C2%,0),"FEDTAX"
.
.
```

You can use the subroutine FINDFILE.SL to have your program automatically build the C1 array. FINDFILE.SL creates a C1 array without opening your files. It also returns the next available channel number. If no PARAM file entry exists for the file, FINDFILE.SL treats the file as a physical file and asks you for the byte offset, record size, and file size. It does not create a PARAM entry.

When FINDFILE.SL ends, you have a C1 array with:

- The 0 column (channel number) blank.
- A variable X$ (an output variable required by FINDFILE.SL) with the filename for the subfile or physical file.
- A variable C% (an output variable required by FINDFILE.SL) with the next available channel number you can use with an OPEN FILE statement.

The following program uses FINDFILE.SL to fill the C1 array with the same values used in the previous C1 array example.

```
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$              :Print 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
```

You can also use the OPEN utility to find a channel for your file and to supply the information you need for the C1 array. OPEN gets the information for the C1 array from the PARAM file. When you set up a C1 array with OPEN, you are restricted to a maximum of 32 subfiles. This is because OPEN passes the information for the C1 array through the common area, which holds

only 512 bytes. Each entry in the C1 array has four elements, and each element is 4 bytes long.

OPEN and FINDFILE.SL are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Modifying a Record in the C1 Array

Business BASIC provides three access routines that you can use to modify a record in the C1 array:

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

GETREC.SL and DELREC.SL both maintain record 0 of a linked-available-record file.

All three subroutines are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Positioning to a Record

Use POSFL.SL to position to a record or to a byte offset in the record. POSFL.SL requires the variables F% for the file number, R1 for the record number you want, and, optionally, V% for the byte location in R1.

POSFL.SL returns 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 master file where record R1 starts; and R8 for the byte position of the subfile where record R1 starts.

You can use the variable R9 or R8 with the POSITION FILE statement and follow it with a WRITE FILE statement to do a quick rewrite of the record. You can also use POSFL.SL to position to the data record and execute a READ FILE to read the record found.

This program positions to a record, reads the record, and uses POSITION FILE with R9 to go back to rewrite the record. Note that code that opens the files and fills the C1 array was not deleted.

```
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 file info into common area.
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 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 from POSFL.SL to position
0220 WRITE FILE (C%),NEWREC$             :and rewrite record R1.
.
.
.
```

Writing a Record in the PARAM Structure

To write a record to a subfile, use the subroutine GETREC.SL to get the number of the next available record. GETREC.SL finds a record in the deleted-record chain, then updates record 0, and returns the record number. Use this record number with POSFL.SL to position to the record before writing to it. Then use WRITE FILE to modify the new record.

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 file, you must copy your file into a larger contiguous file or into a random file.

Deleting a Record in the PARAM Structure

Use DELREC.SL to delete subfile records and place them on the deleted-record chain so that you can reuse the space. DELREC.SL automatically updates record 0 and then deletes the record by setting its status (the first 2 bytes) to 0. DELREC.SL uses the C1 array and calls POSFL.SL.

The following code segment shows a partial update session. The record is deleted using DELREC.SL, and a new one is added using GETREC.SL. This update technique is good to use when the placement of new records does not matter. With indexed data files, placement is not important as long as you use KADD to add the new key to the index.

```
.
.
.
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.
.
.
.
```

Input and Output with the PARAM Database

Business BASIC provides several commands (i.e., commands, statements, and subroutines) for performing input and output operations on subfiles. Some commands are tailored to the PARAM file database while others can be used with both databases. To perform input and output operations on your subfiles, you must OPEN them and set up the C1 array.

In the PARAM structure, much of the linked-available-record access is done through the GETREC.SL, DELREC.SL, and POSFL.SL subroutines. Table 6-8 lists the input and output commands you can use with PARAM database files.

Command	Action
DELREC.SL	Delete a record in a linked-available-record subfile. (PARAM database only)
FINDFILE.SL	Find a subfile and build a C1 array. (PARAM database only)
GETREC.SL	Get the number of the next available record in a linked-available-record chain. (PARAM database only)
KADD	Add a key entry to an index file.
KDEL	Delete a key entry from an index file.
KFIND	Find a key entry in an index file.
KNEXT	Return the next key entry in an index file.
LOCK/UNLOCK	Synchronize the updating of files that are shared between programs.
OPEN	Open physical files and subfiles.
OPEN FILE	Open a physical file.
POSFL.SL	Position the file pointer to a record in a data file. (PARAM database only)
READ FILE	Read binary data from a file or record.
WRITE FILE	Write binary data to a file or a record.

Table 6-8. I/O Commands Used with PARAM Database Files

Converting from a PARAM Database to a Logical Database

Business BASIC provides the PARAMCON utility to allow you to convert from the PARAM file database to the logical file database. When you use PARAMCON, you need to make whatever changes are necessary to your database so that the files meet the requirements for logical files. These are primarily filename changes and are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI* under PARAMCON.

After you convert your files, you can use the logical file input and output statements to access them. This means you will need to modify your programs; for instance, change the READ FILE statements to LREAD FILE. However, you can still use the OPEN utility with these files.

You can also use a special form of the LOPEN statement to define physical files that are not part of the logical structure. This lets you use the logical input and output statements with these files.

To maintain your converted files, use the Logical File Maintenance (LFM).

PARAMCON and LFM are explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*

Comparing Databases

In many cases, the logical file database structure and the PARAM file database structure have features that perform similar operations but have different names. A major difference between the two structures is that the logical database calculates the byte offset to each logical file; the PARAM database does not do this with its subfiles. In addition, you can use the GETREC and DELREC statements with the logical database, whereas the PARAM database uses the GETREC.SL and DELREC.SL subroutines. The advantages of the statements over the subroutines are that the statements perform automatic LOCKing, they are faster than the subroutines, and they free the code space normally used by the subroutines. The three subroutines use a total of 55 lines of code.

Table 6-9 compares some of the features of the two database structures.

Logical Database	PARAM Database	Description
Database file	Master file	Physical file that contains the actual data.
Volume label file	PARAM file	Physical file that contains information on locating the logical files or subfiles.
Logical file	Subfile	A subsection of a physical file.
LFTABL$	C1 array	LFTABL$ is a string variable that holds the information on the logical files. C1 array is an array that holds information on subfiles. In both cases, this information is used by input and output commands.

Table 6-9. Logical, PARAM Database Features

End of Chapter

Chapter 7
The INFOS II® File System (AOS Only)

This chapter provides an overview of the AOS INFOS II file management system. It discusses:

- The components of the INFOS statements, such as argument pairs and channel strings.
- Accessing INFOS statements.
- Error handling.

Introduction to INFOS II

Data General's AOS INFOS II file management system lets you create, maintain, and use many types of databases in batch and multiterminal on-line environments. The INFOS II system's data handling capabilities make it suitable for diverse applications in manufacturing, payroll, accounting, inventory, sales, credit, insurance, and distribution management.

The Business BASIC interface to the INFOS II system allows Business BASIC programs to use the INFOS II facilities. The interface consists of a set of statements that are used like other Business BASIC statements, although their formats are different. These interface statements can be used as statements or commands. Collectively, they are referred to as DB statements, since each one begins with the letters DB.

Table 7-1 lists the DB statements with brief descriptions. For complete descriptions of the statements and their formats, see the section titled "Statements Related to INFOS II Files" in the *Business BASIC Reference Manual for Commands, Statements, and Functions.* The *AOS INFOS II System User's Manual* also contains information on the interface.

The Business BASIC INFOS II interface imposes some restrictions on working with INFOS II files. You cannot create INFOS II files through the interface; you must use the INFOS II ICREATE utility. In addition, INFOS II files are included in Business BASIC's 16-channel limit; thus, you are restricted to a maximum of 16 open INFOS II files at any one time. If you exceed this limit, the error message `NO MORE CHANNELS AVAILABLE` appears.

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 database file.
DBREAD	Reads from an INFOS II file.
DBREINS	Reinstates a logically deleted record.
DBRELEASE	Releases locks and/or current position in the file.
DBRETRIEVE HIGHKEY	Retrieves the high key.
DBRETRIEVE KEY	Retrieves a subindex definition.
DBRETRIEVE STATUS	Returns the INFOS II status.
DBREWRITE	Rewrites an INFOS II database record.
DBSET	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 7-1. Business BASIC INFOS II Statements

Argument Pairs

Each INFOS II statement has a set of arguments called argument pairs. Argument pairs are comparable to switches; they alter the statements. Most argument pairs are optional, though some DB statements require certain argument pairs. In many cases, the argument pairs have default values that are supplied if they are not used. You can use the DBSET statement to redefine the default values.

Argument pairs fall into the following categories:

1) Selecting INFOS II options.

2) Passing numeric values to INFOS.

3) Receiving numeric values from INFOS.

4) Specifying where information is going to or coming from (strings).

5) Handling errors.

Argument pairs use the format:

```
keyword=value
```

where value sets a parameter for INFOS II. The parameter is set only when the statement affected by the argument pair is executed. Depending on the argument pair, the value can be numeric or string. You can use a DBWRITE to set the value. Some examples of argument pairs are:

- ACCESS=REL, which indicates relative access.
- DUPKEY=NO, which indicates the specified key is not a duplicate.
- MOTION=BACK, which indicates the direction of relative motion is backward.

With an argument pair that receives information from INFOS II (category 3), value can be a subscripted variable. You use a DBREAD to place the information in the variable.

Category 4 argument pairs always use string variables for value. You cannot use a subscripted string in these pairs. Once the string is associated with the INFOS channel string, the string variable — not its current contents — is transmitted to INFOS II. In the example:

```
0100 LET KVAR$="XYZZY Computers"
0110 DBSET F$,KEY=KVAR$,REC=RVAR$
0120 LET KVAR$="Data General"
0130 DBREAD F$,ACCESS=KEY
0140 PRINT RVAR$
```

the value of KVAR$ when the DBSET statement is executed is not significant, only the current value of KVAR$ is significant.

Hence, when line 130 is executed, it uses KVAR$ with the contents of "Data General" — not "XYZZY Computers," which KVAR$ contained when the DBSET statement was executed.

Channel Strings

Channel strings under INFOS II are analogous to channel numbers used by Business BASIC when physical files are opened. The INFOS II channel strings identify a particular INFOS II channel and supply Business BASIC with the information it needs to access INFOS II.

To use a channel string, you must first dimension it. The formula for this is:

1) Multiply the argument to the MAXKEYS argument pair by 16.

2) Add 56 to that number.

After you dimension the channel string, use the DBOPEN statement to initialize it. As long as the file that the channel string refers to is open, do not modify the channel string. Also, until the file is closed, use the channel string with INFOS II statements only.

When the INFOS II files have been DBOPENed in one program, that program can CHAIN or SWAP to other programs, passing the channel string through the common area. The program that has been called can then use the channel string to access the INFOS II file without issuing another DBOPEN.

Creating an INFOS II File

When you create an INFOS II file, two logical structures are defined: an index file and a database file. INFOS II automatically links these structures to form a single ISAM file and, using AOS, 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 AOS CLI. Type in:

```
ICREATE filename
```

where filename is the name of the index file. If you do not specify filename, INFOS II prompts you for it. Below is a sample ICREATE dialog.

```
ICREATE ACCOUNTS

****** INFOS FILE CREATION      04/18/86      09:21:21 ******

ACCESS METHOD (I=ISAM, D=DBAM) [D]: I

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

       PAGE SIZE (BYTES) [2048]: 2048
       PARTIAL RECORD LENGTH [0]: 0
       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
```

Accessing INFOS II Files

To access an INFOS II index file, you must set up a channel string and then open the file using the DBOPEN statement. This example opens the INFOS II file ACCOUNTS, which was created in the previous example, and assigns it to the channel string MASTER$, which is then used each time the file is referred to.

```
:
: Dimension channel string, string for data record, and key string
:
0010 DIM MASTER$[72],MDATA$[20],K$[10]
:
: Open the INFOS II database and set default parameters
:
0020 DBOPEN INFOS "ACCOUNTS",MASTER$
0030 DBSET MASTER$,ACCESS=KEY,KEY=K$,REC=MDATA$
:
: Enter information and write it to the INFOS II database
:
0040 FOR I=1 TO 5
0050   INPUT "Enter ACCOUNT NUMBER: ",K$
0060   INPUT "Enter ACCOUNT NAME: ",MDATA$
0070   DBWRITE MASTER$
0080   PRINT
0090   NEXT I
:
: Retrieve selected information from the database
:
0100 FOR I=1 TO 5
0110   LET MDATA$=""
0120   INPUT "Retrieve account name for ACCOUNT NUMBER: ",K$
0130   DBREAD MASTER$
0140   PRINT "ACCOUNT NAME: ";MDATA$
0150   PRINT
0160 NEXT I
:
: Close the database
:
0170 DBCLOSE MASTER$
0180 END
```

Once you have opened a file, you can use any INFOS II statement to manipulate the file. This example uses DBSET to set the INFOS II file parameters,

DBWRITE to place data in ACCOUNTS, and DBREAD to get data from ACCOUNTS. Then DBCLOSE is used to close the INFOS II file. Each of the statements accesses ACCOUNTS by using the channel string MASTER$.

Other common INFOS II operations include deleting keys (DBDELETE) and reinstating logically deleted records (DBREINS). You can also perform more advanced INFOS II features. These include defining subindexes (DBSUBINDEX DEFINE), retrieving subindex definitions (DBRETRIEVE SIDEF), and linking subindexes (DBSUBINDEX LINK).

Error Handling

Error handling for INFOS II files is done by using the optional argument pair ERR=line-number. This argument pair can be used with all INFOS II statements. Line-number represents any valid statement number.

When an INFOS II statement detects a user error, it starts the following sequence of actions (these actions halt when a true condition is met):

1) If the user issued the DB statement as a command, the appropriate error is issued at the terminal.

2) If the user's statement contained ERR=line-number argument pair, the system executes a GOTO line-number; if no line-number exists, the system generates a line number error.

3) If the program had an ON ERR condition in effect, the program executes the THEN clause of that statement.

4) If all of the above conditions fail, the error is directed to your terminal.

Two errors are generated by the interface:

- ERROR 82 (Parameter Range Error)

  This error is issued when you attempt to set an INFOS II parameter to a value that is beyond the range allowed by INFOS II. INFOS II supports 32 levels of subindexes.

- ERROR 83 (INFOS II Error)

  Whenever an INFOS II error occurs, SYS(7) returns a value of 83, and SYS(31) contains an error code indicating the actual INFOS II error code. SYS(31) returns a negative, decimal number. To convert it to the INFOS II error code, make it positive and change it to octal. For example:

  1) SYS(31) = -3705

  2) Change this to a positive number: 3705

3) Convert it to an octal number: 7171

4) Look up 7171 in the INFOS II manual to get the error message; in this case, it is `NOT AN INFOS II INDEX FILE`.

End of Chapter

Chapter 8
Transporting Programs between RDOS and AOS

It is possible to transfer Business BASIC files between RDOS and AOS systems. This chapter discusses what you need to do to move files between operating systems and what compatibility issues you should keep in mind to make your programs portable.

Transferring Files

The commands you use in moving files between operating systems are determined by:

- The sending and receiving operating systems.
- The medium that the file is placed on when it is moved off one system (magnetic tape or diskette).

Table 8-1 summarizes the commands that are used to load files to media and then unload them on other operating systems.

Sending Operating System	Medium Holding File	Command to Move File to Medium	Receiving Operating System	Command to Move File from Medium
RDOS	Tape	RDOS CLI DUMP command	AOS	Execute RDOS.PR program with LOAD command
DG/RDOS	Tape	IMOVE	AOS	AOS CLI LOAD command
DG/RDOS	Diskette	IMOVE	AOS, AOS/WS (Not AOS/VS)	MMOVE
AOS	Tape	Execute RDOS.PR program with DUMP command	RDOS	RDOS CLI LOAD command
AOS	Tape	AOS CLI DUMP command	DG/RDOS	IMOVE
AOS, AOS/WS (Not AOS/VS)	Diskette	MMOVE	DG/RDOS	IMOVE

Table 8-1. Methods of Moving Files between Operating Systems

Moving Files from RDOS to AOS

To transfer files from an RDOS system to an AOS system, certain conversion steps are necessary. These steps vary according to the type of file you move.

Moving Text Files to AOS

To move text files to an AOS system, you need to convert the carriage returns to new lines. When you use the RDOS.PR program to LOAD a Business BASIC text file onto the AOS system, you do this by adding a local /C switch to the command line (this switch is only used with text files):

```
) X RDOS LOAD/V @MTxn:0 filename/C
```

where x is the tape controller type and n is the tape unit. When you use IMOVE to move RDOS text files to AOS, IMOVE handles the conversion.

Moving Logical Database Files to AOS

To move database files and volume label files to an AOS system, you do not use any special switches when you LOAD the files onto AOS. However, unlike text files, you must convert database files and volume label files to the correct AOS file type and element size. Use the BASIC CLI command DBFIX to do this.

DBFIX converts the element size of a database file or a volume label file to a multiple of four and adjusts the file type to DBF for a database file and VLF for a volume label file. It also provides links for the logical files.

You can LOAD logical files that are linked to the volume label file; however, the links may not apply to the AOS directory where you place the files. For example, if you are in a directory called :UDD:USERFILES and a link for a logical file has been loaded as:

```
DATA1     LNK     TEST:PRODUCT.VL
```

then, if DATA1 and PRODUCT.VL are in the same directory, the link needs to be changed to

```
DATA1 LNK PRODUCT.VL
```

Moving SAVE Files to AOS

To move a SAVE file to an AOS system, you do not use any special switches when you LOAD the file onto AOS. However, unlike text files, you must convert the SAVE file to the correct AOS file type and element size. Use the BASIC CLI command FIXFILE to do this.

FIXFILE adjusts the AOS file type of a file to the type you specify (for a SAVE file, this is BBS) and converts the element size to a multiple of four.

FIXFILE and DBFIX are explained in the *Business BASIC System Manager's Guide.*

Moving Files from AOS to RDOS

To transfer files from an AOS system to an RDOS system, certain conversion steps are necessary. These steps vary according to the type of file you move.

Moving Text Files to RDOS

To move text files to an RDOS system, you need to convert the new lines to carriage returns. When you use the RDOS.PR program to DUMP an AOS Business BASIC text file, you can change the new lines by adding a local /C switch to the command line (this switch is only used with text files):

```
) X RDOS DUMP/V @MTxn:0 filename/C
```

where x is the tape controller type and n is the tape unit. When you use IMOVE to LOAD the files onto RDOS, IMOVE handles the conversion.

Moving Logical Database Files to RDOS

To move database files and volume label files to an RDOS system, you do not need any special switches when you DUMP the files from AOS. However, unlike text files, once the logical database files are on the RDOS system, you need to perform two steps before you can use the database:

1) If you moved the links connecting the logical files to the volume label file, then you must DELETE the logical filenames and recreate the links. This is because the link file is loaded as a random file. If you did not DUMP the links, then you must create the links to the volume label file. (Remember a link exists to the volume label file for each logical file contained in the database.)

2) You must set certain attributes on the .DB and .VL files. To do this enter the following:

```
CHLAT filename.DB +P
CHATR filename.VL +&
CHLAT filename.VL +PW
```

Moving SAVE Files to RDOS

To move a SAVE file to an RDOS system, you do not need any special switches when you DUMP the files from AOS. However, unlike text files, once the SAVE file is on the RDOS system, you need to set certain attributes for it:

```
CHATR filename +SW
```

Operating System Differences

The different features of the AOS and RDOS operating systems can affect how you structure your programs. You can have common programs for RDOS and AOS that contain parts that are specific to the operating system you are using.

Use the SYS(30) function to do this; SYS(30) detects which operating system you are using.

If you plan to transport your programs between operating systems, consider these differences as you prepare your programs:

- Under RDOS, Business BASIC runs in either the foreground or the background, and it handles multiplexors for multiple users. In addition, the RDOS Business BASIC system has special requirements for stacks, cells, and buffers. You specify the type of system you need when Business BASIC is generated, and you generate this support in Business BASIC rather than in RDOS. Business BASIC becomes a subsystem running in an RDOS environment. It has its own CLI to handle accessing devices, files, and RDOS CLI operations, as well as an OPCLI program to handle operator responsibilities.

- AOS is a multiprogramming system that allows many programs to run concurrently. Each executing program and its set of system resources is a process. Each user's process resembles a complete computer system: it often has a distinct programming console; it can use all system devices; and, in some cases, it can create many son processes that have their own resources. Under AOS, processes are independent of each other; individual processes usually cannot affect other processes. Each process offers multitasking, so that each user can perform several different tasks concurrently, and each task can respond individually to its own environment. Different processes can execute different versions of Business BASIC (including a run-only version) as well as other AOS utility programs. Business BASIC security under AOS is handled primarily by the operating system. The operating system maintains a username/password security scheme, where any son process you create bears your username. Business BASIC maintains a User Status Table associated with your username.

- System input/output errors are reported with RDOS error codes or their equivalents. SYS(7) holds the RDOS error code and SYS(31) holds the code for the equivalent native operating system error. On an AOS system, SYS(31) has the code of an AOS error message; on an RDOS system, SYS(31) has the code for an RDOS error message, which is the same as the code in SYS(7). You can use SYS(7) with the ERM$ function or SYS(31) with the AERM$ function to retrieve the error message. If an AOS error cannot be translated to an equivalent RDOS error code, the error code -60 is returned in SYS(7).

- AOS has two additional terminal types — 8 and 9. Terminal type 6 on AOS is very compatible with terminal type 6 on RDOS. Terminal type 8 on AOS offers greater functionality with a minor loss of compatibility.

  On AOS, 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, a STOP/END statement, or a full buffer. You can force data from the buffer to the terminal by executing the STMA 8, 5 statement.

Appendix F in the *Business BASIC System Manager's Guide* contains information on terminal types.

- AOS restricts the IKEY to either the excape 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.

- AOS uses new line as the default primary unpend character. In addition, on AOS Business BASIC, text file records must end with new line and, in general, programs that contain carriage return characters in string literals or in numeric variables must be changed to use new line. On RDOS Business BASIC, carriage return is the default primary unpend character.

- AOS Business BASIC handles links the way they are handled under RDOS. To delete a link, use the BASIC CLI command UNLINK. On AOS, the Business BASIC DELETE command deletes the resolution file only, not the link file. Deleting a link deletes the resolution file. Links should contain complete pathnames to the resolution file. In many cases, the AOS search list feature eliminates the need for links.

- AOS does not allow you to run detached jobs.

- AOS systems allow a file to be deleted even if the file is opened by the process doing the delete or by another user. At the time of the DELETE call, AOS systems mark the file for deletion. The file is not actually deleted until the final CLOSE of the file. RDOS returns an error message when you attempt to delete an open file.

- The Business BASIC spooler is not implemented on AOS systems.

- Some features of Business BASIC exist in both RDOS and AOS but do not perform the same function. For example:

  SYS(4) Under RDOS Business BASIC, SYS(4) returns the terminal port number:

  0 = background console
  1 = foreground console
  2 to n = multiplexor consoles

  Under AOS Business BASIC, SYS(4) returns the console number or a zero if the console is not a son process of EXEC.

  STAT Under RDOS Business BASIC, STAT displays the status of all jobs on the system including the Business BASIC program name executing on each terminal port number.

  Under AOS Business BASIC, STAT invokes the AOS utility PED.PR. (On AOS, there is no way to see which Business BASIC program a process is running.)

  STMC 14 Under RDOS Business BASIC, STMC 14 issues an .FGND system call to determine whether a foreground program is already running.

Under AOS a ?PSTAT system call is issued to determine whether a process has sons.

DELETE Under RDOS, you receive an error message when you try to delete an open file.

Under AOS, you can delete a file even if it is open (either by the process doing the delete or by another user). At the time of the DELETE call, AOS systems mark the file for deletion. The file is not actually deleted until the final CLOSE of the file.

GPOS(n) Under RDOS, GPOS(n) returns a 0 when a device is opened on channel n.

Under AOS, GPOS(n) causes an error when a device is opened on channel n.

Under RDOS, when you use mode 0 of OPEN to open a link to a non-existing resolution file, RDOS creates the resolution file.

Under AOS, the message FILE ALREADY EXISTS is returned when you use mode 0 of OPEN to open a link to a non-existing resolution file.

- In addition, there are a number of commands, statements, functions, subroutines, and utilities that are specific to an operating system.

RDOS-only subroutines, utilities, and BASIC CLI commands:

ASG	FLOAD	PED
ATTACH	FREE	RELEASE
CHATR	GSDIR	SDIR
CHLAT	GSYS	SLINE
DBMOVE	INIT	SPDIS
EQUIV	LSPEED	SPEBL
FDUMP	MDIR	SPKILL
		START

RDOS-only commands, statements, and functions:

MSG	STMA 16
STMA 3,0	STMA 17
STMA 4,0	STMA 18

RDOS-only privileged system calls:

STMB 6 - 8	STMC 2 - 4	STMD 0
STMB 14 - 15	STMC 10	STMD 1
STMB 19	STMC 18 - 20	
	STMC 23 - 24	
	STMC 26 - 27	
	STMC 29 - 34	
	STMC 45 - 48	
	STMC 51	
	STMC 53	

RDOS-only system manager utilities:

ACCOUNT	KILL	OPCLI
ACNTNGROLL	LOGDISP	QUICKILL
ACNTNGRPT	LOGINIT	SPOOLER
ANALYZE	NEWS	VACUUM

AOS-only subroutines, utilities, and BASIC CLI:

AOS	PORTS

AOS-only commands, statements, and functions:

STMA 8,5	All INFOS II statements

AOS-only privileged system calls:

STMB 22-24	STME 0-27

AOS-only system manager utilities:

DBFIX	FIXFILE

End of Chapter

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 of your Business BASIC system. Business BASIC allows you to use one-dimensional arrays (vectors) with rows only (each element is a row), and two-dimensional arrays (matrices) with rows and columns. Rows start at row 0, and columns start at column 0. The default dimension for a one-dimensional array is 10 for 11 elements. For a two-dimensional array, the default is 10 by 10 for 121 elements.

ASCII code
The decimal code number assigned to a character (unless octal is specifically stated). All printable characters, nonprintable characters, and other keys on the terminal's keyboard have ASCII code numbers. See also characters.

attributes
All RDOS files can have attributes. The BASIC CLI command CHATR changes a file's resolution attributes (permanent, read-protected, and write-protected). The BASIC CLI command CHLAT changes a file's link access attributes.

BASIC CLI (command line interpreter)
A utility program that emulates an operating system CLI. You can RUN, SWAP, or CHAIN to the BASIC CLI. You can execute BASIC CLI commands interactively, or you can SWAP to the CLI from a program that passes the command through the common area. The BASIC CLI prompt is an exclamation point.

bit
An element of storage. 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).

Boolean logic operators
Operators that evaluate an expression as either true (1) or false (0). Business BASIC supports the Boolean logic operators NOT, AND, and OR.

byte
A sequence of eight adjacent bits (locations 0 to 7).

C1 array
A Business BASIC convention necessary for using the GETREC.SL, DELREC.SL, and POSFL.SL subroutines, which provide fast access to subfiles. The C1 array is used with the PARAM file database structure. 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. Also called the file characteristics array.

carriage return (CR)
A key on some terminals that is used to signal the end of an entry; a line terminator. This manual uses the term new line key to represent both the carriage return key and the new line key. When you see new line, use the line terminator key that is appropriate for your system.

character
A character is stored as an ASCII code number in 1 byte (see also byte). The term character data is used to mean string literals.

command
An instruction directing the system to do something immediately. You execute BASIC commands from keyboard mode (indicated by an 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 picks 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.

concatenation operator
You can concatenate strings by using a comma to separate them in an assignment statement.

database file
A physical file in the logical file database structure that contains logical files (subsections). A database file is indicated by a .DB extension on its filename and is always associated with a volume label file.

delimiter
A character that indicates the start of a character string. Most keyboard editing commands require delimiters. Generally, the delimiter can be any character that is not part of the character string. When a command uses more than one delimiter, the delimiter must be same in each place it is used in the command line.

device name
The name assigned to a device, such as a line printer or tape drive. Under AOS, the names usually begin with an at (@) sign. For example, @LPT is the line printer. Under RDOS, devices 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.

directory
A file that contains entries (pointers) called filenames. Under AOS Business BASIC, directories are DIR-type files or CPD-type files, and you refer to one using a pathname. Under RDOS, 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 form of allocating records to a file by allowing the file to reuse space left by deleted records. Under dynamic allocation, a file maintains a deleted-record chain to indicate which records can be reused. Business BASIC has commands GETREC and DELREC (logical database) and subroutines GETREC.SL, DELREC.SL, and POSF.SL (PARAM database) that access and allocate dynamic file space for random files. See also linked-available-record format.

edit buffer
A one-line buffer that is used by the keyboard editing commands .P, .A, .C, .E, and .I to modify a single line. The edit buffer contains the last statement LISTed, the last statement typed in, or the last statement that caused an error message to appear.

enter
The act of typing in data and then pressing a line terminator key to signal the end of the input, thus sending the data to the system. For example, to enter a program statement, you type in the statement line number followed by the statement contents and then press the new line or carriage return key. You can enter data while in keyboard mode or in response to a program query. Business BASIC also provides a command/statement called ENTER that is used to place programs in working storage.

expression
A numeric expression that consists of some combination of Boolean operators, 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
A collection of data. Business BASIC has program files, listing files, source files, text files, data files, subfiles of data files, logical files, index files, tag files, table files, log files, documentation files, and screen files. Directories and devices are also files.

file characteristics (C1) array
See C1 array.

filename
A name that refers to a file. Filenames in RDOS can have up to 10 alphanumeric characters and an extension of up to two characters after a period.

Filenames in AOS can have up to 31 alphanumeric characters including underscores and extensions. (Some Business BASIC utilities require that filenames follow the RDOS length conventions. These utilities truncate filenames that exceed this limit. Business BASIC does not warn you when a filename is truncated.)

ISAM file
A file containing records that use the Indexed Sequential Access Method. Each file has a unique key, which enables fast reading for selected records. An ISAM file set consists of two files. The actual data 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 data file.

interrupt key
The key(s) you press to interrupt a program or a command execution. It is frequently the ESC key. You can define an interrupt key by using STMA 4,6 and STMA 4,7. This is explained in the *Business BASIC Reference Manual for Commands, Statements, and Functions.*

IKEY
This is the term used by this manual to designate an interrupt key. See interrupt key.

keyboard mode
A Business BASIC operational mode. In keyboard mode, you can enter a command for immediate execution or enter program statements to create a program in working storage. The asterisk prompt indicates you are in keyboard 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) on RDOS and the $SYSLIB (or $SYSLIB3 for triple precision systems) on AOS. All users should have access to it.

linked-available-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.

listing file or list file
A text file. This can be a file created by the LIST command; i.e., an ASCII listing of the program currently in working storage or a file you created in an editor. You can use the BASIC EDIT utility or any system editor to modify a listing file. You use the command ENTER to bring a listing file into working storage.

log file
A file containing a record of all access to a data file in the utility 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
The process of terminating Business BASIC by typing in BYE.

logging on
The process of executing Business BASIC.

logical file
A logical subsection of a physical file. The term logical file is used to indicate files in the logical file database structure.

logical file database structure
One of Business BASIC's two database structures. The logical file database allows you to open more than 16 files simultaneously in one program by using logical files, which are subsections of physical files. See also PARAM file database structure.

master file
A physical file in the PARAM file database structure that contains subfiles.

new line
A key used on some terminals to signal the end of an entry; a line terminator. This manual uses the term new line key to represent both the carriage return key and the new line key. When you see new line, use the line terminator key that is appropriate for your system.

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 the information you need for using subfiles and the OPEN utility program. The PARAM file for your system can be in the library, or you can set up a PARAM file in a directory you specify. You can have more than one PARAM file.

PARAM file database structure
One of Business BASIC's two databases. The PARAM file database structure enables you to open more than 16 files simultaneously in one program by using subfiles.

pathname
A name that uses directory names to identify the location of a file within the system files.

precision
The number of bytes used to store or transfer a numeric variable or array element. The precision is set when your system is generated. You can have either a double precision system or a triple precision system. A double precision system stores numeric values in 4 bytes and transfers numeric variables using either 2 bytes (if the variable name ends in a percent sign) or 4 bytes. Triple precision Business BASIC stores numeric variables in 6 bytes and transfers each variable as 2 bytes (when the variable name ends in a percent sign), 4 bytes (when there is no special character at the end of the variable name), or 6 bytes (when the variable name ends in a pound sign).

prompt
A character output to your terminal to signify that you must enter something.

An asterisk prompt specifies BASIC's keyboard mode. An exclamation point prompt signifies that you are running the BASIC CLI program. Other types of prompts include the line number prompt (000:) in the EDIT utility.

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 that are treated as one unit.

relational operator
A symbol used to compare two expressions. 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 that is stored in binary format. You create a SAVE file when you SAVE (or REPLACE) a program in working storage. You can LOAD, RUN, CHAIN, and SWAP these files.

screen field
An area on your terminal's screen. Use the Conversational Screen Maintenance (CSM) utility or the Screen Maintenance (SM) utililty to define screen fields in screen files.

screen file
A file created by the Conversational Screen Maintenance (CSM) utility or the Screen Maintenance (SM) utility; it usually has an .Sn extension, where n is the terminal type. A screen file can hold multiple screens.

search list
A list of directories that the AOS CLI scans if it cannot find the file you specified in the current directory. You can display and change your search list with the AOS CLI command SEARCHLIST. Search list applies only to AOS systems. See also pathname.

source file
An ASCII text 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 .BA extension.

statement
An instruction in a program. Each Business BASIC statement has a line number, a Business BASIC keyword, and arguments.

string
A combination of characters (letters, digits, spaces, and special characters).

subfile
A file that exists as a logical subsection of 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.

subroutine
A section of Business BASIC code that performs a specialized task. Subroutines are executed using the GOSUB and RETURN commands.

table file
A file used by the FM utility. A table file 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 FM, you need a table file (with a .TB extension) to describe the multiple fields and pages of your datafile and types of data.

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 usually create a tag file in sorted order using the TBUILD utility.

template
A character or a group of characters 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. On AOS Business BASIC systems you can use a plus to represent any string of characters in a filename or extension.

utility
A Business BASIC program that performs a specialized task. Utilities are SAVEd in the library. 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, TINPUT, PACK, and UNPACK. 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.

volume label file
A file used in the logical file database structure to hold the information about the logical file. The volume label file maps the logical file to the database file, which contains the actual data. A volume label file is indicated by a .VL extension.

word
A sequence of two adjacent bytes (bit locations 0-15).

working storage
The portion of memory used to develop programs and where programs that are

ENTERed or LOADed are stored. All programs and data stored in working storage are stored in SAVE file (binary) format. You can restore a program to ASCII format by LISTing it to a file.

End of Glossary

Appendix A
Subroutine, Utility, and Keyword Summary

This table describes the Business BASIC subroutines that come with the software package, their entry points, and the line numbers they occupy. When a subroutine occupies several sections of line numbers, a range of line numbers is given. This does not mean that subroutine occupies every line number within that range. The entry point is the beginning line number of the subroutine unless otherwise specified.

Subroutine	Line Numbers	Description
DELREC.SL	8600-8699	Deletes a record in a linked-available-record file (PARAM file structure) and places it on the deleted-record chain.
FINDFILE.SL	7800-7831	Finds a file (PARAM file structure) and builds a C1 array.
FORM.SL	9300-9544	Handles formatted screen fields for SM screens.
	9300	Entry point to input the screen field to X$ (if a string field) or to X (if a numeric field).
	9500	Entry point to output X$ (if string) or X (if numeric) to a screen field.
FORMIO.SL	9000-9079	Displays edited screen input/output for CSM screens.
GETCM.SL	7500-7560	Creates a BASIC CLI command.
	7500	Entry point to read a field pointed to by Q9; -1 returned in S if at end of command line.
	7550	Entry point to initialize routine variables and to read the common area into T9$.
GETLAST.SL	9950-9976	Retrieves the number of active records and the highest record in use in a linked-available-record file. (Logical file structure).

Table A-1. Business BASIC Subroutines (continues)

Subroutine	Line Numbers	Description
GETREC.SL	8400-8499	Gets the number of the next available record in a linked-available-record file (PARAM file structure).
INITINDEX.SL	7700-7799	Initializes an index file (PARAM file structure).
LFDATA.SL	9900-9915	Gets the file description for a file (logical file structure).
LINITINDEX.SL	7700-7749	Initializes an index file that was opened with the LOPEN file statement.
POSFL.SL	9610-9649	Positions the file pointer to a record in a data file. (PARAM file structure).
	9610	Entry point to position the pointer to the beginning of record R1.
	9612	Entry point to position the pointer to the byte offset in record R1.
PROTFORM.SL	9200-9269	Makes an SM formatted screen field protected or unprotected.
	9200	Entry point to convert a field to an unprotected field.
	9250	Entry point to convert a field to a protected field.
SCRNIO.SL	9200-9262	Enables edited screen input/output.
SFORM.SL	9100-9899	Uses several subroutines to provide formatted handling of CSM screens.
	9100	CLEARFORM clears all unprotected fields.
	9150	INITFORM sets the necessary STMAs and initializes variables.
	9175	ENDFORM resets the STMAs.
	9500	WRITEFORM outputs string and numeric fields.
	9650	READFUNC reads function keys.
	9700	OUTSCRN displays screen number X from the file opened on channel 15 and loads SCRN$.
	9722	Entry point in OUTSCRN to reread SCRN$ without displaying it.
	9750	WAITFUNC waits for a function key to be pressed and call READFUNC.
	9775	WAITCLEAR waits for function key F16 and clears the command line.
	9800	SCRNSET converts a field in F, or in F and R (if 0<F>100), into XROW, XCOL, XPOS, XWID, XFLGS, and XDEC.

Table A-1. Business BASIC Subroutines (continues)

Subroutine	Line Numbers	Description
	9850	ERROUT outputs an error message on the command line.
	9890	ERRCLEAR clears the error message from the screen.
UNFORM.SL	9100-9899	Uses several entry points to position the cursor to the input/output fields and handles unformatted fields.
	9100	Clears all unprotected fields.
	9150	Sets the necessary STMAs and block mode, and locks the keyboard.
	9155	(Used when lowercase is not needed.) Sets the necessary STMAs and block mode, and locks the keyboard.
	9175	Resets the STMAs, turns the interactive mode back on, and clears the terminal screen.
	9400	Reads a field from the screen and returns it in X$. Use 9650 and F (or F and R) to position to the screen field first. If the line is already in LINE$ (known by XLROW), screen field is not read. XLROW is reset whenever you output a field or allow the user to input a field, and whenever LINE$ is destroyed. Trailing spaces are truncated.
	9550	Outputs X$ to the field requested, left justifies, and handles screen overflow.
	9650	Positions the cursor to the requested field.
	9700	Displays screen number SCRN from the file opened on channel 15 and loads SCRN$.
	9735	Rereads SCRN$. (The screen is not displayed.)
	9750	Reads a function key.
	9800	Converts a field in F, or in F and R if F $<$ 100, into XROW, XCOL, XPOS, XWID, XFLGS, and XDEC.
	9850	ERROUT places an error message on the command line.
	9890	ERRCLEAR clears the error message from the screen.

Table A-1. Business BASIC Subroutines (concluded)

This table summarizes the utility programs supplied with the Business BASIC software package.

Utility	Description
ATTACH	Links your terminal to a detached job.
CLI	Executes BASIC CLI program.
CSM	Creates and maintains a screen file.
DBGEN	Builds files for the PARAM file structure.
DOC	Produces printable document files.
DOCTOC	Prepares the table of contents for a document file set up using DOC.
EDIT	Creates and/or edits text files.
FILES	Displays the names of files in the current directory.
FILESORT	Sorts a data file.
FM	Provides file maintenance functions for data files and table files (PARAM file structure).
FMLOG	Displays an FM log file.
FMPRINT	Displays an FM data file.
FMTABPRINT	Prints the records in an FM table file.
IBUILD	Builds an index file from a sorted tag file, a sorted data file, or an index file.
INDEXBLD	Builds or rebuilds an index file (PARAM file structure).
INDEXCALC	Calculates index and data file information.
INDEXPRT	Prints an index file for a logical or PARAM file database structure.
INDEXVRFY	Verifies the structure of an index file.
INITFILE	Creates and/or initializes index or data files (PARAM file structure).
IREBLD	Rebuilds an index file (logical file structure).
LFM	Provides file maintenance functions for the logical file structure.
LFU	Creates and manipulates files (logical file structure).
LIBRARY	Displays the names of files in the library directory.
LINDEXBLD	Builds or rebuilds an index file (logical file structure).
LOCKS	Displays your current locks.
LRELINK	Recreates a deleted-record chain for a linked-available-record file (logical file structure).
LSPEED	Changes the default line speeds.
LXFER	Copies one logical file to another logical file (logical file structure).
MOVETABREC	Copies FM table file records.
OPEN	Opens files in the PARAM file structure.
PARAMCON	Converts a PARAM file structure into a logical file structure.
PARAMPRT	Prints the contents of the PARAM file.

Table A-2. Business BASIC Utilities (continues)

Utility	Description
PD	Displays information about a program in working storage or in a SAVE file.
PED	Displays the system status.
PORTS	Displays processes on the system.
QFILESORT	Quickly sorts a data file.
RELINK	Recreates deleted-record chain of linked-available-record file (PARAM file structure).
RENUM	Renumbers selective lines of a program listing file.
RNAM	Renames program variables.
SCHANS	Displays system channel assignments.
SIZE	Displays the working storage space allocations.
SM	Creates and maintains screen files.
STAT	Displays the status of all jobs.
TABBUILD	Defines arrays for FM table file.
TBUILD	Builds a tag file.
TERM	Changes certain terminal key functions.
UCHANS	Displays your channel assignments.
VAR	Lists the variables in a SAVE file or a program in working storage.
XBUILD	Builds an index file from a data file.

Table A-2. Business BASIC Utilities (concluded)

This table summarizes the commands used with the BASIC CLI.

Command	Description
AOS	Enables you to access the AOS CLI (AOS only).
APPEND	Appends a file to another file.
ASG	Assigns a device for exclusive use (RDOS only).
ATTACH	Attaches your terminal to a detached job (RDOS only).
BLDCOM	Builds a documentation file for PRTCOM.
BUILD	Builds a command file.
BYE	Terminates Business BASIC.
CCONT	Creates a contiguous RDOS file or a random AOS file.
CDIR	Creates a subdirectory.
CHAIN	Executes a program.
CHATR	Changes a file's attributes (RDOS only).
CHLAT	Changes a file's link access attributes (RDOS only).
CPART	Creates a secondary partition (RDOS) or a control point directory (AOS).
CRAND	Creates a random file.
CREATE	Creates a random file (AOS) or sequential file (RDOS).
DBMOVE	Moves logical file structures from one directory to another.
DELETE	Deletes a file, directory, or partition.
DIR	Changes the current directory.
DISK	Displays the amount of disk space used and remaining.
DUMP	Copies one or more disk files in DUMP format to an output file.
EQUIV	Renames a device (RDOS only).
FDUMP	Fast dumps one or more files to magnetic tape (RDOS only).
FILCOM	Compares two files word by word.
FLOAD	Fast loads FDUMPed files (RDOS only).
FPRINT	Displays the contents of a disk file.
FREE	Releases a device from exclusive use (RDOS only).
GDIR	Displays the current directory name.
GQUE	Gets the default queue name.
GSDIR	Displays the current system directory name (RDOS only).
GSYS	Displays the current system name (RDOS only).
GTOD	Displays the time and date.
INIT	Initializes a device (RDOS 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.

Table A-3. BASIC CLI Commands (continues)

Command	Description
MDIR	Displays the master directory name (RDOS only).
MOVE	Moves files from one directory to another.
POP	Exits from the BASIC CLI and clears the common area.
PRINT	Prints a file to the default output queue.
PRTCOM	Prints a BLDCOM documentation file.
QUIT	Exits from the BASIC CLI but retains the common area.
RELEASE	Releases a device or a directory (RDOS only).
RENAME	Changes the name of a file.
SDIR	Sets the system directory (RDOS only).
SLINE	Selects a line (terminal) and attaches a job to it (RDOS 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 only).
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.
VLPRINT	Displays the contents of a volume-label (.VL) file.
XFER	Copies one file to another file.

Table A-3. BASIC CLI Commands (concluded)

This table summarizes the commands and statements that make up the Business BASIC language.

Keyword	Usage	Description
. (period)	Command	Sends the line in the edit buffer to working storage.
.A	Command	Appends a string to a line in the edit buffer.
.C	Command	Changes a string in a line in the edit buffer and passes the line to working storage.
.E	Command	Changes a string in a line in the edit buffer.
.I	Command	Changes a line in the edit buffer.
.P	Command	Displays the contents of the edit buffer.
BLOCK READ	Command, Statement	Retrieves blocks from a file or common area.
BLOCK WRITE	Command, Statement	Places blocks of data in a file or the common area.
BYE	Command, Statement	Logs you off Business BASIC.
CHAIN	Command, Statement	Executes a program.
CLOSE	Command, Statement	CLoses all open files or a specific file.
CON	Command	Continues execution of a stopped program.
DATA	Statement	Specifies values for variables in READ statements.
DEF	Statement	Defines your own functions.
DELAY	Command, Statement	Delays execution of the next program statement or command.
DELETE	Command, Statement	Deletes a file.
DELREC	Command, Statement	Deletes a record from a logical file and puts it on the deleted-record chain.
DIM	Command, Statement	Sets dimensions for arrays and strings.
DIR	Command, Statement	Displays the current directory or moves you to another directory.
END	Statement	Terminates program execution.
ENTER	Command, Statement	Brings program source statements into working storage and merges them with the statements already there (if any).
ERASE	Command, Statement	Deletes program statements.
EXTRACT	Statement	Extracts the next field from a string.
FOR/NEXT	Statement	Defines a program loop.

Table A-4. Business BASIC Commands and Statements (continues)

Keyword	Usage	Description
GETREC	Command, Statement	Gets an available record from a data file that was LOPENed.
GOSUB/ RETURN	Statement	Transfers program control to and from a subroutine.
GOTO	Statement	Transfers program control to a specific statement.
IF	Statement	Transfers program control or executes a statement if a condition is true.
INPUT	Command, Statement	Enters numeric and string data from a terminal or a file.
INPUT USING	Command, Statement	Enters character data from a terminal or a file and allows an error and a delimiter trap.
KADD	Command, Statement	Adds a key to an index file.
KDEL	Command, Statement	Deletes a key from an index file.
KFIND	Command, Statement	Finds a key in an index file.
KNEXT	Command, Statement	Locates the next key in an index file (initially used after a KFIND).
LET	Command, Statement	Assigns a value to a variable.
LIST	Command	Lists the character contents of a program to a file or a terminal.
LISTH	Command	Lists the character contents of a program with header information to a file or terminal.
LOAD	Command	Places a SAVE file in working storage.
LOCK/ UNLOCK	Command, Statement	Locks or unlocks files and records.
LOPEN FILE	Command, Statement	Opens a logical file.
LREAD FILE	Command, Statement	Reads a logical record.
LWRITE FILE	Command, Statement	Writes a logical record.
MSG	Command	Sends a message to another terminal or process.
MTDIO	Command, Statement	Directs input and output control of a magnetic tape or cassette.
NEW	Command, Statement	Clears working storage.

Table A-4. Business BASIC Commands and Statements (continues)

Keyword	Usage	Description
NEXT	Statement	Defines the end of a program loop and increments or decrements loop control variables.
ON ERR	Statement	Traps an error in your program.
ON GOTO, ON GOSUB	Statement	Conditionally transfers control to other statements.
ON IKEY	Statement	Traps an interrupt in your program.
OPEN FILE	Command, Statement	Opens a file or a device in an access mode and gives it a channel number.
PACK	Command, Statement	Composes a record string.
PAGE	Command	Sets the page width.
POSITION FILE	Command, Statement	Positions the file pointer to a byte in a disk file.
PRINT	Command, Statement	Sends numeric and string data to a terminal or a file.
PRINT USING	Statement	Formats output to a terminal, file, or device.
QADD	Command, Statement	Quad precision addition.
QDIV	Command, Statement	Quad precision division.
QLOAD	Command, Statement	Loads a numeric expression into a string variable.
QMUL	Command, Statement	Quad precision multiplication.
QSTORE	Command, Statement	Converts a quad precision string into a double precision variable.
QSUB	Command, Statement	Quad precision subtraction.
RANDOMIZE	Command, Statement	Reseeds the random number generator.
READ	Statement	Assigns DATA statement values to variables.
READ FILE	Command, Statement	Reads data from a file.
REM	Statement	A remark statement.
RENAME	Command, Statement	Changes the directory entry for a file.
RENUMBER	Command	Renumbers lines in the current program.
REPLACE	Command, Statement	Replaces a SAVE program with a new program.
RESTORE	Command, Statement	Resets the DATA list pointer.

Table A-4. Business BASIC Commands and Statements (continues)

Keyword	Usage	Description
RETURN	Statement	Transfers control from a subroutine back to its calling point.
RFORM	Statement	Defines a record string format.
RUN	Command	Executes a program.
SAVE	Command, Statement	Saves a program and its variables.
SCANUNTIL	Statement	Scans a string until characters in a substring are found.
SCANWHILE	Statement	Scans a string while characters match those in a substring.
SIZE	Command	Determines the size of a program in working storage.
STMA	Command, Statement	User system calls to examine or modify some aspects of a job.
STOP	Statement	Halts program execution.
STRPOS	Command, Statement	Finds the starting position of a substring in a string.
SWAP	Command, Statement	Executes a program; then returns to the original program.
TAB	Command	Sets the tab width.
TINPUT	Command, Statement	Performs an INPUT instruction in a specified amount of time.
UCALL	Command, Statement	Calls an assembly language subroutine from Business BASIC.
UNPACK	Command, Statement	Decodes a record string.
VALUE	Command, Statement	Converts a string to a number.
WRITE FILE	Command, Statement	Sends length-sensitive data to a file or terminal.

Table A-4. Business BASIC Commands and Statements (concluded)

This table summarizes the functions that are supported by Business BASIC.

Keyword	Description
ABS	Returns the absolute value of a number.
AERM$	Puts an error message into a string.
AND	Sets bits based on the result of a logical AND of two expressions.
ASC	Gives the ASCII value of a string.
CHR$	Places the binary value of a number in a string.
CRM$	Crams every three bytes of a string into two bytes.
EOF	Checks for the end of the file.
ERM$	Retrieves an error message.
FILL$	Fills a string or substring with a value.
GPOS	Returns the current position of the file pointer.
INT	Truncates a number to make it an integer.
LEN	Finds the current length of a string.
MAX	Finds the larger of two expressions.
MIN	Finds the smaller of two expressions.
MOD	Finds the remainder after dividing two expressions.
OR	Sets bits based on the result of a logical inclusive OR of two expressions.
POS	Determines the position of a substring in a string.
RND	Produces a pseudo-random integer.
SGN	Determines the sign (positive or negative) of an expression.
SHFT	Moves the bits of an expression left or right with zero fill.
SQR	Finds the square root of an expression (truncated to an integer).
SYS	Returns system information.
TRUN$	Truncates a string.
UCM$	Uncrams a crammed string.
VAL	Converts a string of digits to a numeric value.

Table A-5. Business BASIC Functions

This table summarizes the Boolean logic operators supported by Business BASIC.

Operator	Description
AND	Joins two expressions into a single expression and reduces that expression to either 1 (true) or 0 (false). The expression is true only if both expressions evaluate to true.
NOT	Gives an expression the opposite Boolean value.
OR	Joins two expressions into a single expression and reduces that expression to either 1 (true) or 0 (false). The expression is true when either both expressions evaluate to true or when one expression evaluates to true.

Table A-6. Boolean Logic Operators

End of Appendix

Appendix B
ASCII Character Sets

Key(s)	Definition	Octal	Decimal	Hexadecimal
^@	Null	000	000	000
^A	Print form	001	001	001
^B		002	002	002
^C	Enable blink	003	003	003
^D	Disable blink	004	004	004
^E	Read cursor addr	005	005	005
^F		006	006	006
^G	Bell	007	007	007
^H	Cursor home (HOME)	010	008	008
^I	Tab (TAB)	011	009	009
^J	New line (NEW LINE)	012	010	00A
^K	Erase to EOL (ERASE EOL)	013	011	00B
^L	Erase page (ERASE PAGE)	014	012	00C
^M	Carriage return (CR)	015	013	00D
^N	Start blink	016	014	00E
^O	End blink	017	015	00F
^P	Position cursor	020	016	010
^Q	Print	021	017	011
^R	Roll enable	022	018	012
^S	Roll disable	023	019	013
^T	Start underscore	024	020	014
^U	End underscore	025	021	015
^V		026	022	016
^W	Cursor up	027	023	017
^X	Cursor right	030	024	018
^Y	Cursor left	031	025	019
^Z	Cursor down	032	026	01A
^|	Escape (ESC)	033	027	01B
^\	Start dim	034	028	01C
^]	End dim	035	029	01D
^^	Function key prefix	036	030	01E
^_	Cursor addr response	037	031	01F

Table B-1. Standard ASCII Character Set (continues)

Key	Octal	Decimal	Hexadecimal	Key	Octal	Decimal	Hexadecimal
space bar	040	032	020	O	117	079	04F
!	041	033	021	P	120	080	050
"	042	034	022	Q	121	081	051
#	043	035	023	R	122	082	052
$	044	036	024	S	123	083	053
%	045	037	025	T	124	084	054
&	046	038	026	U	125	085	055
'	047	039	027	V	126	086	056
(	050	040	028	W	127	087	057
)	051	041	029	X	130	088	058
*	052	042	02A	Y	131	089	059
+	053	043	02B	Z	132	090	05A
,	054	044	02C	[	133	091	05B
-	055	045	02D	\	134	092	05C
.	056	046	02E	]	135	093	05D
/	057	047	02F	^	136	094	05E
0	060	048	030	_	137	095	05F
1	061	049	031	`	140	096	060
2	062	050	032	a	141	097	061
3	063	051	033	b	142	098	062
4	064	052	034	c	143	099	063
5	065	053	035	d	144	100	064
6	066	054	036	e	145	101	065
7	067	055	037	f	146	102	066
8	070	056	038	g	147	103	067
9	071	057	039	h	150	104	068
:	072	058	03A	i	151	105	069
;	073	059	03B	j	152	106	06A
<	074	060	03C	k	153	107	06B
=	075	061	03D	l	154	108	06C
>	076	062	03E	m	155	109	06D
?	077	063	03F	n	156	110	06E
@	100	064	040	o	157	111	06F
A	101	065	041	p	160	112	070
B	102	066	042	q	161	113	071
C	103	067	043	r	162	114	072
D	104	068	044	s	163	115	073
E	105	069	045	t	164	116	074
F	106	070	046	u	165	117	075
G	107	071	047	v	166	118	076
H	110	072	048	w	167	119	077
I	111	073	049	x	170	120	078
J	112	074	04A	y	171	121	079
K	113	075	04B	z	172	122	07A
L	114	076	04C	{	173	123	07B
M	115	077	04D	|	174	124	07C
N	116	078	04E	}	175	125	07D
				~	176	126	07E
				DEL	177	127	07F

Table B-1. Standard ASCII Character Set (concluded)

Character	Octal	Decimal	Character	Octal	Decimal
space	240	160	Õ	325	213
space	241	161	Ø	326	214
space	242	162	Œ	327	215
space	243	163	Ú	330	216
space	244	164	Ù	331	217
space	245	165	Û	332	218
¤	246	166	Ü	333	219
¢	247	167	space	334	220
£	250	168	space	335	221
space	251	169	space	336	222
space	252	170	space	337	223
!	253	171	á	340	224
¿	254	172	à	341	225
space	255	173	â	342	226
...	...		ä	343	227
space	271	185	ã	344	228
`	272	186	å	345	229
§	273	187	æ	346	230
°	274	188	ç	347	231
¨	275	189	é	350	232
´	276	190	è	351	233
ˆ	277	191	ê	352	234
Á	300	192	ë	353	235
À	301	193	í	354	236
Â	302	194	ì	355	237
Ä	303	195	î	356	238
Ã	304	196	ï	357	239
Å	305	197	ñ	360	240
Æ	306	198	ó	361	241
Ç	307	199	ò	362	242
É	310	200	ô	363	243
È	311	201	ö	364	244
Ê	312	202	õ	365	245
Ë	313	203	ø	366	246
Í	314	204	œ	367	247
Ì	315	205	ú	370	248
Î	316	206	ù	371	249
Ï	317	207	û	372	250
Ñ	320	208	ü	373	251
Ó	321	209	β	374	252
Ò	322	210	space	375	253
Ô	323	211	space	376	254
Ö	324	212	space	377	255

Table B-2. DG International Symbols (8-bit ASCII Character Set)

End of Appendix

Appendix C
Example Programs

This appendix contains examples of programs in the logical database structure and the PARAM database structure.

Setting Up a Logical File Database

This example creates a logical database that can be used with the mailing list program that follows. The one physical file and three logical files are created with the logical file structure commands LFU PCREATE and LFU LCREATE. You use these commands after you define your file layout and run INDEXCALC to determine the number of sectors you need. The *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI* contains explanations of LFU and INDEXCALC.

The logical files consist of two index files, MEMBER and NAME, and one linked-available-record file, CLUB. They have been designed so that MEMBER uses a 4-byte key field and does not allow duplicate keys while NAME uses a 24-byte key field and permits duplicate keys. Both use a 50 percent blocking factor. CLUB holds 200 records, each 110 bytes long. The physical file, CLUBFILE, is a random file.

Once you have designed your files, run INDEXCALC to calculate the numbers for the index files MEMBER and NAME (in that order). This also provides you with the number of sectors required by the data file CLUB.

```
* RUN "INDEXCALC
INDEXCALC VERSION X.XX

BYTES PER KEY : 4
BYTES PER DATA RECORD : 110
MAXIMUM NUMBER OF DATA RECORDS : 200
INDEX BLOCKING FACTOR (% PERCENT) [50]: 50
INDEX BLOCK SIZE (512 or 2048) [512]: 512
DUPLICATE KEYS ALLOWED? (Y or N) [N]: N

63 MAXIMUM KEYS PER INDEX BLOCK
32 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
7 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
9 BLOCKS (512 bytes each) IN INDEX
9 SECTORS IN INDEX
```

```
44 SECTORS IN DATA FILE

CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y or N) [N]: Y

BYTES PER KEY : 24
BYTES PER DATA RECORD : 110
MAXIMUM NUMBER OF DATA RECORDS : 200
INDEX BLOCKING FACTOR (% PERCENT) [50]: 50
INDEX BLOCK SIZE (512 or 2048) [512]: 512
DUPLICATE KEYS ALLOWED? (Y or N) [N]: Y

18 MAXIMUM KEYS PER INDEX BLOCK
9 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
23 BLOCK(S) AT LEVEL 2
3 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
28 BLOCKS (512 bytes each) IN INDEX
28 SECTORS IN INDEX
44 SECTORS IN DATA FILE

CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y or N) [N]: N

*
```

Next, use this information from INDEXCALC and type in the following commands to create the physical file and the three logical files (this example is executing LFU from the BASIC CLI and using its command line format):

```
* !LFU PCREATE CLUBFILE 0

* !LFU LCREATE MEMBER CLUBFILE I 512 8 4 50 N

* !LFU LCREATE NAME CLUBFILE I 512 27 24 50 Y

* !LFU LCREATE CLUB CLUBFILE L 110 200 N
```

These command lines created:

- A physical file CLUBFILE.
- An index file MEMBER with 512-byte blocks, a last usable block number of 8 (since the 9 blocks in the index mean blocks 0-8), a 4-byte member number field, a 50 percent blocking factor, and no duplicate keys.
- An index file NAME with 512-byte blocks, a last usable block number of 27 (since the 28 blocks in the index mean blocks 0-27), a 24-byte name field, a 50 percent blocking factor, and duplicate keys are allowed.
- A linked-available-record file CLUB with records of 110 bytes each, 200 records in the file, and no null filling of the file.

The following program can be used to add records to the database you created.

```
0010 REM * PROGRAM ID: LCREATE - Using a logical file database structure
0020 ON ERR THEN GOTO 0880
0030 CLOSE
```

```
0040 DIM LFTABL$[78],B$[544],X$[30],ER$[132],MKEY$[4],REC$[110]
0050 DIM LAST$[24],FIRST$[16],ADDR$[25],CITY$[25],STATE$[3],ZIP$[5]
0055 DIM PHONE$[8]
0060 LET MKEY$,REC$=FILL$(0)
0070 REM
0080 REM ** Routine to OPEN files and set up the C1 array
0090 LET LFTABL$=FILL$(0)
0100 LOPEN FILE[1,B$],"CLUB"
0110 LOPEN FILE[2,B$],"MEMBER"
0120 LOPEN FILE[3,B$],"NAME"
0130 RFORM ZJLA24A16A25A20A3A5A8
0140 REM
0260 LET R1=-1
0270 REM
0280 REM ** Input variables
0290 PRINT @(-30);@(4,15);"Input Screen"
0300 PRINT
0310 INPUT "Member #: ",MEM
0320 INPUT "Last Name: ",LAST$
0330 INPUT "First Name: ",FIRST$
0340 INPUT "Address: ",ADDR$
0350 INPUT "City: ",CITY$
0360 INPUT "State: ",STATE$
0370 INPUT "Zip: ",ZIP$
0380 INPUT "Phone: ",PHONE$
0390 REM
0400 REM ** SETUP RECORD
0410 PACK 0130,REC$,1,MEM,LAST$,FIRST$,ADDR$,CITY$,STATE$,ZIP$,PHONE$
0420 REM
0510 PACK "L",MKEY$,MEM
0520 REM
0530 REM ** UPDATE FILES
0540 GETREC 1,R1
0545 IF R1<0 THEN GOTO 680
0550 LET SR1=R1
0560 LET T=30
0570 LOCK 1,1,R1,T
0580 IF T=57 THEN GOTO 0560
0590 REM
0600 LWRITE FILE[1,R1],REC$
0610 KADD 2,B$,MKEY$,R1
0620 IF R1<=0 THEN GOTO 0700 : ERROR IN ADDING KEY TO MEMBER INDEX FILE
0630 KADD 3,B$,LAST$,R1
0640 IF R1<=0 THEN GOTO 0760 : ** ERROR IN ADDING RECORD TO NAME INDEX
0650 UNLOCK
0660 REM
0670 GOTO 0280 : ** Input variables
0680 CLOSE
0690 END
0700 REM ** ERROR IN ADDING KEY TO MEMBER INDEX FILE
0710 LET R1=SR1
```

```
0720 DELREC 1,R1
0730 PRINT @(20,5);"ERROR IN MEMBER INDEX-KEY & DATA RECORD NOT ADDED"
0740 GOTO 0840 : ** ERROR ROUTINE FOR FILE HANDLING
0750 REM
0760 REM ** ERROR IN ADDING RECORD TO NAME INDEX
0770 LET R1=SR1
0780 DELREC 1,R1
0790 LET R1=SR1
0800 KDEL 2,B$,MKEY$,R1
0810 PRINT @(20,5);"ERROR IN NAME INDEX - KEYS & DATA RECORD NOT ADDED"
0820 GOTO 0840 : ** ERROR ROUTINE FOR FILE HANDLING
0830 REM
0840 REM ** ERROR ROUTINE FOR FILE HANDLING
0850 PRINT @(21,5);"MEMBER NO.: ";MEM;" Last Name: ";LAST$
0860 GOTO 0970
0870 REM
0880 REM ** GENERAL ERROR ROUTINE
0890 IF SYS(30)>63 AND SYS(7)<0 THEN
0900   LET ER=SYS(31)
0910   LET ER$=AERM$(SYS(31))
0920 ELSE
0930   LET ER=SYS(7)
0940   LET ER$=ERM$(SYS(7))
0950 END IF
0960 PRINT @(20,5);"ERROR # ";ER;"= ";ER$
0970 INPUT USING "",@(23,5),"RECORD ERROR INFORMATION & PRESS 'NL'" ,X$
0980 CLOSE
0990 END
```

Setting Up a PARAM File Database

Creating a PARAM file database involves multiple steps. You must design your database. If you do not have a PARAM file, you must create one and then use the File Maintenance (FM) utility to build record 0 of the file. (A detailed explanation of FM is supplied in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*) You must also run INDEXCALC to determine the number of keys per index block, the number of blocks needed for the index files, and the number of sectors needed for the data file. Next, you must create your subfiles and then run INITFILE to initialize them and enter the necessary information in the PARAM file.

To create a PARAM file, type in:

```
* !CRAND PARAM
* RUN "FM
```

When FM prompts you for a filename, type in PARAM. FM then clears the screen and displays a series of prompts for you to fill in. To set up record 0,

you must ADD a record (for this example, press function key 4) and then supply the following answers:

```
Parameter record #          = 0
Sub file name               = PARAM
Master file name            = PARAM
Sub file position           = 0
Sub file record length      = 42
Last record number          = 100
Highest record number used  = 0
```

The subfile position prompt is asking for the byte pointer to the start of the subfile in the physical file; in this case, it's the beginning of the file. The last record number prompt refers to the number of subfiles that will be catalogued in the PARAM file. Since you are just setting up the file, no records have been used, which is why the highest record number used is set to 0.

When you finish building record 0, you must STOP FM (in this example, press function key 10).

When you have the PARAM file set up, you can create your database. This database contains three subfiles and one master file. The subfiles consist of two index files, MEMBER and NAME, and one linked-available-record file, CLUB. They have been designed so that MEMBER uses a 4-byte key field and does not allow duplicate keys while NAME uses a 24-byte key field and permits duplicate keys. Both use a 50 percent blocking factor. CLUB holds 200 records, each 110 bytes long. The master file, CLUBFILE, is a random file.

Run INDEXCALC to calculate the numbers for the index files MEMBER and NAME (in that order). This also provides you with the number of sectors required by the data file CLUB.

```
* RUN "INDEXCALC"
INDEXCALC VERSION X.XX

BYTES PER KEY : 4
BYTES PER DATA RECORD : 110
MAXIMUM NUMBER OF DATA RECORDS : 200
INDEX BLOCKING FACTOR (% PERCENT) [50]: 50
INDEX BLOCK SIZE (512 or 2048) [512]: 512
DUPLICATE KEYS ALLOWED? (Y or N) [N]: N

63 MAXIMUM KEYS PER INDEX BLOCK
32 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
7 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
9 BLOCKS (512 bytes each) IN INDEX
9 SECTORS IN INDEX
44 SECTORS IN DATA FILE

CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y or N) [N]: Y

BYTES PER KEY : 24
BYTES PER DATA RECORD : 110
```

```
MAXIMUM NUMBER OF DATA RECORDS : 200
INDEX BLOCKING FACTOR (% PERCENT) [50]: 50
INDEX BLOCK SIZE (512 or 2048) [512]: 512
DUPLICATE KEYS ALLOWED? (Y or N) [N]: Y

18 MAXIMUM KEYS PER INDEX BLOCK
9 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
23 BLOCK(S) AT LEVEL 2
3 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
28 BLOCKS (512 bytes each) IN INDEX
28 SECTORS IN INDEX
44 SECTORS IN DATA FILE

CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE (Y or N) [N]: N

*
```

Now run INITFILE to create these files.

```
* RUN "INITFILE
  INDEX (0), DATA (1), STOP (2) [0]: 0
  SUB FILE NAME: MEMBER
  FILE NOT IN PARAM FILE!
  DO YOU WISH TO ADD? (Y OR N) [Y]: Y
  MASTER FILE NAME: CLUBFILE
  INDEX BLOCK SIZE (512 or 2048) [512]: 512
  BYTE OFFSET TO SUB FILE: 0
  MAXIMUM NUMBER OF INDEX BLOCKS: 9
  BYTES PER KEY: 4
  BLOCKING FACTOR (% PERCENT) [50]: 50
  DUPLICATE KEYS ALLOWED? (Y OR N) [N]: N
  INDEX (0), DATA (1), STOP (2) [0]: 0
  SUB FILE NAME: NAME
  FILE NOT IN PARAM FILE!
  DO YOU WISH TO ADD? (Y OR N) [Y]: Y
  MASTER FILE NAME: CLUBFILE
  INDEX BLOCK SIZE (512 or 2048) [512]: 512
  BYTE OFFSET TO SUB FILE: 4608
  MAXIMUM NUMBER OF INDEX BLOCKS: 28
  BYTES PER KEY: 24
  BLOCKING FACTOR (% PERCENT) [50]: 50
  DUPLICATE KEYS ALLOWED? (Y OR N) [N]: Y
  INDEX (0), DATA (1), STOP (2) [0]: 1
  SUB FILE NAME: CLUB
  FILE NOT IN PARAM FILE!
  DO YOU WISH TO ADD? (Y OR N) [Y]: Y
  MASTER FILE NAME: CLUBFILE
  BYTE OFFSET TO SUB FILE: 18944
  BYTES PER DATA RECORD: 110
  MAXIMUM NUMBER OF DATA RECORDS: 200
```

```
SHOULD FILE BE NULL FILLED? N
INDEX (0), DATA (1), STOP (2) [0]: 2
```

You now have in your database the following files:

- A master file CLUBFILE. It contains the subfiles MEMBER, NAME, and CLUB.
- An index file MEMBER with nine 512-byte blocks, a 4-byte member number field, a 50 percent blocking factor, and no duplicate keys.
- An index file NAME with 28 512-byte blocks, a 24-byte name field, a 50 percent blocking factor, and duplicate keys are allowed.
- A linked-available-record file CLUB with 200 records of 110 bytes each.

The following program can be used to add records to the database you created.

```
0010 REM ** PROGRAM ID: CREATE
0020 PRINT @(-30);@(5,20);"FILES ARE BEING CREATED"
0030 CLOSE
0040 DIM C1[2,3],B$[544],MX$[18],NX$[18]
0050 DIM X$[30]
0060 DIM LAST$[24],FIRST$[16],ADDR$[25],CITY$[25],STATE$[3],ZIP$[5],PHONE$[8]
0070 DIM MKEY$[4],NKEY$[24],LNAME$[24],FNAME$[24]
0080 DIM REC$[110]
0090 REM
0100 REM ** Routine to OPEN files and set up the C1 array
0110 LET B$="CLUB,5,MEMBER,5,NAME,5",FILL$(0)
0120 BLOCK WRITE B$
0130 SWAP "OPEN"
0140 BLOCK READ B$
0145 UNPACK "JJ",B$,ERRIN,ERRNO
0150 IF ERRIN <>-1 THEN GOTO 0180
0160 PRINT "ERROR # " ;ASC(B$[3,4]);" - ";B$[5,512]
0170 END
0180 LET K=1
0190 FOR I=0 TO 2
0200   FOR J=0 TO 3
0210     LET C1[I,J]=ASC(B$[K,K+3])
0220     LET K=K+4
0230   NEXT J
0240 NEXT I
0250 LET MX$=CHR$(C1[1,0],2),CHR$(C1[1,1],4),CHR$(0,2),"MEMBER",FILL$(0)
0260 LET NX$=CHR$(C1[2,0],2),CHR$(C1[2,1],4),CHR$(0,2),"NAME",FILL$(0)
0270 LET F%,R1=0
0280 REM
0290 REM ** Input variables
0300 INPUT "Member #: ",MEM
0310 INPUT "Last Name: ",LAST$
0320 INPUT "First Name: ",FIRST$
0330 INPUT "Address: ",ADDR$
0340 INPUT "City: ",CITY$
```

```
0350 INPUT "State: ",STATE$
0360 INPUT "Zip: ",ZIP$
0370 INPUT "Phone: ",PHONE$
0380 REM
: This example uses LET statements to set up the record;
: however, you could have used the PACK statement instead,
: which would reduce the amount of code you have.
0390 REM ** SETUP RECORD
0400 LET REC$[1,2]=CHR$(1,2),FILL$(0)
0410 LET REC$[3,6]=CHR$(MEM,4)
0440 LET REC$[7,30]=LAST$,FILL$[0]
0450 LET REC$[31,46]=FIRST$,FILL$[0]
0460 LET REC$[47,71]=ADDR$,FILL$[0]
0470 LET REC$[72,96]=CITY$,FILL$[0]
0480 LET REC$[97,99]=STATE$,FILL$[0]
0490 LET REC$[100,104]=ZIP$,FILL$[0]
0500 LET REC$[105,112]=PHONE$,FILL$[0]
0510 LET REC$[113,120]=FILL$(0)
0520 LET MKEY$=REC$[3,6]
0530 LET NKEY$=REC$[7,30]
0540 REM
0550 REM ** UPDATE FILES
0560 GOSUB 8400 : \ GETREC.SL
0570 LET SR1=R1
0580 LET T=30
0590 LOCK 1,"CLUB",R1*C1[F%,3],C1[F%,3],T
0600 IF T=57 THEN GOTO 0580
0610 GOSUB 9610 : \ POSFL.SL
0620 WRITE FILE[C%],REC$
0630 KADD MX$,B$,MKEY$,R1
0640 IF R1<=0 THEN GOTO 0720 : ** ERROR IN ADDING KEY TO MEMBER INDEX FILE
0650 KADD NX$,B$,NKEY$,R1
0660 IF R1<=0 THEN GOTO 0810 : ** ERROR IN ADDING RECORD TO NAME INDEX
0670 UNLOCK
0680 REM
0690 GOTO 0290 : ** Input variables
0700 CLOSE
0710 END
0720 REM ** ERROR IN ADDING KEY TO MEMBER INDEX FILE
0730 LET R1=SR1
0740 GOSUB 8600 : \ DELREC.SL
0750 PRINT @(20,5);"ERROR OCCURRED IN MEMBER INDEX - DATA RECORD NOT ADDED"
0760 PRINT @(21,5);"Member No.: ";MEM," Last Name: ";LNAME$
0770 INPUT USING "",@(23,5),"DEPRESS 'NL' TO RETURN TO MENU" ,X$
0780 CLOSE
0790 END
0800 REM
0810 REM ** ERROR IN ADDING RECORD TO NAME INDEX
0820 LET R1=SR1
0830 GOSUB 8600 : \ DELREC.SL
0840 LET MKEY$=REC$[3,6]
```

```
0850 LET R1=SR1
0860 KDEL MX$,B$,MKEY$,R1
0870 PRINT @(20,5);"ERROR OCCURRED IN NAME INDEX; NO RECORD ADDED TO DATA FILE"
0880 PRINT @(21,5);"Member No.: ";MEM," Last Name: ";LNAME$
0890 INPUT USING "",@(23,5),"DEPRESS 'NL' TO RETURN TO MENU",X$
0900 CLOSE
0910 END
0920 REM
8400 REM \ GETREC.SL
8405 LET C%=C1[F%,0]
8410 LET W0=C1[F%,1]
8415 POSITION FILE[C%,W0]
8420 READ FILE[C%],X0%,Y0,Z0
8430 IF Y0=-1 THEN GOTO 8472
8440 LET R1=Y0
8450 GOSUB 9610 ! \ POSFL.SL
8460 READ FILE[C%],Y0%,X0
8470 IF Y0%>0 THEN LET Y0%=1/0
8471 GOTO 8480
8472 IF Z0=-1 THEN GOTO 8494
8474 LET Z0=Z0+1
8475 LET X0=Y0
8476 LET Y0=Z0
8478 IF Z0>C1[F%,2] THEN GOTO 8496
8480 POSITION FILE[C%,W0]
8491 WRITE FILE[C%],X0%,X0,Z0
8492 LET R1=Y0
8493 RETURN
8494 LET R1=Z0
8495 RETURN
8496 LET R1=-1
8497 RETURN
8499 REM * END GETREC.SL
8600 REM \ DELREC.SL
8602 LET X0=R1
8605 IF R1<=0 THEN LET X0=1/0
8610 LET C%=C1[F%,0]
8615 LET W0=C1[F%,1]
8620 POSITION FILE[C%,W0]
8630 READ FILE[C%],X0%,Y0
8650 GOSUB 9610 ! \ POSFL.SL
8652 READ FILE[C%],Y0%
8655 IF Y0%<>0 THEN GOTO 8658
8656 PRINT @(25,50); RECORD ALREADY DELETED
8657 RETURN
8658 GOSUB 9610 ! \ POSFL.SL
8660 LET Y0%=0
8670 WRITE FILE[C%],Y0%,Y0
8680 POSITION FILE[C%,W0]
8692 WRITE FILE[C%],X0%,X0
8695 RETURN
```

```
8699 REM * END DELREC.SL
9610 REM \ POSFL.SL
9611 LET V%=0
9612 REM \ POSFL WITH OFFSET V%
9613 LET C%=C1[F%,0]
9615 IF R1<0 THEN LET V%=1/0
9620 IF R1>C1[F%,2] THEN LET V%=1/0
9625 LET R8=R1*C1[F%,3]
9630 LET R9=C1[F%,1]+R8+V%
9640 POSITION FILE[C%,R9]
9645 RETURN
9649 REM * END POSFL.SL
```

Comparing Logical, PARAM Code

There is a marked difference in the code needed to open files in the logical file database structure and the PARAM file database structure. The logical files require less code. These next two pieces of code illustrate opening files in the logical database and in the PARAM database.

Code to open files in the logical file database structure:

```
0010 DIM LFTABL$[78],B$[544]
0020 LET LFTABL$=FILL$(0)
0030 LOPEN FILE[1,B$],"CLUB"
0040 LOPEN FILE[2,B$],"MEMBER"
0050 LOPEN FILE[3,B$],"NAME"
```

Code to open files in the PARAM file database structure:

```
0010 DIM C1[2,3],B$[544],MX$[18],NX$[18]
0020 LET B$= CLUB,5,MEMBER,5,NAME,5 ,FILL$(0)
0030 BLOCK WRITE B$
0040 SWAP "OPEN
0050 BLOCK READ B$
0060 IF ASC(B$[1,2])<>-1 THEN GOTO 0090
0070 PRINT "ERROR # ";ASC(B$[3,4]);" - ";B$[5,512]
0080 END
0090 LET K=1
0100 FOR I=0 TO 2
0110   FOR J=0 TO 3
0120     LET C1[I,J]=ASC(B$[K,K+3])
0130     LET K=K+4
0140   NEXT J
0150 NEXT I
0160 LETMX$=CHR$(C1[1,0],2),CHR$(C1[1,1],4),CHR$(0,2),"MEMBER",FILL$(0)
0170 LET NX$=CHR$(C1[2,0],2),CHR$(C1[2,1],4),CHR$(0,2),"NAME",FILL$(0)
```

In terms of code, another difference between the logical structure and the PARAM structure is the use of the DELREC and GETREC statements as opposed to the DELREC.SL, GETREC.SL, and POSFL.SL subroutines. The

logical structure uses the statements, which also perform automatic locking of record 0, while the PARAM structure uses the subroutines. The three subroutines occupy a total of 55 lines of code compared to the 2 lines required by the statements.

Enlarging a Logical Database

If you discover that your logical database is too small for your needs, you can enlarge it by following these steps:

1) Use LFU PCREATE to create a dummy database of the desired size.

2) Use LFU LCREATE to create a dummy logical file in this database for each logical file in your current database. You can increase the size of the files. (Note that the dummy logical filename for each file must be different from the original logical filename.)

3) Use the Business BASIC utility LXFER to copy each logical file in the current database to the corresponding logical file in the larger dummy database. You can only copy one file at a time. (LXFER is explained in the *Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI.*)

4) Use LFU PDELETE to delete the old database.

5) Use LFU PRENAME to rename the dummy database to the old database name.

6) Use LFU LRENAME to rename the logical files to the same names that existed in the old database.

In this example, a database named CUSTDB needs to be enlarged so the following is done:

```
* !LFU PLIST CUSTDB
  DB file: CUSTDB

  File    File    Starting    # of    Record    Last    # of
  Name    Type     Sector    Sectors  Length   Record  Bytes
CUST       L           0       10       50      100    5050
CUSTNO     I          10        6      512        5    3072
                             ---                      ----
              Total Sectors:   16             Bytes:   8122
```

Currently, this database holds 100 records. To expand the data file and the index file to allow 500 records, you run INDEXCALC to determine the number of sectors needed for the new database file. Then you create a dummy database file named TEMPDB and dummy logical files named TCUST and TCUSTNO.

```
* !LFU PCREATE TEMPDB 67
* !LFU LCREATE TCUST TEMPDB L 50 500 N
* !LFU LCREATE TCUSTNO TEMPDB I 512 17 4 50 N
* !LFU PLIST TEMPDB
```

```
  DB file: TEMPDB
 File    File    Starting    # of    Record    Last    # of
 Name    Type     Sector    Sectors  Length   Record  Bytes
TCUST      L          0       49      50      500    25050
TCUSTNO    I         49       18     512       17     9216
                             ---                     -----
              Total Sectors:  67            Bytes:   34266
```

Next use the LXFER utility to copy the original logical files to the new logical files and then delete the old database.

```
* !LXFER/V CUST TCUST          :Copies old data file into new file
5120 bytes transferred

* !LXFER/V CUSTNO TCUSTNO      :Copies old index file new index file;
3072 bytes transferred         :Last-available-block pointer in 0 is adjusted.

* !LFU PDELETE CUSTDB          :Delete the old database.

* !LFU PRENAME TEMPDB CUSTDB   :Rename new database with the old name.

* !LFU LRENAME TCUST CUST      :Rename new data file with the old name.

* !LFU LRENAME TCUSTNO CUSTNO  :Rename new index file with the old name.

* !LFU PLIST CUSTDB            :Get a listing of the enlarged database.

  DB file: CUSTDB
  File    File    Starting    # of    Record    Last    # of
  Name    Type     Sector    Sectors  Length   Record  Bytes
CUST       L          0       49      50      500    25050
CUSTNO     I         49       18     512       17     9216
                             ---                     -----
              Total Sectors:  67            Bytes:   34266
```

To see how the database has changed, compare the output of INDEXVRFY on the CUSTNO index file before it was moved to the new database with the output of INDEXVRFY on the new CUSTNO index file. This is the original CUSTNO file:

```
* !INDEXVRFY CUSTNO

    INDEX FILE NAME = CUSTNO
       ** VERIFYING **

Index file size             -  6   512 byte blocks
Number of index blocks used -  5
Empty index blocks          -  0
Key length                  -  4    Duplicates not allowed
Max keys per block          -  63
Min key count               -  3
Max key count               -  37
Avg key count               -  26
Total keys at bottom level  -  100
```

```
Number of index levels       - 2
INDEX STRUCTURE VERIFIED CORRECT

  ******* VERIFY COMPLETE ******
```

This is the output from INDEXVRFY on the new CUSTNO index file:

```
* !INDEXVRFY CUSTNO

  INDEX FILE NAME = CUSTNO
    ** VERIFYING **

Index file size              - 18   512 byte blocks
Number of index blocks used  - 5
Empty index blocks           - 0
Key length                   - 4    Duplicates not allowed
Max keys per block           - 63
Min key count                - 3
Max key count                - 37
Avg key count                - 26
Total keys at bottom level   - 100
Number of index levels       - 2
INDEX STRUCTURE VERIFIED CORRECT

******  VERIFY COMPLETE  ******
```

End of Appendix

Related Documents

Business BASIC Reference Manual for Commands, Statements, and Functions **093-000351-00**

An alphabetical directory of Business BASIC statements, commands, and functions. It is intended to be used as a reference manual for programmers.

Business BASIC Reference Manual for Subroutines, Utilities, and BASIC CLI **093-000389-00**

An alphabetical directory of Business BASIC subroutines, utilities, and BASIC CLI commands. It is intended to be used as a reference manual for programmers.

Business BASIC System Manager's Guide **093-000388**

Describes how to load and generate Business BASIC on AOS, AOS/VS, AOS/WS, RDOS, and DG/RDOS systems. It is intended for the system manager or system operator.

AOS INFOS® II System User's Manual **093-000152**

Provides information on how to use the AOS INFOS II file management system.

AOS/VS INFOS® II System User's Manual **093-000299**

Provides information on how to use the AOS/VS INFOS II file management system.

Business BASIC (AOS/RDOS) Reference Card **069-705010**

A pocket-sized reference card for Business BASIC programmers on AOS/VS, AOS, RDOS or DOS.

DASHER D2 File Maintenance and Screen Maintenance Template **093-000212**

DASHER D200 File Maintenance and Screen Maintenance Template **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**

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

*DEL 6-3
. (period) command 2-8
.A command 2-9
.C command 2-8, 2-9
.DB file 6-2
.E command 2-9
.I command 2-9
.P command 2-8, 2-9
.SL 4-1
.VL file 6-2

A

AND 3-9
AOS file types 8-2
Arguments 1-2
Arithmetic operators 3-8
Arrays 1-2, 3-4
  indexing 3-4
ASC 4-7

B

BASIC CLI 1-3, 4-3, 4-6, 4-7
BASIC CLI commands 1-4
BBS 8-2
Binary format 1-4, 2-18, 2-19
BLOCK READ 2-14, 4-8
BLOCK WRITE 2-14, 4-8
Boolean logic operators 3-9
BYE 1-6, 1-7

C

C1 array 6-8, G-2
Carriage return ii, G-2
CHAIN 2-12
Channel 5-9, 6-1, 6-9
Channel strings 7-3
CHR$ 4-7
CLI 1-3, 1-4
Colon comments 2-18, 4-3
Command 1-2
Comments, program 2-18
Common area 2-14, 3-20, 4-8
CON 2-14
  debugging aid 2-17
Continuing a program 2-13
CTRL-U 2-7
CTRL-X 2-7

D

Data transfer 5-9
Database file 6-1, 6-2, 8-2, 8-3
DB statements 7-1
DBF 8-2
DBFIX 8-2
Decimals 3-11
DEL 6-3
DEL key 2-7
Deleted-record chain 5-11
Delimiter 2-8, G-2
DELREC.SL 6-8, 6-12, 6-13, 6-14, C-10
Direct random access files 5-11

E

EDIT 2-10
Edit buffer 2-7
END 2-2
ENTER 2-3, 2-6, 4-2
Error messages 2-18, 3-19, 8-4
ESCAPE key ii, G-4

F

File attributes 8-3
Files
  deleting 6-3, 8-5
  limits 6-1, 7-1
FINDFILE.SL 6-10
FIXFILE 8-2
Flat files 5-11
Flow-control constructions 2-4

FOR/NEXT 2-4
Function 1-2

G

GETREC.SL 6-12, 6-14, C-10
GOSUB 2-4, 4-3
GOTO 2-4

I

IF 2-4, 3-10
IKEY ii, G-4, 2-7, 2-11, 2-14
IMOVE 8-2, 8-3
Indexing, strings 3-15
INFOS II files 7-1
INPUT 2-5, 3-15
Integer 1-2
Interactive programs 2-5
Interrupts 2-14, 2-16
ISAM 5-12

K

Keyboard mode 1-4, 2-1

L

LET 3-11, C-8
LFTABL$ 6-4
LFU 6-3
Line length 2-3
LIST 2-3, 2-5, 2-10, 2-19
Listing file 2-10, 2-19
LOAD 2-11
LOCKS 4-7
Logical files 6-1, 6-2
LOPEN 6-4
LRELINK 5-12

M

Master file 6-1, 6-6

N

Nesting subroutines 4-4
New line ii, G-5
NOT 3-9
Null string 3-14
Numeric variables 1-2, 3-3

O

ON GOSUB 4-3
OPEN 6-13
Open files 5-7, 6-1, 7-1
Operations, precedence 3-14
Optional keywords 2-4
OR 3-9

P

PARAM file 6-1, 6-6, 6-7
PARAMCON 6-14
Physical files 6-1
Portability 8-1
POSFL.SL 6-11, 6-14, C-10
POSITION FILE 5-10, 6-11
Precedence of operations 3-11
Precision 3-3
PRINT 2-2, 2-5
Program comments 2-18
Program statement,
  maximum length 2-3
Program statements 2-2

R

RDOS.PR 8-2, 8-3
Records, deleting 6-12
Relational operators 3-9
Relative position 5-10
RELINK 5-12
REM 2-4, 2-18
RENUM 2-6
RENUMBER 2-6
Renumbering statements 2-6, 2-7
RETURN 4-3
RUN 2-11

S

SAVE 2-10
SAVE file 2-10, 2-19, 8-2, 8-3
Sharing program data 2-14
Special characters 3-1
Statement 1-2
STMA 1 2-14
String arrays 3-17
String functions 3-19, 3-21
String literals 3-15
String variables 3-15
Strings 1-2, 3-14
  concatenating 3-19
  storage 3-15
Subfiles 6-1
  limits 6-10
Subroutine 1-3, 2-4, 4-1
Subroutine comments 4-3
Subscripts 3-5, 3-16
Substrings 3-16
SWAP 2-12, 4-6
  sharing program data 2-14

SWAP THEN CON 2-13
SYS(30) 8-4
SYS(31) 2-18, 3-19, 8-4
SYS(7) 2-18, 3-19, 8-4
System library 4-3

T

Text files 2-19, 8-2, 8-3
Truncation 2-3
TYPE 4-3

U

Utilities
  run-only 4-6
  SWAP-only 4-6
  stand-alone 4-6

V

Variable names 3-1
Variables 1-2
  changing values 2-14
  numeric 3-3
  values 2-2, 2-11, 2-13, 2-14
VLF 8-2
Volume label file 6-2, 8-2, 8-3

W

Working storage 1-4, 2-7, 2-18

Data General

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

SHIP TO: (if different)
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No._____ Expiration Date_____

☐ Purchase Order Number:_____

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other _____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order _____ Phone _____ Extension _____

Data General Corporation
Educational Services/TIPS
MS G214
2400 Computer Drive
Westboro, MA 01580
(617) 366-8911, Extension 1610

_____
Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

_____
Title

_____
DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

CUT ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. **PRICES**
   Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. **PAYMENT**
   Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. **SHIPMENT**
   Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. **TERM**
   Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. **CUSTOMER CERTIFICATION**
   Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. **DATA AND PROPRIETARY RIGHTS**
   Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. **DISCLAIMER OF WARRANTY**
   DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. **LIMITATIONS OF LIABILITY**
   IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. **GENERAL**
   A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

DataGeneral

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   Data General Corporation
   Educational Services/TIPS
   MS G214
   2400 Computer Drive
   Westboro, MA 01580
   (617) 366-8911, Extension 1610
8. We'll take care of the rest!

educational services

Data General Corporation, Westboro, MA 01580

093-000480-00