Data General

Customer Documentation

Subroutines, Utilities, and Business BASIC CLI Commands

Subroutines, Utilities, and Business BASIC CLI Commands

093-000389-02

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. 093-000389

Printed in the United States of America
Rev. 02, May, 1991

Notice

Subroutines, Utilities, and Business BASIC CLI Commands
093-000389-02

Revision History:

Original Release - December, 1984
First Revision - December, 1989
Second Revision - May, 1991

Effective with:

Business BASIC for AViiON™ Systems, Rev. 1.10
Business BASIC for INTERACTIVE UNIX Systems, Rev. 1.10
AOS Business BASIC, Rev. 4.20
AOS/VS Business BASIC, Rev. 5.20
RDOS Business BASIC, Rev. 8.20
DG/RDOS Business BASIC, Rev. 8.30

A vertical bar in the margin of a page indicates substantive technical change from the previous revision.

Preface

This manual describes how to use the subroutines, utilities, and BASIC CLI commands that make up Business BASIC. This manual explains these features and how they work on each of the operating systems supporting Business BASIC—AOS, AOS/VS, AOS/VS II, DG/RDOS, RDOS, DG/UX™, and INTERACTIVE UNIX® systems.

Since DG/UX and INTERACTIVE UNIX software are related to UNIX software, this document sometimes refers to those two software products as UNIX products. These references are solely for the purpose of improved readability and occur only where there are no significant differences between DG/UX, INTERACTIVE UNIX, and UNIX software. UNIX® is a registered trademark of AT&T.

This manual is for the experienced Business BASIC programmer who is familiar with the particular operating system being used. The programmer who is not familiar with the operating system should consult the documentation related to the system before using this manual.

NOTE: INTERACTIVE Systems Corporation has replaced the name 386/ix™ with INTERACTIVE UNIX. References to INTERACTIVE UNIX and 386/ix refer to the same product line.

Document Set

Business BASIC is documented by a set of manuals that describe the language, the operating system features that affect its use, and its utilities. This manual is a companion manual to *Commands, Statements, and Functions in Business BASIC*. Both of these manuals apply to Business BASIC on all operating system platforms. Other manuals in the Business BASIC manual set apply only to certain operating systems. For information on the other manuals in the set and their ordering numbers, see the "Related Documents" section at the end of this manual.

Scope

Subroutines, Utilities, and Business BASIC CLI is a reference manual for experienced Business BASIC programmers. This manual provides the following information for each subroutine, utility, and Business BASIC CLI command in the language:

- Which operating systems it works with
- How to code it
- What it does
- How to use it

You can use the Business BASIC subroutines, utilities, and Business BASIC CLI commands to make your Business BASIC work easier. The following sections provide general information about these Business BASIC features and the database structures they are designed to work with.

Database Structures

Business BASIC supports two database structures: the PARAM file database structure and the logical file database structure. Certain Business BASIC features work with one database structure and not the other. To distinguish between the two database formats, this manual uses the terms *master file* and *subfile* to refer to files in the PARAM structure and *database file* and *logical file* to refer to files in the logical file structure. These terms are defined below:

Master file	A physical file in the PARAM database structure.
Subfile	A file within a master file in the PARAM database structure.
Database file	A physical file in the logical file database structure. (Database files have a .DB filename extension.)
Logical file	A file within a database file in the logical file database structure.

The tables in Appendix B show which subroutines and utilities can be used with each database structure.

Subroutines

The subroutines in this manual are portions of Business BASIC code designed to perform specific functions from within a program. They reside in the Business BASIC library directory, and their filenames have .SL extensions to distinguish them from utility programs and other files.

The first time you use a subroutine, enter it into your program and then save the program. This makes the subroutine part of the program, so that you don't need to enter it each time you run your program.

To avoid overwriting parts of your program, check the subroutine's line numbers before you add it to the program. You can do this by entering **!TYPE** *subroutine-name*.SL at your terminal. The typed subroutine also displays the colon comments that explain the subroutine's function. These comments are lost when you enter the subroutine into a program.

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.

Utilities

Utilities are Business BASIC programs that perform specialized tasks. Most utilities can be executed using **RUN**, **CHAIN**, or **SWAP** "*utility-name*. You can also execute most utilities from the Business BASIC CLI (either by entering the utility name while in the Business BASIC CLI or by entering !*utility-name*). The Business BASIC CLI performs a swap to the program named.

Some utilities have restricted execution modes. The only way to use **FILESORT**, **IBUILD**, **OPEN**, **QFILESORT**, **TBUILD**, and **XBUILD** is to swap to them from within programs. They cannot be run from keyboard mode or executed through the Business BASIC CLI. Other utilities, such as **DBGEN**, are run-only utilities and don't work with **SWAP**. Do not execute them by entering "*utility-name*, !*utility-name*, or **SWAP** "*utility-name*.

Once executed, the interactive utilities prompt you for the information they need. In some cases you can execute the utility in command line format by entering *!utility-name arguments*, where the arguments provide information the utility needs in order to work.

Business BASIC CLI

Business BASIC has its own Command Line Interpreter (CLI). The Business BASIC CLI allows you to perform operating system functions without forcing you to leave Business BASIC. (Business BASIC simulates the DG/RDOS CLI environment.)

Execute the Business BASIC CLI by entering **RUN, CHAIN**, or **SWAP "CLI** or **!CLI**. An exclamation point (!) prompt indicates you are in the Business BASIC CLI. You can issue commands by entering the command word from the Business BASIC CLI or by entering *!command* while in Business BASIC keyboard mode. If the Business BASIC CLI does not recognize a command, it performs a swap to the file named, allowing you to execute some utilities through the Business BASIC CLI.

Organization

The Business BASIC subroutines, utilities, and CLI commands are listed in alphabetical order. In some cases, subroutines and/or commands are used only with a certain utility. They are explained with the utility and often contain individual examples. A general example for the utility also appears at the end of the description for that utility.

Terms Used in This Book

The term "AOS/VS" is used to refer to AOS, AOS/VS, and AOS/VS II systems, and the term "DG/RDOS" is used to refer to DG/RDOS and RDOS systems. For example, any explanation that contains the phrase "for AOS/VS systems" also applies to AOS and AOS/VS II systems, unless the documentation states otherwise. Differences between these products are noted.

When the term "search path" is used in this book, it has the following meanings:

AOS/VS systems:	The directories you have selected using the AOS/VS **SEARCHLIST** command.
DG/RDOS systems:	In DG/RDOS systems, you do not have a search path, so Business BASIC searches your directory first and then the library directory ($LIB or $LIB3 for triple precision).
UNIX systems:	The directories you have listed in the UNIX **BBPATH** environment variable.

When the term "switch" is used, it has the following meanings:

AOS/VS and DG/RDOS systems:	A switch that is preceded by a slash.
UNIX systems:	An option that is preceded by a hyphen.

The phrase "Business BASIC user's guide" refers to *Business BASIC System Manager's Guide* if you are using an AOS/VS or DG/RDOS system and *Using Business BASIC on DG/UX™ and INTERACTIVE UNIX® Systems* if you are using a UNIX system.

Coding Conventions

The coding conventions used in this manual are described below.

UPPERCASE BOLD	Indicates a Business BASIC command, statement, or function.
lowercase italics	Indicates a placeholder to be replaced by your variable name or literal.
{ }	Enclose a part of the format from which you must make a single selection. Do not enter the braces.
[ ]	Enclose an optional part of the format; do not enter the brackets.
...	Indicates that the preceding item can be repeated.

The following boxes indicate whether a subroutine, utility, or Business BASIC CLI command is available on a particular operating system. If only an AOS/VS box appears above the "Format" section for a utility, that utility is available only on AOS/VS systems. If all three boxes appear, the utility can be used on all operating systems that run Business BASIC.

AOS/VS	DG/RDOS	UNIX

How To Use the Examples in This Book

Some examples in this book use AOS/VS conventions, and others use DG/RDOS or UNIX conventions. Unless a particular example states otherwise, if a utility is available on all operating systems, its examples work on all operating systems, provided you make any changes necessary to conform to an operating system's conventions. For example, pathnames in AOS/VS and DG/RDOS use a colon (:) as the pathname delimiter. UNIX systems use a slash (/) as the pathname delimiter unless you specified pathname conversion by including the -P option on the command line you used to execute Business BASIC. To run an AOS/VS example on a UNIX system, you must change the colons to slashes if you did not specify pathname conversion when you executed Business BASIC.

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please use the enclosed TIPS order form (United States only) or contact your local Data General sales representative.

Telephone Assistance

If you are unable to solve a problem using any manual you received with your system, and you are within the United States or Canada, contact the Data General Service Center by calling 1-800-DG-HELPS for toll-free telephone support. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

Free telephone assistance is available with your hardware warranty and with most Data General software service options. Lines are open from 8:30 a.m. to 8:30 p.m., Eastern Time, Monday through Friday.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining the Users Group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive FOCUS monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual Member Directory, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1-800-877-4787 or 1-512-345-5316.

End of Preface

Contents

Chapter 1
Subroutines, Utilities, and Business BASIC CLI Commands

Tables

Table

Figures

Figure

Chapter 1
Subroutines, Utilities, and Business BASIC CLI Commands

This chapter describes the Business BASIC subroutines, utilities, and CLI commands. They are listed in alphabetical order. Where subroutines or commands are used only with a certain utility, they are explained with that utility.

Subroutines. These are portions of Business BASIC code designed to perform specific functions from within a program. They reside in the Business BASIC library directory, and their filenames have .SL extensions to distinguish them from utility programs and other files.

Utilities. These are Business BASIC programs that perform specialized tasks. Most utilities can be executed using **RUN**, **CHAIN**, or **SWAP** "*utility-name*. You can also execute most utilities from the Business BASIC CLI (either by entering the utility name while in the Business BASIC CLI or by entering !*utility-name*). The Business BASIC CLI performs a swap to the program named.

Business BASIC CLI. Business BASIC has its own Command Line Interpreter (CLI). The Business BASIC CLI allows you to perform operating system functions without forcing you to leave Business BASIC. (The Business BASIC CLI emulates the DG/RDOS CLI.)

ANALYZE *Utility*

Displays the status of an RDOS Business BASIC system.

DG/RDOS

What It Does

Since this utility can only be used on DG/RDOS systems by someone with system manager privileges, **ANALYZE** is explained in the *Business BASIC System Manager's Guide*.

AOS

BASIC CLI Command

Brings up an AOS/VS CLI environment.

AOS/VS

Format

!AOS

What It Does

The **AOS** command creates a son process to run the AOS/VS CLI. The son process starts in the environment where the **AOS** command was executed. Once in the son process, you can use the AOS/VS command **CURRENT** to display the AOS/VS environment's settings. Any changes you make in the AOS/VS environment (such as moving to another directory or changing your search path) are effective only while you are in the son process. When you exit the AOS/VS CLI, the system returns you to your original environment but retains any non-environmental changes you made under AOS/VS, such as deleting files.

How To Use It

To use the **AOS** command, you must have **CLI.PR** on your search path, and your user profile must allow you to create a son process.

Execute the command by entering **AOS** from the Business BASIC CLI. A right parenthesis prompt indicates you are in the AOS/VS CLI. To leave the AOS/VS CLI, enter **BYE**.

Example

Once you execute the Business BASIC CLI, the Business BASIC CLI command **GDIR** shows you are in subdirectory **SHEILA**. Then you execute the **AOS** command and move to directory **MAIN**. You delete the file **TEST.AP** and return to the Business BASIC CLI. The **BYE** command returns you to the subdirectory **SHEILA** but does not restore the deleted file **TEST.AP**.

```
* RUN "CLI
CLI REV. X.XX
!GDIR
:UDD:MAIN:SHEILA
!AOS

AOS CLI     REV. XX.XX     14-JAN-90     10:38:12
)DIR :UDD:MAIN
)DELETE TEST.AP
)BYE
AOS CLI     TERMINATING    14-JAN-90     10:40:33
!GDIR
:UDD:MAIN:SHEILA
```

APPEND

BASIC CLI Command

Combines two or more files.

AOS/VS	DG/RDOS	UNIX

Format

!APPEND *newfile oldfile1* [*oldfile2* ...]

Arguments

newfile — The file that receives the concatenated files. If this file does not exist, **APPEND** without the **/N** switch creates it.

oldfile — The file or files you want to append to *newfile*. The old files are concatenated in the order in which you list them as arguments to **APPEND**. **APPEND** does not change the old files.

Global Switches

/N — Do not create *newfile*. (If *newfile* does not exist, **APPEND** displays an error message.)

/S — (DG/RDOS only) Organizes *newfile* sequentially.

What It Does

APPEND concatenates *oldfile* to *newfile*. If *newfile* doesn't exist, **APPEND** without the **/N** switch creates it. When the **/N** switch is used, **APPEND** returns an error message if *newfile* is missing instead of creating *newfile*.

On DG/RDOS systems, *newfile* is a randomly organized file unless you use the **/S** switch to specify sequential organization.

How To Use It

Execute the command by entering **APPEND** from the Business BASIC CLI. **APPEND** must be followed by the name of the file to receive the existing file and the name of the file to be added. You can add several files to a *newfile* with a single **APPEND** command. If used, the optional global switches are added either to **APPEND** or to another global switch.

Example

APPEND concatenates to **MAIN.SR** the programs **SUB1.SR** and **SUB2.SR**, and the subroutines **FORM.SL** and **UNFORM.SL**. The files are added in the order in which they are listed in the command line. Because the **/N** switch is used, **MAIN.SR** must exist or an error is generated.

```
!APPEND/N MAIN.SR SUB1.SR SUB2.SR FORM.SL UNFORM.SL
```

ASG

BASIC CLI Command

Assigns a device for exclusive use by an operator.

DG/RDOS

Format

!ASG *devicename*

Arguments

devicename The name of the device assigned to exclusive use by the operator.

What It Does

ASG assigns the device represented by *devicename* to you until you log off or use the Business BASIC CLI command **FREE** to free the device.

How To Use It

Execute the command by entering **ASG** from the Business BASIC CLI. The command must be followed by the name of a device.

Example

ASG gives your job exclusive use of the line printer.

```
!ASG $LPT
```

ATTACH *Utility*

Links your terminal to a detached job.

DG/RDOS

Format

```
{ RUN   }
{ SWAP  } "ATTACH
{ CHAIN }
```

What It Does

ATTACH prompts you for the job number of a detached job and then connects your terminal to that job.

A detached job is a job running independently of a terminal. You can begin a job's execution at your terminal and then detach the job by using the detach key. (The default detach key is Ctrl-D. Use **STMA 4,0** or the Business BASIC **TERM** utility to change the detach key.) You can also execute a detached job using the Business BASIC **START** utility. The system discards any terminal output from a detached job.

If a detached job executes an **INPUT** statement, the job waits ten minutes for input. If nothing is entered, the job logs off. Use **ATTACH** to connect to a job and supply data for the **INPUT** statement before the job logs off.

How To Use It

First, run the **STAT** utility to get the number of the detached job. Detached jobs are indicated by a -1 in the eighth column of the **STAT** display (the column immediately following the job ID).

Then start the **ATTACH** dialog by entering **RUN, CHAIN,** or **SWAP "ATTACH**.

ATTACH asks you for the job number of the detached job. If the detached job is waiting for input, the message `JOB IS WAITING ON INPUT` appears as soon as **ATTACH** adds the job. If you try to attach to a job that's not in use, a message appears telling you that the job is not in use. If you try to attach to your own job, the message `THIS IS YOUR JOB` appears.

Example

Enter **RUN "STAT** to get the number of a detached job. The output from **STAT** defines your job as 00—running **STAT**—and the detached job **MYPROG** is job 2, the number given in the first column. Detached jobs have a -1 in the eighth column (the column after the account codes). The account code for **MYPROG** is ABTLB6.

ATTACH

Continued

```
* RUN "STAT

00 O R 13 DOC  STAT     ABTLB6  03  0  5874 3428  55.7   2507  14:47
01 I R 13 OMI  SCRATCH  ABEAS6  01  0  9416 8914 113.9   1462  15:49
02 I R 13 DOC  MYPROG   ABTLB6  -1  0  6450 3428 137.7   8807  14:53
```

Now, enter **RUN "ATTACH** and use the utility to attach your terminal to job number 2, MYPROG. Once you are attached to MYPROG, a question mark prompt appears, indicating the job is waiting on numeric input. For this example, enter the number 123.

```
* RUN "ATTACH

JOB NUMBER:2
JOB IS WAITING ON INPUT
?123
```

bb_port *Utility*

Converts listing files to UNIX save files.

UNIX

What It Does

The **bb_port** utility aids you in porting your programs to a UNIX system. Information on this utility, like other program conversion tools, is documented in the on-line file **CONVERT.DOC**, located in the Business BASIC directory **DOC**.

BLDCOM *BASIC CLI Command*

Builds a documentation file to be used by PRTCOM.

AOS/VS	DG/RDOS

Format

!BLDCOM *filename1* [*filename2* ...]

Argument

filename The name of a program source file or listing file. Names with an .FR extension indicate FORTRAN source files. Filename templates can be used; they function as they do on your operating system.

Global Switch

/V Display the list of filenames at the terminal.

What It Does

BLDCOM is used in connection with **PRTCOM**. **BLDCOM** creates a file containing the comments from BASIC, Business BASIC, FORTRAN, or assembly language program source files. You must execute **PRTCOM** to display the **BLDCOM** documentation file (see **PRTCOM**).

How To Use It

Execute this command by entering **BLDCOM** from the Business BASIC CLI. The command must be followed by the name of the input file or files. **BLDCOM** then requests the name of an output file to receive the program comments. If the output file exists, **BLDCOM** overwrites the output file with the current program comments.

BLDCOM places comment sections from the source file in the documentation file it creates. A comment section starts with the programming language's comment symbol followed by an exclamation point. The comment symbol is a colon (:) in BASIC and Business BASIC, a semicolon (;) in assembly language, and a C in FORTRAN. Terminate a comment section with a double comment symbol, such as ::, ;;, or CC. (See **PRTCOM** for editing conventions concerning comments.)

BLDCOM *Continued*

Example

In this example, **BLDCOM** creates the documentation file **TEMP**, which contains the comments for all subroutine library modules (-.SL) and any programs ending in .SR that have six-character names, ending in BUILD (*BUILD). A verification list of these files appears at the terminal.

```
!BLDCOM/V -.SL *BUILD.SR
DOCUMENTATION FILE: TEMP
GETCM.SL
GETLAST.SL
INITINDEX.SL
    .
    .
    .
!
```

BUILD

BASIC CLI Command

Creates a file containing a list of the files in the current directory.

AOS/VS	DG/RDOS

Format

!BUILD *outputfile* [*inputfile*[/N]...]

Arguments

outputfile	The new file that contains the filenames. If *outputfile* exists, it is deleted and recreated.
inputfile/**N**	A file in the current directory. Links are required to use a file in another directory. Filename templates can be used; they function as they do on your operating system. With the /N switch, include all files in the current directory except those matching the filename or template.

Global Switches

/A	(DG/RDOS only) Include all files, permanent and nonpermanent. You do not need to specify an *inputfile* argument with this switch.
/E	(DG/RDOS only) Do not include extensions to the *inputfile* argument.
/K	Do not include links.
/N	Include only links.
/P	Sort the list by the date of the last file access (the oldest file is listed first).
/S	Sort the list alphabetically.
/W	Sort the list by the time of the last write to each file (the oldest file is listed first).
/X	Sort the list in ascending order of file size.

Local Switches

mm-dd-yy/**A**	Include only files created on or after the date *mm-dd-yy* (e.g., 04-13-91/A).
mm-dd-yy/**B**	Include only files created before the date *mm-dd-yy*.

What It Does

BUILD creates a file containing the names of files in the current directory. You can then use the resulting output file as an indirect reference to the files specified by the *inputfile* argument.

BUILD *Continued*

Since the line continuation characters for AOS/VS and DG/RDOS differ, an output file created on an AOS/VS system cannot be used on a DG/RDOS system and vice versa. (AOS/VS uses the ampersand, while DG/RDOS uses the up-arrow.)

How To Use It

Execute this command by entering **BUILD** from the Business BASIC CLI. You must enter an output filename with **BUILD**. Global switches attach either to the keyword **BUILD** or to another global switch, while local switches appear as separate arguments.

Examples

1. You can use **BUILD** to create an alphabetical list of all the files in a directory. On a DG/RDOS system, the **/A** global switch calls for both permanent and nonpermanent files to be included in the list, and the **/S** global switch puts the list in alphabetical order.

   ```
   !BUILD/A/S TEST
   ```

2. **BUILD** creates the file **TEMP**, which contains the names of all nonpermanent files in the current directory. In this case, the current directory is the system library for both the AOS/VS and the DG/RDOS versions of the example.

   In AOS/VS:

   ```
   !DIR
   :UTIL:BBASIC:$SYSLIB
   !BUILD/S TEMP +.SL
   !TYPE TEMP
   DELREC.SL,FINDFILE.SL,FORM.SL,FORMIO.SL,GETCM.SL,GETLAST.SL,&
   GETREC.SL, INITINDEX.SL, LFDATA.SL, LINITINDEX.SL, POSFL.SL,&
   PROTFORM.SL, SCRNIO.SL, SFORM.SL, UNFORM.SL&
   !
   ```

   In DG/RDOS:

   ```
   !GDIR
   $LIB
   !BUILD/S TEMP -.SL
   !TYPE TEMP
   DELREC.SL,FINDFILE.SL,FORM.SL,FORMIO.SL,GETCM.SL,GETLAST.SL,^
   GETREC.SL, INITINDEX.SL, LFDATA.SL, LINITINDEX.SL, POSFL.SL,^
   PROTFORM.SL, SCRNIO.SL, SFORM.SL, UNFORM.SL^
   !
   ```

BUILD

Continued

3. **BUILD** creates the file **BUILDLINKS**, which holds the names of all permanent and nonpermanent files that are links to files in other directories. The files are sorted by date of last access; the file opened most recently is the last filename in **BUILDLINKS**.

   !BUILD/A/N/P BUILDLINKS

4. The new file **LISTING** contains files created before 11-11-91 that are not link files and do not begin with TEMP.

   !BUILD/K LISTING 11-11-91/B TEMP-.-/N

BYE

BASIC CLI Command

Logs you out of Business BASIC from the Business BASIC CLI.

AOS/VS	DG/RDOS	UNIX

Format

!BYE

What It Does

When you enter **BYE** from the Business BASIC CLI, you are logged off Business BASIC.

On AOS/VS and UNIX systems, **BYE** moves you back one level. For example, if you execute Business BASIC from the AOS/VS CLI or the UNIX shell, **BYE** returns you there. If you execute Business BASIC using a CLI macro, **BYE** returns control to the CLI macro. If you log on to AOS/VS Business BASIC directly, **BYE** logs you off the system.

On DG/RDOS systems, **BYE** renders your terminal inactive until you or someone else logs on. When you execute **BYE** as a program statement, Business BASIC logs off the job executing the program. **BYE** also logs off a detached job.

How To Use It

Execute **BYE** by entering it from the BASIC CLI, or use it as a program statement.

Examples

First, execute the Business BASIC CLI; then, after you use **GTOD** to check the time and the date, enter **BYE** to terminate Business BASIC.

```
* RUN "CLI
CLI REV. X.XX
!GTOD
15:37:18   11/09/91
!BYE
```

CCONT

BASIC CLI Command

Creates a contiguous file.

AOS/VS	DG/RDOS	UNIX

Format

!CCONT *filename1 blocks1* [*filename2 blocks2* ...]

Arguments

filename	The name of the file you want to create.
blocks	The number of contiguous blocks you need for file length. A block is 512 bytes, so your file length in bytes is the number of blocks multiplied by 512.

Global Switches

/C	Create a contiguous file of up to 65534 blocks and write a null to the last byte in the file to allocate space for the file (AOS/VS only).
/N	Do not null fill the file (DG/RDOS only).

What It Does

For each *filename* you specify, **CCONT** creates a contiguous DG/RDOS file or a random AOS/VS or UNIX file.

On DG/RDOS systems, **CCONT** sets the DG/RDOS C attribute (for contiguous) for the new file. The command also reserves and fills with nulls the number of contiguous blocks you specify with each filename.

On AOS/VS systems, **CCONT** without the /C switch creates a contiguous file with a maximum index level of 0 and an element size defined by the *blocks* argument, which cannot be greater than 32,767. If the /C switch is used, then the *blocks* argument can be up to 65,534. In both cases, space is reserved when you create the file. The element size is a multiple of the system default element size. (On AOS/VS and AOS/VS II, you can set this value when you generate your system.) When you enter a block argument that is not a multiple of the default element size, the system rounds the block number up until it becomes a multiple of the default value.

On UNIX systems, **CCONT** creates each file, reserves space for it, and fills the file with nulls. The access privileges are determined by the value of the environment variable BBDEFACL.

CCONT

Continued

How To Use It

Execute the command by entering **CCONT** from the Business BASIC CLI. The command must be followed by at least one filename and the number of contiguous blocks needed for that file. You can create more than one file with a single **CCONT** command, but each filename must be accompanied by a block number.

Examples

1. This example creates the file **CARL** with a length of 25 contiguous blocks (12,800 bytes).

   !CCONT CARL 25

2. This example creates **MASTER** as a contiguous file of 200 blocks (102,400 bytes) and **TMP** as a contiguous file of 22 blocks (11,264 bytes) in the directory **NBASIC**.

   !CCONT MASTER 200 NBASIC:TMP 22

3. This AOS/VS example creates a contiguous file with an element size of 40,000 blocks (20,480,000 bytes).

   !CCONT/C C$FILE 40000

CDIR

BASIC CLI Command

Creates a DG/RDOS subdirectory or an AOS/VS or UNIX directory.

AOS/VS	DG/RDOS	UNIX

Format

!CDIR *directoryname*

Argument

directoryname The name of the directory or subdirectory that you want to create.

What It Does

On DG/RDOS systems, **CDIR** attaches a .DR extension to *directoryname* and creates the subdirectory in the current or given partition.

On AOS/VS and UNIX systems, **CDIR** creates *directoryname* as a directory file. On AOS/VS systems, the file is of type DIR.

See your operating system manual for more information about subdirectories, directories, and/or partitions.

How To Use It

Execute the command by entering **CDIR** from the Business BASIC CLI. The command must be followed by the name of the subdirectory or directory you want to create.

Examples

1. This creates the subdirectory **KEITH.DR** in DG/RDOS and the directory **KEITH** in the current directory in AOS/VS and UNIX. For DG/RDOS systems, the current directory must be a primary or secondary partition and not a subdirectory.

   !CDIR KEITH

2. On DG/RDOS systems, this creates the subdirectory **SHEILA.DR** in the secondary partition **JEFF.DR** of the primary partition **DE0**.

   !CDIR DE0:JEFF:SHEILA

CHAIN *BASIC CLI Command*

Executes a program and returns you to keyboard mode.

AOS/VS	DG/RDOS	UNIX

Format

!**CHAIN** *filename*

Arguments

filename The name of a SAVE file.

What It Does

CHAIN executes a program and then returns you to keyboard mode, rather than returning you to the Business BASIC CLI. **CHAIN** does this by overwriting the Business BASIC CLI with the program referred to by *filename* (i.e., your program replaces the CLI) and then executing the program. When the program ends, **CHAIN** leaves you in Business BASIC keyboard mode—not in the Business BASIC CLI.

CHAIN is also a statement and keyboard mode command (see *Commands, Statements, And Functions in Business BASIC.*)

How To Use It

Execute the command by entering **CHAIN** *filename* from the Business BASIC CLI.

Example

In this example, **CHAIN** replaces the CLI with **MYPROG** and executes it.

```
!CHAIN MYPROG
```

CHATR *BASIC CLI Command*

Changes a file's resolution access attributes.

DG/RDOS

Format

!CHATR *filename1 attributes1* [*filename2 attributes2*] ...

Arguments

filename — A string representing a filename in your directory.

attributes — One or more of the following attributes:

N	No linking is permitted to this file.
P	Permanent file; cannot be deleted or renamed.
R	Read-protected file; cannot be read.
S	SAVE file; the **SAVE** command automatically sets this attribute.
W	Write-protected file; cannot be written to.
0	Deletes existing removable attributes.
?	User-defined attribute. Use this symbol to define an attribute for a unique access specification. (See your DG/RDOS system manual for more information on user-defined attributes.)
&	User-defined attribute. Use this symbol to define an attribute for a unique access specification. This attribute is set on the volume label (.VL) files. (See your DG/RDOS system manual for more information on user-defined attributes.)
*	Retains existing attributes.

What It Does

CHATR modifies the existing attributes of a file or adds attributes to a file. It affects only the resolution access attributes of resolution files. (Use **CHLAT** to modify link access attributes of resolution files.)

If the DG/RDOS A attribute is set on a file, the file is attribute-protected. This means **CHATR** cannot change the file's attributes.

On AOS/VS systems, use the AOS/VS CLI commands **ACL** and **PERMANENCE** to change a file's access control list and permanence attribute.

On UNIX systems, use the shell commands **chown**, **chgrp**, and **chmod** to change a file's owner, group, and access permissions.

CHATR

Continued

How To Use It

Execute the command by entering **CHATR** from the Business BASIC CLI. At least one filename and attribute must follow **CHATR**, but you can use a single **CHATR** command to change the attributes on more than one file. Provide at least one attribute with each filename.

Example

This example adds permanent and write-protected attributes to the file **PAYROLL** without removing any existing attributes, while it removes the current attributes from file **ALAN**.

```
!CHATR PAYROLL *PW ALAN 0
```

CHLAT

BASIC CLI Command

Changes a file's link access attributes.

DG/RDOS

Format

!CHLAT *filename1 attributes1* [*filename2 attributes2*] ...

Arguments

filename — The name of a file in your directory.

attributes — One or more of the following attributes:

N	Do not allow linking to this file (actually, the link can be created but not used).
P	Permanent file; cannot be deleted or renamed.
R	Read-protected file; cannot be read.
S	SAVE file.
W	Write-protected file; cannot be written to.
0	Removes existing attributes.
?	User-defined attribute. Use this symbol to define an attribute for a unique access specification. (See your DG/RDOS system manual for more information on user-defined attributes.)
&	User-defined attribute. Use this symbol to define an attribute for a unique access specification. (See your DG/RDOS system manual for more information on user-defined attributes.)
*	Retains existing attributes.

What It Does

CHLAT removes, modifies, or adds link access attributes for a resolution file that regulates link access from other directories. It affects only the link access attributes. (Use **CHATR** to modify resolution access attributes.) **CHLAT** enables you to restrict what users linking to the filename argument can do.

If the DG/RDOS A attribute is set on a file, the file is attribute-protected. This means **CHLAT** cannot change the file's attributes.

Under AOS/VS, you can use the CLI commands **ACL** and **PERMANENCE** to change a file's access control list and permanence attribute.

On UNIX systems, use the shell commands **chown**, **chgrp**, and **chmod** to change a file's owner, group, and access permissions.

CHLAT

Continued

How To Use It

Execute the command by entering **CHLAT** from the Business BASIC CLI. At least one filename and attribute must follow **CHLAT**, but you can use a single **CHLAT** command to change the attributes on more than one file. Provide at least one attribute with each filename.

Example

Permanent and write-protected attributes are added to the file **TAPEUTIL** without changing the existing attributes. This way a user in another directory can link to and execute **TAPEUTIL** but cannot write to, delete, or rename it.

```
!CHLAT TAPEUTIL *PW
```

CLI

Utility

Executes the Business BASIC CLI.

AOS/VS	DG/RDOS	UNIX

Format

```
{ RUN  }
{ SWAP } "CLI
{ CHAIN}
```

What It Does

The Business BASIC Command Line Interpreter (CLI) is a utility written in Business BASIC that acts as an interface between Business BASIC and your operating system. The Business BASIC CLI simulates a DG/RDOS CLI environment and allows you to use operating system features without leaving Business BASIC. In addition, you can execute any command or saved program with switches and arguments by entering the command or program name after the Business BASIC CLI exclamation point (!) prompt. (The Business BASIC keyboard mode prompt is an asterisk.) Entering an exclamation point before a filename is the same as entering **SWAP** *filename*.

Be careful about running utilities from the Business BASIC CLI because it closes all files when it executes a command or program. Upon your return from the Business BASIC CLI, these files must be reopened and repositioned. Don't use the Business BASIC CLI to execute utilities that work with open files. These utilities expect to have the files open when they start; if they call the files from the CLI, the files are closed. Also, don't use the Business BASIC CLI to execute utilities such as **DBGEN** that can't be swapped to.

The Business BASIC CLI builds the equivalent of a command file in the common area for any command it does not recognize and swaps to the filename that is the first element of the command. You can create your own Business BASIC CLI command by writing a program that uses the **GETCM.SL** subroutine to interpret the information passed to the common area.

You can also log out of Business BASIC by entering **BYE** from the Business BASIC CLI.

How To Use It

Execute the Business BASIC CLI by entering **!CLI**, **RUN "CLI**, **SWAP "CLI**, or **CHAIN "CLI**. Entering **RUN "CLI** causes a chain to the CLI, while **!CLI** causes a swap to the CLI. You can also enter **SWAP "CLI** or **CHAIN "CLI** from within a program. If you use **SWAP "CLI**, your program in working storage is saved. No programs are saved when you use **RUN** or **CHAIN "CLI**.

CLI

Continued

If you plan to use the Business BASIC CLI on AOS/VS, you must execute Business BASIC with the /C switch. If you plan to use the Business BASIC CLI on UNIX systems, you must execute Business BASIC with the **-c** and **-P** options. UNIX systems require that you use pathname conversion if you use the Business BASIC CLI.

When you execute the Business BASIC CLI, all files are closed. Upon your return from the Business BASIC CLI, these files must be reopened and repositioned.

To leave the Business BASIC CLI, use the **POP**, **QUIT**, **CHAIN**, or **BYE** command. **POP** terminates the CLI program, clears the common area, and returns you to your previous level. **QUIT** responds the same way **POP** does, except that **QUIT** retains the common area. **CHAIN** executes the program you chain to and then places you in keyboard mode. **BYE** logs you out of Business BASIC.

Once the Business BASIC CLI exclamation point (!) prompt appears, each line you enter is interpreted as a Business BASIC CLI command. Keywords that are not CLI commands or executable programs can cause errors.

You can execute a Business BASIC CLI command in keyboard mode (i.e., !*command*). This causes a swap to the CLI, and the command is passed through the common area. After the command finishes, Business BASIC pops you back to keyboard mode. However, immediate execution of CLI commands uses your common area, and your response time slows down. To preserve the information in the common area or to execute more than one CLI command at a time and get a faster response, first run the Business BASIC CLI program and then enter the commands directly from the Business BASIC CLI.

Business BASIC CLI commands let you use global and local switches. A global switch defines what the command does to all of its arguments. Append a global switch to the command itself or to another global switch. A local switch defines what the command does to a specific argument and is appended to that argument or follows that argument.

If you call the Business BASIC CLI from your program, you can pass a command line string to the common area for the Business BASIC CLI to find it. This command line string must start with "CLI.CM <0>" followed by a series of commands separated by semicolons (see example 2). These commands can be Business BASIC CLI commands with optional switches or the names of programs to be executed. The size of your command line is limited to the size of the common area.

When you enter **RUN "CLI**, you can continue a command line from line to line by entering an up-arrow (^) in DG/RDOS, an ampersand (&) in AOS/VS, or a backslash (\) in UNIX before pressing CR or New Line to get to the next line. As long as you enter a line continuation character before pressing CR or New Line, CLI regards this as one command line. You can have multiple commands on this line. To end the command line, press CR or New Line without entering a line continuation character (see example 4).

CLI

Continued

In the Business BASIC CLI, you can use the DG/RDOS (@) or AOS/VS ([]) indirect file conventions (see your operating system manual for more information on these). However, the indirect reference can be used at only one level; in other words, *@filename@* cannot occur within a file you are using as an indirect reference. You can use parentheses to repeat commands with different arguments, but you cannot nest indirect references within parentheses or vice versa.

Table 1-1. CLI Command Table

Command	Description
AOS	Enables you to access the AOS/VS CLI (AOS/VS only).
APPEND	Appends a file to another file.
ASG	Assigns a device for exclusive use (DG/RDOS only).
ATTACH	Attaches your terminal to a detached job (DG/RDOS only).
BLDCOM	Builds a documentation file for PRTCOM (AOS/VS and DG/RDOS only).
BUILD	Builds a command file (AOS/VS and DG/RDOS).
BYE	Terminates Business BASIC.
CCONT	Creates a contiguous DG/RDOS file or a random AOS/VS or UNIX file.
CDIR	Creates a directory or subdirectory.
CHAIN	Executes a program.
CHATR	Changes a file's attributes (DG/RDOS only).
CHLAT	Changes a file's link access attributes (DG/RDOS only).
CPART	Creates a secondary partition (DG/RDOS) or a control point directory (AOS/VS).
CRAND	Creates a random file.
CREATE	Creates a random file (AOS/VS and UNIX) or a sequential file (DG/RDOS).
DBMOVE	Moves logical file structures from one directory to another (DG/RDOS only).
DELETE	Deletes a file, directory, or partition (AOS/VS and DG/RDOS only).
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 (DG/RDOS only).
FDUMP	Fast dumps one or more files to magnetic tape (DG/RDOS only).
FILCOM	Compares two files word by word.

CLI

Continued

Table 1-1. CLI Command Table *(continued)*

Command	Description
FLOAD	Fast loads "fast-dumped" (**FDUMP**) files (DG/RDOS only).
FPRINT	Displays the contents of a disk file.
FREE	Releases a device from exclusive use (DG/RDOS only).
GDIR	Displays the current directory name.
GQUE	Gets the default queue name.
GSDIR	Displays the current system directory name (DG/RDOS only).
GSYS	Displays the current system name (DG/RDOS only).
GTOD	Displays the time and date.
INIT	Initializes a device (DG/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 (AOS/VS and DG/RDOS only).
LSTMERGE	Merges two listing files (AOS/VS and DG/RDOS only).
MDIR	Displays the master directory name (DG/RDOS only).
MOVE	Moves files from one directory to another.
POP	Exits from the Business BASIC CLI and clears the common area.
PRINT	Prints a file to the default output queue (AOS/VS and DG/RDOS only).
PROGPRT	Prints reference information for a program.
PRTCOM	Prints a BLDCOM documentation file (AOS/VS and DG/RDOS only).
QUIT	Exits from the Business BASIC CLI but retains the common area.
RELEASE	Releases a device or a directory (DG/RDOS only).
RENAME	Changes the name of a file.
SDIR	Sets the system directory (DG/RDOS only).
SLINE	Selects a line (terminal) and attaches a job to it (DG/RDOS only).
SPDIS	Disables device spooling (DG/RDOS only).
SPEBL	Enables device spooling (DG/RDOS only).
SPKILL	Deletes the spool queue (DG/RDOS only).
SQUE	Sets the default queue.
START	Starts a detached job (DG/RDOS only).
TABLE	Prints a program cross-reference. (AOS/VS and DG/RDOS only)

CLI

Continued

Table 1-1. CLI Command Table (*concluded*)

Command	Description
TCOPY	Copies from tape to tape (AOS/VS and DG/RDOS only).
TFER	Copies a file between tape and disk (AOS/VS and DG/RDOS only).
TPRINT	Prints the tuning report (DG/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 (AOS/VS and DG/RDOS only).
VLPRINT	Displays the contents of a volume-label (.VL) file.
XFER	Copies one file to another file (AOS/VS and DG/RDOS only).

Examples

1. This example swaps to the Business BASIC CLI, checks the time and the date with the Business BASIC CLI command **GTOD**, and then leaves the CLI with **POP**. The prompt changes from an asterisk (*) for Business BASIC keyboard mode to an exclamation point (!) for Business BASIC CLI and then back to an asterisk for the Business BASIC keyboard mode.

```
* SWAP "CLI
CLI REV. X.XX
!GTOD
17:34:12          5/20/91
!POP
*
```

2. This program sets up a command line in X$, writes it out to the common area, and then swaps to the Business BASIC CLI.

```
* LIST
00010 DIM X$(512)
00020 LET X$="CLI.CM<0>GTOD",FILL$(0)
00030 BLOCK WRITE X$
00040 SWAP "CLI
00050 PRINT "DONE"
00060 STOP
* RUN
11:31:36          1/16/91
DONE
STOP AT 00060
 *
```

CLI

Continued

3. The Business BASIC CLI command **GDIR**, which displays your current directory, is issued while in Business BASIC keyboard mode (indicated by the asterisk prompt) by typing an exclamation point before the command. The command displays your current directory and then returns you to keyboard mode.

   In DG/RDOS:

```
* !GDIR
BBASIC
*
```

   In AOS/VS:

```
* !GDIR
:UDD:SETH:BBASIC

*
```

4. This AOS/VS example continues a CLI command line by using the AOS/VS line continuation character (an ampersand). When the ampersand precedes CR or New Line, the CLI moves the cursor to the next line and waits for you to enter the rest of the command. When the CR or New Line key is pressed without being preceded by an ampersand, the CLI executes the **LIST** command.

```
* RUN "CLI
 CLI REV. X.XX
!LIST/S NAME &
 ITEM+
:UDD:TEST
                                    9/14/91  11:23:15
ITEMS      10801     UDF
ITEMS2     11498     UDF
ITEMS3     1744      UDF
NAME       6868      TXT

 Listed space = 63 sectors, 32256 bytes
 Total space = 369 sectors, 188928 bytes
!
```

CPART

BASIC CLI Command

Creates a secondary partition (DG/RDOS) or a control-point directory (AOS/VS).

AOS/VS	DG/RDOS

Format

!CPART *partname blockcount*

Arguments

partname — The name of the secondary partition or control-point directory (CPD) that you want to create.

blockcount — In DG/RDOS systems, a multiple of 16 greater than or equal to 48 that represents a number of contiguous blocks. In AOS/VS systems, this number represents the maximum block size of the CPD.

What It Does

In DG/RDOS, **CPART** adds a .DR extension to *partname* and sets it up as a secondary partition within a primary partition. The secondary partition has the block size specified by *blockcount*.

In AOS/VS, **CPART** creates a control-point directory called *partname* with a maximum block size specified by *blockcount*. When first created by **CPART**, the CPD is shown as having a size of zero. This changes when something is placed in the new CPD.

How To Use It

Execute this command by entering **CPART** from the Business BASIC CLI. Both arguments must follow the command.

Under DG/RDOS, the *blockcount* must be a multiple of 16 that is greater than or equal to 48. If *blockcount* is not a multiple of 16, the system sets the block size at the greatest multiple of 16 that is lower than *blockcount*.

Examples

1. In this DG/RDOS example, **CPART** creates the secondary partition **PART01.DR** in partition **DE0**. The new partition is 48 blocks long, since that is the greatest multiple of 16 that is under 50.

   ```
   !CPART DE0:PART01 50
   ```

2. In this AOS/VS example, **CPART** creates the CPD **REPORTS** with a maximum block size of 50 and then creates the CPD **MONTHLY** as a subdirectory of **REPORTS**. **MONTHLY** has a maximum block size of 20.

   ```
   !CPART REPORTS 50
   !CPART REPORTS:MONTHLY 20
   ```

CRAND

BASIC CLI Command

Creates a random file.

AOS/VS	DG/RDOS	UNIX

Format

!CRAND [*dirname*:]*filename1* [[*dirname*:]*filename2* ...]

Arguments

filename	The name of a randomly organized file to be created in your directory.
dirname	The name of an existing directory or partition where you want *filename* to reside.

What It Does

CRAND sets up a randomly organized file in the directory you specify. If no directory is specified, **CRAND** creates the file in your working directory (the default).

On DG/RDOS systems, each file has an initial length of 0 and grows as necessary. The D attribute, indicating a random file, is also set.

Under AOS/VS, **CRAND** and **CREATE** are equivalent—both create random files with default access control lists (ACLs).

On UNIX systems, **CRAND** creates a random file. The access privileges are determined by the value of the environment variable BBDEFACL.

How To Use It

Execute **CRAND** by entering the command word from the Business BASIC CLI. At least one filename must follow the command. A single command can be used to create multiple files. If you want the files created in a directory or partition other than your current one, specify a directory name immediately preceding each filename. If the directory does not exist, **CRAND** returns an error message and does not create the file.

Example

In the following example, **CRAND** creates three random files—**ANNTEST**, **MYFILE**, and **JEAN**. **ANNTEST** and **MYFILE** are created in the working directory, while **JEAN** is created in directory **GILES**.

```
!CRAND ANNTEST MYFILE GILES:JEAN
```

CREATE

BASIC CLI Command

Creates a file.

AOS/VS	DG/RDOS	UNIX

Format

!CREATE [*dirname*:]*filename1* [[*dirname*:]*filename2*] ...

Arguments

filename	The name of your new file.
dirname	The name of an existing directory where you want *filename* to reside.

What It Does

CREATE creates one or more files in a given directory. The files have no attributes and a length of zero.

On DG/RDOS systems, **CREATE** sets up a sequential file. No attributes are set on the file.

On AOS/VS systems, **CREATE** and **CRAND** are equivalent—both create random files with access control lists matching the default ACL.

On UNIX systems, **CREATE** creates a random file. The access privileges are determined by the value of the environment variable BBDEFACL.

How To Use It

Execute the command by entering **CREATE** from the Business BASIC CLI. At least one filename must follow **CREATE**. You can use a single **CREATE** command to set up multiple files. If you want the files created in a directory other than your current one, specify the name of an existing directory immediately before each filename. If the directory does not exist, **CREATE** displays an error message and does not set up the file.

Example

The files **JOEL.JC** and **FORECAST.JC** are created in your working directory, while **CAROL** is created in directory **SUSAN**.

```
!CREATE JOEL.JC FORECAST.JC SUSAN:CAROL
```

CSM *Utility*

Creates and maintains a screen file.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "CSM

What It Does

Conversational Screen Maintenance (CSM) is used to create and edit screen files on Data General terminals that have function keys. The screen files can contain multiple screen records. Each record represents one screen.

With CSM, you can specify each character position on a screen as either a prompt field or an input/output field.

The prompt fields are screen literals. When the screen is used by a program, the prompt fields appear exactly as you typed them. Use these fields to prompt the user for information or to explain something to the user. These fields are considered protected fields because you can use cursor placement subroutines to keep a user from entering data in them. Prompt fields appear in low intensity on your screen.

The input/output fields contain field definition characters to indicate the field format (for example, numeric with a floating decimal point). You use function keys and field definition characters to set up these fields (see "CSM Field Definition Characters"). These fields receive input from the user and/or display the output. These fields are considered unprotected fields because the user can input data in them. Input/output fields appear in high intensity on your screen.

The CSM subroutines **FORMIO.SL**, **SCRNIO.SL**, and **SFORM.SL** access CSM screens. These subroutines control cursor placement and input requests. Use these subroutines in your program to display the CSM screens, solicit input from the user, and wait for the user to press certain function keys to enter the input (see **FORMIO.SL**, **SCRNIO.SL**, and **SFORM.SL** in the "CSM Subroutines" section of this CSM description).

CSM uses different subroutines from the Screen Maintenance (SM) utility, but the entry points for both subroutine sets are compatible for the most part. This lets you make an easy transition between the two sets of subroutines. There are, however, a few places where the entry points differ, and, to use the SM screens, sometimes you must make more than one call to accomplish the same thing that the CSM subroutines can accomplish in one call. In addition, the subroutines for CSM have some new features.

CSM

Continued

CSM screen files differ from SM screen files because CSM uses the @() function and supports 8-bit mode. The embedded control characters that CSM uses do not conflict with 8-bit mode.

How To Use It

Since you use CSM function keys, commands, field definition characters, and subroutines (FORMIO.SL, SCRNIO.SL, and SFORM.SL) with this utility, more detailed information on these subjects follows the general overview of how to execute CSM.

Overview

On an AOS/VS system, execute the command CHAR/ON/EB1/OFF/EB0/ST/FKT before executing Business BASIC. This enables you to use the cursor positioning keys, such as Home.

On a UNIX system you must determine the terminal and mode where you will be running CSM. If your terminal uses non-DG mode and you plan to run CSM, you must bring Business BASIC up with the -C option. Since CSM is function-key driven, it runs only on terminals that support function keys. The **vt100_bbux terminfo** packet does not support function keys, so you cannot run CSM on a terminal using this packet. Also, if the terminal on which CSM is being run does not provide shifted function keys, you can use the F7 key followed by the function key you need shifted. If you select Business BASIC's DG mode (by using the -D option when you bring Business BASIC up), then you can run CSM on a Data General terminal in DG hardware mode. See *Using Business BASIC on DG/UX™ and INTERACTIVE UNIX Systems* for more information on options and terminal modes.

NOTE: On UNIX systems, you cannot use -F switch on the command line you use to bring up Business BASIC if you plan to run CSM. This is because -F emulates AOS/VS' FKT/ON, which prevents the arrow keys from working.

Execute CSM by entering RUN, CHAIN, or SWAP "CSM. CSM begins by requesting the name of the screen file. If you are going to use the screen with the Logical File Maintenance (LFM) utility, the name of the screen file must have a .S*n* extension, where *n* is the terminal type. (With LFM, you omit the .S*n* extension when you add the filename to the table file. LFM appends the .S*n* to the screen file when it reads it from the table.)

After you enter the filename, CSM clears the screen and places the cursor at row one, column one so you can create your screen.

You set up a screen by typing the prompt fields and by using function keys and field definition characters to specify input/output fields. You can enter a prompt field or define an input/output field at any place on the screen except the bottom line, which CSM reserves for command entry and error messages. In general, you define an input/output field by pressing the START FIELD function key (F1), entering the field

CSM *Continued*

definition characters, and then pressing the END FIELD function key (F2). (See the sections "CSM Function Keys" and "CSM Field Definition Characters" in this CSM description.)

When you finish setting up the screen or if you want to look at an existing screen, press the ENTER COMMAND function key (F8). This puts you on the command line. You must press F8 each time you want to enter a CSM command. After pressing F8, enter the letter of the command. CSM clears the command line and then displays the command function followed by a request for a Y or N confirmation on whether to execute the command. If the command requires additional information (e.g., a screen number), CSM then prompts you for that information (see the "CSM Commands").

For example, to define a 20-character alphanumeric input/output field on your screen, you first move the cursor to the point on the screen where the field begins. Then press the START FIELD key, type X 20 times, and press the END FIELD key. To save the screen, you press the ENTER COMMAND key and issue either a W (write) or C (change) command.

If a file contains multiple screens, use a screen number in your commands to refer to a specific screen. The first screen number is always zero.

CSM Function Keys

CSM uses function keys F1 through F8 as well as the functions associated with the key sequence SHIFT-function key (F1 through F8). You can use the SHIFT function key (F7) instead of the SHIFT key with the function keys to generate these functions. This section explains the function keys, which are listed in Table 1-2.

CUT (Shift-F5)

Clears the screen of everything from the cursor to the end of the screen and stores this information in memory. This is like cutting away information on the screen that you want to move somewhere else on the screen. You execute CUT before you execute PASTE.

DELETE CHARACTER (Shift-F3)

Deletes the character at the current cursor location.

DELETE LINE (Shift-F4)

Deletes the line at the current cursor location.

DISPLAY SCREEN (Shift-F7)

Redisplays a screen with all its fields.

DUPLICATE TO EOL (F5)

Duplicates the previous line.

CSM

Continued

END FIELD (F2)

Ends an input/output field. Characters following this key appear in low-intensity and are used as prompt information.

ENTER COMMAND (F8)

Positions the cursor to the command line and displays the prompt "CMND?". Enter a one-letter command. (See the "CSM Commands" section.)

ERASE TO EOL (Shift-F1)

Erases everything from the right of the cursor to the end of the line.

INSERT LINE (F4)

Inserts a blank line at the current cursor location.

INSERT SPACE (F3)

Inserts a space at the current cursor location.

PASTE (Shift-F6)

Restores a previously stored (**CUT**) portion of the screen at the current cursor location. The screen is treated as a continuous stream of characters without line boundaries; thus, additional cutting and pasting may be required to realign field positions on the screen. You can use the **CUT** and **PASTE** functions to move blocks of type on your screen.

SCALE TO EOL (F6)

Fills the line from the current cursor location to end of the line with a repeating scale of the digits 1234567890. The cursor then returns to the beginning of the scale so that text can be entered over the numbers.

SHIFT (F7)

Provides an optional SHIFT key. Use this key (or the keyboard SHIFT key) with keys F1 through F8. Press F7 followed by the second function key.

START FIELD (F1)

Starts an input/output field. Press this key before entering field definition characters. Characters following this key appear in high-intensity and indicate the type of data expected for the field.

TAB (Shift-F8)

Tabs to the next input/output field.

CSM

Continued

Table 1-2. CSM Function Key Summary

Function Key	Operation
F1	START FIELD
F2	END FIELD
F3	INSERT SPACE
F4	INSERT LINE
F5	DUPLICATE TO EOL
F6	SCALE TO EOL
F7	SHIFT
F8	ENTER COMMAND
Shift-F1	ERASE TO EOL
Shift-F3	DELETE CHARACTER
Shift-F4	DELETE LINE
Shift-F5	CUT
Shift-F6	PASTE (follows CUT)
Shift-F7	DISPLAY SCREEN
Shift-F8	TAB

CSM Commands

You enter **CSM** commands by pressing the ENTER COMMAND Key (F8). **CSM** then moves you to the command line and prompts you for the command. Enter the letter

that represents the appropriate command. **CSM** automatically unpends when you enter the command letter. Then **CSM** clears the command line, displays the command function, and asks for verification. Enter **Y** or **N** and press CR or New Line. All **CSM** prompts are displayed on the command line.

Some commands request additional information; however, you do not need to enter data for the documentation fields PROJECT, SYSTEM, and PROGRAM.

The **CSM** commands and responses are explained in this section.

Change Screen (C)

```
CMND? C
CMND? REPLACE CURRENT SCREEN    O.K.? Y
SCREEN [0]?
PROJECT []?
SYSTEM []?
PROGRAM []?
```

Overwrites the screen record you specify at the screen number prompt with the current screen. The default for the screen number is the last screen you referred to, and the defaults for the documentation fields PROJECT, SYSTEM, and PROGRAM are the last values given for the displayed screen.

CSM

Continued

Clear Screen (E)

```
CMND? E
CMND? ERASE SCREEN                O.K.? Y
```

Clears the terminal screen but does not affect the screen record in the file. You must execute a **D** (delete) or **C** (change) command to alter or delete the screen in the file.

Delete Screen (D)

```
CMND? D
CMND? DELETE SCREEN               O.K.? Y
```

Deletes the current screen from the screen file so that its record no longer exists.

Display Screen Prompts (O)

```
CMND? O
CMND? OUTPUT SCREEN               O.K.? Y
```

Displays the prompt portion of the current screen without displaying the input/output fields.

Exit CSM (X)

```
CMND? X
CMND? EXIT SCREEN MAINTENANCE  O.K.? Y
```

Closes the screen file and terminates the execution of **CSM**.

Insert Screen (I)

```
CMND? I
CMND? INSERT SCREEN FROM ANOTHER FILE       O.K.? Y
FILENAME []?
SCREEN [0]?
```

Inserts a screen from another screen file. The screen can be edited and then added to the file as a new screen using the **W** (write) command, or you can use the screen to replace an existing screen by using the **C** (change) command. You need to provide the name of the other screen file and the record number of the screen you want. The default screen number is zero.

Print Screen (P)

```
CMND? P
CMND? PRINT CURRENT SCREEN      O.K.?  Y
OUTPUT DEVICE [default output queue]?
```

Writes the current screen to the specified output file or default output queue. You can use **STMA 9,3** to determine the default output queue.

CSM *Continued*

Read Screen (R)

```
CMND? R
CMND? READ NEW SCREEN                    O.K.? Y
SCREEN [0]?
```

Displays the screen you specify at the screen number prompt, showing the prompt/display-only fields and the formatted input/output fields. The screen defaults to the last screen you referred to or 0.

Write Screen (W)

```
CMND? W
CMND? WRITE NEW SCREEN                   O.K.? Y
SCREEN [0]?
PROJECT []?
SYSTEM []?
PROGRAM []?
```

Writes the current screen to the new screen record you specify at the screen number prompt. The screen number cannot refer to an existing screen record. The defaults for the documentation fields PROJECT, SYSTEM, and PROGRAM are the last values given for the displayed screen.

Table 1-3. CSM Command Summary

Command	Function
C	Replaces the screen record specified by screen number with the current screen.
D	Deletes the current screen.
E	Erases the displayed screen (but this does not affect the screen in the file).
I	Inserts a screen from another screen file.
O	Displays the prompt fields of the screen specified.
P	Writes the current screen to the specified output file or default output queue.
R	Displays a screen with all its field definition characters.
W	Writes the current screen to the screen file.
X	Exits from CSM.

CSM Field Definition Characters

Use the field definition characters to specify the type of data a user can enter in an input/output field and the type of data that is output by that field.

Field definition for input/output fields and screen composition usually are done simultaneously; however, you can edit fields later if you prefer. The field definition

CSM

Continued

characters determine the field's format (see **CSM** example 1). You enter these characters after pressing the START FIELD function key. The field ends when you press the END FIELD function key. **CSM** ignores invalid characters.

Use these formats if you are going to use the subroutine **SFORM.SL** or the File Maintenance (**FM**) utility.

Table 1-4. CSM Field Definition Characters Summary

Field Code	Meaning
8,9	Numeric field. 8 Input/output fields. 9 Output fields.
- (hyphen)	Field allowing negative numeric entries. The hyphen must appear in the first position of the field.
. (period)	Decimal point location in a numeric field requiring a fixed number of decimal places on input.
: (colon)	Implied location of a floating decimal point. On input, the decimal point keyed in is scaled to the specified implied location.
>	Right-justified field. The > must appear in the first column of the field.
L,M	Lowercase alphanumeric field. L Input/output fields. M Output fields.
X,Y	Alphanumeric field. X Input/output fields. Y Output fields.

An input/output field's size is determined by the number of contiguous characters between the START and END FIELD function keys (F1 and F2). At least one character must separate any two fields.

You can define input/output numeric and alphanumeric fields by using a special set of field characters.

For numeric fields, use the character 8 to indicate an input/output field or 9 to indicate an output-only field. If the field is negative, enter a hyphen (-) in the first position of the field. A period (.) or a colon (:) marks the location of an assumed decimal point.

The period (.) indicates that you must enter exactly the number of digits specified to the right of the decimal point. Thus, for a field defined as 8888.88, an input value of 12.34 is valid and returns a value of 1234, while an input value of 1.234, 123.4, or 1234 results in an error.

CSM *Continued*

The colon (:) means that the number of digits after the decimal point can vary. Thus, for a field defined as 8888:88, an input value of 12.3 returns 1230 while an input value of 12.345 or 12.34567 returns a value of 1234. Note that an input of 1234 returns a value of 123400.

To indicate a right-justified field, type an angle bracket (>) in the first column of the field.

You define an alphanumeric field by entering Xs for input/output fields and Ys for output-only fields. An alphanumeric field that allows lowercase characters is defined by Ls for input/output fields and Ms for output-only fields.

CSM Subroutines

The subroutines **FORMIO.SL**, **SCRNIO.SL**, and **SFORM.SL** work in conjunction with **CSM** as follows:

FORMIO.SL	Displays edited screen input/output.
SCRNIO.SL	Enables edited screen input/output.
SFORM.SL	Provides formatted handling of fields.

Since these subroutines share scratch variables, the scratch variables are summarized in the following list. The input and output variables that are specific to a subroutine are listed as arguments to that subroutine. Each of the subroutines is explained in this section.

CSM

Continued

Scratch Variable List for CSM Subroutines

LINE$	A string used for input and output to screen. Dimension LINE$ to at least 132 bytes.
OLDX$	The old value of X$. Dimension OLDX$ to the length of X$.
OLDX	The old value of X.
RFORM1$	A string variable dimensioned to 8.
RFP	A pointer into RFORM$.
SCRN$	A string describing each field built by **CSM**. Dimension SCRN$ to at least 72 plus 3 times the number of fields.
SPACES$	A string dimensioned to 80 bytes or larger that is filled with spaces. SPACES$ is initialized in the subroutine at line 9150.
X10	The value of **SYS(10)** after a **READFORM** (9300). X10 indicates whether input was terminated by a function key (1) or a normal unpend (0).
XCOL	The column number of the current field.
XDEC	The number of digits past the decimal. XDEC is used to edit numeric fields.
XFLGS	Bit flags describing the field. You can set the following flags:

Bit	Meaning When Set
128	A numeric field; if not set, a string.
64	A display-only field.
32	A left-justified display field.
16	A numeric field that can have varying numbers of decimals on input (when bit 128 is also set).
16	A string field that can have lowercase characters on input (when bit 128 is not set).
8	A numeric field that can have negative input (when bit 128 is also set). If bit 128 is not set, bit 8 has no meaning.
4,2,1	Values placed in XDEC if the field is numeric.

XLEN	The length of the keyed input to subroutine **READFORM** at line number 9300. This is used since X$ is truncated to remove trailing spaces, which could create a null string.
XROW	The line number of the current field.
XSHFT	The bias factor for function keys (Shift-F7).
XWID	The width in columns of the current field.
Y%	Temporary variable.
Z%	Temporary variable.

CSM *Continued*

FORMIO.SL

Format

GOSUB 9000

Input Variables

REC$	The data record containing all fields.
RFORM$	The list of **RFORM** formats separated by commas. These formats correspond to the fields in REC$ and control the type of fields. Include the @ with each format. An **S** format indicates a crammed field. Adjacent commas indicate fields to be skipped.

This subroutine uses **SCRNIO.SL** and **SFORM.SL**, so check the input variables for those subroutines also.

Scratch Variables

See "Scratch Variable List for **CSM** Subroutines."

Output Variables

REC$	The updated record string.

This subroutine uses **SCRNIO.SL** and **SFORM.SL**, so check the output variables in those subroutines.

Line Numbers

9000	Entry point to **FORMIO.SL**
9000-9079	Line numbers for **FORMIO.SL**

What It Does

FORMIO.SL lets you display, enter, or edit an entire record with one subroutine call. The subroutine supports tab and back tab and returns to the calling program if you press either FINISH or STOP. It also returns if you tab or back tab beyond the screen field. **FORMIO.SL** uses **SCRNIO.SL** and **SFORM.SL**.

FORMIO.SL starts at a screen field indicated by the **SFORM.SL** variable F. Based on the value in the **SCRNIO.SL** input variable XMODE, **FORMIO.SL** displays and/or lets you input one or more fields in REC$. If you are in EDIT mode, **FORMIO.SL** displays the current value of the field and positions the cursor at the beginning of the field for input. If you enter the default response, the subroutine leaves the field without modifying it.

The string RFORM$ contains a list of paired **RFORM** formats, separated by commas. These formats define the fields in REC$. List the formats in the order that they appear on the screen. The RFORM$ pairs take the form *@bbf*, where *@bb* is the byte

CSM

Continued

where the field begins in REC$, and *f* is the format of the field. Unlike the **PACK** and **UNPACK** commands, **FORMIO.SL** interprets the **S** format as a crammed string. If necessary, **FORMIO.SL** automatically crams or uncrams the field for screen input/output.

This subroutine also returns the **SCRNIO.SL** output variables FUNC (with the value of the last function key pressed) and X10 (with a value indicating a tab, back tab, FINISH, or STOP). An error message indicates an invalid numeric input. **FORMIO.SL** then allows you to retype or edit the field input.

If you try to use the combination of input only (**SCRNIO.SL**'s XMODE = 1) and back tab, **FORMIO.SL** fills the preceding fields with nulls. You must retype each field to preserve its contents. A better method for **FORMIO.SL**'s input is to null fill REC$ or affected pieces of REC$ and use edit mode (XMODE = 2). This allows you to tab and back tab through fields freely.

How To Use It

To use **FORMIO.SL**, perform the following steps:

1. Enter "**FORMIO.SL**, enter "**SCRNIO.SL**, and enter "**SFORM.SL** into your program.
2. Specify the field in RFORM$.
3. Dimension the variables and set the edit mode.
4. Position the cursor to the screen field.
5. Enter the program statement **GOSUB 9000** to call **FORMIO.SL**. **FORMIO.SL** calls **SCRNIO.SL** and **SFORM.SL**.

Example

This example edits all of the data fields on the screen. The fields are in REC$.

```
00300 LET RFORM$="@3S8,@11S14,@33J,@36B,@37B,@35B,@25L"
:                 |    |      |    |    |    | *price 4-byte integer
:                 |    |      |    |    |  *year 1-byte binary
:                 |    |      |    |    *day 1-byte binary
:                 |    |      |    *month 1-byte binary
:                 |    |      *quantity on hand 2-byte integer
:                 |    *description 14-byte crammed
:                 *product code 8-byte crammed
00310 LET XMODE=2          :Set edit mode.
00320 LET F=-2             :Begin with second field (product code).
00330 GOSUB 9000           :Call FORMIO.SL
00340 ON X10 THEN GOTO 0250, 0900, 0240   :Process special function
                  :definitions: previous field, done, or abort.
```

CSM *Continued*

SCRNIO.SL

Format

GOSUB 9200

Input Variables

X$, X	The contents of a screen field. Use X$ for a string field or X for a numeric field. Do not use both X$ and X. If used, X$ must be dimensioned to at least 80 bytes.
XMODE	Specifies the type of display. 0 Display only. 1 Input only (field set to 0 or "" initially). 2 Edit (display and then input).

This subroutine uses **SFORM.SL**, so check its input variables.

Scratch Variables

See "Scratch Variable List for **CSM** Subroutines."

Output Variables

F	The next field number (adjusted for the function key pressed).
FUNC	The function key terminator (or -1).
XLEN	The length of new input (if any).
X10	The results of pressing a function key. 0 Tab (Return or any other function key). 1 BACK TAB (F8). 2 FINISH (F6). 3 STOP (F14).

This subroutine uses **SFORM.SL**, so check its output variables.

Line Numbers

9200	Entry point to **SCRNIO.SL**
9200-9262	Line numbers for **SCRNIO.SL**

What It Does

SCRNIO.SL works with the subroutines in **SFORM.SL** to enable an additional level of editing for terminals that use **CSM** formatted screens. **SCRNIO.SL** specifies a field for terminal editing, field display, or input. **SCRNIO.SL** uses **SFORM.SL** to perform the actual input and output to the terminal.

Whether **SCRNIO.SL** inputs, displays, or edits the current field depends on the value of XMODE. A value of 0 means to display the field indicated by the **SFORM.SL**

CSM

Continued

input variable F. X$ contains the contents of a string field, while X contains the contents of a numeric field. Only field relative addressing (-F) is allowed in **SCRNIO.SL**. A value of 1 in XMODE indicates the field is for input only. String input goes to X$, while numeric input goes to X. A value of 2 in XMODE indicates the field is for editing. Editing means displaying the current field with the cursor positioned at its beginning so you can enter a replacement value or use the default response to return the original value. If you give a default response, the subroutine returns the old value as the new value, returns FUNC, and updates F.

The edit routine also returns the **SFORM.SL** output variable FUNC if a function key terminated the input. It returns X10 to indicate TAB, BACK TAB, FINISH, or STOP.

How To Use It

To use **SCRNIO.SL**, follow these steps:

1. Enter "**SCRNIO.SL** and enter "**SFORM.SL** into your program.
2. Dimension the variables, and specify the fields to be edited.
3. Include the program statement **GOSUB 9200** to call **SCRNIO.SL**.

Numeric fields are formatted according to the specifications in the screen file. **SCRNIO.SL** indicates invalid numeric inputs. Initiate the normal error sequence by using function key sequence Shift-F8 as an error acknowledgment. This lets you re-edit the field.

Example

This sample program uses the edit mode to rekey the month, day, and year fields from the **CSM** print screen example.

```
00620 LET F=-5                  :Month is at relative field 5.
00630 LET XMODE=2               :XMODE is set for edit.
00640 LET X=MONTH
00650 GOSUB 09200               :Entry point for SCRNIO.SL.
00655 IF X<1 THEN GOTO 00640    :Check for legal number for month.
00656 IF X>12 THEN GOTO 00640
00660 LET MONTH=X               :Month gets value in X.
00670 ON X10 THEN GOTO 00550, 00900, 00240  :Handle special function
                                :descriptions:
                                :previous function, done, abort.
00680 LET X=DAY                 :Edit the day.
00690 GOSUB 09200               :Call SCRNIO.SL.
00695 IF X<1 THEN GOTO 00680    :Rough check for legal day.
00696 IF X>31 THEN GOTO 00680
00700 LET DAY=X                 :Day gets value in X.
00710 ON X10 THEN GOTO 00640, 00900, 00240
00720 LET X=YEAR                :Edit the year.
```

CSM

Continued

```
00730 GOSUB 09200               :Call SCRNIO.SL.
00735 IF X<69 THEN GOTO 00720   :Check range of acceptable years.
00736 IF X>99 THEN GOTO 00720
00740 LET YEAR=X                :Year gets value in X.
00750 ON X10 THEN GOTO 00680, 00900, 00240
```

CSM

Continued

SFORM.SL

Format

GOSUB *linenumber*

Argument

linenumber	The entry point to the specific **SFORM.SL** routine you want. The entry points are listed under "Line Numbers."

Input Variables

E	A Business BASIC error code number.
F	The number of a field. F can be a field number within a row when R contains a row number, or F can contain a row number times 100 plus the field number within the row. The field number can also be -1 to *n*, where *n* is the number of fields. This enables order referencing of fields.
R	The row number. R is used with F as a field specification when 0 < F < 100.
X	Variable used to display numeric fields.
X$	Variable used to display string fields. Dimension X$ to at least 80 bytes.

Scratch Variables

See "Scratch Variable List for **CSM** Subroutines."

Output Variables

E	Variable containing an error code. The value in E indicates data errors on input.
FUNC	The function number returned on a READ or WAIT for FUNC (function).
X	The value read from a numeric field.
X$	The value read from a string field. Dimension X$ to at least 80 bytes.

Line Numbers

9100	Entry point to CLEARFORM—clears all unprotected fields to underlines.
9150	Entry point to INITFORM—sets the necessary **STMA** statements and initializes internal variables.
9175	Entry point to ENDFORM—resets the **STMA** statements set by entry 9150 and clears the screen.
9300	Entry point to READFORM—puts the field in either X$ (string field) or X (numeric field). The routine returns in E either 0 if

CSM

Continued

	the input conversion succeeds or 128 if an error occurs. String input is converted to uppercase, unless lowercase is allowed. Pressing CR or a function key terminates input. READFORM calls READFUNC (9650) to return FUNC.
9500	Entry point to WRITEFORM—outputs the field from either X$ (string field) or X (numeric field). Numeric fields are formatted according to specifications in the screen file.
9650	Entry point to READFUNC—an internal routine that reads a function key and returns a value of 1 to 16 in FUNC. FUNC contains 0 if no function key was pressed and -1 if an illegal function key was detected. READFUNC normally is called by READFORM, not by a direct program statement. If you do any inputs besides those in READFORM, you should set X10 to the value of **SYS(10)**. Then call READFUNC to check X10. If X10 does not equal 0, a function key ended the last input.
9700	Entry point to OUTSCRN—displays screen number X from file number 15 and loads SCRN$.
9722	Similar to 9700 (OUTSCRN), except that the screen is not displayed (i.e., reread SCRN$ only).
9750	Entry to WAITFUNC—waits for a function key to be pressed and then calls READFUNC (9650) to return a value from 1 to 16 in FUNC (FUNC contains -1 if the input was an illegal function key).
9775	Entry point to WAITCLEAR—waits at the current field for function key F16. It then clears the command (line 24) before returning.
9800	Entry point to SCRNSET—converts a field in F (or in F and R if 0 < F < 100) into XROW, XCOL, XPOS, XWID, XFLGS, and XDEC.
9850	Entry point to ERROUT—displays an error message on line number 24 of the screen. If E does not equal 0, then ERROUT uses the message from the **ERM$** function. If E equals 0, then the user-supplied error message in X$ is displayed.
9890	Entry point to ERRCLEAR—clears the error message from the screen.

What It Does

SFORM.SL contains routines that provide formatted handling of numeric and string fields for a **CSM** screen. The subroutine functions include displaying a screen, entering a function, and entering and displaying string and numeric fields.

SFORM.SL has several entry points accessed by **GOSUB** statements. Arguments are passed through the appropriate variables.

CSM

Continued

How To Use It

To use **SFORM.SL**, perform the following steps:

1. Enter "**SFORM.SL** into your program.
2. Set the terminal's input/output characteristics by dimensioning the string variables and executing a **GOSUB 9150** (INITFORM entry).
3. Include a **GOSUB** *linenumber* statement for each of the **SFORM.SL** routines your program uses.
4. Execute a **GOSUB 9175** (ENDFORM) before you exit the program to reset the previous terminal characteristics.

SFORM.SL Routines

To display a **CSM** screen file, open the screen file for reading on Business BASIC channel 15. Then assign the number of the screen desired in the variable X and execute a **GOSUB 9700** (OUTSCRN entry). This displays the screen file and builds the SCRN$ table that defines the screen fields for future input and output.

Example

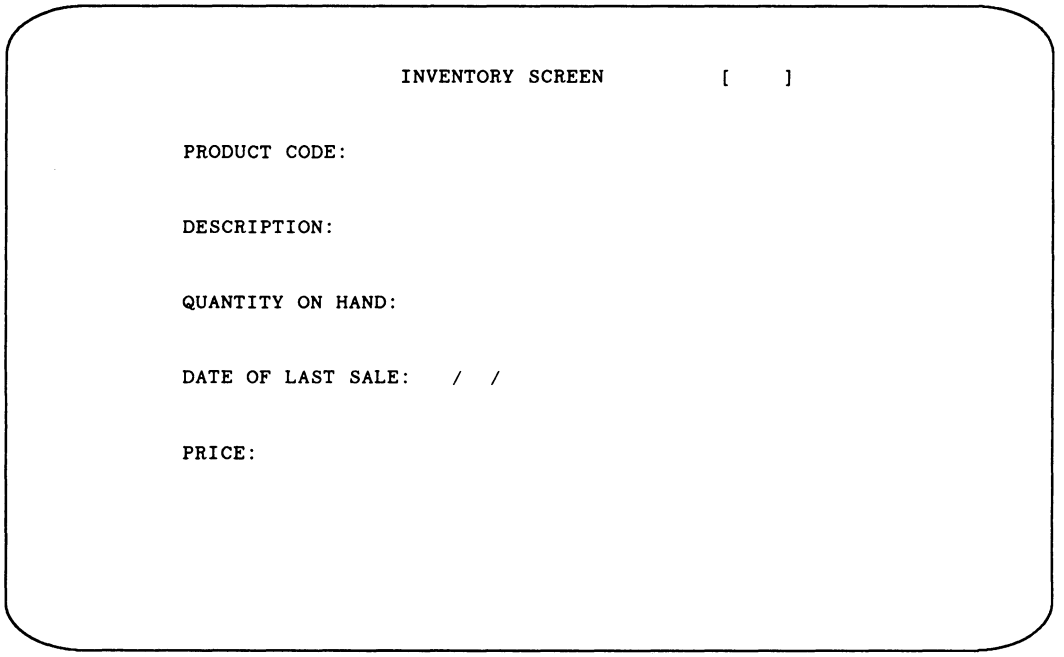
If you enter this program segment, a screen similar to the following one appears:

```
00050 DIM X$[80],SPACES$[80],LINE$[132],SCRN$[93]
00090 OPEN FILE[15,4],"INVII.S6"      :Open the screen file.
00110 GOSUB 09150                     :Prepare the terminal
                                      :characteristics.
00200 LET X=0                         :Display screen number 0.
00210 GOSUB 09700                     :Call OUTSCRN.
```

CSM

Continued

Output screen:



Screen Field Input/Output

SFORM.SL has entries at 9300 (READFORM) and 9500 (WRITEFORM) that allow input/output to the screen fields. Most routines require a field specification. There are three ways to supply a field specification (the value of F determines which method is used):

- Set R to the line number on the screen and F to the field within the line as defined by the **CSM** program.
- Set F to the line number times 100 plus the field within the line; in this case, R is not needed.
- Set F to the negative of the actual field number on the screen.

Examples

1. This code segment reads in the PRODUCT CODE field of the INVENTORY SCREEN using the first method of setting the row and field using R and F.

```
00300 LET R=3          :Product code is on row (line) 3
00310 LET F=1          :and is the first field on that row.
00320 GOSUB 09300      :Call READFORM.
00330 LET PCODE$=X$    :Save the product code returned in X$.
```

CSM

Continued

2. This code segment reads in the PRODUCT CODE field of the INVENTORY SCREEN using the second method, where F is set to the field number plus 100 times the row number.

```
00300 LET F=301          :Address is combined as R*100+F.
00310 GOSUB 09300        :Call READFORM.
00320 LET PCODE$=X$      :Save the product code.
```

3. This code segment reads in the PRODUCT CODE field of the INVENTORY SCREEN using the third method, where F is set to the negative field number. The product code is the second field; the function display field is the first. This field could also have addresses of R=1 and F=1 or F=101.

```
00300 LET F=-2           :Product code is the second field
                         :on the screen.
00310 GOSUB 09300        :Call READFORM.
00320 LET PCODE$=X$      :Save the product code.
```

To operate READFORM, you position the cursor to the field location specified by the F and R coordinates. Then you execute an **INPUT USING** statement with a limited maximum field size. READFORM calls READFUNC (9650) to determine whether CR or a function key terminated the input and then returns the variable FUNC with the function code. You should check FUNC following READFORM. Thus, add the following code to the earlier examples:

```
00340 ON FUNC THEN GOTO 00100,02000,03500,09990
                  :Vector on function key.
                  :Return here if no function key or an
                  :invalid function key was pressed.
```

FUNC now contains one of the following:

- A number from 1 through 16, indicating that the function key was pressed.
- A 0, indicating the last unpend was not a function key header.
- A -1, indicating the function key character was invalid.

If you want to ignore other key strokes and wait for a function key, you can use the entry WAITFUNC (9750). If F equals 0, the cursor is left at its current location; otherwise, the cursor is positioned to the field indicated by F and R. All input is discarded until a function key is pressed; then FUNC is returned.

CSM

Continued

READFORM (9300) and WRITEFORM (9500) use the same methods of addressing fields. To display values with WRITEFORM, you set either the variable X$ (a string field) or X (a numeric field) before entering a **GOSUB** statement. Use the following code to display the DESCRIPTION and QUANTITY ON HAND on the INVENTORY SCREEN:

```
00600 LET F=501              :Description is field 1 on row 5.
00610 LET X$=DESC$           :Use X$ since field is string.
00620 GOSUB 09500            :Call WRITEFORM.
00650 LET F=701              :Quantity on hand is field 1 on row 7.
00660 LET X=QTYOH%           :USE X SINCE FIELD IS NUMERIC.
00670 GOSUB 09500            :CALL WRITEFORM.
```

If display characters exceed the defined width of the field, the entry WRITEFORM places asterisks (*) in the field to indicate a field overflow.

To clear all of the fields on the screen, use **GOSUB 09100** to call the CLEARFORM entry of **SFORM.SL**. This sets all field columns to spaces or underscores, depending on the particular screen.

Frequently, a program checks the input data for range, type, or some other validation criteria. **SFORM.SL** displays error messages and any non-field oriented dialog on line 24 (the command line). The ERROUT (9850) entry places a message on the command line. If the variable E does not equal 0, ERROUT uses the **ERM$** function to determine the error code in E. If E equals 0, then ERROUT displays the user-supplied error message in X$ at the command line.

The ERRCLEAR (9890) routine clears the command line. The WAITCLEAR (9775) routine waits for function key F16 as an error acknowledgment by the operator and follows it with an automatic clearing of the command line. During a WAITCLEAR, the cursor remains at the field indicated by F and R. This designates the field in error. The READFORM entry sets E to an appropriate error code when you enter non-numeric data in a numeric field. The following code lets you read the QUANTITY ON HAND field used in the INVENTORY SCREEN:

```
00420 LET F=701              :Quantity on hand is field 1, row 7.
00430 GOSUB 09300            :Call READFORM.
00440 IF E=0 THEN GOTO ---- :Next field.
00450 GOSUB 05000            :Call my error handler.
00460 GOTO 00420             :Try it again.
.
.
.
05000 REM MY ERROR HANDLER
05010 GOSUB 09850            :Call ERROUT for message in ERM$(E).
```

CSM

Continued

```
05020 GOSUB 09775              :Call WAITCLEAR.
05030 LET E=0                  :Use my message.
05040 LET X$="PROGRAM ERROR":Error message.
05050 GOSUB 09850              :Call ERROUT.
05060 DELAY 20
05070 GOSUB 09890              :Call ERRCLEAR.
05080 RETURN
```

CSM Screen File Layout

The screen file contains the definition of the screens used with **SFORM.SL**. The **CSM** utility builds and maintains the screen file. If the file contains multiple screens, each screen makes up one record of 5,120 bytes. The length for the last screen varies. (If the file contains only one screen, then that screen is the last screen.) The first record is always record number zero. To use the screen file with **SFORM.SL**, you must open the file on channel 15 and use the formats found in Table 1-5.

Section A has the start of the screen label information section and includes fields 1 through 8. This section documents the screen and is not normally used by the program unless label checking is required by design. Field 1 is significant in that a 1 in this field identifies an active screen record, and a 0 indicates a deleted record.

Section B is the field definitions section. This section makes up a variable-length record within the screen record. It contains the definition of SCRN$. Field 1 is the length of SCRN$, starting with the first field definition (field 2) for line 1. Subsequent field definitions follow immediately.

In field 2, the variables XCOL, XWID, and XFLGS contain fields 2A, 2B, and 2C for the current field accessed.

The number of screen definitions (section B) matches the number of fields on the screen plus one additional field per line (24 lines). The extra field per line contains all zeros and marks the end of a line's definition.

Section C is the screen literals section. It consists of one or more sets of field 1 (fields 1A through 1D). Field 1A contains the relative row number of the 1D field immediately following it. Field 1B contains the relative column number of the following 1D field. Field 1C contains the length of the following 1D field. Field 1D is a variable-length string containing a portion of some screen literal beginning at the coordinates @(1A,1B) for length 1C. Successive literal definitions follow, until a definition with zero coordinates terminates the list.

CSM

Continued

Table 1-5. Screen File Record

Field Contents	Location	Size	Type
Section A: Screen label information	1	128	—
1. Record status (1 or 0)	1	2	numeric
2. Last modification time (HHMMSS)	3	4	numeric
3. Last modification date (MMDDYY)	7	4	numeric
4. Last modifier's account number	11	6	string
5. Project identification	17	30	string
6. System identification	47	30	string
7. Program identification	77	30	string
8. Reserved	107	22	—
Section B: Field definitions (SCRN$)	129	variable	—
1. Length of SCRN$	129	2	numeric
2. Field definitions (1/field/line)	131	#fields*3	—
2A. Field column (XCOL)	—	1	numeric
2B. Field width (XWID)	—	1numeric	
2C. Field flags (XFLGS)	—	1	numeric
Section C: Screen literals	1537	variable	—
1. Screen literal record	1537	3+len(literal)	—
1A. Literal row	—	1	numeric
1B. Literal column	—	1numeric	
1C. Literal length	—	1	numeric
1D. Literal	—	variable	string

CSM *Continued*

CSM Examples

1. The following **CSM** screen was printed using the P command. The information at the top tells you about the screen; in this case, the optional documentation fields were not filled in.

```
PROJECT:

SYSTEM:                              PROGRAM:

SCREEN ID: CSMEMP.S8[0] LAST MODIFIED BY: CAROL8 AT 16:50:41 ON 03/05/85

SCRN$[84]
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
.................................................................................
1:                          EMPLOYEE FILE MAINTENANCE                           : 1
2:                                                                              : 2
3:                                                                              : 3
4:                                                                              : 4
5:                                                                              : 5
6:EMPLOYEE LAST NAME:  LLLLLLLLLL                                               : 6
7:                                                                              : 7
8:EMPLOYEE FIRST NAME:  LLLLLLLLLL                                              : 8
9:                                                                              : 9
10:EMPLOYEE EXTENSION:  8888                                                    :10
11:                                                                             :11
12:EMPLOYEE ID:  99-888                                                         :12
13:                                                                             :13
14:                                                                             :14
15:                                                                             :15
16:                                                                             :16
17:                                                                             :17
18:                                                                             :18
19:                                                                             :19
20:                                                                             :20
21:                                                                             :21
22:                                                                             :22
23:                                                                             :23
 :.............................................................................:
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

You can create this screen by using the following **CSM** commands.

CSM

Continued

```
* RUN "CSM
```

CSM then clears the screen and begins a multi-screen dialog. **CSM** first asks for a screen filename. Enter the name of the file you want to create or the name of an existing screen file. If you are going to use the file with **LFM**, enter the name with a **.S***n* extension, where *n* is your terminal type. (You can use **STMA 1,0** to determine your terminal type.)

```
DG SCREEN FILE MAINTENANCE  REV X.XX
SCREEN FILENAME:
```

CSM then clears your terminal screen and places the cursor at row 1, column 1. Now you can enter your prompt fields.

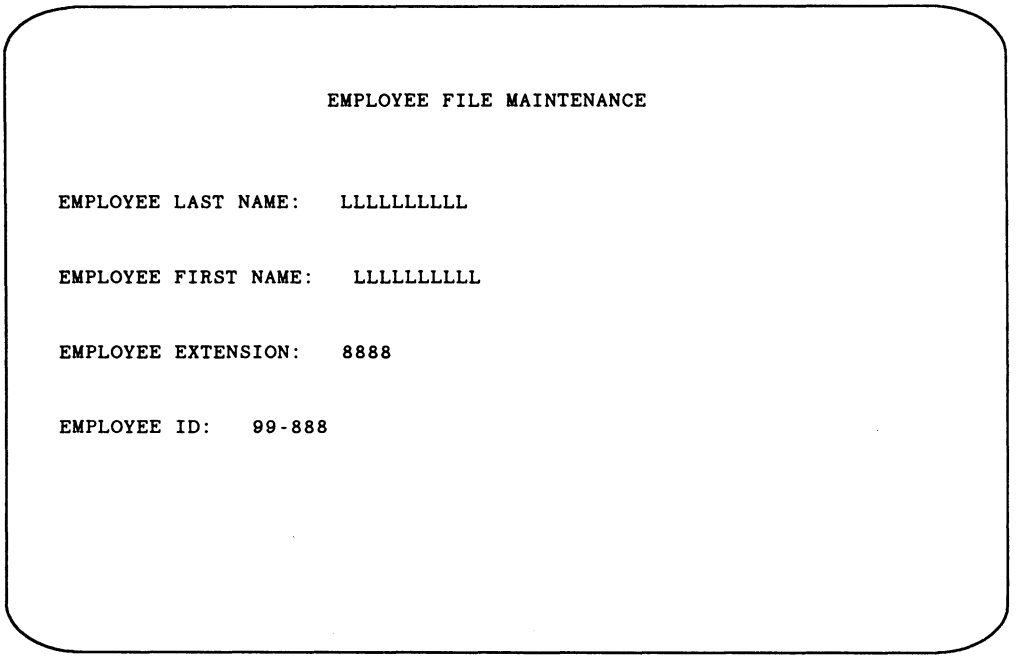
To create the screen at the beginning of the example, space to column 25, and type the screen heading, **EMPLOYEE FILE MAINTENANCE**. Then position the cursor to row 6, column 1. To enter prompts for your **CSM** screen, do the following:

1. Type the prompt.
2. Press the START FIELD (F1) function key to begin an input/output field.
3. Type field definition characters necessary to format the input/output fields (see Table 1-4, "Field Definition Characters Summary").
4. Press the END FIELD (F2) key.
5. Position the cursor to the next prompt line.
6. Repeat steps 1-5, until you have entered everything.

When you finish entering your prompts, press the ENTER COMMAND (F8) key. This moves the cursor to the command line at the bottom left of the screen and displays the prompt, `CMND?`. Enter a **W** to write the screen record to the file. **CSM** then prompts for a screen number. If there are no screen records in this file, use the default value 0; otherwise enter the number for a new screen record in the file. This record cannot exist. If you keyed everything in correctly, the following is displayed:

CSM

Continued



Press the ENTER COMMAND (F8) key again and enter **X** to exit **CSM**.

2. After the screen is built, enter the following program **TEST1**. You must also include **SFORM.SL**, **POSFL.SL**, and **GETREC.SL** in this program. The program uses the **CSM** screen you created, so you can run **TEST1**.

```
* LIST
00010 STMA 6,5                        :Do not allow interrupts
00020 DIM C1[2,3],B$[544],LX$[18],FX$[18]   :Scratch variable
                                            :for opening files
00030 DIM LAST$[10],FIRST$[10],REC$[64]  :Variables for data file
                                         :records
00040 DIM X$[80],LINE$[132],SCRN$[87],SPACES$[80]:Variables for
                                         :screen file
00050 LET F%,R1,Y%,E,X,X10=0             :Scratch variables
00060 REM ** Routine to OPEN files and set up the C1 array
00070 LET B$="EMP,5,LAST,5,FIRST,5",FILL$(0)
                                         :Data and Index files
00080 BLOCK WRITE B$
00090 SWAP "OPEN"                        :Swap to the OPEN utility
00100 BLOCK READ B$
00105 UNPACK "JJ",B$,ERRIN,ERRNO
00110 IF ERRIN<>-1 THEN GOTO 00140    :Check for errors from OPEN
00120 PRINT "ERROR # ";ERRNO;" - ";B$[5,512]
00130 END
00140 LET K=1
00150 FOR I=0 TO 2                    :Build or assign a C1 array
00160   FOR J=0 TO 3
```

CSM

Continued

```
00170     LET C1[I,J]=ASC(B$[K,K+3])

00180     LET K=K+4
00190   NEXT J
00200 NEXT I
00210 LET LX$=CHR$(C1[1,0],2),CHR$(C1[1,1],4),CHR$(0,2),"LAST",FI
L$(0)
00220 LET FX$=CHR$(C1[2,0],2),CHR$(C1[2,1],4),CHR$(0,2),"FIRST",F
ILL$(0)
00230 OPEN FILE[15,4],"CSMSCREEN"  :Open screen file
00240 GOSUB 09150 : \ INITFORM     :Set keyboard and STMAs (SFORM.
SL)
00250 LET SCRN=0                   :Screen number
00260 GOSUB 09700 : \ OUTSCRN      :Display screen (SFORM.SL)
:
: ** Display INPUT/OUTPUT fields of the screen
:
00280 LET F=601                    :Row 6, field 1
00290 LET X$=""                    :Initialize X$
00300 LET X$[1,10]=FILL$(32):Fill string with spaces for output
00310 GOSUB 09500 : \ WRITEFORM:Output string to field (SFORM.SL)
00320 LET F=801                    :Row 8, field 1
00330 LET X$=""                    :Initialize X$
00340 LET X$[1,10]=FILL$(32):Fill string with spaces for output
00350 GOSUB 09500 : \ WRITEFORM:Output string to field (SFORM.SL)
00360 LET F=1001                   :Row 10, field 1
00370 LET X=0                      :Initialize X
00380 GOSUB 09500 : \ WRITEFORM:Output X to field (SFORM.SL)
00390 LET F=1201                   :Row 12, field 1
00400 LET X=20                     :Set output-only field
00410 GOSUB 09500 : \ WRITEFORM:Output X to field (SFORM.SL)
00420 LET F=1202                   :Row 12, field 2
00430 LET X=0                      :Initilize X
00440 GOSUB 09500 : \ WRITEFORM:Output X to field (SFORM.SL)
:
: ** Request input into INPUT/OUTPUT fields
:
00460 LET F=601                    :Row 6, field 1
00470 GOSUB 09300 : \ READFORM  :Read formatted input (SFORM.SL)
00480 LET LAST$=X$,FILL$(32)    :Assign input to record variable
00490 LET F=801                    :Row 8, field 1
00500 GOSUB 09300 : \ READFORM  :Read formatted input (SFORM.SL)
00510 LET FIRST$=X$,FILL$(32)   :Assign input to record variable
00520 LET F=1001                   :Row 10, field 1
00530 GOSUB 09300 : \ READFORM  :Read formatted input (SFORM.SL)
00540 LET EXT=X                    :Assign input to record variable
00550 IF E=0 THEN GOTO 00610    :Check for error
00560 LET E=128                    :If error, make it error 128
00570 GOSUB 09850 : \ ERROUT    :Output error (SFORM.SL)
```

CSM

Continued

```
00580 GOSUB 09775 : \ WAITCLEAR:Wait for SHIFT-F8 to clear error
00590 LET E=0                  :Reset error to 0
00600 GOTO 00520               :Go back and read field again
00610 LET F=1202               :Row 12, field 2
00620 GOSUB 09300 : \ READFORM :Read formatted input (SFORM.SL)
00630 LET ID=X+20000           :Assign input to record variable
00640 IF E=0 THEN GOTO 00700   :Check for error
00650 LET E=128                :If error, make it error 128
00660 GOSUB 09850 : \ ERROUT   :Output error (SFORM.SL)
00670 GOSUB 09775 : \ WAITCLEAR:Wait for SHIFT-F8 to clear error
00680 LET E=0                  :Reset error to 0
00690 GOTO 00610               :Go back and read field again
00700 PACK "ZJA10A10LJ",REC$,1,LAST$,FIRST$,EXT,ID
                               :Build record string
00710 LOCK 1,"EMP",0,C1[F%,3]  :Lock record 0 of data file
00720 GOSUB 08400 : \ GETREC.SL:Get next available record in R1
00730 GOSUB 09610 : \ POSFL.SL :Position to record R1
00740 WRITE FILE[C%],REC$      :Write record
00750 KADD LX$,B$,LAST$,R1     :Add key to LAST index using R1
00760 IF R1<=0 THEN GOTO 00880 : ** INDEX ERROR ROUTINE
                               :Check for error
00770 KADD FX$,B$,FIRST$,R1    :Add key to FIRST index using R1
00780 IF R1<=0 THEN GOTO 00880 : ** INDEX ERROR ROUTINE
                               :Check for error
00790 UNLOCK                   :Unlock data record
00800 LET X$="F6=STOP   F8=CONTINUE"  :Operator choices
00810 LET E=0                  :Let error = 0
00820 GOSUB 09850 : \ ERROUT   :Output choice message (SFORM.SL)
00830 PRINT @(24,30);          :Position cursor to receive
                               :function key
00840 GOSUB 09750 : \ WAITFUNC :Read function key (SFORM.SL)
00850 IF FUNC=6 THEN GOTO 00910:Look for F6
00860 IF FUNC=8 THEN GOTO 00280:Look for F8
00870 GOTO 00810               :If neither, go back and try again
:
00880 REM ** INDEX ERROR ROUTINE
00890 LET X$="<7>ERROR IN ADD<7> - F10=STOP"   :Assign error mess
age
00900 GOTO 00810               :Go display error message
:
: ** Routine to end and print data records
:
00910 GOSUB 09175 : \ ENDFORM  :Reset terminal for normal use
00920 STMA 7,5                 :Allow interrupts
00930 LET LAST$=""             :Null key
00940 KFIND LX$,B$,LAST$,R1    :Find 1st record
00950 LET R1=ABS(R1)           :Expecting a negative return
00960 IF R1<=0 THEN GOTO 01040    :Check for end-of-file
00970 GOSUB 09610 : \ POSFL.SL    :Position to record
```

CSM

Continued

```
00980 READ FILE[C%],REC$          :Read record
00990 UNPACK "JA10A10LJ",REC$,X,LAST$,FIRST$,EXT,ID
                                  :UNPACK record
01000 PRINT USING "A10,3X,A10,3X,D5.0,3X,D6.0",LAST$,FIRST$,EXT,I
D                                 :Print
01010 KNEXT LX$,B$,LAST$,R1       :Get next key
01020 GOTO 00970                  :Go read until the end of file
01030 CLOSE
01040 END
```

```
* ENTER "SFORM.SL
* ENTER "POSFL.SL
* ENTER "GETREC.SL
* SAVE "TEST1
```

DBFIX *Utility*

Adjusts the characteristics of logical database files.

AOS/VS

What It Does

DBFIX aids you in porting files by converting the element size and adjusts the file type of logical database files (.VL and .DB) loaded from DG/RDOS. Information on this utility, like other program conversion tools, is documented in the on-line file **CONVERT.DOC**, located in the Business BASIC directory **DOC**.

DBGEN *Utility*

Builds files for the PARAM file structure.

AOS/VS	DG/RDOS	UNIX

Format

RUN "DBGEN

What It Does

The Database Generator (**DBGEN**) utility is an interactive ISAM file builder used with the PARAM file database structure. It creates a random ISAM file that uses linked-available records. The **DBGEN** multi-screen dialog lets you define the parameters and characteristics of the file structure. To do this, the **DBGEN** utility program chains to these utility programs:

DBINIT	(A modification of **INITFILE**.) Initializes the database and the indexes of the ISAM file based on the information **DBGEN** provides.
DBOPEN	Opens the file to enable the File Maintenance (**FM**) utility to write to it.
DBFM.T6	(A modification of **FM**.) Builds a table file, using information from the files created by **DBGEN**.
DBPRINT	An interactive utility that asks whether you want a line printer copy of your table file information. A yes (**Y**) response sends a copy of the table file to the line printer. A no (**N**) response displays the table file information at your terminal.

With these utilities, **DBGEN**:

- Allows you to define your database structure via a multi-screen question and answer session. In this phase you define the number of records, the types of records, the record length, and any index files your database requires. You can have a maximum of three indexes.
- Builds index files from the index file parameters you specify during the **DBGEN** dialog. **DBGEN** supports 512-byte index blocks (on AOS/VS, DG/RDOS, and UNIX systems) and 2048-byte blocks (on AOS/VS and UNIX systems only).
- Initializes and opens all files in your database structure.
- Builds and displays a table file containing the parameters and specifications for your database. The table file is used by **FM**, thus enabling you to update your files.

The versions of **DBGEN** are:

- Double precision **DBGEN**. Valid lengths for numeric fields are two, three, and four.

DBGEN *Continued*

- A triple precision **DBGEN** that stores and retrieves fields with a maximum of four bytes. Valid lengths for numeric fields are two, three, and four.
- **DBGENT**, a triple precision version of **DBGEN** that stores and retrieves numeric fields with a maximum of six bytes. Valid lengths for numeric fields are two, three, four, five, and six.

FMPRINT works with double and triple precision **DBGEN** only. It does not work with **DBGENT**. (Use **FMPRINTT** with **DBGENT**.)

In addition, you must use **FMT** with files created by **DBGENT**. **FM** works only with **DBGEN** files.

How To Use It

The only way to execute **DBGEN** is to enter **RUN "DBGEN**. Once executed, **DBGEN** starts a multi-screen dialog.

Do not swap to **DBGEN** (i.e., do not enter **!DBGEN**, **"DBGEN** or **SWAP "DBGEN**).

When answering the **DBGEN** prompts, use capital letters for all single character responses. On AOS/VS systems, press New Line after each entry. (Using CR as a terminator causes **DBGEN** to abort on AOS/VS.) On DG/RDOS systems, press CR after each entry.

DBGEN returns you to keyboard mode when it finishes.To leave **DBGEN** before the dialog finishes, press the interrupt key.

Example

This AOS/VS example uses a tax application to show how **DBGEN** works. The application requires a record with four fields (name, address, income, and tax), so you enter 4 in response to the first **DBGEN** question.

DBGEN *Continued*

```
* RUN "DBGEN

......DEFINING FILE STRUCTURE.....

A RECORD FIELD IS A GROUP OF CHARACTERS OR NUMBERS DESCRIBING
A DATA ITEM IN A DATA RECORD (E.G., NAME, ADDRESS, TELEPHONE ).

HOW MANY FIELDS DO YOU WANT IN YOUR DATA RECORDS?:  4
```

After you enter the numeral 4, **DBGEN** clears your screen and displays the second screen of its dialog, where you define your fields. For each field, **DBGEN** prompts you for the field size, then the field description, and the field type (a numeric (N) or string (S) field). The utility waits until you answer a prompt before displaying the next one. **DBGEN** automatically assigns a numeric field a length of four bytes. This example uses two string fields, NAME (with a field length of ten bytes) and ADDRESS (with a length of fifteen bytes), and two numeric fields, INCOME and TAX, both of which have a length of four bytes.

```
......DEFINING FILE STRUCTURE.....

THE SIZE OF A CHARACTER FIELD IS THE NUMBER OF CHARACTERS,
SPACES AND DIGITS IT CONTAINS. THE MAXIMUM SIZE OF A
NUMERIC FIELD IS 4 BYTES (2147483647). THE FIELDS MAY BE
NUMERIC (N) OR STRING (S).

DEFINING 4 FIELDS

FIELD SIZE 1: 10 DESCRIPTION: NAME          (N OR S): S
FIELD SIZE 2: 15 DESCRIPTION: ADDRESS       (N OR S): S
FIELD SIZE 3:  4 DESCRIPTION: INCOME        (N OR S): N
FIELD SIZE 4:  4 DESCRIPTION: TAX           (N OR S): N
```

DBGEN *Continued*

Screen three deals with the index file definition. **DBGEN** asks how many indexes you want. It then prompts you for the field number and asks you to verify the information by entering either **Y** or **N**. For each index, the utility asks if duplicate records are allowed. This example uses two index files, which are pointers to the records via the NAME and TAX fields. The index block size is 512 bytes per block. (With AOS/VS, **DBGEN** prompts you for the index block size, which can be either 512 bytes or 2048 bytes. With DG/RDOS, **DBGEN** displays INDEX BLOCK SIZE 512.) Duplicate keys are allowed for both indexes.

```
.....DEFINING INDEX STRUCTURE.....

AN INDEX CONSISTS OF KEYS POINTING TO COMPARABLE FIELDS IN ALL THE
DATA RECORDS. FOR EXAMPLE YOU CAN HAVE AN INDEX CONTAINING ALL THE
NAME KEYS.

A NAME INDEX ALLOWS DUPLICATE KEYS (E.G., 2 SMITHS). SOME NUMERIC IN-
DEXES ALLOW DUPLICATE KEYS (E.G., CREDIT LIMIT).
OTHERS MUST BE UNIQUE (E.G., ACCOUNT NUMBER).

NUMBER OF INDEXES TO BE BUILT (MAX=3): 2

FIELD NUMBER FOR INDEX 1 1 DESCRIPTION NAME       CORRECT (Y,N): Y
FIELD NUMBER FOR INDEX 2 4 DESCRIPTION TAX        CORRECT (Y,N): Y

INDEX BLOCK SIZE (512 OR 2048) [512]: 512
DUPLICATE KEYS (Y OR N): Y
```

The fourth screen deals with defining the file structure. It requests the number of records needed by your database and asks you to supply the names of your database and your index files. This example uses a database with 1,000 records. The database name is **TAXES**, and the label names for the index files are **NAME.IX** and **TAX.IX**. It does not matter whether the index filenames are identical to the record field names or different.

DBGEN *Continued*

```
.....DEFINING FILE STRUCTURE.....

AN INDEX LABEL SHOULD REFLECT ITS FUNCTION. FOR EXAMPLE, THE LABEL OF
A NAME FIELD COULD BE NAMES

WARNING: TRUNCATES LABELS TO MAXIMUM OF 10 CHARACTERS.


NUMBER OF RECORDS: 1000
 DATA BASE NAME TO BE: TAXES

INDEX 1 POINTS TO NAME LABEL OF INDEX : NAMES.IX
INDEX 2 POINTS TO TAX  LABEL OF INDEX : TAX.IX
```

The fifth **DBGEN** screen summarizes your file specifications for the database. At the bottom line, **DBGEN** asks you to confirm your data structure. If you enter **NO** (or **N**), **DBGEN** redisplays the first screen so that you can start over again. If you enter **YES** (or **Y**), **DBGEN** chains to the utilities listed under "What It Does."

```
YOU HAVE DEFINED THE FOLLOWING STRUCTURE:

DATABASE FILE.......TAXES
CONSISTS OF........1000 RECORDS OF 4 FIELDS EACH

FIELD   SIZE
1       10       DESCRIPTION: NAME       INDEX LABEL: NAMES.IX
2       15       DESCRIPTION: ADDRESS
3        4       DESCRIPTION: INCOME
4        4       DESCRIPTION: TAX        INDEX LABEL: TAX.IX

CONFIRM (YES OR NO): Y
```

Once you verify your database setup, the database structuring proceeds automatically. **DBGEN** enters the initialization phase. This phase takes several minutes to complete.

DBGEN *Continued*

Screens six and seven are displayed during these sequences, but you are not prompted for any information.

Screen six displays information on the database initialization.

```
DBINIT REV. X.XX


ADDING       NAMES.IX
INDEX BLOCK SIZE IS 512
ADDING       TAX.IX
INDEX BLOCK SIZE IS 512
ADDING       TAXES
RECORD SIZE IN BYTES:    35
```

Screen seven appears when the initialization period ends. The **DBFM.T6** utility automatically creates a table file when the initialization phase ends.

```
END OF INIT PHASE

INDEX INITIALIZED        NAMES.IX
INDEX INITIALIZED        TAX.IX

DATA FILE      TAXES

INCLUDES       TAXES.TB
```

DBGEN *Continued*

When the table file creation phase is completed, **DBGEN** clears the screen and displays the following prompt:

```
OUTPUT TO LINE PRINTER:
```

Enter **Y** to get a printed copy of the table file showing the design specifications of the database structure. If you enter N, **DBGEN** displays the information on your screen. This example produces the following table file display:

```
FILENAME:    TAXES     LINKED

TABLE FILE: TAXES.TB
RECORD LENGTH: 35       MAXIMUM NUMBER OF RECORDS: 1000
DBGEN FILE MAINTENANCE
DBGEN TAX RECORD (TYPE 1)
REC NAME     SEQ   DESCRIPTION     SIZE  TYPE  POS.  FORMAT   EDIT
INDEX FILE: NAMES.IX  KEY LENGTH:  10
16 NAME   1 NAME                    10    S     2     A10      U

INDEX FILE: TAX.IX    KEY LENGTH:  4
 19 TAX      1 TAX                  4     D     31    F11      N
 16 NAME     1 NAME                 10    S     2     A10      U
 17 ADDRESS  2 ADDRESS              15    S     12    A15      U
 18 INCOME   3 INCOME               4     D     27    F11      N
 19 TAX      4 TAX                  4     D     31    F11      N
```

The display specifies:

- A record length of 35. That is the total of the field lengths specified in screen 2 plus two bytes for status information.
- A maximum of 1,000 data records. This was specified in screen 4 of the **DBGEN** dialog.
- The labels for the index files and the record fields and also the attributes for the record fields.

DBMOVE

BASIC CLI Command

Moves logical file structures from one directory to another.

DG/RDOS

Format

!DBMOVE *directory* [*database* ...] [*filename*/**L**]

Arguments

directory	The directory you want to move one or more databases to.
database	The databases you want to move. This is an optional argument; if it is not used, all nonpermanent databases are moved.
filename/**L**	The output file containing the names of the databases and files moved. If this argument is used, it must have the **/L** switch appended to it. (**Note:** you cannot use this argument and the global **/L** switch in the same command line.)

Global Switches

/A	Move permanent and nonpermanent databases.
/L	List moved database names to the default output queue. (**/L** overrides **/V**.)
/O	Delete *directory*:*database-name* and *directory*:*volume-label-name*, if they exist, before moving the database. Existing links are unlinked, and new links to the volume-label file are created.
/P	Move databases in order of the date last opened.
/R	Move most recent copy of database. If an entry exists in the destination directory with a more recent creation date, the database is not moved.
/S	Move databases in alphabetical order.
/V	Verify moved databases with list of names at your terminal.
/W	Move databases in order of the time of the last write.
/X	Move databases in ascending order of size.

Local Switches

mm-dd-yy/**A**	Move only databases created on or after the date *mm-dd-yy*.
mm-dd-yy/**B**	Move only databases created before the date *mm-dd-yy*.
/N	Move only databases that do not match this template.

DBMOVE *Continued*

What It Does

DBMOVE works with files in the logical file database structure. **DBMOVE** moves entries with .DB and .VL extensions for a database or databases in the current

directory to a given directory. All logical file links are recreated in the destination directory and linked to the .VL file created in that directory. Error checking is performed before the move to avoid any conflict with logical filenames in the destination directory.

How To Use It

To use **DBMOVE**, go to the source directory. Then execute **DBMOVE** from the Business BASIC CLI. You must specify the name of the destination directory. If you don't specify the databases to be moved, all nonpermanent databases are moved. You can use templates with the database names.

Examples

1. Move a copy of the database **PHONES** (**PHONES.DB** and **PHONES.VL**) to the directory **EMPLOYEES** and link all logical files to the volume-label file.

   !DBMOVE/A EMPLOYEES PHONES

2. Move the most recent copies of the databases in the current directory to the directory **EMPLOYEES** and link all logical files to the appropriate volume-label file.

   !DBMOVE/R EMPLOYEES
   or
   !DBMOVE/R EMPLOYEES -.DB

DELETE

BASIC CLI Command

Deletes a file, directory, or partition.

AOS/VS	DG/RDOS

Format

!DELETE *itemname1* [*itemname2* ...]

Argument

itemname	The name of a file, a directory, or a partition. You can use filename templates.

Global Switches

/C	Confirm each deletion. The system displays each *itemname*, then waits for you to confirm the deletion by pressing CR or New Line. To prevent a deletion, press any key except CR or New Line.
/L	List deleted files to the default output queue. (**/L** overrides **/V**.)
/P	Delete files in order of the date last opened.
/S	Delete files in alphabetical order.
/V	Display a list of the deleted files at your terminal.
/W	Delete files based on the time of the last write to the file.
/X	Delete files in ascending order of size.

Local Switches

mm-dd-yy/**A**	Delete only files created on or after this date.
mm-dd-yy/**B**	Delete only files created before this date.
/N	Do not delete files matching this *itemname*. When used, this switch must always be appended to *itemname*.

What It Does

DELETE eliminates the specified files, directories, or partitions.

On DG/RDOS systems, if *itemname* is a link entry, **DELETE** eliminates the resolution file (when its link attributes allow it), not the link entry. Use **UNLINK** to delete a link entry.

On AOS/VS systems, **DELETE** eliminates a file that has been opened by you or another user. You can delete an open file; the system does not warn you that the file is open. At the time of the **DELETE** call, the operating system marks the file for deletion. The file is not actually deleted until the final close occurs.

DELETE

Continued

How To Use It

Execute this command by entering **DELETE** from the Business BASIC CLI. An *itemname* must follow the command. You can use a single command to delete several

files. To use the optional switches, append global switches to either the keyword **DELETE** or another global switch and enter local switches, except for **/N**, as separate arguments. When used, the **/N** is appended to *itemname*. You can use the filename templates allowed by your operating system.

On DG/RDOS, to delete a directory or partition, first release it. Then enter the complete directory name, including the .DR extension, with the **DELETE** command.

Example

This example removes from your directory all files beginning with **TEST** that have a two-character extension and the file **MIKE**. The names of the deleted files are displayed on the terminal.

```
!DELETE/V TEST-.** MIKE
DELETED TEST.SR
DELETED TEST1.SR
DELETED TEST101.SV
DELETED MIKE
```

DELREC.SL *Subroutine*

Deletes a record in a linked-available-record file (PARAM file structure) and places it on the deleted record chain.

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 8600

Input Variables

F%	The number in the file characteristics array (C1) of the subfile with linked available records.
C1	The file characteristics array.
R1	The number of the record to be deleted.

Scratch Variables

X0 %	The record status of record 0.
Y0 %	The record status of first record on deleted record chain.
X0	The next new available record.
Y0	The next available record on a deleted record chain.
Z0	The last record used from record 0.

Line Numbers

8600	Entry point to **DELREC.SL**.
8600-8695	**DELREC.SL** occupies these line numbers.
9610-9645	**POSFL.SL** occupies these line numbers.

What It Does

DELREC.SL deletes record R1 from the linked-available-record file F% in the PARAM file database structure. **DELREC.SL** changes the record status (the first two bytes in the record) to zero and includes in the next four bytes a pointer to the deleted record chain. Then, **DELREC.SL** updates record 0 in the linked-available-record file to contain the number of the last deleted record.

DELREC.SL uses **POSFL.SL** to position to record R1.

If you use the logical database file structure, you can use the **DELREC** statement instead of **DELREC.SL**. The advantages of the **DELREC** statement are that it performs automatic locking, is faster than **DELREC.SL**, and frees the code space normally occupied by the subroutine. See *Commands, Statements, and Functions in Business BASIC*.

How To Use It

To use **DELREC.SL**, perform the following steps:

1. Enter "**DELREC.SL** and enter "**POSFL.SL** into your program.

DELREC.SL

Continued

2. Build the C1 array and assign the file number (the row in the C1 array) for the file to F%.
3. Assign the number of the record to be deleted to R1.
4. Lock record 0 of the file.
5. Enter the program statement **GOSUB 8600**.
6. Unlock the file.

Example

DELREC.SL and **POSFL.SL** are added to the program **TEST**, which is in working storage. The program is listed and the segment shown illustrates setting up the C1 array and establishing the record to be deleted. The record is deleted from the data file and its key is removed from the index file.

*** LIST**

```
00010 DIM C1[1,3],B$[544],D$[18],KEY$[4],REC$[52]
00020 LET B$="EMPIX,5,EMP,5",FILL$(0) :Open file routine.
00030 BLOCK WRITE B$
00040 SWAP "OPEN
00050 BLOCK READ B$
00052 UNPACK "JJ",B$,ERRIN,ERRNO
00054 IF ERRIN<>-1 THEN GOTO 00060    :Check for errors from OPEN
00056 PRINT "ERROR # ";ERRNO;" - ";B$[5,512]
00058 END
00060 LET K=1
00070 FOR I=0 TO 1
00080   FOR J=0 TO 3
00090     LET C1[I,J]=ASC(B$[K,K+3])
00100     LET K=K+4
00110   NEXT J
00120 NEXT I
00130 LET D$=CHR$(C1[0,0],2),CHR$(C1[0,1],4),CHR$(0,2),"EMPIX",
        FILL$(0)
00140 LET F%=1                        :Subfile 1(row 1 of C1) is EMP.
00150 LET R1=0
00160 INPUT "Enter the number of the employee to be deleted: ",NUM
00170 LET KEY$=CHR$(NUM,4)
00180 KFIND D$,B$,KEY$,R1
00190 IF R1<=0 THEN PRINT "INVALID EMPLOYEE NUMBER"
00200 IF R1<=0 THEN GOTO 00160
.                                     :Before deleting the record, it
.                                     :could be displayed so the
.                                     :operator could verify.
00320 LET T=30
00330 LOCK 1,"EMP",R1*C1[F%,3],C1[F%,3],T  :Lock record 0 of EMP.
00340 IF T=57 THEN GOTO 00320         :If lock times out, try again.
```

DELREC.SL

Continued

```
00350 GOSUB 08600 : \ DELREC.SL       :Delete the record.
00380 KDEL D$,B$,KEY$,R1     :Delete the key in the index file.
00390 IF R1<=0 THEN GOTO 02000        :Check for error in index.
00400 UNLOCK                          :Unlock record 0 of EMP.
00410 GOTO 00160
.
.
.
* ENTER "DELREC.SL
* ENTER "POSFL.SL
* SAVE "TEST
```

DIR

BASIC CLI Command

Changes the current directory to the directory specified.

AOS/VS	DG/RDOS	UNIX

Format

!DIR *pathname*

Arguments

pathname The full pathname of the initialized directory where you want to go. On DG/RDOS systems, the pathname is required. It can include the name of a secondary partition where the directory resides as well as the directory name. On AOS/VS and UNIX systems, the pathname is optional. On AOS/VS, it cannot exceed 255 characters, including colons. If it is too long, AOS/VS recognizes only the first 31 characters. On UNIX, the pathname cannot exceed 255 characters.

What It Does

This command moves you to the directory you specify. That directory then becomes the current directory. In DG/RDOS, **DIR** does not change the current system directory.

DIR initializes the directory (by executing **INIT**). When you change to a directory, then that directory and everything in its pathname are initialized. The directory remains initialized until you release it.

If you enter **!DIR** without an argument, **DIR** displays the name of your current directory.

How To Use It

There are certain restrictions on using the **DIR** command.

Under DG/RDOS, only AA accounts (system managers) can use **DIR** to change to all directories. These are also the only accounts with access to **$SYS** (**$SY3** in triple precision) and **$LIB** (**$LIB3** in triple precision). Other accounts can move everywhere else if they meet the following requirements:

1. The **DIR** flag is set in your **ACCOUNT** file. It allows you to use the Business BASIC CLI command **DIR**. Without this flag, then you cannot move anywhere, and the system does not perform any more checks.
2. If you are issuing a **DIR** command to a directory with a **U$ERS** file, your account must be listed in the **U$ERS** file. You are automatically allowed access to directories without **U$ERS** files.

On AOS/VS and UNIX systems, you must have write access and execute access in the directory you are changing to.

DIR

Continued

Execute **DIR** from the Business BASIC CLI. To change directories, **DIR** must be followed by an argument specifying the directory you want to move to.

On DG/RDOS systems, you don't need to include the **.DR** extension on the directory name.

Examples

1. This changes the current directory to **USER.DR** in DG/RDOS and to **USER** in AOS/VS.

   !DIR USER

2. In DG/RDOS, this changes your current directory to the **USER.DR** directory that is a directory in the partition **PART01.DR**.

   !DIR PART01:USER

3. In AOS/VS, this example uses a pathname to change your current directory to **USER**, which is within the directory **CARL**, which is within the directory **UDD**.

   !DIR :UDD:CARL:USER

DISK

BASIC CLI Command

Displays the amount of disk space used and remaining.

AOS/VS	DG/RDOS	UNIX

Format

!DISK

What It Does

DISK returns a display showing the amount of disk space used and remaining for the current partition (DG/RDOS) or directory (AOS/VS and UNIX). If you are in a subdirectory, **DISK** returns the information for the parent partition.

The **DISK** display shows the number of 512-byte blocks actually allocated.

Under AOS/VS, **DISK** gives no indication of the available contiguous blocks. If you are not in a Control Point Directory (CPD), Business BASIC supplies information about the CPD above you.

How To Use It

Execute **DISK** from the Business BASIC CLI.

Examples

1. On a DG/RDOS system, **DISK** shows the number of blocks (sectors) used and remaining as well as the number of contiguous blocks. In this example, 113,661 blocks of the 187,408 blocks in your directory are still available for use. Of these, the largest group of contiguous blocks is 89,086.

```
!DISK
                    USED          LEFT      CONTIGUOUS
SECTORS           73,747       113,661          89,086
BYTES         37,758,464    58,194,432      45,612,032
```

2. On an AOS/VS system, **DISK** shows the maximum number of blocks (sectors) in your directory, the current number that are being used, and the number of blocks that are available for use. Here, of the 1,000,000 total blocks, 9,093 are being used and 990,907 are available for use.

```
!DISK
                  MAXIMUM       CURRENT       REMAINING
SECTORS:          1000000          9093          990907
BYTES:        512,000,000   4,655,616     507,344,384
```

DOC

Utility

Produces printable document files.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "DOC

What It Does

DOC uses automatic text justification, line centering, paging, and page headings to produce printable documents. The documents have chapters and sections. DOC sends each document to either the line printer, a terminal, or a file you specify.

DOC accepts input from the keyboard or text files.

If you are using text files, they must have been set up using the EDIT utility, and they must contain control commands, which affect the way the output is printed but do not appear on printouts. (See "Control Commands.") The names of the text files must take the form *chapter_number.section_number* (e.g., 1.12 = chapter 1, section 12). In addition, you need a section called *chapter_number*.0, which has the chapter heading beginning in column 16. This is necessary for creating the table of contents with DOCTOC.

Using ascending order, DOC searches for input files to correspond to sections in a chapter. The utility begins with the first chapter and section specified and continues through the last chapter and section specified. An out-of-sequence filename indicates the end of one chapter and the beginning of the next. DOC then resets the section count to 0 and increments the chapter count by 1.

How To Use It

Execute DOC by entering RUN, CHAIN, or SWAP "DOC. That starts the DOC dialog.

DOC first asks:

```
FIRST CHAPTER:      LAST CHAPTER:
```

Enter the chapter numbers. If your first chapter and last chapter are different, DOC starts at the first chapter, section 0 (or the first section in the chapter), and continues through the last chapter.

If the first and last chapters are the same, DOC asks for the first and last sections numbers. This lets you print portions of a chapter.

DOC then asks for the name of the output file. If you press CR, DOC sends the output to your terminal. You can specify any output file.

After getting information on an output file, DOC reads the default header information from the file DOC.HD (you can add information to DOC.HD using EDIT or another text editor). Then DOC asks:

```
LINE:
```

DOC

Continued

This prompt lets you enter text or control commands that are output before **DOC** displays the text file sections. **DOC** repeats the `LINE:` prompt until you press CR or New Line without entering any text. **DOC** then outputs the text files.

Control Commands

Control commands enable you to change the default margins, page length, number of printed lines per page, and headings. They also allow you to start a new page, center lines, specify conditional paging, and turn justification on and off.

All control commands begin with a backslash (\). **DOC** assumes that any line in the text file (or in your keyboard input to the prompt `LINE:`) that does not start with a backslash is text and prints it. The control commands are summarized in Table 1-6.

Table 1-6. Control Command Summary

Command	Function
\CPAGE [*n*]	Page-up to next page if *n* lines do not fit on this page. If *n* is not specified, read the input file until an **\ECPAGE** is found, set *n* to the number of noncontrol lines read between **\CPAGE** and **\ECPAGE**.
\ECPAGE	Used with **\CPAGE** to indicate the number of lines that must appear on the same page (see **\CPAGE**).
\EQC	Turn off centering (see **\QC**).
\ETX	Turn off justification (see **\STX**).
\HEAD *n*, *text*	Print a heading containing *text* and the section/page number at the top of a page. If *n* is 0, suppress the section/page heading. If *n* is positive, print a section heading in the form `SECTION` *X.X*. If *n* is negative, print a page heading in the form `PAGE` *X-X*.
\MAR *l,r*	Set the left margin to *l*, the right margin to *r*.
\NLF	Print the next line of text without a line feed.
\PAGE [*p,l*]	If you include no optional arguments, page-up to the next page. If you specify *p* and *l*, set the number of printed lines per page to *p* and the number of lines per page to *l*.
\SPAGE *n*	Set the current page number to *n*.
\STX	Turn on justification, continue to justify until encountering an **\ETX** command.
\QC [*n*]	If an argument n is included, center the next n lines of text. If you do not specify *n*, center until encountering an **\EQC**.

Justifying Text

Use the **\STX** command to turn justification on and the **\ETX** command to turn justification off. All text between the **\STX** and **\ETX** commands is justified to right

DOC

Continued

and left margins. These are either default margins or margins set by the **\MAR** command.

During justification, a single blank line indicates the end of a paragraph and is not printed. To have a blank line between output paragraphs, place two blank lines in the text file.

Centering Lines

The **\QC** command centers the lines between the two default or set margins. Centering continues for the number of lines you specify in the **\QC** command or until **DOC** encounters an **\EQC** command. The **\EQC** command turns off centering.

The format for the **\QC** command is:

\QC [*number*]

For example, **\QC 1** centers the line of text following the command.

Setting Margins

The **\MAR** command sets the left and right margins for the page. The format is:

\MAR *left,right*

For example, **\MAR 10,50** sets the left margin to column position 10 and the right margin to column 50. The default left and right margins are at columns 1 and 70. You must set margins outside an **\STX...\ETX** sequence. The margins are effective only within such a sequence.

Unconditional Paging

Without arguments, the **\PAGE** command starts a new output page (page-up). With arguments, the **\PAGE** command sets the page length, according to this format:

\PAGE [*printed_lines_per_page, lines_per_page*]

For example, **\PAGE 50,60** specifies that the output page is 60 lines long and that only 50 lines will be printed on a page. The default values are 66 lines per page with 55 printed lines per page.

Conditional Paging

The **\CPAGE** command allows you to specify the number of lines that must appear on the same page. If there is room for the lines at the bottom of the current page, **\CPAGE** pages after printing those lines. Otherwise, it pages where it is entered. This is useful when you are printing a table or chart, and you want the entire chart either at the bottom of the current page or at the top of the next page, but you do not want to split the chart.

If you do not specify a number of lines, **DOC** looks for an **\ECPAGE** command. If lines between **\CPAGE** and **\ECPAGE** fit at the bottom of the page (control command lines not counted), **DOC** does not page-up; if the number of lines do not fit, **DOC** pages-up at the **\CPAGE** command line.

DOC

Continued

The format for the \CPAGE command is:

\CPAGE [*number_of_lines*]

For example, \CPAGE 10 indicates to DOC that if ten lines fit from here to the bottom of the page, do not page-up; if ten lines do not fit, page-up now.

Overprinting

The \NLF command combines two lines to make one output line without a line feed. The line of text following the \NLF control command is retained and the next line of text is merged with it. To be effective, a \NLF command must be followed by a \QC command. For example:

```
\MAR 1,20
\NLF
\QC 1
COLUMN A
\MAR 21,4O
\QC 1
COLUMN B
```

becomes:

```
          COLUMN A            COLUMN B
```

Headings

DOC lets you print up to two lines of headings containing section/page numbers on each page except the first page. DOC omits headings on the first page because its default page number is zero. You can use the \HEAD command or the default heading in the DOC.HD file. The heading can include text and a section number heading or a page heading. Also, if you include the string !D! in the heading text, DOC prints the current date in the form *mm/dd/yy*. The \HEAD command must be the first command in the file. The command uses the format:

\HEAD *column-number,heading-text* [!D!]

column-number indicates the column where printing of the section/page number begins. With a positive column number, DOC prints the text and a section heading in the form SECTION *section-number*. The section number is a string of numbers separated by periods indicating the current chapter, section, and subsection. It is set when a string of numbers (e.g., 12.22.1) is encountered in line 1 of the text and the first two numbers equal the current chapter and section. For example:

```
\HEAD 55,THIS IS A MAJOR HEADING
\HEAD 55,THIS IS THE SECOND HEADING
```

appears as:

```
THIS IS A MAJOR HEADING                SECTION 1.1
THIS IS THE SECOND HEADING
```

DOC

Continued

If *column-number* is negative, a page number heading is printed instead of the section number heading. The page number heading takes the form text followed by **PAGE** *chapter-page number*. If your current chapter is 0, the command displays **PAGE** *page-number*.

```
\HEAD -55,THIS IS A MAJOR HEADING !D!
\HEAD -55,THIS IS THE SECOND HEADING
```

appears as:

```
THIS IS A MAJOR HEADING  4/1/91        PAGE 1-1
THIS IS THE SECOND HEADING
```

When *column-number* is 0, **DOC** suppresses the section/page heading.

```
\HEAD 0,THIS IS A MAJOR HEADING
\HEAD 0,THIS IS THE SECOND HEADING
```

appears as:

```
THIS IS A MAJOR HEADING
THIS IS THE SECOND HEADING
```

The file **DOC.HD** contains two **\HEAD** commands with the default header information. Use a text editor to change the information in **DOC.HD**. When the **\HEAD** command is used in a **DOC** text file, it overrides the **DOC.HD** headings.

You can replace both heading lines. If you enter only one **\HEAD** command in your section, its text appears as the second heading printed. The first heading is replaced by the old second heading.

Setting Page Numbers

The **\SPAGE** command sets the current page number. The format is:

\SPAGE *number*

where *number* is the current page's number. Zero is the default beginning page number.

Example

The following series of examples illustrate how to set up a **DOC** file. Below is the section 1.0 text file created using the **EDIT** utility. The heading, "Writing and Running Programs," begins in column 16.

```
1.              Writing and Running Programs
```

DOC

Continued

Next, you need the section 1.1 text file created using the **EDIT** utility, and then you can start entering your text and the **DOC** control commands.

```
1.1             Preparing a program
```

\STX

Business BASIC programs are made up of statements. Each statement is preceded by a line number. The number you give the statement determines the order in which it is executed and listed.

Each statement is on a separate line. The programmer terminates each line at the terminal with a carriage return. If you enter an error, it can be corrected using special control keys.

\ETX
\MAR 10,60
\NLF
    1.
\STX

Pressing Ctrl-H erases the last character entered (RUBOUT) on some terminals. The cursor is backed up on a CRT.

\ETX
\NLF
    2.
\STX

Pressing Ctrl-X deletes the entire line. A backslash is printed, representing line deletion.

\ETX

Using the preceding text file, you can generate the documentation file. You do this by responding to the **DOC** prompts. This is a short text file where the first and last chapters are the same, so you enter 1 in response to both prompts. Enter 0 for the first section number so that **DOC** picks up the heading. When you press CR at the OUTPUT FILE: prompt, **DOC** displays the justified text file at your terminal first.

```
* RUN "DOC
 FIRST CHAPTER: 1   LAST CHAPTER: 1
 FIRST SECTION: 0   LAST SECTION: 1
 OUTPUT FILE: <CR>
 1-1
 LINE: <CR>


1.      Writing and Running Programs
1.1     Preparing a program

Business BASIC programs are made up of statements. Each statement
is preceded by a line number. The number you give the statement
determines the order in which it is executed and listed.
```

DOC

Continued

Each statement is on a separate line. The programmer terminates each line at the terminal with a carriage return. If you enter an error, it can be corrected using special control keys.

1. Pressing Ctrl-H erases the last character entered (RUBOUT) on some terminals. The cursor is backed up on a CRT.
2. Pressing Ctrl-X deletes the entire line. A backslash is printed, representing line deletion.

DOCTOC *Utility*

Prepares a table of contents for a document file set up using DOC.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "DOCTOC

What It Does

DOCTOC sets up a table of contents for a file formatted using **DOC** that can be printed or displayed at your terminal. **DOCTOC** searches the current directory for filenames with the form:

chapter_number.section_number

DOCTOC then searches each of these files for a line of text with the form:

chapter.section text

where *text* starts at column 16 (see **DOC** for an example of this text format). This line is then printed. In the table of contents, section 0 is not indented; all other sections are indented by two spaces. A blank line separates each chapter. When a section is not found, the section count is reset to 0 and the chapter count is incremented.

How To Use It

Execute the utility by entering **RUN**, **CHAIN**, or **SWAP "DOCTOC**.

The utility then asks for the first and last chapters. Enter the chapter numbers. Next, **DOCTOC** asks for the output file, which may or may not exist. If you press CR without naming a file, the output appears at your terminal.

Example

The following example refers to the example explained in **DOC**. Since CR is entered at the prompt `OUTPUT FILE:`, the output is displayed at the terminal.

```
* RUN "DOCTOC
 FIRST CHAPTER:  1   LAST CHAPTER:  1
 OUTPUT FILE: <CR>
 1.         Writing and Running Programs
   1.1        Preparing a program
```

DUMP

BASIC CLI Command

Copies one or more disk files in DUMP format to an output file.

AOS/VS	DG/RDOS	UNIX

Format

!DUMP *outputfile* [*filename*[/N] ...]

Arguments

outputfile — The name of the file or device to receive the dump.

filename[/N] — A file you want to dump. If you specify *filename*, then only files matching filename are dumped. You can use the filename templates allowed by AOS/VS or DG/RDOS (UNIX systems use AOS/VS templates). If you include the /N switch, only files that do not match the filename template are dumped.

Global Switches

/A — Dump permanent and nonpermanent files. Only nonpermanent files are dumped without this switch (DG/RDOS only).

/K — Do not dump links.

/L — List dumped filenames to the default output queue. (/L overrides /V).

/N — Dump only links.

/P — Dump in sorted order by the date of last access.

/R — Dump noncontinguous files as if they were random files rather than sequential files (AOS/VS only).

/S — Dump in alphabetical order.

/V — Verify dump by listing the filenames on the terminal.

/W — Dump in sorted order by the time of the last write.

/X — Dump in ascending order by file size.

Local Switches

mm-dd-yy/A — Dump only files created on or after the date *mm-dd-yy*.

mm-dd-yy/B — Dump only files created before the date *mm-dd-yy*.

What It Does

DUMP copies files from the current directory to a given file or device. The argument *outputfile* can be a device such as a disk file, a magnetic tape, or a cassette tape file. If you specify *filename*, only files with a matching *filename* are dumped. If you omit *filename*, all nonpermanent files in the current directory are dumped.

Under DG/RDOS, the format of a Business BASIC **DUMP** is compatible with a DG/RDOS **DUMP**. The **DUMP** command does not allow the dumping of partitions

DUMP *Continued*

containing embedded directories or of directories containing files. You can only use **DUMP** to dump files.

On AOS/VS and UNIX systems, if you dump to a tape from AOS/VS or UNIX Business BASIC, then you must load that tape from AOS/VS or UNIX Business BASIC, not from your AOS/VS or UNIX operating system. **DUMP** does not dump links to files in a different directory. Therefore, links should not be transported between AOS/VS and UNIX systems.

How To Use It

Execute **DUMP** from the Business BASIC CLI. The argument *outputfile* must always be entered with **DUMP**, while *filename* is optional.

Examples

1. On a DG/RDOS system, dump all permanent and nonpermanent files in the current directory to magnetic tape **MT0:0**.

   !DUMP/A MT0:0

2. Dump all files in the current directory (except links) that have the .SR extension and were created on or after Feb. 2, 1991, to the output file **SOURCE.DP**.

   !DUMP/K SOURCE.DP -.SR 2-2-91/A

EDIT *Utility*

Creates and/or edits a text file.

AOS/VS	DG/RDOS

Format

```
{ RUN  }
{ SWAP } "DOC
{ CHAIN}
```

What It Does

EDIT lets you create a text file or modify an existing text file. The utility has three modes of operation (Command, Insert, and Comment) and two modes of screen display (Brief and Verify). Use Command Mode to open files for editing, to move or copy lines in the edit buffer, to establish file formats, to search for a given character sequence, to change characters, and to send lines to and from the edit buffer. Use Insert Mode to add text anywhere in your edit buffer and Comment Mode to add quickly comments to program listings. Verify Mode, which is the default for **EDIT**, displays at your terminal lines you have modified, while Brief Mode suppresses this display.

Table 1-7. EDIT Modes

Mode	Function
Brief	Do not automatically display modified text lines at the terminal. Enter **BR** to execute this mode.
Command	Perform file input and output and text editing. Press the interrupt key to leave this mode or to return to this mode from Insert or Comment mode.
Comment	Add comments to listing files. Enter **COM** to execute this mode and press the interrupt key to return to Command Mode.
Insert	Insert text to and from the edit buffer. Press the interrupt key to execute this mode; press the interrupt key again to return to Command Mode.
Verify	Display modified lines at the terminal. This mode is always on unless suppressed by Brief Mode. Enter **V** to return to Verify Mode from Brief Mode.

EDIT

Continued

How To Use It

Execute **EDIT** by entering **RUN, CHAIN,** or **SWAP "EDIT**. This puts you in Command Mode. When the 000: prompt appears, use the **F** command to enter an input filename (either an existing file or the name of one you want to create). As you enter text, the 000: prompt changes to show the line you are working on.

To switch to Insert Mode, press the interrupt key (usually the Esc key). For Comment Mode, enter **COM**. (Command, Insert, and Comment modes are explained later in this section.)

In addition to the operation modes, **EDIT** uses several special commands. These are summarized in the **EDIT** Command Summary table and are explained in more detail later in this section. To leave EDIT, enter **END, STOP,** or **OMIT** from Command Mode. You can return to Command Mode from Insert Mode or Comment Mode by pressing the interrupt key.

END writes the buffer and the remaining input file to the output file. Since the output file now has an edited version of the input file, **END** deletes the input file. **END** then renames the output file, giving it the name of the primary input file.

STOP executes an **END** and then exits from the **EDIT** utility.

OMIT stops the **EDIT** utility without changing the input or output file. Use **OMIT** if you made an error in your buffer that you cannot fix, if you forgot to output a buffer, or if you made some other error you cannot recover from. **OMIT** keeps both the input and the output file open. It does not give the output file the input file's name. The temporary output filenames are:

EDIT*processor#job#***.TM**
EDIT$$$*processor#job#*

where *processor#* comes from **SYS(25)** and *job#* comes from **SYS(9)**. **EDIT** does not delete these files, so you can examine them after an **OMIT**. **EDIT***processor#job#***.TM** is a temporary file and **EDIT$$$***processor#job#* is the file that you were editing. It includes your changes to the file.

Table 1-8. EDIT Command Summary

Command	Function
+,-	Go forward (+) or backward (-) the number of lines specified.
A	Bring lines of text into the buffer.
ACC	Allow control characters.
ALC	Allow lowercase characters.
B	Move to the last line (bottom) of the edit buffer and start Insert Mode.

EDIT

Continued

Table 1-8. EDIT Command Summary (*concluded*)

Command	Function
BR	Enter Brief Mode.
C	Change the contents of one string to another string.
COM	Enter Comment Mode.
COPY	Write lines from the input file directly to the output file without changing the buffer.
D	Delete either the current line or the specified line(s).
DCC	Prohibit control characters (turns off ACC).
DITTO	Move lines in the edit buffer without deleting the original lines.
DLC	Prohibit lowercase characters (turns off ALC).
END	Write the buffer and the rest of the input file to the output file; then delete the input file and rename the output file to the name of the deleted input file.
F	Specify an input file.
FIND	Use templates to locate all occurrences of strings matching a search string.
I	Insert one line following the current line.
IB	Insert one line before the current line.
L, LC	Locate strings in the edit buffer.
LINES	Set the number of lines to be brought into the edit buffer. The default is 60.
MAX	Set the maximum line length. The default is 132.
MOVE	Take specified lines in the edit buffer and place them after the current line. Then delete the original lines and renumber lines.
N	Move to the next line after the current line.
O, OW	Send information from the buffer to an output file.
OMIT	Leave the **EDIT** utility without changing the input file and without closing the input file or the temporary files.
P, P!	Print lines from the edit buffer.
PAGE	Send the buffer contents to the output file; then place one page from the input file into the buffer.
R	Replace the current line of text with a new line.
SET	Set tab stops for use with the Tab key.
SKIP	Discard lines from the input file.
STOP	Terminate the **EDIT** utility by executing an **END**.
T	Move to the top of the edit buffer (the first line).
V	Enter Verify Mode; this is the default mode. Use the **BR** command (for Brief Mode) to suppress it.

EDIT

Continued

Command Mode

To conserve memory in the system, **EDIT** uses a disk resident buffer to hold the lines being modified. This buffer has a maximum size of 132 lines with 132 characters per line. The number of characters in a line consists of everything that appears at the terminal, including **EDIT** prompts. Use the commands **F, A, O, OW, PAGE, END,** and **STOP** to move data between the edit buffer and the file.

To use more than one input file, append a colon (:) to commands that refer to files (see the **F** command, **A** command, and **PAGE**). However, you can have only one temporary output file. The temporary output file is renamed to the primary input file when you enter **END** or **STOP**.

Setting File Formats

This section contains **EDIT** commands you can use to set formats for editing a file. These commands are **ACC, ALC, DCC, DLC, LINES, MAX,** and **SET**. The first four affect the type of characters you can write to the buffer while **LINES** and **MAX** set the buffer page size and line length, and **SET** provides tabs.

ACL, ACC Commands

These commands allow you to place lowercase (**ACL**) and/or control characters (**ACC**) in your edit buffer. Without these two commands, you cannot type lowercase and control characters in the buffer even if they exist in the input file. Use **DCL** and **DCC** commands to turn off **ACL** and **ACC** commands.

DCL, DCC Commands

After you use **ACL** and **ACC**, use the **DLC** and **DCC** commands to restore the default condition in the edit buffer. The **DLC** command prohibits lowercase characters in the edit buffer while the **DCC** command prohibits control characters.

LINES Command

Use the **LINES** command to set the number of lines that you want brought into the edit buffer with an **A** command or a **PAGE** command. The format is:

LINES *n*

where *n* is the number of lines you want brought in from the edit buffer each time it brings in a page of input.

MAX Command

Use the **MAX** command to set the maximum number of characters in a line. The format is:

MAX *n*

EDIT

Continued

where *n* is the number of characters you want in a line. The default is 132 characters in a line.

SET Command

Use the **SET** command to establish up to eight tab stops. You can use the Tab key to reach the stops when you are in Insert or Comment Mode. For example:

SET 5,10,20,35,42,50

sets tab stops at locations 5, 10, 20, 35, 42, and 50.

Input/Output Commands

This section contains **EDIT** commands dealing with file input and output. These commands are the **F**, **A**, **O**, **OW**, **P**, **PAGE**, **SKIP** and **COPY**. Use the **F** command to establish your input file. The **A**, **O**, **OW**, **P**, and **PAGE** commands deal with transferring text between the input file and the buffer or the buffer and the output file. The **SKIP** and **COPY** commands act on the input and output files, but they do not change what is in the buffer.

F Command

Use the **F** command to specify your input file. When the 000: prompt appears, enter **F** *filename*, where *filename* is the name of your input file (if it exists) or the name of the file you are creating. For instance, with

000: **F 1.2**

the input filename is **1.2**. The **F** command opens the filename as the primary input file and creates a temporary output file, called **EDIT$$$***process#job#* where *process#* is from **SYS(25)** and *job#* from **SYS(9)**.

To use an alternate input file, append a colon to **F**. Thus, **F: TEMP.SR** opens **TEMP.SR** as an alternate input file.

A Command

Use the **A** command with an optional line number argument to bring lines of text into your edit buffer. The form is:

A [*n*]

where *n* indicates the number of lines you want to bring into the buffer. If you do not specify *n*, **A** brings in one page (60 lines or the number you set by **LINES**). You can have up to 132 lines in your buffer.

After working with the lines in your buffer, you must place the buffer in the output file with either the **O**, **OW**, or **PAGE** command.

EDIT

Continued

O, OW Commands

The **O** and **OW** commands allow you to send either the entire buffer or a portion of it to the output file. The commands are identical except that **OW** writes a page mark (consisting of the line -- :\P) as the last line that it outputs. Both the **O** and the **OW** command delete the lines in the buffer as they move the lines to the output file. This changes the position of the lines in the buffer.

The **O** command takes the form:

O [*n*[,*m*]],

where *n* and *m* are optional. When *n* alone is used, it represents the number of lines from the current line to be output. When *n* and *m* are used, they represent the exact lines to be output. If no arguments are specified, **O** sends the entire buffer to the output file. This empties the buffer, allowing you to bring in more lines.

P Command

Use the **P** command to print lines in the edit buffer. The following command prints all the lines in the edit buffer:

P!

You can also specify the number of lines to print or a range of line numbers to print:

P [*n*] [*line1*, *line2*]

n is the number of lines, while *line1* and *line2* specify a range of lines.

PAGE Command

The **PAGE** command brings in a new page from the primary input file as it sends the entire buffer to the output file, thus combining the action of the **A** and **O** commands. An input page is 60 lines or the number you set by the **LINES** command. If you use the **PAGE** command with a colon (**PAGE:**), it brings in a page from the alternate input file.

SKIP Command

Without changing the buffer, **SKIP** reads the number of lines you specify in the input file and discards them. If no line number is specified, **SKIP** reads and discards a page (60 lines or the number in **LINES**). **SKIP** refers to the primary input file, and **SKIP:** refers to the alternate input file. The format is:

SKIP [*n*]

where *n* is an optional argument for specifying line numbers.

EDIT

Continued

SKIP also acts on lines until a string is found. If you specify:

SKIP *string*

SKIP discards lines until it finds *string*; then it backs up the file pointer to save the line with *string*, allowing you to bring the line and the lines following it into the edit buffer. *string* does not have to be within quotation marks, and you can use the dash (-), asterisk (*), and plus (+) templates to match a string. The dash matches any number of characters occurring at the location of the dash except a period, the asterisk matches any single character occurring at its location, and the plus matches all characters.

COPY Command

COPY writes lines from the input file directly to the output file. Since **COPY** bypasses the buffer, no lines are changed there. If you output the buffer after copying lines to the output file, the buffer lines appear in the output file after the lines that were copied. The format is:

COPY [*n*]

where *n* is an optional number of lines. If no line number is specified, **COPY** reads and copies a page (60 lines or the number in **LINES**) to the output file. If you specify **COPY** *string*, **COPY** copies lines until the string is found.

Line Commands

This section contains **EDIT** commands dealing with moving from one line to another line in the edit buffer. These commands are +/- *linenumber*, **B**, **N**, and **T**. The +/- *linenumber* and the **N** commands work from the current line while **B** and **T** take you to the bottom or top of the edit buffer. (For information on deleting, inserting, moving, and replacing lines, see "String and Text Editing Commands.")

+/- *linenumber* command

The line indicated by the prompt is the current line in the edit buffer. For instance, the 003: prompt means the current line is line 3. You can change the current line by entering the desired line as a command to the prompt. If you are at line 32 and you want to go to line 21, you enter:

```
032: 21
021: THIS IS LINE 21.
021:
```

EDIT *Continued*

Unless you are in Brief Mode (described later), **EDIT** automatically displays the new current line. If the line you want is higher than the last line in the buffer, **EDIT** takes you to the last line of the buffer and gives you an error message. You can then enter the line number you want, or you can enter a plus (+) or a minus (–) sign prefixed to a number, where number is used as an offset. For example:

```
032: +5
037: THIS IS LINE 37.
037: -10
027: THIS IS LINE 27.
027:
```

The +5 moves you from line 32 to line 37 while the –10 moves you from line 37 to line 27.

B Command

The **B** command moves you to the last line of the edit buffer and automatically puts you in Insert Mode, allowing you to add lines to the bottom of the buffer. For example:

```
001: B

I061: I AM INSERTING LINE 61.
I062:
```

Whenever you use the **B** command or respond by pressing CR or New Line to the last buffer line prompt, you automatically enter Insert Mode.

N Command

The **N** command moves you to the next line after the current line. For example:

```
000: N
001: THIS IS THE FIRST LINE.
001:
```

T Command

The **T** command moves you to the top of the buffer. For instance:

```
032: T
000:
```

Line 0 of the buffer never contains a text line; text starts at line 1. You go to line 0 to insert lines before the current line 1 or to insert lines into an empty buffer.

String and Text Editing Commands

This section contains **EDIT** commands dealing with strings and text editing: **C**, **D**, **I**, **IB**, **R**, **FIND**, **L**, **LC**, **MOVE**, and **DITTO**. The **C**, **D**, **I**, **IB**, and **R** commands

EDIT

Continued

change strings, delete lines of text, insert lines, and replace lines of text. Use the **FIND**, **L**, and **LC** commands to locate strings. The **MOVE** and **DITTO** commands allow you to move a block of lines within the edit buffer.

C Command

Use the **C** command to change the contents of the current line. It takes the form:

C */oldstring/newstring/*

where the slash (/) is a string delimiter. You can use any printable character (except a space) for a delimiter if the character is not used in *oldstring* or *newstring*.

The next example uses an asterisk (*) as the delimiter, while other examples use the slash (/).

```
005:  01000 LET  ABC=0 \ ABCD=1
005:  C *ABC*XYZ*
005:  01000 LET XYZ=0 \ XYZD=1
```

The current line is line 5. This command changes the first occurrence of ABC in the current line to XYZ. (**EDIT** displays the changed line when you are in Verify mode.) To change all occurrences of string in the current line, append **/G** to the command like this:

C */oldstring/newstring/***G**

The example now looks like:

```
005:  C /ABC/XYZ/G
```

and it changes all occurrences of ABC in the current line to XYZ.

To change all occurrences of *oldstring* in the edit buffer to *newstring*, use **/A** instead of **/G**:

C */oldstring/newstring/***A**

Now the example:

```
005:  C /ABC/XYZ/A
```

changes all occurrences of ABC in the edit buffer to XYZ, no matter what the current line is. You can specify a range of lines or a line increment from the current line.

To change the first occurrence of the string for a number of lines above or below the current line, use a line increment argument with the form:

C */oldstring/newstring/***I** [+/−] *n*

EDIT

Continued

The new example looks like:

005: **C /ABC/XYZ/I 10**

and it changes the first occurrence in each line of ABC to XYZ for the 10 lines beginning with and following the current line, (line 5 through line 14).

To change all occurrences of *oldstring* to *newstring* for each line from *n* to *n* or for a number of lines from the current line, use **/G**:

C */oldstring/newstring/***G** *n n*
C */oldstring/newstring/***G** [+/–] *n*

Now the example:

005: **C /ABC/XYZ/G -4**

changes all occurrences of ABC to XYZ from line 2 through line 5. Use the + and – signs to indicate a line range either beginning at the current line and extending *+n* lines or beginning at *–n* lines from the current line and extending to the current line.

005: **C /ABC/XYZ/G 25 35**

This example changes all occurrences of ABC to XYZ from line 25 through line 35, regardless of the current line.

D Command

The **D** command deletes the current line. Thus:

035: **D**

deletes line 35 from the edit buffer. The **D** command followed by one number uses the number as an increment from the current line. It deletes all lines beginning with the current line until it reaches the maximum set by the increment. For example:

035: **D 10**

deletes lines 35 through 44 (+10 lines from 35). In another example:

035: **D -5**

deletes lines 31 through 35 (–5 lines from 35). The **D** command followed by two numbers uses those numbers as a range of lines within the edit buffer to be deleted. Thus:

035: **D 10,20**

deletes line 10 through line 20 regardless of the current line.

EDIT *Continued*

I, IB Commands

Use the **I** command to insert one line following the current line. The **IB** command inserts one line before the current line. This is not the same as using Insert Mode.

For example:

035: **I INSERTS THIS LINE AFTER LINE 35 (LINE 36).**

Here, line 35 is used to set up line 36. The **IB** command inserts the line before the current line.

035: **IB INSERTS THIS LINE BEFORE LINE 35 MAKING THIS LINE 35.**

This inserts a new line before the current line, so the new line becomes line 35 and the current line (35) becomes line 36.

To put a line in your edit buffer, specify the line number as a command with a plus sign (+) if you want to insert the line after the specified line or a minus sign (–) if you want to insert the line before the specified line. For example:

035: **+27:THIS IS NEW LINE 28.**

inserts THIS IS NEW LINE 28. after line 27 regardless of the current line, and

035: **-26:THIS IS NEW LINE 26.**

inserts THIS IS NEW LINE 26. before line 26, making it the new line 26 (and renumbering old line 26 to line 27, etc.). To insert more than one line, use Insert Mode.

R Command

Use the **R** command to replace the current line of text with a new line. For example:

035: **R THIS IS NEW LINE 35.**

replaces the text in line 35 (the current line) with THIS IS NEW LINE 35.

To change any line of text in the edit buffer, use the line number:

001: **35:THIS IS NEW LINE 35.**

This changes the text of line 35 (regardless of the current line) to THIS IS NEW LINE 35.

FIND Command

The **FIND** command locates all occurrences of strings matching a search string using dash (-), asterisk (*), and plus (+) templates. The asterisk matches any single

EDIT *Continued*

character occurring in the position of the asterisk, the dash matches any number of characters except a period occurring in the position of the dash, and the plus matches all characters. For instance:

FIND -ABC-

locates all occurrences of any line that has the characters ABC together in a sequence (**LET X$="RSJKL MN OPABCDR XY"** is such a line).

FIND *ABC*

finds all occurrences of any line that has a five-character string with ABC as the second, third, and fourth characters (**LET X$="RSJKL RABCK"** is such a line).

L, LC Commands

Use the **L** and **LC** commands to locate strings in the edit buffer. The **LC** command locates all occurrences of a string, while the **L** command finds only the first occurrence of a string. If you are in Brief Mode, the **LC** command displays all occurrences of the specified string at your terminal. The search starts with the first nonspace character in the string and continues to a carriage return delimiter. For example:

LC GOSUB 09610

locates all occurrences of the string: GOSUB 09610.

MOVE Command

You can move lines from one location in the edit buffer to another. The **MOVE** command moves a block of type starting with the first line given through the last line given and inserts these lines immediately after the current line. The original lines are deleted, and the space they occupied is closed up. All the lines are then renumbered accordingly. For example:

010: **MOVE 4,7**

moves lines 4 through 7 to lines 11 through 14. It then deletes the original lines 4 through 7 and renumbers all the lines to use up the deleted space. The original line 10 becomes line 6, line 11 becomes line 7, line 12 becomes line 8, etc.

DITTO Command

The **DITTO** command moves lines in the edit buffer but does not delete the original lines. This allows you to duplicate blocks of text. The **DITTO** command looks like the **MOVE** command, but it places a copy of lines 4 through 7 at lines 11 through 14:

010: **DITTO 4,7**

EDIT

Continued

Insert Mode

Use Insert Mode to enter new lines into the **EDIT** buffer. To enter Insert Mode, use your interrupt key (usually the Esc key). The prompt changes so that an I precedes the current line number (such as I003: for Insert Mode, line 3) to indicate that you are in Insert Mode. Insert Mode always inserts lines after the current line. Do not insert more lines than the **EDIT** buffer's 132-line maximum.

To leave Insert Mode, press the interrupt key again.

Comment Mode

Comment Mode allows you only to append text to existing lines, such as a source or listing program. The **COM** command puts you in Comment Mode. Use the interrupt key to leave comment mode.

When you enter Comment Mode, **EDIT** displays the current line with a C prefix and places the cursor at the end of the line's text so you can append new text.

Business BASIC stores a program's colon comments in a character version of your program, not in the program file. That is why normally a .SR or .LS extension is used to denote listing (character-format) files.

When you add comments to programs, remember to enter a colon (:) first to mark the comment. Otherwise, if you type **ENTER "PROGRAM.LS** in Business BASIC to check the listing program, the system tries to interpret your comments as Business BASIC statements and gives you error messages.

The comments stay in the .LS file and do not take up working storage space. They are not copied from the .LS file into the program file.

Brief and Verify Modes

Normally, **EDIT** operates in Verify Mode. This mode displays the new line of text automatically after you enter a command that changes the current line number or modifies a line.

You can suppress Verify Mode by setting Brief Mode with the **BR** command. Restore Verify Mode with the **V** command.

EDIT

Continued

Examples

1. This example begins in Command Mode and uses the interrupt or Esc key to call Insert Mode. The new lines are added after the line 35, which was the current line when the switch was made. When the example returns to Command Mode, it is at line 38.

```
035: <interrupt key>
I036: THESE ARE NEW LINES BEING TYPED AFTER LINE 35.
I037: THE LAST ONE WAS LINE 36, AND THIS IS LINE 37.
I038: THIS IS LINE 38. USE ESC KEY TO GET OUT OF INSERT MODE.
1039: <interrupt key>
038:
```

2. This example uses **PROGRAM.LS** as an input file. It brings 20 lines into the buffer, allows the use of lowercase letters, sets four tabs, returns to the first line in the buffer, and then calls Comment Mode with the command **COM**. It uses the tab key to insert the comments until the end of the program is reached. After the Esc key is pressed, Command Mode returns with a current line number of 6.

```
000:F PROGRAM.LS
000:A 20
020:ALC
020:SET 30,40,50,60
020:1
001:COM
C001: 00010 DIM C1(2,3) (press tab key)        :Dimension C1 to
                                               :be a 3*4 array.
C002: 00020 DIM X$(10)       :Dimension X$ to max file
                             :name size.
C003: 00030 LET X$="SUB1"    :SUB1 is a subfile in MASTER.
C004: 00040 LET F %=0        :and is logical file 0.
C005: 00050 GOSUB 07800      :Go to FINDFILE.SL subroutine.
C006: 00060 GOSUB 00200 (Press the interrupt or Esc key)
006:
```

EQUIV

BASIC CLI Command

Renames a device.

DG/RDOS

Format

!EQUIV *newname oldname*

Arguments

newname	The name you want to give the device temporarily.
oldname	The current name of a device or a new name assigned in a previous **EQUIV**.

What It Does

EQUIV assigns a temporary name to an uninitialized device. You cannot use **EQUIV** to rename a master disk device. *newname* remains in effect until you release the device by its *newname* or until you rename the device again. **RELEASE** restores the original device name (*oldname*). If you use **EQUIV** to give a device a new name, it remains in effect throughout the system.

EQUIV gives programs device independence. You can write a generic device specifier in your programs and then use **EQUIV** to give the device to that generic name at runtime.

Under AOS/VS, use **LINK** to give your programs device independence.

How To Use It

Execute **EQUIV** from the Business BASIC CLI. **EQUIV** must be followed by two arguments—the new name and the current device name.

Examples

1. This gives MT0 the new name of MYTAPE. Once **EQUIV** is executed, the system no longer recognizes the name MT0.

```
!EQUIV MYTAPE MT0
!INIT MT0
FILE DOES NOT EXIST:MT0
!INIT MYTAPE
```

2. This example follows the first one. MYTAPE is released so the system no longer recognizes the name. You must use **EQUIV** again to get systemwide acceptance of MYTAPE.

```
!RELEASE MYTAPE
!INIT MYTAPE
FILE DOES NOT EXIST:MYTAPE
!EQUIV MYTAPE  MT0
!INIT MYTAPE
```

FDUMP

BASIC CLI Command

Fast dumps one or more files to magnetic tape.

DG/RDOS

Format

!FDUMP *magtapefile* [*filename*[/N] ...]

Arguments

magtapefile	A tape file that has a name in the form **MT***n*:*file*, where *n* is the drive number and *file* is the tape file number.
filename[/N]	The name of any file in the current directory. *filename* is used to match the files in the current directory. You can use the filename templates allowed by your operating system with this argument. With the /N switch, only files that do not match *filename* are dumped.

Global Switches

/A	Dump permanent and nonpermanent files.
/B	Dump small tape blocks for compatibility with unmapped systems so that you can use **FLOAD** to load them onto an unmapped system.
/D	Delete files that have been dumped.
/E	Append dump to end-of-tape file.
/K	Do not dump links.
/L	List dumped filenames to the default output queue (**/L** overrides **/V**).
/M	Prompt the user to input label; then add label to tape.
/N	Dump only links.
/V	Verify dump by listing filenames on the terminal.

Local Switches

mm-dd-yy/**A**	Dump only files created on or after the date *mm-dd-yy*.
mm-dd-yy/**B**	Dump only files created before the date *mm-dd-yy*.

What It Does

FDUMP writes blocks of up to 8,192 bytes to a magnetic tape, making it much faster than **DUMP**. If you use **FDUMP**, you must use **FLOAD** to load the tape. **LOAD** is not compatible with **FDUMP**.

When the **/B** switch is used, **FDUMP** writes blocks of up to 6,144 bytes. These smaller blocks are compatible with unmapped DG/RDOS Business BASIC systems.

FDUMP

Continued

FDUMP dumps all files in the current directory or the files specified by *filename* and/or the switches to a given magnetic tape file. **FDUMP** performs an implicit **INIT** and **RELEASE** for the tape drive. If data exceeds the tape capacity, **FDUMP** asks that additional tape volumes be mounted.

When the **/M** switch is used, **FDUMP** requests an input label for the tape. Business BASIC displays the label you give the tape when it is reloaded.

The **FDUMP** and **FLOAD** commands used under the operating system CLI are incompatible with those used under the Business BASIC CLI . Tapes backed up with the Business BASIC CLI command **FDUMP** must be loaded with the BASIC CLI command **FLOAD**.

FDUMP does not allow the dumping of partitions with embedded directories.

How To Use It

Enter **FDUMP** from the Business BASIC CLI. The argument *magtapefile* must follow **FDUMP**, but the argument *filename* is optional. If used, global switches are appended either to the keyword **FDUMP** or to another global switch, while the local switches appear as separate arguments.

Examples

1. Dump all files in the current directory to magnetic tape unit 0:file 0.

   !FDUMP/A MT0:0

2. Dump all nonpermanent files with the .SR extension created on or after Feb. 2, 1991, to the file **MT1:4**, excluding links.

   !FDUMP/K MT1:4 -.SR 2-2-91/A

FILCOM

BASIC CLI Command

Compares two files and displays dissimilar word pairs.

AOS/VS	DG/RDOS	UNIX

Format

!FILCOM *filename1 filename2* [*outputfile*/**L**]

Arguments

filename	The files you want to compare.
outputfile/**L**	Optional file to receive **FILCOM**'s output. This filename must be followed by the **/L** switch. This argument overrides the global **/L** switch.

Global Switches

/B	Print output in byte format.
/D	Print all numbers in decimal.
/H	Print all numbers in hexadecimal.
/O	Print output in octal (this is the default value).
/L	Print output to the default output queue.

What It Does

FILCOM compares two files word by word and displays dissimilar word pairs. The command prints the displacement in octal and then the dissimilar word pair. The word pair appears in decimal, hexadecimal, or octal, depending on which global switch you use. When no switch is used, the word pair appears in octal. The output goes to the terminal unless you use *outputfile*/**L** or the global **/L** switch.

How To Use It

Execute this command by entering **FILCOM** from the Business BASIC CLI. The names of two files must follow the command. If used, global switches are appended either to the keyword **FILCOM** or to another global switch. You can also use an optional argument with a **/L** switch to specify an output filename.

Example

Compare **PROG1.LS** to **PROG2.LS**, writing the dissimilar word pairs in decimal to the terminal.

PROG1.LS contains:

```
* LIST
00010 DIM A$[132],REPLY$[1]
```

FILCOM

Continued

```
00030 INPUT "ERROR # OR <ESC> TO END ",E
00040 LET A$=AERM$(E)
00050 PRINT "ERROR ";E;" (AERM$) - ";A$
00100 GOTO 00030

*
```

PROG2.LS contains:

```
* LIST
00010 DIM A$[132],REPLY$[1]
00030 INPUT "ERROR # OR <ESC> TO END ",E
00040 LET A$=ERM$(E)
00050 PRINT "ERROR ";E;" (ERM$) - ";A$
00100 GOTO 00030

* !FILCOM/D PROG1.LS PROG2.LS
    47     15681     15685    =A     =E
    50     17746     21069    ER     RM
    51     19748      9256    M$     $(
    52     10309     17705    (E     E)
    53     10506      2608    ).     .0
    54     12336     12341    00     05
    55     13616     12320    50     0
    56      8272     20562    P      PR
    57     21065     18766    RI     IN
    60     20052     21536    NT     T
    61      8226      8773    "      "E
    62     17746     21074    ER     RR
    63     21071     20306    RO     OR
    64     21024      8226    R      "
    65      8763     15173    ";     ;E
    66     17723     15138    E;     ;"
    67      8736      8232    "      (
    70     10305     17746    (A     ER
    71     17746     19748    ER     M$
    72     19748     10528    M$     )
    73     10528     11552    )      -
    74     11552      8763    -      ";
    75      8763     16676    ";     A$
    76     16676      2608    A$     .0
    77      2608     12592    .0     10
   100     12592     12320    10     0
   101     12320     18255    0      GO
   102     18255     21583    GO     TO
   103     21583      8240    TO     0
   104      8240     12339    0      03
   105     12339     12298    03     0.
   106     12298      ----    0.     ..
```

FILES *Utility*

Displays names of files in current directory.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "FILES

What It Does

FILES displays your current directory and its contents.

On DG/RDOS systems, **FILES** displays an L beside a link file and an S next to a SAVE file.

On AOS/VS systems, **FILES** displays an L beside a link file.

How To Use It

Execute this command by entering **RUN**, **CHAIN** or **SWAP "FILES**.

Examples

1. On AOS/VS, **FILES** displays at your terminal your directory pathname, the time, the date, and the files in your directory. An L next to the filename indicates a link file.

```
* RUN "FILES
:UDD:SETH:BASIC1                                    7/04/91  14:35:29

CH6F.MS             CH16T.MS     CH19X.MS     BB_MEMO_103
L BIZ_SPECS         REPORTS      MISC_NOV
LISTED SPACE = 1688 SECTORS,  864256 BYTES
 TOTAL SPACE = 1688 SECTORS,  864256 BYTES
```

2. On DG/RDOS, **FILES** displays the date, the time, and the files in your directory. An L next to a filename indicates a link file, while an S next to a filename indicates a SAVE file.

```
* !FILES
TEST                                                5/09/91  14:07:03

TEST3.LS      TEST.LS       COM.CM        GLSUMMARY.LS
XFER1.LS      S PRT80.SV    S TEST.SV     PRTFL1.LS
S GLSUMMARY   L INDX.       PRDX1.        S CALC.
L INDX2.      ARRAY.LS      CALC.LS       L LPA.
  PROD.       LREAD.LS      U$ERS.        COLOR.LS
LISTED SPACE = 226 SECTORS,   115712 BYTES
 TOTAL SPACE = 285 SECTORS,   145920 BYTES
```

FILESORT *Utility*

Sorts a data file.

AOS/VS	DG/RDOS	UNIX

Format

SWAP "FILESORT

What It Does

FILESORT sorts a data file in ascending or descending order based on key fields. You can have multiple key fields, enabling you to have both ascending and descending sorts within a file. The utility also tests the status of deleted records and places them at the end of the file.

Unlike **QFILESORT**, **FILESORT** allows both signed integer fields and unsigned binary (alphanumeric) fields as key fields, and the **FILESORT** key fields can be any size.

How To Use It

FILESORT is always used within a program statement. Follow these steps to execute **FILESORT**:

1. Open your data file.
2. Set up an argument string containing the key fields and the channel number of your file.
3. Use a **BLOCK WRITE** statement to pass the argument string through the common area.
4. Enter the program statement **SWAP "FILESORT**.
5. Use a **BLOCK READ** statement to check the returned argument string for an error.

   Any error that occurred during **FILESORT** is returned in the first two bytes (location 1,2) of the argument string. A zero indicates a successful **FILESORT**, a negative value represents an operating system or I/O error, and a positive value is a Business BASIC error (see Appendix B). The rest of the argument string is unused.

FILESORT *Continued*

FILESORT does not work with variable length records.

All arguments are considered binary, and bit flags are numbered 0-7. Table 1-9 describes the argument locations and the arguments in the string.

Table 1-9. FILESORT Argument String

Location	Size	Contents
1,1	1	Channel number of file you want to sort.
2,5	4	Number of records you want to sort. The records must exist in the file unless you set the value of substring location 12,12 to have a value of 1 to check for deleted records.
6,7	2	Record size in bytes.
8,11	4	Byte offset to first record to be sorted.
12,12	1	Flag to check for deleted records (linked-available-record file): 1 Check for deleted records. 0 Don't check for deleted records.
13,13	1	Number of key fields in record.
*14,15	2	First byte of key (1 is first byte of record).
*16,17	2	Last byte of key.
*18,18	1	Sort flag: **Bits** **Meaning** 0 = 1 Sort field in descending order, else ascending. 1 = 1 Signed binary field, else unsigned.
19,*	5	Next key field's descriptors. Repeat the sequence of entering the first byte of the key, the last byte of the key, and the key type. Since you can have multiple keys, you can enter this information several times.

FILESORT

Continued

Example

This program segment fills the string to be sent to **FILESORT**, which returns arguments to the common area. The program then prints the first two bytes of the returned string to see if an error occurred.

```
00010 DIM X$[512]                    :X$ to get argument string for common.
.                                    :Put code to open the file.
00100 LET X$=CHR$(0),CHR$(100,4) :Channel number; number of records.
00110 LET X$[0]=CHR$(32,2),CHR$(32,4) :Bytes per record;
                                       :offset-skip record 0.
00120 LET X$[0]=CHR$(1),CHR$(1)      :Check deleted records;
                                       :# of key fields.
00130 LET X$[0]=CHR$(3,2),CHR$(6,2)  :Key field bytes 3 to 6.
00140 LET X$[0]=CHR$(1,1)            :Sort in descending order.
00150 LET X$[0]=FILL$(0)             :No more fields.
00155 BLOCK WRITE X$                 :Send into common area.
00160 SWAP "FILESORT"                :Execute FILESORT and return.
00170 BLOCK READ X$                            :Read common area.
00180 PRINT ASC(X$[1,2])             :Check for errors.
```

FINDFILE.SL *Subroutine*

Finds a file and builds a C1 array (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 7800

Input Variables

X$	The name of the subfile. Dimension X$ to 10.
F%	The file number in the C1 array.

Output Variables

X$	The name of the physical file in which the subfile resides.
C1	The file characteristics array. Only columns 1, 2, and 3 are set, column 0 is not set.
C%	The next available channel number where you can open a file.

Scratch Variables

T9$	The subfile entry read from the PARAM file. T9$ must be dimensioned to at least 42 bytes.
Z	A pointer to the subfile entries in PARAM file.

Line Numbers

7800	Entry point to **FINDFILE.SL**.
7800-7831	**FINDFILE.SL** occupies these line numbers.

What It Does

FINDFILE.SL builds a C1 array without opening your file. It leaves column 0 (channel number) of the C1 array blank and returns the next available channel number (C%) where you can open your file. **FINDFILE.SL** also returns the name of the physical file where the subfile resides. The subroutine uses information from the PARAM file to build columns 1, 2, and 3 for row F% (the file number) of the C1 array.

If no PARAM file entry is found for the subfile, **FINDFILE.SL** treats the file as a physical file and asks you for the byte offset to the first record, the record size, and the number of records in the file. **FINDFILE.SL** does not create a PARAM entry for the file. (See **GETREC.SL** for more information on the C1 array and **OPEN** for more information on PARAM records.)

FINDFILE.SL

Continued

How To Use It

To use this subroutine, perform the following steps:

1. Enter "**FINDFILE.SL** into your program. Program line numbers 7800 to 7831 must be free for **FINDFILE**.
2. Dimension your C1 array, dimension T9$ to at least 42 bytes, dimension and assign a subfile name to X$, and assign a file number (row in C1 array) to F%.
3. Enter the program statement **GOSUB 7800**.

FINDFILE.SL returns the name of the physical file in the variable X$. You can then use your C1 array, and X$ and C% in an **OPEN FILE** statement to open your file. Be sure to set the row in the C1 array, C1(F%,0), to the available channel number, C%.

FINDFILE.SL does not swap to programs or open the files.

Example

The program **TEST** builds a C1 array with the subfiles **SUB1** and **SUB2** and the physical file **PHYS**.

```
* LIST
00010 DIM C1(2,3)          :Dimension C1 to be a 3*4 array.
00020 DIM X$(10)           :Dimension X$ to maximum filename size.
00030 LET X$="SUB1"        :SUB1 is a subfile in MASTER,
00040 LET F%=0             :and is file number 0 in C1 array.
00045 DIM T9$(42)          :Dimension T9$ to 42 bytes.
00050 GOSUB 07800          :Go to FINDFILE.SL subroutine.
00060 GOSUB 00200          :Go to verification routine.
00070 LET X$="SUB2"        :SUB2 is in MASTER,
00080 LET F%=1             :and is file number 1 in C1 array.
00090 GOSUB 07800          :Go to FINDFILE.SL subroutine.
00100 GOSUB 00200          :Go to verification routine.
00110 LET X$="PHYS"        :PHYS is a physical file,
00120 LET F%=2             :and is  file number 2 in C1 array.
00130 GOSUB 07800          :Go to FINDFILE.SL subroutine.
00140 GOSUB 00200          :Go to verification routine.
00150 STOP
00200 REM -- VERIFICATION ROUTINE
00210 PRINT X$             :Prints name of physical file.
00220 LET C1(F%,0)=C%      :Assign channel number to 0 column in
                           :C1 array.
00230 PRINT C1(F%,0),C1(F%,1),C1(F%,2),C1(F%,3)
00240 RETURN

*
```

FIXFILE *Utility*

Adjusts AOS/VS file types.

AOS/VS

What It Does

FIXFILE adjusts the AOS/VS file type of *filename* to be the specified type. Information on this utility, like other program conversion tools, is documented in the on-line file **CONVERT.DOC**, located in the Business BASIC directory **DOC**.

FLOAD

BASIC CLI Command

Loads files from a magnetic tape that were stored using FDUMP.

DG/RDOS

Format

!FLOAD *magtapefile* [*filename*[/N] ...]

Arguments

magtapefile	A tape file that has a name in the form **MT***n*:*file*, where *n* is the drive number and *file* is the tape file number.
filename[/N]	The files you want to load. You can use the filename templates allowed by your operating system with *filename*. With the **/N** switch, only files that do not match the filename template are loaded.

Global Switches

/A	Load permanent and nonpermanent files.
/F	Begin loading after the first tape volume. Use this when you are loading selected files that are not on the first tape volume.
/K	Do not load links.
/L	List loaded filenames to the default output queue (overrides **/V** and **/P**).
/N	Load only links.
/O	Delete the current file if it exists and replace it with the file being loaded.
/P	Do not load files; list their filenames on the terminal.
/R	Load the most recent version of the file.
/V	Verify loaded files by listing their filenames on the terminal.

Local Switches

mm-dd-yy/**A**	Load only files created on or after the date *mm-dd-yy*.
mm-dd-yy/**B**	Load only files created before the date *mm-dd-yy*.

What It Does

FLOAD quickly loads from magnetic tape specified files that were stored using **FDUMP**. **FLOAD** is the only way to load **FDUMP** stored files (see **FDUMP**).

FLOAD loads either all nonpermanent files into the current directory or, if you use the **/A** switch, all files. You can load or list the files specified in *filename*.

FLOAD *Continued*

The **FDUMP** and **FLOAD** commands used under the operating system CLI are incompatible with those used under the Business BASIC CLI . Tapes backed up with the Business BASIC CLI command **FDUMP** must be loaded with the BASIC CLI command **FLOAD**.

FLOAD does not let you load partitions with embedded directories.

How To Use It

Enter **FLOAD** from the Business BASIC CLI. The argument *magtapefile* must follow **FLOAD**, but additional arguments and switches are optional. If used, global switches are appended either to the keyword **FLOAD** or to another global switch, while the local switches appear as separate arguments.

Example

Load from the tape file **MT0:0** the most recent copies of all files with the .SR extension, created on or after Feb. 2, 1991, that do not begin with TMP, and list their names on the terminal.

```
!FLOAD/V/R MT0:0 -.SR 2-2-91/A TMP-.-/N
```

FM

Utility

Provides file maintenance functions for data files and table files (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "FM

What It Does

The File Maintenance (**FM**) utility is an interactive Business BASIC utility that works with table files or data files with table files. **FM** uses Business BASIC utility programs to give you keyboard mode access to data and index files. With **FM** you can:

- Set up linked-available-record data files or direct-access data files.
- Access records by record number or key.
- Add, change, and delete records and keys.
- Define screen formats for data fields used on screen "pages" or on screens set up by the Screen Maintenance utility.
- Establish multiple screen formats or pages for up to nine data record types.
- Specify user passwords.
- Allow multiple users to access a single data file (**FM** handles file locking).

FM works with files in the PARAM file database structure, so all data and index files used with the utility must have PARAM entries with fixed-length records. **FM** uses dynamic space allocation.

Business BASIC provides five other utilities that work with files created by **FM**. They are **FMTABPRINT**, **FMPRINT**, **FMLOG**, **TABBUILD**, and **MOVETABREC**.

How To Use It

Execute **FM** by entering **RUN**, **SWAP** or **CHAIN "FM**. This starts the **FM** dialog.

There are two versions of the utility: **FM**, which uses function keys, and **FM.RM** (roll-mode), which uses commands.

Depending on your terminal type, when you execute **FM** on AOV/VS and DG/RDOS systems, if the utility does not recognize your terminal or if your terminal does not

FM

Continued

have function keys, **FM** automatically executes its roll-mode version. This is not true on UNIX systems. If you are on a UNIX system and your terminal does not support function keys, you must explicitly execute **FM.RM**; **FM** will not automatically execute it for you. Once executed, **FM.RM** displays the prompt `FUNCTION:` and waits for you to enter a command (see the Table 1–11). When **FM.RM** starts, enter the **AUTO** command to turn on the automatic display for the **FIND** command.

If your terminal has function keys, **FM** places the cursor in the command brackets at the top right of the screen. Enter commands by pressing a function key (see the "**FM** Function Keys" section). **FM** then displays the function in the command brackets.

To use **FM** with a data file, first set up a table file (with a .TB extension on its name) containing information on the data file and up to three index files. The table file holds heading information, record and file sizes, a list of valid users, record format descriptions, field descriptions, and key descriptions.

The table file uses six record types (formats) for entering this information. (These types are described in the section "Entering Table File Data.")

Table 1–10. Table File Records

Type	Record #	Description
1 (control record)	0	Names of data and index files, number of record types, number of fields, number of pages, and pointers to other descriptions.
2 (control record)	1	Heading and record and file size of data file, with optional log file and screen filenames.
3 (user record)	2	Eight valid user IDs and passwords.
4 (format descriptor record)	3-11	Pointer to the first field descriptor record (type 5 record) with the screen format for each record type and number of pages.
5 (field descriptor record)	16-400	Record for each field on each page for each type. You can have nine different types, 385 different fields, and up to seven pages per type.
6 (index descriptor record)	13-15	Record for each key to describe each key field.

FM

Continued

To set up a table file, perform the following steps:

1. Create the table file with a .TB extension to its name. You can use the **CRAND** command to do this. For example:

   *** !CRAND filename.TB**

2. Execute **FM**.

3. Enter data in the six table file record types.

Table record types 1, 2, and 3 must be in records 0, 1, and 2 in the table file. Record type 4 must start at record 3. **FM** is set up so that record type 5 starts at record 16 and record type 6 starts at record 13. With both record types 5 and 6, subsequent records of the same type must follow in sequential order. You can have up to 400 records for record type 5 and up to 15 records for record type 6.

After setting up a table file, use **INITFILE** to create and initialize the data file, the index file, and, if used, the log file. The first two bytes of each record in the data file must be reserved for the record status (a zero here indicates a deleted record) even if you don't use the linked-available-record format. You do not have to describe these two bytes in the table file.

FM Function Keys

Operate **FM** by using function keys. Once executed, **FM** positions the cursor to the top right of the screen and waits for you to press a function key. If your terminal does not support function keys, you should explicitly execute **FM.RM**.

ADD. Use the ADD key to add a record to a direct-access file. You must specify the record number. (Linked-available-record format files allocate records dynamically.) When you add data, **FM** checks all fields for range and data type errors. A duplicate key causes an error if the index file does not allow them.

CHANGE. Press the CHANGE key to enter new data over old data. You must enter the entire field. Press CR or New Line to step over fields without changing them. The record is not updated until you finish (indicated by pressing CR or New Line at the last field, or the Home key), so you can interrupt a change session without disturbing the record. To change one field in the record without going through the entire record, first make the change and press CR or New Line to enter it. Then press the Home key.

CHANGE FORMAT. Use the CHANGE FORMAT key to change the record type in a file with multiple record types. A data file can have up to nine record types while a table file can have up to six record types. Press CHANGE FORMAT and enter the record type (1–9). You can change the format if you're adding a new record not of the type currently displayed (or if you are changing record types in a file with no record status).

FM

Continued

Table 1-11. FM Function Keys

Key	Function	Roll-Mode Command
F1	Find a record	FIND
F2	Find next record	NEXT
F3	Change a record	CHANGE
F4	Add a new record	ADD
F5	Format change	FORMAT
F6	Delete a record	DELETE
F7	Page change	PAGE
F10	Stop FM	STOP
F11	Clear an error	--
--	Display record	DISPLAY
--	Automatic display	AUTO
CR/New Line	Next field	Tab or CR/New Line key
Ctrl-R	Previous field	--
Home	Ends a **CHANGE** session.	--

CHANGE PAGE. Use the CHANGE PAGE key to display another page of a current record (a page is the amount of information that appears on your screen). Press CHANGE PAGE and enter the page number instead of the record number. The record number and any key fields are retained between pages. If you add or change a page's field, **FM** changes the data in the record.

CLEAR ERROR. Press the CLEAR ERROR function key when you get an error message.

DELETE. Before using the DELETE key, you have to find the record first. When you delete a record, **FM** sets its status flag (the first two bytes of the record) to zero.

FIND, FIND NEXT. With the FIND key, **FM** prompts you for a record number. If you don't enter a record number, it prompts you for a key entry. It gives you up to three chances to enter a key. You can enter either the record number or a key; however, once you enter something, **FM** retrieves either that record number or a record with that key. Press CR or New Line to step through the **FM** prompts. After a FIND, you can use the FIND NEXT key to go sequentially to the next record based on the record number (or key) you entered with FIND. You can use FIND NEXT to advance through a file by record numbers; however, a deleted record stops this action. Enter AUTO before using FIND to turn on the automatic display.

STOP. The STOP key halts **FM** and returns you to your previous level.

Setting Up a Sample Database

The following examples show how to set up a database for a list of employees. The list includes each employee's last name, first name, telephone extension, and

FM

Continued

identification number. There are two key fields: last name and first name. The name of the table file is **EMP.TB**. It includes a data file called **EMP**. Each record in **EMP** has four fields: LAST, FIRST, EXT, and EMPID. The two key fields are LAST and FIRST.

Before you can build a table file, you must know the record size and number of records in the data file. In these examples, the records are 52 bytes long, and the file contains a maximum of 100 records. Also, you must use **INITFILE** to initialize your data and index files before using **FM**.

Entering Table File Data

Before entering data into a table file, create the table file and then execute **FM**.

```
* !CRAND EMP.TB
* RUN "FM
```

This starts the multi-screen **FM** display. **FM** prompts you for the filename and data file name. For the filename, enter **FM.TB**, which is part of your Business BASIC package. This is the master table file; it contains the table file parameters and record descriptions. For the data file, enter the name of the table file in the form *filename*.**TB**; in this case, **EMP.TB**.

```
DATA FILE MAINTENANCE  REV X.XX
FILENAME: FM.TB
DATA FILE NAME: EMP.TB
```

In the next screen, **FM** positions the cursor in the command brackets, indicating it is ready for a command. You must press a function key before you can enter data into the table file.

FM

Continued

```
FILE MAINTENANCE TABLE FILE                          [       ]
RECORD # _____
```

In general, you enter data into the table file by performing these steps:

1. Press the FORMAT CHANGE key.
2. Enter record type (1, 2, 3, 4, 5, or 6). A table file starts at record 0 with a type 1 record and progresses to a type 6 record.
3. Press the ADD key.
4. Enter a record number.
5. Enter data for a field. If a field does not require data entry, press New Line (AOS/VS) or CR (DG/RDOS). **FM** automatically enters a 0 in numeric fields and leaves alphanumeric fields blank.
6. Repeat steps 3-5 to add records that are the same type.
7. Repeat steps 1-5 to add records of different types.
8. Press the STOP key to end the **FM** session.

Perform these steps for each record type in the screen examples.

The next screen prompts you for records. Press the FORMAT CHANGE key and enter 1 to indicate a type 1 record.

```
FILE MAINTENANCE TABLE FILE                          [FORM]
RECORD # 1
```

FM

Continued

When the next screen appears, press the ADD key and begin entering data.

```
FILE MAINTENANCE TABLE FILE                                  [ADD]
#0 CONTROL RECORD # 0      FILENAME  EMP        TYPE (D,L) L
INDEX FILENAME  LAST       KEY DESCRIPTOR RECORD #  13
INDEX FILENAME  FIRST      KEY 14   INDEX FILENAME 0      KEY  0_
# RECORD TYPES  1      INDICATOR  31416  MAX FIELDS 4  DEFAULT FORMAT 1
DEFAULT PAGE  1
```

Type 1 Record

For a type 1 record, enter the following data in response to the **FM** prompts.

0 CONTROL RECORD #

Enter 0. The control record is always located a record 0 in a table file.

FILENAME

Enter the data subfile name.

TYPE (D,L)

Enter L for linked-available-record format, which uses deleted records, or D for direct format, which does not use deleted records.

INDEX FILENAME

Enter the index subfile name.

KEY DESCRIPTOR RECORD #

Enter 13, 14, or 15, whichever is the record number of the type 6 record that points to the index named in INDEX FILENAME. Type 6 records usually start at record number 13.

FM

Continued

INDEX FILENAME

Enter the index subfile name used for the second key.

KEY

This is an abbreviation for KEY DESCRIPTOR RECORD # (see that entry) since you can have more than one key.

RECORD TYPES

Enter the number of different record types you want for the data file. A data file can have up to nine record types while a table file can have up to six record types.

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

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

INDICATOR

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

MAX FIELDS

Enter the number of fields per record. The number of fields per record must be greater than or equal to the largest number of fields on any page; otherwise, **FM** generates an error.

DEFAULT FORMAT

Enter a number *N* where:

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

FM uses this information to find a pointer to your first type 4 record. Type 4 records start at record number 3, and you can have up to nine type 4 records for each of your nine record types. **FM** uses the default format first.

If you have more than one format (more than one type 4 record), **FM** uses the first format specified in this entry. It finds the other type 4 records by following the formula:

N + 2

where *N* is the record number that you give **FM** each time you press the FORMAT CHANGE key and answer the subsequent RECORD # query. Each record is numbered sequentially.

FM *Continued*

DEFAULT PAGE

Enter 1 to display the first page of the table file when you begin running **FM** on your data file.

Type 2 Record

To enter data for another record type, press the FORMAT CHANGE key; then enter the number of the record type you want (in this case, 2).

```
FILE MAINTENANCE TABLE FILE                                    [FORM]
#0 CONTROL RECORD #  2
```

Next, press the ADD key and begin entering data.

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]
#1 CONTROL RECORD (CON'T) RECORD # 1
MAINTENANCE DESCRIPTION  EMPLOYEE MAINTENANCE   BYTES/RECORDS 52
MAX RECORDS 100   SCREEN FILENAME EMP   LOG FILENAME EMP.LG
```

#1 CONTROL RECORD (CON'T) RECORD #

FM

Continued

Enter 1 for the record number of the type 2 record. Type 2 records are always record number 1 in table files.

MAINTENANCE DESCRIPTION

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

BYTES/RECORDS

Enter the number of bytes in a record.

MAX RECORDS

Enter the number of records in the data file.

SCREEN FILENAME

This entry is optional. If you leave this field blank, **FM** uses the default **FM** page format.

FM uses the EDIT CODES listed under type 5, field descriptor records, when displaying fields instead of the Field Definition Characters that are documented under the SM utility. For example, if you have defined your field as a right-justified numeric field in SM using >9999.99, you should use the NR code in the field descriptor record. FM ignores the SM codes.

To use a screen, enter the screen filename without an extension. Use the Screen Maintenance (**SM**) utility to create a screen file with the name *filename*.**Sn**. The *n* represents your terminal type. Any **SM** screen you use must exist and contain enough fields to hold all fields in a given format/page combination. Assign a special number to your **SM** screen by using this formula:

screen_num=(*page_num* - 1)*(*number of record types*) + (*format_num* - 1)

where:

screen_num	the number for your screen
page_num	the number of the page for this screen
format_num	the number you entered to answer the DEFAULT FORMAT query in your type 1 record

For this example, create an **SM** screen named **EMP.S6** and enter **EMP** at this field. Screen file 0 corresponds to page 1 (see Type 4 records).

LOG FILENAME

FM

Continued

This entry is optional. If you have a log file, enter a log filename. A log file shows all additions and changes made to the data file. Log filenames should carry the extension .LG for easy identification, though this convention is optional. Create log files with the utility **INITFILE**. The record size in bytes should be the size of the data record plus 18 bytes.

Type 3 Record

A type 3 record allows you to maintain a password system for your data file. To enter type 3 records, press the FORMAT CHANGE key and enter 3.

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

Next, press the ADD key and begin entering data.

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]
#2   VALID USERS RECORD # 2     USER #1 0     INITIALS
USER #2  0       INITIALS       USER #3 0     INITIALS
USER #4  0       INITIALS       USER #5 0     INITIALS
USER #6  0       INITIALS       USER #7 0     INITIALS
USER #8  0       INITIALS       USER #9 0     INITIALS
```

#2 VALID USERS RECORD #

Enter 2. Type 3 records always reside in record 2.

USER #1

This entry is optional. If you want the file to be public, enter 0 (or press New Line or CR).

FM

Continued

If you want a password, enter a nonzero number of up to 10 digits to match against the initials entered in INITIALS.

INITIALS

Enter the user's initials. The user must use the password number and the initials to gain access to this file.

AOS/VS systems treat characters 3 through 5 of USERNAME as the user initials. DG/RDOS systems treat characters 3 through 5 of ACCOUNT as the user initials. UNIX systems treat characters 3 through 5 of Logon as the user initials.

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

USER #2-USER #9

You can specify up to nine passwords.

Type 4 Record

A type 4 record describes formats used to display fields. Each format can have up to eight pages. To enter data for type 4 records, press the FORMAT CHANGE key.

```
FILE MAINTENANCE TABLE FILE                              [FORM ]
#2   VALID USERS RECORD #  4
```

Next, press the ADD key and begin entering data.

FM

Continued

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

#3-#12 FORMAT DESCRIPTOR RECORD #  3
TYPE DESCRIPTION  EMPLOYEE      FIELDS/RECORD 4
FIRST FIELD (REC #) 18   # PAGES 1   # FIELDS/PAGE 1 2   PAGE 2 0
PAGE 3 0   PAGE 4 0   PAGE 5 0    PAGE 6 0   PAGE 7 0   PAGE 8 0
```

#3-#12 FORMAT DESCRIPTOR RECORD #

Enter 3 for the first format descriptor record. They start at record 3 and proceed sequentially to 11. This allows up to nine formats.

TYPE DESCRIPTION

Enter an arbitrary name for the format you are using.

FIELDS/RECORD

Enter a number *N* for total fields displayed, where:

N = (all fields) - (key fields)

FM automatically displays key fields, so do not count them. For this example, enter 2 (4 data fields minus 2 key fields).

FIRST FIELD (REC #)

Enter the record number of the field descriptor record (type 5) that describes the first non-key field. Type 5 records usually start at record 16. **FM** expects subsequent field descriptor records to follow consecutively. In the example, enter 18 (records 16 and 17 describe key fields).

FM

Continued

PAGES

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

FIELDS/PAGE 1

Enter the number of total fields on page 1 (see FIELDS/RECORD). Do not count key fields.

PAGE 2-PAGE 8

Enter the number of total fields for each page of your format.

Type 5 Record

A type 5 record describes the individual fields used for each data value that Business BASIC sends to the screen. These field descriptor records must exist for key and data fields and for any fields that you display and maintain. To enter data for a type 5 record, press the FORMAT CHANGE key and enter 5.

```
FILE MAINTENANCE TABLE FILE                                    [FORM]
#3-#12 FORMAT DESCRIPTOR RECORD #  5
```

Press the ADD key and begin entering data.

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]
#16-#400 FIELD DESCRIPTOR RECORD #  16       EDIT CODE U
FIELD DESCRIPTION LAST NAME                  SIZE 10     TYPE S
POSITION 2       DISPLAY FORMAT A10      MINIMUM 0
MAXIMUM 0        FIELD ID LAST
```

FM

Continued

#16-#400 FIELD DESCRIPTOR RECORD #

Enter 16. Type 5 records start at record number 16. Enter consecutive record numbers, one field per record, in the order that you want the fields displayed. There must be a field descriptor record for every field in the data record including key fields.

EDIT CODE

Enter one of four edit codes:

U	Unedited alphanumeric, left-justified.
N	Edited numeric, left-justified.
UR	Unedited alphanumeric, right-justified.
NR	Edited numeric, right-justified.

NOTE: To right justify a field that is defined as right justified in a custom screen (not the default FM page format), you must also enter a UR or NR edit code here.

FIELD DESCRIPTION

Enter up to 32 characters for the field prompt.

SIZE

Enter the length of the field in bytes. The following applies for numeric fields:

Number of Bytes	Minimum	Maximum
1	-128	127
2	-32768	32767
3	-8388608	8388607
4	-2147483648	2147483647
5	-549755813888	549755813887
6	-14073748855328	14073748855327

TYPE

Enter a one (or two) character code indicating the format of the field. Some combinations of character types are possible, such as CS, BF, or BN. These codes are:

C	Crammed field (3 characters per 2 bytes)
S	String field
I	Integer field, single precision (2 bytes)
D	Double integer (4 bytes)
B	Binary integer (1 or 3 bytes, also 5 bytes in triple precision)

FM

Continued

N	Digit subfield (part of a larger number)
F	Bit flag field
T	Triple precision (6 bytes)

POSITION

Enter the starting byte of the field relative to byte 0. Do not enter a number less than 2. Business BASIC reserves the first two bytes in a data record for the status flag.

DISPLAY FORMAT

Enter a format code for how **FM** should display the field. (The format codes are similar to the formats used with the Business BASIC **PRINT USING** command.) Enter one of four formats: **F***w.d*, **A***w*, **B***w.d*, or **N***w.d*.

F*w.d* displays a right-justified binary or integer field, where *w* is the maximum number of digits, including the decimal, and *d* is the number of digits to the right of the decimal. For example, with **F***w.d*:

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

A*w* displays a string or crammed field, where *w* is the maximum number of characters. For example, with **A***w*:

Data	Field Length	Display Format	Output
SMITH	5	A5	SMITH
ARLINGTON	6 (crammed)	A9	ARLINGTON

B*w.d* displays a bit flag field, where *w* is the number of spaces setting the bit corresponding to 2 ^ *d*; *w* must always equal 1, and *d* is 0 to 31. In this case, with **B***w.d*:

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

N*w.d* displays a subfield, where *w* is the maximum number of digits, and *d* is the number of digits to the left of the rightmost digit. In this **N***w.d* example:

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

MINIMUM

If you specified an unedited (U or UR) field in EDIT CODE, skip this entry. If you specified a numeric (N or NR) field, enter a minimum numeric value for the field. Do

FM

Continued

not enter decimal points; you define them in DISPLAY FORMAT. The **FM** Character Types table shows the minimum values for certain fields.

MAXIMUM

If you specified an unedited (U or UR) field in EDIT CODE, skip this entry. If you specified a numeric (N or NR) field, enter a maximum numeric value for the field. Do not enter decimal points; you define them in DISPLAY FORMAT. The **FM** Character Types table shows maximum values for certain fields.

FIELD ID

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

In the next three screens, type 5 records are created for the remaining three fields. Press the ADD key to enter data.

```
FILE MAINTENANCE TABLE FILE                                    [ADD  ]

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

FM

Continued

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

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

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

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

FM

Continued

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

#13-#15 INDEX DESCRIPTOR RECORD # 13      KEY LENGTH 10
FORMAT 1-KEY FIELD 1 16     KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 2-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 3-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 4-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 5-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 6-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 7-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 8-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 9-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
```

#13-#15 INDEX DESCRIPTOR RECORD #

Enter the record number. Type 6 records are usually record numbers 13-15 (one for each index file allowed). If this is the first type 6 record, enter the same record number you entered in the first KEY DESCRIPTOR RECORD # field of the type 1

record. The key descriptor record number points to the type 6 record, so the two entries must be the same.

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

KEY LENGTH

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

FORMAT 1-KEY FIELD 1

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

FM

Continued

KEY FIELD 2

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

KEY FIELD 3

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

FORMAT 2 through FORMAT 9

Enter values as described above for each field in each key in each format.

To create type 6 records for the second key, press the ADD key, which allows you to enter data.

```
FILE MAINTENANCE TABLE FILE                                    [ADD ]

#13-#15 INDEX DESCRIPTOR RECORD # 14      KEY LENGTH 10
FORMAT 1-KEY FIELD 1 17     KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 2-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 3-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 4-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 5-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 6-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 7-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 8-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
FORMAT 9-KEY FIELD 1 0      KEY FIELD 2 0      KEY FIELD 3 0
```

FM

Continued

Using FM on a Data File

First, execute **INITFILE** to create and initialize the data and index files. **INITFILE** also adds entries for these files to the PARAM file. Then **RUN "FM**. When **FM** prompts you for a filename, enter **EMP**, which is the name of the data file.

```
* RUN "FM
FILENAME: EMP
```

FM displays the screen defined in the table file for **EMP** and positions the cursor in the command brackets. After you press the function key, you can enter data into the data and index files.

FMLOG *Utility*

Displays an FM log file.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "FMLOG

What It Does

FMLOG displays the log file that the File Maintenance (**FM**) utility maintained for your data file. This file contains all **FM** transactions for that data file. Not all data files have log files. You specify whether to have a log file when you set up the table file for the data file. Log file records are 18 bytes longer than the data file's records.

How To Use It

Execute **FMLOG** by entering **RUN**, **CHAIN** or **SWAP "FMLOG**.

FMLOG then asks you to choose an output format: spaced, packed, or full. You can also specify an output file. Enter the device name where you want the log file printed or the name of a file to receive the contents of log file. When no output file is specified, the log file is displayed on your terminal (the default). **FMLOG** also asks if you want to clear the log file.

Example

Once run, **FMLOG** prompts you for information. In this case, **FMLOG** places the information on the file **EMP** in the output file **EMP012891**. You can use **TYPE** to display **EMP012891** and check the information in it.

```
* RUN "FMLOG
DATA FILE MAINTENANCE  REV X.XX
FILENAME: EMP
EMPLOYEE MAINTENANCE
EMPLOYEE
OUTPUT FILE: EMP012891
FORM FEED BETWEEN RECORDS? NO
SPACING (SPACED, PACKED, FULL)? PACKED
CLEAR LOG FILE? YES
```


FMLOG *Continued*

```
* !TYPE EMP012885

RECORD ADDED  BY AAARP8 ON  1/28/91 AT 13:18:27
REC#: 1 LAST: JONES FIRST: TOM LAST: JONES FIRST: TOM EXT: 2138 EMPID: 31648
UNUSED:


RECORD ADDED  BY AAARP8 ON  1/28/91 AT 13:18:45
REC#: 2 LAST: SMITH FIRST: SHEILA LAST: SMITH FIRST: SHEILA EXT: 2015 EMPID: 31738
UNUSED:


RECORD BEFORE CHANGE  BY AAARP8 ON  1/28/91 AT 13:18:53
REC#: 1 LAST: JONES FIRST: TOM  LAST: JONES FIRST: TOM EXT: 2138 EMPID: 31648
UNUSED:


RECORD AFTER CHANGE  BY AAARP8 ON  1/28/91 AT 13:19:01
REC#: 1 LAST: JONES FIRST: ALAN LAST: JONES FIRST: ALAN EXT: 2138 EMPID: 31648
UNUSED:


RECORD ADDED  BY AAARP8 ON  1/28/91 AT 13:19:30
REC#: 3 LAST: BROWN FIRST: DIANA LAST: BROWN FIRST: DIANA EXT: 2505 EMPID: 28916
UNUSED:


RECORD DELETED  BY AAARP8 ON  1/28/91 AT 13:19:35
REC#: 3 LAST: BROWN FIRST: DIANA LAST: BROWN FIRST: DIANA EXT: 2505 EMPID: 28916
UNUSED:


*
```

FMPRINT *Utility*

Displays an FM data file.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "FMPRINT

What It Does

FMPRINT prints data records from an **FM** data file as if you were using the FIND NEXT key.

For triple precision files that use **FMT** and **DBGENT**, use **FMPRNTT** instead of **FMPRINT**.

How To Use It

Execute the utility by entering **RUN**, **CHAIN**, or **SWAP "FMPRINT**. This starts the **FMPRINT** dialog.

FMPRINT prompts you for the first and last record numbers or the first and last key values. You can also specify your output file. Enter the device name where you want the data file printed or the file that you want the data file placed in. When no output file is entered, the data file is displayed at your terminal (the default). **FMPRINT** also asks for an output format: spaced, packed, or full.

Your data file must have been set up using **FM** and a table file. When running **FMPRINT**, Business BASIC may ask you to supply a user number to gain access to the data file.

FMPRINT can also be used to print the actual record of a table file. Enter **FM.TB** (or **FMT.TB** for triple precision) in response to the first prompt and the filename of the file containing the records in response to the second prompt.

FMPRINT

Continued

Examples

1. This example illustrates the **FMPRINT** dialog and output when using a data file as the input file.

```
* RUN "FMPRINT
DATA FILE MAINTENANCE  REV X.XX
FILENAME: EMP
EMPLOYEE MAINTENANCE

EMPLOYEE
OUTPUT FILE:
FORM FEED BETWEEN RECORDS? NO
SPACING (SPACED, PACKED, FULL)? PACKED
FIRST RECORD
REC#: 1
LAST RECORD
REC#: 5

REC#: 1  LAST: JONES  FIRST: ALAN  LAST: JONES  FIRST: ALAN  EXT: 2138
EMPID: 31648 UNUSED:

REC#: 2  LAST: SMITH  FIRST: SHEILA  LAST: SMITH  FIRST: SHEILA  EXT: 2015
EMPID: 31738 UNUSED:

REC#: 3  LAST: BROWN  FIRST: DIANA  LAST: BROWN  FIRST: DIANA  EXT: 2505
EMPID: 28916 UNUSED:

REC#: 4  LAST: ADAMS  FIRST: CARL  LAST: ADAMS  FIRST: CARL  EXT: 2290
EMPID: 29876 UNUSED:

REC#: 5  LAST: HILL  FIRST: BARRY  LAST: HILL  FIRST: BARRY  EXT: 2242
EMPID: 28773 UNUSED:

FIRST RECORD
REC#:       LAST: BROWN
LAST RECORD
REC#:       LAST: JONES
REC#: 3 LAST: BROWN FIRST: DIANA LAST: BROWN FIRST: DIANA EXT: 2505 EMPID: 28916
UNUSED:

REC#: 5 LAST: HILL FIRST: BARRY LAST: HILL FIRST: BARRY EXT: 2242 EMPID: 28773
UNUSED:

REC#: 1 LAST: JONES FIRST: ALAN LAST: JONES FIRST: ALAN EXT: 2138 EMPID: 31648
UNUSED:

FIRST RECORD
REC#:                                  :Press IKEY to end.
*
```

FMPRINT *Continued*

2. This example illustrates the **FMPRINT** dialog and output when using a table file as the input file.

```
* RUN "FMPRINT
DATA FILE MAINTENANCE  REV X.XX
FILENAME: FM.TB
DATA FILE NAME: EMP.TB
FILE MAINTENANCE TABLE FILE


OUTPUT FILE:
FORM FEED BETWEEN RECORDS? NO
SPACING (SPACED, PACKED, FULL)? PACKED
FIRST RECORD
REC#: 0
LAST RECORD
REC#: 1

#0 CONTROL
REC#: 0  FILENM: EMP  FLTYPE: L  IFILE1: LAST  KEYDR1: 13  IFILE2: FIRST
KEYDR2: 14 IFILE3: 0 KEYDR3: 0 NRCTYP: 1 INDCTR: 31416 MXFLDS: 8 DFFORM: 1
DFPAGE: 1

#1 CONTROL RECORD (CON'T)
REC#: 1  MDESC: EMPLOYEE MAINTENANCE  RECLEN: 52  MAXREC: 100  SCRNFN:
LOGFN: EMP.LG

FIRST RECORD
REC#:                                  :Press IKEY to end.


*
```

FMTABPRINT *Utility*

Prints the records in an FM table file.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "FMTABPRINT

Global Switch

/L — Send the output to the default output queue. Use this switch only when you are executing **FMTABPRINT** using the Business BASIC CLI command format (i.e., **!FMTABPRINT/L** *filename*).

What It Does

FMTABPRINT prints the records in an **FM** table file. **FMTABPRINT** searches for a file with a .DS extension where the filename without the .DS matches the filename of the **FM** table file without the .TB extension. **FMTABPRINT** treats the .DS file as the comment file for the table file. If you do not have a comment file, **FMTABPRINT** prints the table file with default information only.

The .DS file contains special lines that take the form:

/field

where *field* is the name of a field in the field descriptor records (type 5 table file records; see **FM**). **FMTABPRINT** puts the text following one of these lines after the corresponding field's information. Any text in the .DS file before the first field line is printed as a heading.

For triple precision files that use **FMT** and **DBGENT**, use **FMTABPRNTT** instead of **FMTABPRINT**.

How To Use It

Execute **FMTABPRINT** by entering **RUN**, **CHAIN**, or **SWAP "FMTABPRINT**. This starts the **FMTABPRINT** dialog.

The utility asks you for the name of the table file and if you want the output to go to the line printer. Enter either the name of the device to print the table file or the name of an output file. When no output file is entered, the table file is displayed at your terminal.

FMTABPRINT *Continued*

You can also execute **FMTABPRINT** in a command line format from the Business BASIC CLI. Enter the name of at least one table file with the command. You can also use the optional **/L** switch to send the output to the line printer. If you don't specify this switch, the output is displayed on your terminal.

Example

When you execute **FMTABPRINT** as a Business BASIC CLI command without the **/L** switch, the listing for the table file **PARAM.TB** appears at your terminal.

```
* !FMTABPRINT PARAM.TB

FILENAME: PARAM                              DIRECT

THIS FILE CONTAINS DESCRIPTIVE RECORDS OF THE SUBFILES USED IN
A SYSTEM.
RECORD 0 OF THIS FILE MUST BE AN ENTRY DESCRIBING THE PARAM FILE
ITSELF.  FIELD HIREC MUST BE CORRECT.

TABLE FILE: PARAM.TB
RECORD LENGTH: 42                                MAXIMUM NUMBER OF RECOR
DS: 100
PARAMETER FILE MAINTENANCE
PARAMETER RECORD  ( TYPE 1 )
REC  NAME    SEQ DESCRIPTION             SIZE TYPE POS. FORMAT  EDIT
5    SUBFIL  1   SUB FILE NAME           10   S    2    A10     U
     THE LOGICAL NAME OF THE SUBFILE

6    MASFIL  2   MASTER FILE NAME        10   S    12   A10     U
     THE NAME OF THE PHYSICAL FILE CONTAINING THE SUBFILE

7    SUBPOS  3   SUB FILE POSITION       4    D    22   D10.0   N

     0       21000000
     THE PHYSICAL BYTE POSITION OF THE START OF THE SUBFILE WITHIN
     THE MASTERFILE.  MUST BE A MULTIPLE OF 512.

8    RECLEN  4   SUB FILE RECORD LENGTH 2    I    26   D5.0    N

     1       32768
9    LSTREC  5   LAST RECORD NUMBER      4    D    28   D6.0    N

     0       100000
10   HIREC   6   HIGHEST RECORD NO.USED 4    D    32   D6.0    N

     0       100000
     THIS FIELD IS NEEDED ONLY FOR RECORD 0 OF THE PARAM FILE


*
```

FORM.SL *Subroutine*

Used with SM to provide formatted screen input/output.

AOS/VS	DG/RDOS	UNIX

What It Does

FORM.SL is part of the Screen Maintenance utility. It is explained under **SM**.

Even though UNIX systems do not support **SM**, you can use this subroutine on UNIX systems. If you are using Business BASIC DG mode (specified by including the **-D** option on the command line to execute Business BASIC), you must use 7-bit mode. **SM** screens do not support 8-bit mode. If you are using Business BASIC in non-DG mode, you must specify the **-C** option when you use **SM** screens. This is because the screens contain embedded DG characters.

FORMIO.SL

Subroutine

Displays edited screen input and output for CSM screens.

AOS/VS	DG/RDOS	UNIX

What It Does

FORMIO.SL is part of the Conversational Screen Maintenance utility. It is explained under **CSM**.

FPRINT

BASIC CLI Command

Displays the contents of a disk file.

AOS/VS	DG/RDOS	UNIX

Format

!FPRINT *filename* [*outputfile*/**L**]

Arguments

filename	The name of a file in the current directory.
o*utputfile*/**L**	The name of the output file to receive *filename*. This argument overrides a global /**L** switch. If this argument is used, it must have /**L** appended to it.

Global Switches

/**B**	Display *filename* in byte format.
/**D**	Display *filename* in decimal format.
/**H**	Display *filename* in hexadecimal format.
/**L**	Display *filename* on the default output queue.
/**N**	Allow duplicate lines of output.
/**O**	Display *filename* in octal format.

Local Switches

[**O**]*n*/**F**	Print the contents of *filename* starting at octal location *n*.
D*n*/**F**	Print the contents of *filename* starting at decimal location *n*.
X*n*/**F**	Print the contents of *filename* starting at hexadecimal location *n*.
[**O**]*n*/**T**	Stop the printing of *filename* at octal location *n*.
D*n*/**T**	Stop the printing of *filename* at decimal location *n*.
X*n*/**T**	Stop the printing of *filename* at hexadecimal location *n*.

What It Does

FPRINT displays the contents of a file in octal, decimal, byte, or hexadecimal. The file's location counter is always printed in octal. The default listing device is the terminal, and the default format is octal.

The **FPRINT** command under Business BASIC always assumes a starting location of 0 for all files.

How To Use It

Execute the command by entering **FPRINT** from the Business BASIC CLI. The argument filename must follow **FPRINT**. If used, global switches are appended either

FPRINT

Continued

to the command word or to another global switch while local switches appear as separate arguments that follow filename.

The default format for **FPRINT** is octal and the default output queue is your terminal. If you use the global /L switch, the output is sent to a line printer. The argument *outputfile*/L overrides the global /L switch and indicates a file to receive filename.

Examples

1. Display all of **MYFILE** in decimal at your terminal.

   *** !FPRINT/D MYFILE**

```
    0       8      63       9       2       1      32       0       0...?.....
   10       0       0       0       0       0       0       0       0.........
   20       0       0       0       0       0       0       0       0.........
***
  400       4   65535       0     101       0       1       0     102......e.......f
  410       0       2       0     103       0       3   65535   65535......g........
  420   65535   65535   65535   65535   65535   65535   65535   65535.........
  430   65535   65535   65535   65535   65535   65535   65535   65535.........
```

2. Display **MYFILE** in decimal at your terminal. The display begins at octal location 400 in **MYFILE** and continues to octal location 420.

   *** !FPRINT/D MYFILE 400/F 420/T**

```
  400       4   65535       0     101       0       1       0     102......e.......f
  410       0       2       0     103       0       3   65535   65535......g........
  420    ----    ----    ----    ----    ----    ----    ----    ----.........
```

3. Display **MYFILE** in decimal at your terminal. The display starts at decimal location 256 in **MYFILE** and continues to decimal location 272.

   *** !FPRINT/D MYFILE D256/F D272/T**

```
  400       4   65535       0     101       0       1       0     102......e.......f
  410       0       2       0     103       0       3   65535   65535......g........
  420    ----    ----    ----    ----    ----    ----    ----    ----.........
```

4. Display **MYFILE** in decimal at your terminal. The display starts at hexadecimal location 100 in **MYFILE** and continue to hexadecimal location 110.

   *** !FPRINT/D MYFILE X100/F X110/T**

```
  400       4   65535       0     101       0       1       0     102......e.......f
  410       0       2       0     103       0       3   65535   65535......g........
  420    ----    ----    ----    ----    ----    ----    ----    ----.........
```

FREE

BASIC CLI Command

Frees a device from exclusive (ASG) use.

DG/RDOS

Format

!FREE *filename*

Argument

filename — The name of a reserved device that was assigned to a job by the Business BASIC CLI command **ASG**.

What It Does

FREE releases a device from the exclusive use of a job that was assigned to the device by the **ASG** command. Only the job that made the **ASG** assignment can free the device.

How To Use It

Execute the command by entering **FREE** from the Business BASIC CLI.

Example

Release the line printer from the exclusive use of your job.

```
!FREE $LPT
```

GDIR

BASIC CLI Command

Displays the current directory name.

AOS/VS	DG/RDOS	UNIX

Format

!GDIR

What It Does

GDIR returns the name of the current user directory.

On DG/RDOS systems, **GDIR** displays the directory name without the .DR extension.

On AOS/VS and UNIX systems, **GDIR** returns the full pathname of the current directory.

How To Use It

Execute the command by entering **GDIR** from the Business BASIC CLI.

Examples

1. Under AOS/VS, **GDIR** displays the full pathname for your current directory, **BBASIC**.

```
!GDIR
:UDD:SHEILA:BBASIC
!
```

2. Under DG/RDOS, **GDIR** tells you that you are in directory **NBASIC.DR**.

```
!GDIR

NBASIC

!
```

GETCM.SL *Subroutine*

Creates a Business BASIC CLI command.

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 7550
GOSUB 7500

Input Variables

Q9	A byte pointer; Q9 points to the field in the command string that you want to read.
T9$	The string that receives the command string from the common area. Dimension T9$ to 512 bytes.

Output Variables

Q9	A byte pointer that shows the start of the next field
X$	A string containing the last field that was read from T9$
S	The switches associated with the field read from T9$ (A -1 in S indicates the end of the command string.)

Line Numbers

7550	Entry point to initialize routine variables and read the common area into T9$
7500	Entry point to read the field pointed to by Q9
7500-7560	Line numbers for **GETCM.SL**

What It Does

Use **GETCM.SL** in a program to generate your own Business BASIC CLI command. When the Business BASIC CLI does not recognize a command, it looks for a command string in the Business BASIC common area and swaps to the filename in the first field of the string.

When you execute the statement **GOSUB 7550**, **GETCM.SL** initializes its variables and places the command string in T9$ with Q9 pointing to the first field. Then, when you execute **GOSUB 7500**, **GETCM.SL** extracts a field from T9$ and places the field and any switches associated with it in the output variable X$. Each time **GETCM.SL** extracts a field from T9$, it updates Q9 to point to the start of the next field, and it returns the sum of the switch values in the output variable S.

You can set up your program so that **GETCM.SL** continues to extract fields from the string until the output variable S is returned with a value of -1, indicating the end of the command string.

GETCM.SL

Continued

For example, if switches for a field were /B/K/L, then the value returned by GETCM.SL for S would be 1076887552 (the sum of bits 30, 21, and 20).

Switch	Bit Position	Value Returned in S
/A	31	-2147483648
/B	30	1073741824
/C	29	536870912
/D	28	268435456
/E	27	134217728
/F	26	67108864
/G	25	33554432
/H	24	16777216
/I	23	8388608
/J	22	4194304
/K	21	2097152
/L	20	1048576
/M	19	524288
/N	18	262144
/O	17	131072
/P	16	65536
/Q	15	32768
/R	14	16384
/S	13	8192
/T	12	4096
/U	11	2048
/V	10	1024
/W	9	512
/X	8	256
/Y	7	128
/Z	6	64

How To Use It

To execute the subroutine, code the statement **ENTER "GETCM.SL** in your program. Dimension T9$ to 512 bytes. Then enter the program statement **GOSUB 7550** to initialize your variables and to read the common area into T9$. Next, enter **GOSUB 7500** to extract information from T9$ and place the sum of its switch values in the output variable S.

Since the entry point at 7550 initializes variables, you should have only one **GOSUB 7550** each time you use **GETCM.SL**. However, you can continue to use **GOSUB 7500** until S is returned with a value of -1, indicating the end of the command string. Do this by checking the value in S each time you use **GOSUB 7500**. If S does not equal -1, execute **GOSUB 7500** again and extract the next field from T9$.

To see the Business BASIC CLI command that is in T9$, print X$ each time GETCM.SL extracts a field from T9$.

GETCM.SL

Continued

Example

Write the program **GET**, which returns the combined values of **/B** and **/C**, and then add the subroutine **GETCM.SL** to the program. You only need to code the statement **ENTER "GETCM.SL** into **GET** one time and then save the program. When Business BASIC saves the program, it makes **GETCM.SL** a part of **GET**. **GET** has one **GOSUB 07550** to initialize the **GETCM.SL** values; however, line 00060 in **GET** loops back to **GOSUB 07500** until S is returned with a -1. In line 00050, **GET** prints the Business BASIC CLI command and the value of S.

```
* LIST
00010 DIM T9$[512],X$[24]
00020 GOSUB 07550
00030 GOSUB 07500
00040 IF S=-1 THEN STOP
00050 PRINT X$, TAB(35),S
00060 GOTO 00030



* ENTER "GETCM.SL
* SAVE "GET
* !GET/B/C

GET                1610612736


STOP AT 40
```

GETLAST.SL *Subroutine*

Retrieves the number of active records and the highest record in use in a linked-available-record file (logical file structure).

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 9950

Input Variable

F%	The logical file number of a type L file (linked-available-record file).

Output Variables

ACTREC	Number of active records in the logical file.
LSTREC	Number of the highest record in use in the logical file.
E	Error code (0 = normal return).

Line Numbers

9950-9976	Line numbers for **GETLAST.SL**.

What It Does

GETLAST.SL retrieves the last active record number (highest record number in use) and the number of active records from record 0 of a type L (linked-available-record) logical file.

An error occurs if the file is not type L or if the file has not been initialized. In that case, **GETLAST.SL** returns the variables LSTREC and ACTREC as undefined.

How To Use It

To execute the subroutine, add the statement **ENTER "GETLAST.SL** to your program. Perform an **LOPEN FILE** statement on the logical file and assign the logical file number to F%. Then enter the program statement **GOSUB 9950**.

Example

The program **CUSTOMER** opens a customer data file and then calls **GETLAST.SL** to determine the last record number in use and the number of active records. This example assumes that the data file was set up to contain a maximum of 500 records, of which 100 were added to the file and then 15 were deleted from throughout the file. Thus, the last record number in use is 100 while only 85 of those records are in use.

GETLAST.SL

Continued

You only need to enter **GETLAST.SL** into **CUSTOMER** one time and then type **SAVE "CUSTOMER**. This makes **GETLAST.SL** a part of **CUSTOMER**.

```
* LIST
00010 DIM LFTABL$[52],T9$[544]
00020 DIM ACTREC[10],E[10],LSTREC[10]
00030 LET LFTABL$=FILL$(0)            :Initialize local file table.
00040 LOPEN FILE[2,T9$],"CUST"        :Open customer data file.
00060 LET F%=2                        :Assign input file variable
                                      :for GETLAST.SL
00070 GOSUB 09950                     :Call GETLAST.SL.
00080 PRINT "LAST RECORD NUMBER IN USE: ";LSTREC
                                      :Print last record number,
00090 PRINT "ERROR CODE IS ";E        :error code (0 if no error),
00100 PRINT "ACTIVE RECORD COUNT: ";ACTREC
                                      :and number of active records.
00110 END


* ENTER "GETLAST.SL
* SAVE "CUSTOMER
* RUN "CUSTOMER


LAST RECORD NUMBER IN USE: 100
ERROR CODE IS 0
ACTIVE RECORD COUNT: 85
*
```

GETREC.SL *Subroutine*

Gets the number of the next available record in a linked-available-record file (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 8400

Input Variables

F%	The file number (row in the C1 array for this subfile).
C1	The file characteristics array.

Output Variable

R1	The number of the next available record (-1 if the file is full).

Scratch Variables

X0 %	Record status of record 0.
Y0 %	Record status of first record on the deleted record chain.
X0	New next available record.
Y0	Next available record from record 0.
Z0	Last record used from record 0.

Line Numbers

8400-8495	**GETREC.SL** occupies these line numbers.
9610-9645	**POSFL.SL**, which is used by **GETREC.SL**, occupies these line numbers.

What It Does

GETREC.SL retrieves the next available record number in a linked-available-record subfile. Use **GETREC.SL** with files in the PARAM file database structure that have a minimum record size of 10 bytes. Linked-available-record files use dynamic allocation and maintain their own space if you use **GETREC.SL** and **DELREC.SL**.

GETREC.SL maintains record 0 in a linked-available-record file to show which records are in use. The next free record is either the next record in a deleted record chain or the lowest numbered unused record. If the next available record is a deleted record, **GETREC.SL** uses **POSFL.SL** to set a pointer to the deleted record and checks the record status bytes to verify that this record is a deleted record. It is the responsibility of the user programs to assign the status bytes in the first 2 bytes of each record before writing the record.

GETREC.SL

Continued

NOTE: If you use the logical database file structure, you can use the **GETREC** statement instead of this subroutine. The advantages of the **GETREC** statement are that it performs automatic locking, is faster than **GETREC.SL**, and frees the code space normally occupied by the subroutine. See **GETREC** in *Commands, Statements, and Functions in Business BASIC*.

How To Use It

To use **GETREC.SL**, perform the following steps:

1. Add the statements **ENTER "GETREC.SL** and **ENTER "POSFL.SL** to your program.
2. Build the C1 array and assign the file number (the row in the C1 array) for the file to **F%**.
3. Lock record 0 of the file.
4. Enter the program statement **GOSUB 8400**.
5. Enter the program statement **GOSUB 9610**.
6. Use the **WRITE FILE** statement to write to record R1.
7. Unlock the file.

GETREC.SL

Continued

Example

This segment of code uses **GETREC.SL** to get an available record of **EMP**.

```
00010 DIM C1[1,3],B$[544],D$[18],KEY$[4],REC$[52]
00020 LET B$="EMPIX,5,EMP,5",FILL$(0)    :Open file routine
00030 BLOCK WRITE B$
00040 SWAP "OPEN
00050 BLOCK READ B$
00060 LET K=1
00070 FOR I=0 TO 1
00080   FOR J=0 TO 3
00090     LET C1[I,J]=ASC(B$[K,K+3])
00100     LET K=K+4
00110   NEXT J
00120 NEXT I
00130 LET D$=CHR$(C1[0,0],2),CHR$(C1[0,1],4),CHR$(0,2),"EMPIX",
FILL$(0)
00140 LET F%=1                      :Subfile 1 (row 1 of C1) is EMP
00150 LET R1=0
00160 LET REC$[1,2]=CHR$(1,2)
.                                   :Input record data
.
.
00300 LET KEY$=REC$[3,6]            :Assign key value
00320 LET T=30
00330 LOCK 1,"EMP",R1*C1[F%,3],C1[F%,3],T      :Lock record 0 of EMP
00340 IF T=57 THEN GOTO 00320          :If lock times out, try again
00350 GOSUB 08400 : \ GETREC.SL     :Get next available EMP record
00360 GOSUB 09610 : \ POSFL.SL      :Position using R1 from GETREC.SL
00370 WRITE FILE[C%],REC$           :Write record using C% from POSFL
00380 KADD D$,B$,KEY$,R1            :Add new key; R1 is record pointer
00390 IF R1<=0 THEN GOTO 02000      :If error occurred in index,
                                    :go to an error routine.
00400 UNLOCK                        :Unlock record 0 of EMP
00410 GOTO 00150
.
.
.
```

GQUE

BASIC CLI Command

Displays the name of the default queue.

AOS/VS	DG/RDOS	UNIX

Format

!GQUE

What It Does

GQUE displays the name of the default output queue at your terminal. You can set the default output queue with the SQUE Business BASIC CLI command or the STMA 10,1 statement. To set the default output queue on an AOS/VS or UNIX system, you can also use the /Q (AOS/VS) or -q (UNIX) global switch when you execute Business BASIC.

How To Use It

Execute GQUE by entering the command from the Business BASIC CLI.

Examples

1. On an AOS/VS system, GQUE shows that the default output queue is the line printer.

```
!GQUE
@LPT
```

2. On DG/RDOS, GQUE shows that the default output queue is the line printer.

```
!GQUE
$LPT
```

GSDIR *BASIC CLI Command*

Displays the name of the system directory.

DG/RDOS

Format

!GSDIR

What It Does

GSDIR returns the name of the system directory. The system directory is the directory from which you executed Business BASIC. It is the directory where your Business BASIC system or links to it reside. The system directory can be changed with the Business BASIC CLI **SDIR** command or with the **DIR** command (see *Commands, Statements, and Functions in Business BASIC*).

NOTE: Use the Business BASIC CLI **DIR** command to change directories. This prevents you from accidentally changing the system directory.

How To Use It

Execute **GSDIR** by entering the command from the Business BASIC CLI.

Example

GSDIR shows that the system directory is **DE0**.

```
!GSDIR
DE0
```

GSYS

BASIC CLI Command

Displays the operating system name.

DG/RDOS

Format

!GSYS

What It Does

GSYS returns the name of the operating system.

How To Use It

Execute GSYS by entering the command from the Business BASIC CLI.

Example

GSYS shows that the current operating system is MYRDOS.

```
!GSYS
MYRDOS
```

GTOD

BASIC CLI Command

Displays the time and date.

AOS/VS	DG/RDOS	UNIX

Format

!GTOD

What It Does

GTOD displays the time as *hh:mm:ss* and the date as *mm/dd/yy*.

How To Use It

Execute **GTOD** by entering the command from the Business BASIC CLI.

Example

GTOD returns the time as 5:35 a.m. (and 16 seconds) and the date as March 14, 1991.

```
!GTOD
05:35:16     3/14/91
```

IBUILD

Utility

Builds an index file from a sorted tag file, a sorted data file, or an index file.

AOS/VS	DG/RDOS	UNIX

Format

SWAP "IBUILD

What It Does

IBUILD builds an index file in one pass using **BLOCK WRITE** instead of **KADD**. Your input file must be an index, a data file, or a tag file sorted by key in ascending order.

IBUILD reads the argument string from the common area and then uses the index, sorted data, or tag file to build an index with the specified blocking factor. For a sorted data file, you must specify the starting and ending byte of the key, and **IBUILD** calculates the data record pointer directly. With a tag file, **IBUILD** assumes that the key starts at the first byte of the tag file record and that the record pointer associated with the key is four bytes immediately after the key.

When **IBUILD** encounters an error, it fills block 0 of the output index with nulls to prevent you from using an index that was incorrectly rebuilt. Therefore, you should back up all files before rebuilding. Note that **INDEXBLD**, **LINDEXBLD**, and **IREBLD** can use **IBUILD**, depending on your responses to their prompts.

How To Use It

Execute **IBUILD** by entering **SWAP "IBUILD** from within a program. You can write your own program to call **IBUILD** or use **INDEXBLD**.

To use **IBUILD** from within a program, do the following:

1. Open the input and output (index) files.
2. Set up an argument string that includes the channel numbers for the input and index files.
3. Use **BLOCK WRITE** to put the argument string in the common area.

IBUILD

Continued

Table 1-12. IBUILD Argument String

Substring Location	Size in Bytes	Contents
1,4	4	Channel number of the open input file.
5,8	4	Byte offset to record 0 of input file.
9,12	4	Maximum number of records in an input file.
13,16	4	Number of bytes per record in input file. If the input file is a tag file, the number of bytes per record is the number of bytes in a tag file record plus four bytes for the pointer.
17,20	4	Channel number of opened index file.
21,24	4	Byte offset to record 0 of index file.
25,28	4	Maximum number of blocks in index file (use **INDEXCALC** to calculate this).
29,32	4	Number of bytes per block in index file, 512 (AOS/VS, DG/RDOS, and UNIX) or 2048 (AOS/VS and UNIX).
33,34	2	Keys per block at the specified blocking factor (use the **INDEXCALC** utility).
35,36	2	Flag to allow duplicate keys: 1 Allow duplicate keys. 0 Do not allow duplicate keys.
37,38	2	Flag to check for deleted records: 1 Check for deleted records. 0 Do not check for deleted records.
39,40	2	Total length of key field in bytes.
41,42	2	Number of key fields. -1 Tag file input. -2 Index file input.
*43,44	2	First byte of key (1 is the first byte of the record).
*45,46	2	Last byte of the key.
47,*	4	Next key field's descriptors. Repeat the sequence of entering the first and last byte of the key. Since you can have multiple keys, you can enter this information several times.

4. Enter the program statement **SWAP "IBUILD**.

5. Check **STMA 1,1** to see if an error occurred. The utility returns either a 0 for a successful **IBUILD** or the appropriate error code. If an error occurred in an **ON ERR** trap, the line number of the error is returned in **STMA 1,2**.

IBUILD

Continued

The following error codes can occur when you are using **IBUILD**:

Error Code	Meaning
45	Illegal record length.
68	Index file full.
77	Illegal record number.
146	Key already exists and duplicates are not allowed.
148	File not on sector boundary.
149	Record out of sequence.
150	Illegal blocking factor.
151	Illegal key length.

Data files used as input must be sorted in ascending order on the key for which you are building an index. The key field must be in the same location in each record in the data file. The key can have multiple fields, but you must specify the locations of the key fields and their sizes.

Example

This program segment uses **IBUILD** to create an index file (CI%) for the input file whose channel number has been placed in C%.

```
00010 DIM X$(512)              :Get argument string from common area.
00020 LET ERCODE=0             :Initialize ERCODE and LINENO for
00030 LET LINENO=0             :use with STMAs.
.                              :Put code to open both files here.
.                              :Channel of input file is in C%;
.                              :channel of new index is in CI%.
00100 LET X$=CHR$(C%,4),CHR$(0,4)        :Data file has 0 offset,
00110 LET X$[0]=CHR$(100,4),CHR$(100,4)  :100 records and 100 bytes
                                         :per record.
00120 LET X$[0]=CHR$(CI%,4),CHR$(0,4)    :Index file has 0 offset,
00130 LET X$[0]=CHR$(8,4),CHR$(512,4) :8 blocks, 512 bytes per block.
00140 LET X$[0]=CHR$(36,2)   :Blocking factor is 36 keys per block.
00150 LET X$[0]=CHR$(1,2),CHR$(1,2)   :Allow duplicates,
                                      :check deleted recs.
00160 LET X$[0]=CHR$(10,2),CHR$(2,2)  :Total key 10 bytes, 2 fields.
00170 LET X$[0]=CHR$(3,2),CHR$(6,2)   :First field, bytes 3 to 6.
00180 LET X$[0]=CHR$(15,2),CHR$(20,4) :Second field, bytes 15 to 20.
00190 LET X$[0]=FILL$(0)              :No more fields.
00200 BLOCK WRITE X$           :Send into common area.
00210 SWAP "IBUILD"            :Execute IBUILD and return.
00220 STMA 1,1,ERCODE          :Get error code from IBUILD.
00230 STMA 1,2,LINENO          :Get line of error if ON ERR trap.
00240 IF ERCODE THEN GOTO 00700       :If IBUILD not successful,
                                      :go to 700.
. . .
```

INDEXBLD *Utility*

Builds or rebuilds an index file (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "INDEXBLD

What It Does

INDEXBLD is an interactive utility that builds an index file from a data file, a tag file, another index file in the PARAM file database structure, or a physical file. (Use **LINDEXBLD** for files in the logical file database structure.)

With a data file, **INDEXBLD** prompts you on whether to use **IBUILD** or **KADD** (**XBUILD**) to build the index file. If you select **IBUILD**, **INDEXBLD** calls **TBUILD** to build a tag file and then calls **QFILESORT** to sort the tag file. When you select **KADD**, **INDEXBLD** sets the blocking factor at 50%.

When the input file is a sorted tag file, **INDEXBLD** bypasses **TBUILD**. If the input file is another index file, **INDEXBLD** assumes you want to rebuild the index with a new blocking factor. **INDEXBLD** always uses **IBUILD** to generate the new index file when the input file is a sorted tag file or another index file.

Anytime **INDEXBLD** encounters an error, the utility returns an error message and fills block 0 of the output index file with nulls. If you are building in place (i.e., both the input file and the output file are the same), this action corrupts your original index. To recover you must have a backup copy of your index. When building in place, **INDEXBLD** creates a tag file using **TBUILD**. If the system crashes after **INDEXBLD** has displayed the message `TEMP FILE BUILD COMPLETED... BUILDING INDEX`, the tag file (**TAG-.TM**) may exist on the disk. Then you can use it as the input file to rebuild the index.

How To Use It

Execute **INDEXBLD** by entering **RUN**, **CHAIN**, or **SWAP "INDEXBLD**. This starts the **INDEXBLD** dialog.

INDEXBLD asks for the name of the input file and whether it is an index, data, or sorted tag file. If the input file is a data file, **INDEXBLD** asks you to choose either **KADD** (**XBUILD**) or **IBUILD** for building the index. Index and tag files are sorted, so **INDEXBLD** automatically uses **IBUILD** for them.

INDEXBLD looks for a PARAM entry defining the input file. If no PARAM entry exists, **INDEXBLD** issues a warning and asks for the byte offset, record size, and file

INDEXBLD

Continued

size of the input file. With data files, **INDEXBLD** asks if you want to omit deleted records (indicated by a 0 in the first two bytes of the record) from the intermediate file.

INDEXBLD asks for the number of fields in the key. With data files, it also asks for the starting and ending byte locations for each field in the key. The key field must be in the same location in each record in the data file.

Finally, **INDEXBLD** asks for the output filename of the index file you are building. If you are not building in place, the output file must exist as a null file. With an output file that is a physical file, **INDEXBLD** asks for the byte offset to the first record in the subfile and the index block size. All operating systems permit an index block size of 512 bytes, while AOS/VS and UNIX systems also allow block sizes of 2048 bytes. **INDEXBLD** also asks for blocking factor and whether duplicate keys are allowed.

Example

Once executed, **INDEXBLD** begins to prompt you for information. The utility provides you with the options shown in parentheses. User responses for this example are shown following a colon (:). The default responses appear in brackets [ ]. This is a DG/RDOS example, so **INDEXBLD** displays the message `INDEX BLOCK SIZE IS 512` instead of prompting you for the block size.

```
* RUN "INDEXBLD


INPUT FILENAME: CUST
INPUT FILE IS INDEX(0), DATA(1), OR SORTED TAG(2): 1
USE IBUILD(0) OR KADD(1): 1
CHECK FOR DELETED RECORDS: (Y OR N) [N]: Y
TOTAL KEY LENGTH IN BYTES: 10

NUMBER OF FIELDS IN KEY: 2
KEY FIELD # 1 LOCATION IN DATA RECORD(BYTE 0-N): 2
FIELD LENGTH IN BYTES: 4
KEY FIELD # 2 LOCATION IN DATA RECORD(BYTE 0-N): 18
FIELD LENGTH IN BYTES: 6
INDEX FILE NAME: CUSTINDEX
INDEX BLOCK SIZE IS    512
BLOCKING FACTOR ( % PERCENT): 50
DUPLICATE KEYS ALLOWED? (Y OR N) [N]: Y
```

INDEXCALC *Utility*

Calculates index and data file information.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "INDEXCALC

What It Does

INDEXCALC is an interactive utility that provides information on index and data files. You use this data when you run other utilities, such as **INITFILE**.

INDEXCALC displays at your terminal:

- The maximum number of keys per index block
- The number of keys per index block at the specified blocking factor
- The number of blocks at each level in the index
- The number of blocks (either 512 or 2048 bytes per block) in the index
- The number of sectors (each 512 bytes) in the index
- The number of sectors in the data file

The information **INDEXCALC** displays is based on:

- The size of the index key
- The number of bytes in your data record
- The maximum number of data records
- A specified blocking factor as a percentage
- The index block size
- The presence or absence of duplicate keys

The maximum number of records in an index file always equals the number of blocks in the index, as returned by **INDEXCALC**.

The index blocking factor is the relative percentage of each index block that you want to fill with key entries (a larger percentage makes the index denser, and a smaller percentage leaves space for future entries). A 50-percent blocking factor with 10-byte key entries tells the index to pack 18 keys in an index block. With a larger blocking factor (i.e., 90 percent), each index block is denser, and the index file is smaller. Once the index file has been built, the blocking factor is ignored.

INDEXCALC *Continued*

If you build an index using **KADD**, supply a blocking factor of 50 percent. All operating systems permit an index block size of 512 bytes, while AOS/VS and UNIX systems also allow block sizes of 2048 bytes.

How To Use It

Execute **INDEXCALC** by entering **RUN**, **CHAIN**, or **SWAP "INDEXCALC**. This initiates the **INDEXCALC** dialog. In response to the BYTES PER RECORD prompt, you must include two extra bytes if you want linked-available records.

You can use blocks 1 - 65534. An error occurs if the number of blocks in the index exceeds 65535 (0 - 65534). Also, since DG/RDOS supports only one block size, **INDEXCALC** on DG/RDOS systems displays INDEX BLOCK SIZE IS 512, instead of querying the user for a block size.

Example

The **RUN "INDEXCALC** command starts the **INDEXCALC** dialog. First, **INDEXCALC** prompts you for information on bytes per key, bytes per record, maximum records, index blocking factor, index block size, and whether duplicate keys are allowed. The default responses are shown in brackets, and the user responses appear after the colon. **INDEXCALC** then displays its calculations and asks if you want to calculate data for another file. This is a DG/RDOS example, so **INDEXCALC** does not prompt you for an index block size.

```
* RUN "INDEXCALC

BYTES PER KEY: 10
BYTES PER DATA RECORD: 52
MAXIMUM NUMBER OF DATA RECORDS: 100
INDEX BLOCKING FACTOR ( % PERCENT) [50]: 50
INDEX BLOCK SIZE IS 512
DUPLICATE KEYS ALLOWED? (Y OR N) [N]: N

36 MAXIMUM KEYS PER INDEX BLOCK
18 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
6 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
8 BLOCK(S) (512 BYTES EACH) IN INDEX
8 SECTORS IN INDEX
11 SECTORS IN DATA FILE

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

On AOS/VS and UNIX systems, **INDEXCALC** asks you to choose either a 512- or 2048-byte index blocking size. ■

INDEXCALC *Continued*

```
* RUN "INDEXCALC

BYTES PER KEY: 10
BYTES PER DATA RECORD: 52
MAXIMUM NUMBER OF DATA RECORDS: 100
INDEX BLOCKING FACTOR ( % PERCENT) [50]: 50
INDEX BLOCK SIZE (512 OR 2048) [512]: 2048
DUPLICATE KEYS ALLOWED? (Y OR N) [N]: N

145 MAXIMUM KEYS PER INDEX BLOCK
73 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
2 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
4 BLOCK(S) (2048 BYTES EACH) IN INDEX
16 SECTORS IN INDEX
11 SECTORS IN DATA FILE

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

INDEXPRT *Utility*

Prints an index file for a logical or PARAM file database structure.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "INDEXPRT

What It Does

INDEXPRT prints all levels of an index, including the index entry pointers to other levels and index file information. It lets you print a key that has multiple fields, each with different key types.

When you specify the line printer as the destination for an **INDEXPRT** listing, your terminal continues to display index information, but only the printer provides data on the actual keys.

How To Use It

Execute **INDEXPRT** by entering **RUN, CHAIN,** or **SWAP "INDEXPRT**. This initiates the **INDEXPRT** dialog. **INDEXPRT** asks you whether the output goes to the line printer or to your screen.

Next, it requests the index file name. **INDEXPRT** opens the index file exclusively, which means that no one else can have the index file open while it is being printed. **INDEXPRT** first tries to open the input file as a logical database file; if that fails, **INDEXPRT** searches the PARAM file. If no PARAM entry exists for the file, the utility displays an error message and asks for the byte offset to the subfile. If the file does not exist, **INDEXPRT** displays an error message and ends.

You can print a key with multiple fields, each with different key types, so **INDEXPRT** prompts you for the number of fields in the key. With only one field, **INDEXPRT** asks you for the key type (integer, string, or crammed). With multiple key fields, **INDEXPRT** requests the location in the key, the field length in bytes, and the key type. Integer keys must not exceed four bytes. String keys are printed with control and lowercase characters represented as decimal numbers within angle brackets. Crammed keys are uncrammed using the default special characters for **CRM$**.

In addition, **INDEXPRT** requests a beginning key. Press CR to indicate that the key starts at the beginning of the file or enter a key. If your key contains multiple fields, **INDEXPRT** prompts you for the beginning key for each field. This is the same as setting up a key string for a **KFIND**. If you do not enter the complete length for each field, **INDEXPRT** pads the incomplete field with nulls.

INDEXPRT *Continued*

INDEXPRT displays level indicators for each index block followed by the keys in that index block. The utility places a blank line between each index block. The level indicators begin with the level 0 entries. These are printed on separate lines and indented so that the level 0 entry is indented more than any other level entry. These entries are printed in three columns and represent:

1. A reference number beginning at 1 and incremented by 1 for each entry in the block at this level.
2. The high key in the block to which this entry points.
3. A pointer representing the block number of the next higher level block.

Immediately following the level indicators are the keys in this block. These three columns represent:

1. A reference number beginning at 1 that is incremented by 1 for each key printed until the end of the first block.
2. The key.
3. The number of the record in the data file to which the key points.

If the index being printed contains 2048-byte blocks and duplicate keys are allowed, then an additional column is printed for the level indicator and for each key in the block. This column is displayed between the key and the pointer (columns 2 and 3 described above) and contains the occurrence number.

Example

Once executed, **INDEXPRT** begins prompting you for information. The default responses to **INDEXPRT** prompts are shown in brackets, and user responses appear after the colons. **INDEXPRT** skips a line after prompting you for the key type and then displays the output. In this example, GENTRY is the high key in block 1, and -1 is the high key in block 2.

INDEXVRFY *Utility or Business BASIC CLI Command*

Verifies the structure of an index file.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "INDEXVRFY

Or

!INDEXVRFY *filename* [, *filename* ...]

Argument

filename The name of the index file to be checked.

Global Switch

/L List to the default output queue.

What It Does

INDEXVRFY detects most of the structural flaws that can occur in a Business BASIC index file. The utility completely traverses the index tree structure.

How To Use It

Start the **INDEXVRFY** dialog by entering **RUN**, **CHAIN**, or **SWAP "INDEXVRFY**, or bypass the dialog by executing **INDEXVRFY** as part of a Business BASIC CLI command line.

If you use **RUN**, **CHAIN**, or **SWAP** to execute **INDEXVRFY**, the utility asks whether to send the output to the printer. Then **INDEXVRFY** asks for the name of the index file to verify.

When you execute **INDEXVRFY** as a Business BASIC CLI command, the name of the file to be verified must be supplied on the command line. You can use a single **INDEXVRFY** command with multiple filenames, but each file must be part of either the logical file database structure or the PARAM file database structure. You can use the **/L** switch to have the output sent to the default output queue.

INDEXVRFY first tries to open the index file as a logical database file; if that fails, the utility searches the PARAM file. If no PARAM entry exists for the file, **INDEXVRFY** asks for the byte offset to the file. When the index file does not exist, **INDEXVRFY** displays a missing file message and ends. If it finds the file, the utility sends a message to the screen saying the file is being verified.

INDEXPRT

Continued

```
* RUN "INDEXPRT

OUTPUT TO PRINTER (Y OR N) [N]: N
INDEX FILE NAME: INDEX1
NUMBER OF FIELDS IN KEY [1]: 1
KEY TYPE (1-INTEGER,2-STRING,3-CRAM) [1]: 2

INDEX NAME                  = INDEX1
BYTES PER KEY               = 24 DUPLICATE KEYS NOT ALLOWED
MAXIMUM KEYS PER BLOCK      = 18
BLOCKING FACTOR             = 9 KEYS PER BLOCK
MAXIMUM INDEX BLOCKS        = 40  512 BYTE BLOCKS
NEXT AVAILABLE BLOCK        = 4
LEVEL ZERO BLOCK            = 3

BEGINNING KEY =

01 GENTRY                   1
01 ACME                     1
02 ALLELEC                  2
03 AUDIOSHAK                16
04 BURCORP                  5
05 COMPUTONE                10
06 CRAMER                   4
07 DATACON                  7
08 FISCHER                  6
09 GENTRY                   8

02 -1  2
01 MCFARLAND                3
02 METAL                    20
03 PHONE                    19
04 POWELL                   9
05 POWERMATE                15
06 RESEARCH                 17
07 SPEARS                   14
08 STANDARD                 18
09 SYKES                    13
10 TETRONIX                 12
11 WABASH                   11
12 -1  -1
```

INDEXVRFY *Continued*

Once it has found a file, **INDEXVRFY** traverses its logical structure. **INDEXVRFY** starts at block zero and checks every block used by the index. The utility also checks each block pointer to see that it falls within the index file limits. If block pointers point outside the index file limits, **INDEXVRFY** displays the message INVALID BLOCK ADDRESS along with the address of the block with the invalid pointer, the invalid pointer, and the maximum legal block address. **INDEXVRFY** then stops processing that file.

In addition, **INDEXVRFY** checks every key on every level for correct sequence on that level. Each key in the level above is compared with the last key in the block it points to. **INDEXVRFY** displays in octal any out-of-sequence keys or a key in an upper level index block that is less than the last key in the block it points to.

The utility also checks the forward pointers on the bottom level to ensure that they match the tree structure. When differences are found, **INDEXVRFY** displays the message FORWARD POINTER MISMATCH WITH UPPER LEVEL and stops processing that file.

INDEXVRFY also checks each block for a key count less than the maximum. Each level must end with a key value of -1. If the link word in the level zero block is not a 0 or a -1, **INDEXVRFY** displays the message INVALID POINTER TO NEXT IN LEVEL ZERO and stops processing that index file.

When you are using 2048-byte index blocks, **INDEXVRFY** also checks the backward pointers on the bottom level to ensure that they are correct.

After successfully traversing the index, **INDEXVRFY** prints statistics about the index file. In the output display, the message EMPTY INDEX BLOCKS refers to blocks that were in use but the keys in them have been deleted. The maximum, minimum, and average key counts include all index blocks used at all levels.

INDEXVRFY *Continued*

Example

Executed as a Business BASIC CLI command, **INDEXVRFY** checks on the index file **TSTIV**. Since the **/L** switch is not appended to **INDEXVRFY**, the output goes to the terminal.

```
!INDEXVRFY TSTIV

          INDEX FILE NAME = TSTIV
          ** VERIFYING **

Index file size              - 100  512 byte blocks
Number of index blocks used  - 34
Empty index blocks           - 0
Key length                   - 4       Duplicates not allowed
Max keys per block           - 62
Min key count                - 31
Max key count                - 40
Avg key count                - 31
Total keys at bottom level   - 1000
Number of index levels       - 2
INDEX STRUCTURE VERIFIED CORRECT

******    VERIFY COMPLETE    ******
```


INIT

BASIC CLI Command

Initializes a device, partition, or directory.

DG/RDOS

Format

!INIT *itemname*

Argument

itemname The name of a device, directory, or partition.

What It Does

INIT initializes *itemname*, making it accessible for input/output. Files on multiple file devices, such as disks, cannot be accessed until the device and any directories containing these files are initialized. (A **DIR** command implies an **INIT** and initializes a directory.)

How To Use It

Execute the command by entering **INIT** from the Business BASIC CLI. The *itemname* argument must follow **INIT**.

Examples

1. This initializes **DP1** (disk unit 1), the partition **USER.DR** in **DP1**, and the subdirectory **MYTOWN.DR** in the partition **USER.DR**.

   !INIT DP1:USER:MYTOWN

2. This initializes tape drive 0.

   !INIT MT0

INITFILE *Utility*

Creates and/or initializes index or data files (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

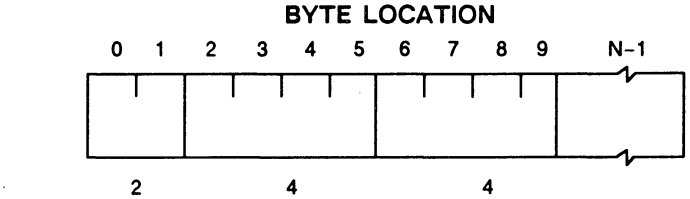
Format

```
{ RUN  }
{ SWAP } "INITFILE
{ CHAIN}
```

What It Does

INITFILE builds record 0 of a linked-available-record file or builds the level 0 block of an index file and adds the -1 key to indicate an index file. **INITFILE** can also make a PARAM entry for your file. The PARAM file, however, must exist; **INITFILE** cannot create it.

When you use **INITFILE** to add records to the PARAM file, it also updates the number of the last record containing data (bytes 32-35 in the PARAM file). If you add records to the PARAM file using **FM**, you must update this count yourself.



Where:

N = maximum number of bytes per record

Byte Locations:

0-1 = status flag (always -2 for record 0).
2-5 = record number of next available record (-1 if there are no deleted records).
6-9 = record number of last number used—either last record in file or highest-number record written to.
N-1 = unused. Record 0 is always the same size as the data records.

Figure 1-1. Record 0 of a Linked-available-record Format File

INITFILE

Continued

How To Use It

Start the **INITFILE** dialog by entering **RUN, CHAIN,** or **SWAP "INITFILE.**

Note: All 2048-byte block indexes must start on a 2048-block boundary, which is a multiple of four sectors in the master file. This is not a concern when all 2048-byte block indexes are at the start of the master file.

INITFILE first asks you for the filename of your file and whether it is an index or a data file. If there is no PARAM file entry for your file, **INITFILE** asks if you want to make a PARAM entry and requests the byte offset to your file, the master file containing it, the record size in bytes, and the last record number (maximum number of data records) in the file. The minimum record size is 10 bytes when you are using **INITFILE.**

For index files, **INITFILE** asks for the index block size, the number of bytes per key, the blocking factor percentage, and whether duplicate keys are allowed. Index blocks can be 512 bytes (AOS/VS, DG/RDOS, and UNIX systems) or 2048 bytes (AOS/VS and UNIX systems only) in length. Sectors are always 512 bytes long. Since there is only one index block size for DG/RDOS systems, **INITFILE** displays the message `INDEX BLOCK SIZE IS 512` instead of a question. To find the maximum number of blocks needed for an index file, use **INDEXCALC.** An error occurs if the number of index blocks exceeds 65534.

The blocking factor is the relative percentage of each index block that you want to fill with key entries (a larger percentage makes the index denser, and a smaller percentage leaves space for future entries). It is recommended that you use a 50 percent blocking factor. **KADD** uses a blocking factor of 50 percent.

To rebuild an existing index and change anything besides the key size, the blocking factor, or the duplicate keys flag, run **FM** on the PARAM file and change the entry for the file. You need to do this if you want to make the new index file larger than the old one.

Finally, **INITFILE** asks if you want to fill the data file with nulls. If you answer **Y,** **INITFILE** rebuilds record 0 and null fills the rest of the data file. If you answer **N,** **INITFILE** only rebuilds record 0. You should null fill reinitialized data files before using any of the index build utilities.

INITFILE *Continued*

Example

The following dialog appears when you enter **RUN "INITFILE**. The default responses are shown in brackets, and the user responses appear after the colon.

```
* RUN "INITFILE
INDEX(0), DATA(1), STOP(2) [0]: 0
SUB FILE NAME: FIRST
FILE NOT IN PARAM FILE!
DO YOU WISH TO ADD (Y OR N) [Y]: Y
MASTER FILE NAME: FIRST
INDEX BLOCK SIZE (512 OR 2048) [512]: 512
BYTE OFFSET TO SUB FILE: 0
MAXIMUM NUMBER OF INDEX BLOCKS: 8
BYTES PER KEY: 10
BLOCKING FACTOR ( % PERCENT) [50]: 50
DUPLICATE KEYS ALLOWED? (Y OR N) [N]: N
INDEX(0), DATA(1), STOP(2) [0]: 2
```

INITINDEX.SL

Subroutine

Initializes an index file (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 7700

Input Variables

Y$	An index file descriptor string (same one used with **KADD**).
X	The key length in bytes.
Y	A flag to allow duplicate keys: 1 Duplicates allowed. 0 No duplicates allowed.
Z	The index file blocking factor (percentage).
X1	The number of the last block; you can get this value from **INDEXCALC**. The maximum for X1 is 65534.
YY	A flag to specify index block size: 1 2048-byte blocks (AOS/VS and UNIX systems only). 0 512-byte blocks (AOS/VS, DG/RDOS, and UNIX systems). If YY is not specified, it defaults to 0 or a 512-byte block index.

Scratch Variables

T9$	A string to hold a block, must be dimensioned to at least 512 bytes.
QZQZ$	A temporary string (**INITINDEX.SL** dimensions this string).
Y1	Temporary variable.
Z1	Temporary variable.
YY0	Temporary variable.
YY1	Temporary variable.

Line Numbers

7700	Entry point to **INITINDEX.SL**.
7700-7790	**INITINDEX.SL** occupies these line numbers.

What It Does

INITINDEX.SL initializes an index file. **INITINDEX.SL** does an **INITFILE** for your index file (see **INITFILE**) by building block 0 and creating block 1. It does not, however, make a PARAM entry for you or initialize a data file.

INITINDEX.SL

Continued

INITINDEX.SL builds block 0 from input arguments that contain the file descriptor string, the key length, the blocking factor, and a duplicate key flag.

How To Use It

To use **INITINDEX.SL**, perform the following steps:

1. Add the statement **ENTER "INITINDEX.SL** into your program.
2. Open your index file and set up your input arguments. The descriptor string contains the byte offset to record 0 in the physical file for this index file, the channel number on which the index file is open, an automatic locking flag, and the logical filename of the index.
3. Enter the program statement **GOSUB 7700**.

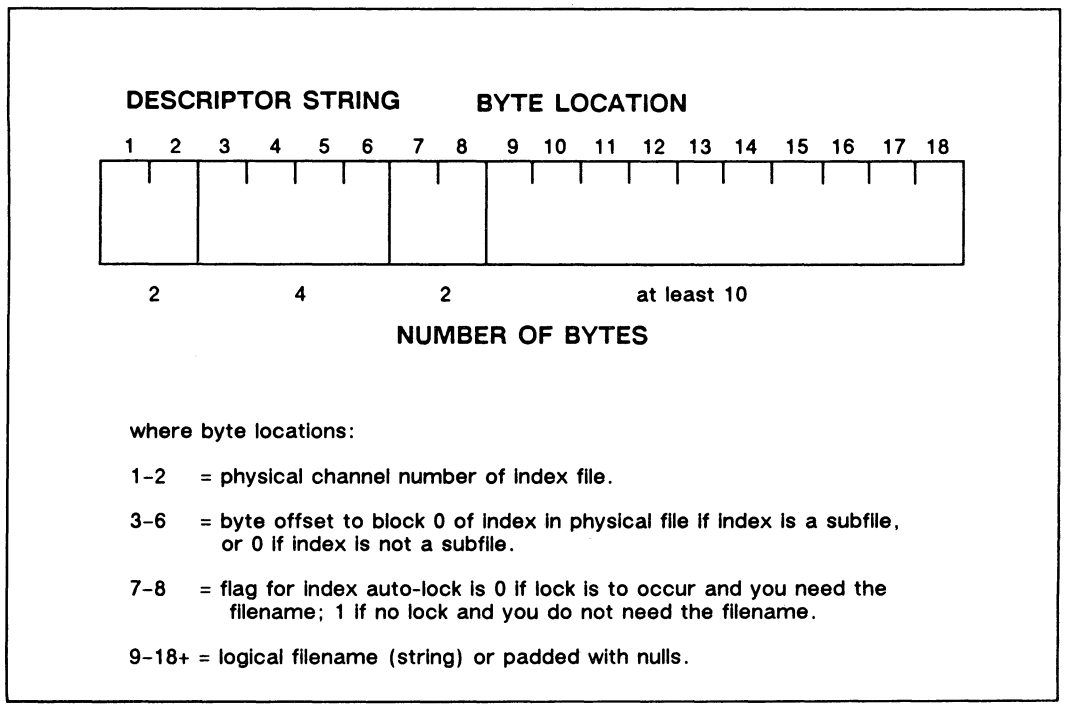


Figure 1-2. Descriptor String

INITINDEX.SL

Continued

Example

This program segment sets up the arguments for **INITINDEX.SL** and then calls the subroutine.

```
00010 DIM Y$[18],T9$[512].                    :Dimension the input variables.
.                                             :Code to open file goes here; Y$
.                                             :is descriptor string, C% is
.                                             :channel number.
00050 LET Y$=CHR$(C%,2),CHR$(0,4)             :Index file opened on C%, 0 offset.
00060 LET Y$(0)=CHR$(0,2),"INDX1",FILL$(0)          :INDX1 has auto lock.
00070 LET X=10                                :10 bytes per key.
00080 LET Y=1                                 :Allow duplicate keys.
00090 LET Z=50                                :50% blocking factor.
00100 LET X1=8                                :INDX1 has 8 blocks; INDEXCALC
                                              :returns this.
00110 GOSUB 07700                             :Go to INITINDEX.SL.
.
.
.
```

IREBLD *Utility*

Conditionally rebuilds an index file (logical file structure).

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "IREBLD

Or

!IREBLD [*indexname block-percentage key-percentage blocking-factor*]

Arguments

indexname	The name of an index file in the logical file database structure.
block-percentage	The maximum percentage of blocks in the index file that you would like to see in use.
key-percentage	At the lowest level in an index file, there is room to accommodate a certain number of keys depending on the number of bytes per entry. This argument represents the minimum percentage of that maximum number that you find acceptable.
blocking-factor	The percentage of each index block that you want to fill with key entries.

Global Switches

/A	Abort program execution on any error. By default the operation continues with the next index block if a nonfatal error occurs.
/L	Send to the default output queue a list of processed indexes and the resulting actions.
/N	Do not initialize empty indexes.
/V	Send to the terminal a list of processed indexes and the resulting actions.

What It Does

IREBLD conditionally rebuilds a 512-byte block index (AOS/VS, DG/RDOS, and UNIX systems) or a 2048-byte block index (AOS/VS and UNIX systems only). If you are not satisfied with the percentage of blocks in the file that are currently in use and the percentage of space at the lowest level that is currently filled with key entries, the utility rebuilds the index using the blocking factor you specify; otherwise, the program leaves the index alone.

When **IREBLD** rebuilds a file, it uses the **KFIND/KNEXT** loop in **IBUILD** to rebuild the index in place. It then uses a **BLOCK WRITE** to write the index to a temporary

IREBLD

Continued

index. Next, **IREBLD** overwrites the original index with the temporary index via a **READ/WRITE** loop.

To control program activity during an **IREBLD** session, the utility locks block 0 before processing begins and unlocks it after processing is completed. Once **IREBLD** is finished, the jobs that access the index can resume processing.

How To Use It

Execute **IREBLD** by entering **RUN**, **CHAIN**, or **SWAP "IREBLD** or **!IREBLD**. This starts the interactive mode for the utility. As the utility comes up, it displays information about your index file and then asks you if you want to rebuild the index. The most important statistics that **IREBLD** reports are `BLOCK 0 PERCENT LOADING`, which indicates the percentage of all the blocks in the file that are currently being used, and `PERCENT IN USE`, which is a percentage calculated by comparing the total number of keys at the lowest level with the maximum possible number of keys at that level. If you find that too many blocks are in use and that the percentage of keys is too low, type **Y** after the `REBUILD INDEX` prompt and then indicate the blocking factor you want **IREBLD** to use as it rebuilds the index.

If the index is empty (i.e., there are no keys at the lowest level), **IREBLD** initializes block 0 of the index. To specify that you do not want this block initialized, you can answer **N** at the `REBUILD INDEX` prompt if you are running the utility interactively, and you can use the **/N** switch if you are running **IREBLD** through the Business BASIC CLI and using the utility's four arguments.

To bypass the interactive mode, execute **IREBLD** in the Business BASIC CLI command line format and include all four arguments. If the value you supply with the *block-percentage* argument is less than the value **IREBLD** refers to as `BLOCK 0 PERCENT LOADING` and the value you supply with key-percentage is greater than the value that the utility refers to as `PERCENT IN USE`, **IREBLD** rebuilds your index file using the blocking factor you supplied on your command line.

Global switches can only be used when **IREBLD** is executed through the Business BASIC CLI. Append the switches either to **IREBLD** or another global switch. **IREBLD** should be the only job accessing an index file when you run it.

IREBLD

Continued

Example

The following illustrates the **IREBLD** dialog when the command is executed in interactive mode. Default responses are enclosed in brackets, and user responses follow the colons.

```
* RUN "IREBLD
INDEX FILE NAME [END] : INDX
BLOCK 0 PERCENT LOADING : 88.9%
TOTAL NUMBER OF KEYS  : 10
TOTAL NUMBER OF BLOCKS : 7
MAXIMUM NUMBER OF KEYS : 210
PERCENT IN USE  : 4.8%
EMPTY INITIAL SECTORS : 6
REBUILD INDEX (Y,N) [N] : Y
BLOCKING FACTOR [50] : 95
```

You could obtain similar results by executing **IREBLD** through the Business BASIC CLI with the command line:

```
* !IREBLD INDX 50 80 95
```

This command causes the utility to rebuild **INDX** if more than 50 percent of the file's blocks are in use and if the blocks at the lowest level are, on average, more than 80 percent full.

KILL *Utility*

Terminates Business BASIC and returns control to DG/RDOS.

DG/RDOS

What It Does

Since this utility can only be used on DG/RDOS systems by someone with system manager privileges, **KILL** is explained in the *Business BASIC System Manager's Guide*.

LFDATA.SL *Subroutine*

Gets the file description for a file (logical file structure).

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 9900

Input Variable

F%	The file number of a file opened using **LOPEN FILE**.

Output Variables

CHAN	The channel number of the database (.DB) file.
FLGS	The flag to indicate that automatic locking should occur.
LFILE$	The name of the logical file. Dimension LFILE$ to at least 10.
LNK	Reserved.
LSTREC	The last (highest) usable record number in logical file.
POSN	The position of the logical file in the database or .DB file (byte offset to record 0).
RECLEN	The record length in bytes of the records in the logical file.
TYP$	The type of logical file (D, I, or L for direct, index, or linked-available record). Dimension TYP$ to at least 1.

Line Numbers

9900	Entry point into **LFDATA.SL**.
9900-9915	**LFDATA.SL** occupies these lines.

What It Does

LFDATA.SL retrieves the file definition of a logical file from LFTABL$ and places the information in the output variables. LFTABL$ received that information when the file was opened with **LOPEN FILE**. Use **LFDATA.SL** with files created in the logical file database structure.

LFDATA.SL

Continued

The transfer of information from LFTABL$ to **LFDATA.SL** goes as follows:

Bytes in LFTABL$	Description	Variable in LFDATA.SL
2	Channel number	CHAN
4	Starting byte	POSN
2	Flags signaling automatic locking for type I and L files	FLGS
10	Name of the logical file	LFILE$
2	Record length in bytes	RECLEN
4	Last (highest) usable record number in the logical file	LSTREC
1	File type (I, L, or D)	TYP$
1	Reserved	LNK

How To Use It

To use **LFDATA.SL**, perform the following steps:

1. Add the statement **ENTER "LFDATA.SL** to your program.
2. Dimension the **LFDATA.SL** output variables within your program.
3. Include an **LOPEN FILE** statement to open the logical file and get its file number.
4. Assign the file number to the F% input variable.
5. Enter the program statement **GOSUB 9900**.

Example

Use the **LFU** utility to create the logical index file, **TMPI**. Then use **LFDATA.SL** to retrieve the file description for **TMPI**.

LFDATA.SL

Continued

```
00010 DIM CHAN[2],POSN[4],FLGS[2],LFILE$[10]    :DIM output variables
00020 DIM RECLEN[2],LSTREC[4],TYP$[1],LINK[1]   :used by LFDATA.SL.
00030 DIM LFTABL$[52],BUF$[512]                 :DIM LFTABL$ and buffer.
00040 LET LFTABL$=FILL$(0)                      :Null fill LFTABL$.
00050 LOPEN FILE[2,BUF$],"TMPI"                 :Open Logical file
00060 LET F%=2                                  :Store logical file
                                                :number of TMPI in F%.
00070 GOSUB 09900                               :Call LFDATA.SL.
00080 PRINT "CHANNEL NUMBER:";CHAN
00090 PRINT "STARTING BYTE:";POSN
00100 PRINT "LOCKING FLAG:";FLGS
00110 PRINT "LOGICAL FILE NAME: ";LFILE$
00120 PRINT "RECORD LENGTH IN BYTES:";RECLEN
00130 PRINT "LAST USABLE RECORD NUMBER:";LSTREC
00140 PRINT "FILE TYPE: ";TYP$
00150 END

* ENTER "LFDATA.SL
* SAVE "TEST
* RUN
CHANNEL NUMBER: 0
STARTING BYTE: 52224
LOCKING FLAG: 0
LOGICAL FILE NAME: TMPI
RECORD LENGTH IN BYTES: 512
LAST USABLE RECORD NUMBER: 31
FILE TYPE: I

*
```

LFM *Utility*

Provides file maintenance functions for the logical file structure.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "LFM

What It Does

The Logical File Maintenance (**LFM**) utility maintains files converted from the PARAM file database structure to the logical file database structure by the **PARAMCON** utility. Use **LFM** to keep the File Maintenance (**FM**) table file for these data files. **LFM** assumes that your files conformed to the logical file database structure naming conventions when you ran **PARAMCON** (see **PARAMCON**).

How To Use It

Execute **LFM** by entering **RUN**, **CHAIN**, or **SWAP "LFM**. This begins the **LFM** dialog.

The utility prompts you for the name of your file. To use **LFM** with a data file, enter the name of the data file. To use **LFM** on the table file, enter **FM.TB** for the `FILENAME:` prompt, and enter the table filename (with a .TB extension) that associated with your database file for the `DATA FILE NAME` prompt.

Example

This example runs **LFM** on the data file **EMP** from Business BASIC keyboard mode. The utility then displays the screen that was defined using **FM** for the **EMP** database before it was converted to the logical file database structure.

```
* RUN "LFM

DATA FILE MAINTENANCE     REV. X.XX
FILENAME: EMP

EMPLOYEE MAINTENANCE                                              [     ]
EMPLOYEE RECORD #__________ LAST NAME__________ FIRST NAME__________
LAST NAME_____________ FIRST NAME_____________ TELEPHONE EXT____
IDENTIFICATION NUMBER 2 _____
```

LFU *Utility*

Creates and manipulates files (logical file structure).

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "LFU

Or

!LFU [*command* [*parameter* ...]]

Arguments

command	An **LFU** command.
parameter	One or more arguments associated with the **LFU** command you are using.

What It Does

The Logical File Utility (**LFU**) manipulates files in a logical file database structure. Use **LFU** and its commands to create a logical file database structure and to work with the files associated with this structure.

Business BASIC lets you implement logical files as subsections of physical files. The logical file database structure is a file set that includes a volume label file (indicated by an .VL extension) and a database file (indicated by a .DB extension). Logical files are implemented as links that point to the volume label file. The volume label file contains an entry with the information specified when the logical file was created and maps the logical file to the database file that contains the actual data. Thus, the simple logical file database called **CUST** consists of the physical files **CUST.DB** and **CUST.VL**. The database file **CUST.DB** consists of the three logical files: the data file **CUST**, and the index files **CUSTI1**, and **CUSTI2**. The volume label file **CUST.VL** contains information about each logical file.

How To Use It

Execute **LFU** by entering **RUN**, **CHAIN**, or **SWAP "LFU**. You can use **LFU** interactively or supply all required parameters in a command line and let the Business BASIC CLI drive **LFU**. With the command line format, **LFU** only goes into interactive mode if you do not enter all the required parameters. An **LFU** command parameter list is order sensitive, so missing parameters can cause a misinterpretation of later commands in the list.

With index files, run **INDEXCALC** first to get the information required by **LFU**.

The **LFU** commands are listed in Table 1-13 and are explained in more detail later in this section.

LFU

Continued

Table 1-13. LFU Command Summary

Command	Function
LCREATE	Creates a logical file of type D, I, or L.
LDELETE	Deletes a logical file.
LINIT	Initializes a type D, I, or L 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**.

Except for **STOP**, each **LFU** command can be used with one or more arguments. These arguments make up the parameters for the commands. Since some of the commands use the same arguments, a common argument table precedes the discussion of individual commands.

LFU

Continued

Arguments

D	Type D (direct random data file). The user handles all record assignments.
I	Type I (index file). These are maintained via the ISAM statements and utilities.
L	Type L (linked-available-record file). Business BASIC dynamically makes the record allocation assignments.
block-factor	The index file's blocking factor.
block-length	The size of the index block; either 512 bytes (AOS/VS, DG/RDOS, and UNIX systems) or 2048 bytes (AOS/VS and UNIX systems only).
duplicate-keys	Y or N indicating whether duplicate keys are allowed.
fill	Y or N indicating whether to null fill the file.
key-length	The key length in bytes.
last-block	The last usable block number.
last-record	The last usable record number.
logical-file	Name of a logical file without an extension. The name can be up to 10 characters long. For data dictionary files, the name can only be up to six characters long.
new-logicalfile	The new name you want for the logical file.
new-dbname	The new name you want for the database and volume label files.
old-logicalfile	The current name of the logical file.
old-dbname	The current name of the database and volume label files.
dbname	The names of the physical files that are associated with logical file database set. *dbname*.DB contains the logical database files, and *dbname*.VL contains the volume label information for each logical file.
record-length	The length of the records in bytes.
size	The number of blocks (sectors) in the database file that you want to create.

LFU

Continued

LCREATE

Direct Format

!LFU LCREATE *logical-file dbname* **D** *record-length last-record* [*fill*]

Index Format

!LFU LCREATE *logical-file dbname* **I** *block-length last-block key-length block-factor duplicate-keys*

Linked Format

!LFU LCREATE *logical-file dbname* **L** *record-length last-record* [*fill*]

What It Does

LCREATE creates a logical file of type D (direct), I (index), or L (linked-available-record). The argument *logical-file* is the name given to the file. The *dbname* argument must be an existing physical database file (see **LFU PCREATE**). The remaining parameters differ depending on the file type (D, I, or L).

You must provide the record length for direct or linked-available-record data files.

For index files, the index block length can be set to 512 bytes (AOS/VS, DG/RDOS, and UNIX systems) or 2048 bytes (AOS/VS and UNIX systems only). With direct or linked-available-record files, you provide the last usable record number (last-record). For index files, this is the last usable block number (last-block). You receive an error message if the last block exceeds 65534.

Direct and linked-available-record files require a **Y** or **N** for *fill* to indicate whether the file should be null filled. However, type L files with a record length less than 10 bytes always fill, so no *fill* argument is necessary. If you answer **Y** to null fill the file, then the space for the data file is allocated on disk.

Index files require the key length and the blocking factor as a percent to be used for repacking an index. Also, index files require a **Y** or **N** for *duplicate-keys* to indicate whether duplicate keys are allowed.

NOTE: All 2048-byte block indexes must start on a 2048-block boundary, which is a multiple of four sectors in the database file. If the next available sector in the database file is not a multiple of four, **LFU** automatically starts the index at the next 2048-byte boundary when the index is created with **LCREATE**. Therefore, you could waste up to three sectors in the database file every time a 2048-byte block index file follows a data file. This never happens when all

LFU

Continued

2048-byte block indexes are at the start of the database file. It happens only once if all 2048-byte block indexes are grouped together elsewhere in the database file. This factor must be taken into consideration when determining the size of the database file for **LFU PCREATE**.

Examples

1. Create a logical file named **CUSTL** in the database file named **CUST.DB** (.DB is implied) of type L with 128-byte records with 100 as the last usable record number and no null fill of the file.

   !LFU LCREATE CUSTL CUST L 128 100 N

2. Create a logical file of type I named **CUSTI1** in the database file **CUST**. Because parameters are missing, **LFU** starts an interactive dialog and asks for the index block size, the last usable block number, the blocking factor, and whether duplicate keys are allowed. The last two questions receive the default responses, indicated when the user presses CR or New Line. **LFU** loops in interactive mode; therefore, you must enter the command STOP to leave **LFU**.

```
!LFU LCREATE CUSTI1 CUST I
BLOCK SIZE (512 OR 2048 BYTES): 512
LAST BLOCK NUMBER: 10
BYTES PER KEY [0]: 4
REBUILD BLOCKING FACTOR (PERCENT) [50]:
DUPLICATE KEYS ALLOWED (Y,N) [N]:

COMMAND: STOP
```

LFU

Continued

LDELETE

Format

!LFU LDELETE *logical-file*

What It Does

LDELETE deletes a logical file by removing the link for the logical file's name from your directory and placing the name ***DEL** in the name field of the .VL file entry. **LFU** reuses this space if another logical file of the same size is created.

Example

The logical file **CUSTI2** is removed from your directory.

```
!LFU LDELETE CUSTI2
```

LFU

Continued

LINIT

Direct or Linked Format

!LFU LINIT *logical-file* [*fill*]

Index Format

!LFU LINIT *logical-file key-length block-factor duplicate-keys*

What It Does

LINIT initializes an existing logical file of type D (direct), I (index), or L (linked). The type is determined automatically. The arguments differ according to the file type.

Direct and linked files require a **Y** or **N** for *fill* to indicate whether the file should be null filled. Type L files with a record length less than 10 bytes always fill, so no *fill* argument is necessary. If you answer **Y**, **LFU** rebuilds record 0 (type L files only) and null fills the rest of the data file. If you answer **N**, **LFU** only rebuilds record 0 (type L files only). Note that you should null fill reinitialized data files before using any of the index build utilities.

Index files require a key length and a blocking factor (shown as a percent) to be used for repacking the index. Index files also require a **Y** or **N** for *duplicate-keys* to indicate whether duplicate keys are allowed. These are the same parameters required by **LCREATE**, since **LCREATE** performs an **LINIT** of the file after its creation.

Examples

1. Use **LFU LINIT** to initialize the logical file **CUSTL** as a type L file. Since *fill* was not supplied, **LFU** prompts you for that information.

   ```
   !LFU LINIT CUSTL
   NULL FILL FILE (Y,N) [N]:
   ```

2. Initialize the logical, type I file **CUSTI3** to have a four-byte key with a 50 percent blocking factor and no duplicate keys.

   ```
   !LFU LINIT CUSTI3 4 50 N
   ```

LFU

Continued

LLIST

Format

!LFU LLIST *logical-file*

What It Does

LLIST displays the type, location in the database file, size in blocks (sectors), record length, last valid record number, and size in bytes of a logical file.

Example

LLIST displays the following information about the logical file **CUSTL**.

!LFU LLIST CUSTL

```
DB FILE: CUST
      FILE      FILE     STARTING   # OF     RECORD     LAST      # OF
      NAME      TYPE      SECTOR   SECTORS   LENGTH    RECORD    BYTES
     CUSTL        L          0        26       128       100      12928
```

LFU

Continued

LRENAME

Format

!LFU LRENAME *old-logicalfile new-logicalfile*

What It Does

LRENAME changes the name of the logical file, *old-logicalfile*, to *new-logicalfile*. This is accomplished by removing the old link, creating a new link, and altering the name appearing in the volume label file of the database.

Example

The logical file **CUSTI3** in the database **CUST** is renamed **CALPHA**. This example displays the information on the database **CUST** before **CUSTI3** is renamed and after it is renamed.

!LFU PLIST CUSTL

```
DB FILE: CUST
       FILE      FILE     STARTING   # OF     RECORD     LAST       # OF
       NAME      TYPE     SECTOR   SECTORS   LENGTH   RECORD     BYTES
      CUSTL        L          0       26       128      100       12928
     CUSTI1        I         26       11       512       10        5632
     CUSTI2        I         37       11       512       10        5632
     CUSTI3        I         48       11       512       10        5632
                                     -----                         ------
            TOTAL SECTORS:           59              BYTES:       29824
```

!LFU LRENAME CUSTI3 CALPHA

!LFU PLIST CUST

```
DB FILE: CUST
       FILE      FILE     STARTING   # OF     RECORD     LAST       # OF
       NAME      TYPE     SECTOR   SECTORS   LENGTH   RECORD     BYTES
      CUSTL        L          0       26       128      100       12928
     CUSTI1        I         26       11       512       10        5632
     CUSTI2        I         37       11       512       10        5632
     CALPHA        I         48       11       512       10        5632
                                     -----                        --------
            TOTAL SECTORS:           59              BYTES:       29824
```

LFU

Continued

PCREATE

Format

!LFU PCREATE *dbname* [*size*]

What It Does

PCREATE creates the physical files associated with a database file set. **PCREATE** sets up the *dbname*.DB file, which will contain the logical database file, and the *dbname*.VL file, which will contain the volume label information for each logical file. If you omit the *size* argument or give it a value of 0, then **PCREATE** sets up *dbname*.DB as a random file. When *size* is greater than 0, the *dbname*.DB file is created contiguously with the number of sectors indicated in *size*. On AOS/VS and UNIX systems, you can specify up to 65532 blocks for the size argument when you are creating a contiguous database file; on DG/RDOS, you can specify up to 65536 blocks for the size argument.

NOTE: On AOS/VS systems, if the default element size for your system is greater than the value you supply for the *size* argument, your system uses the default element size when it creates the file.

For more information on how to compute the number of sectors for a database file, see the general **LFU** examples.

Examples

1. Create a random database file set (.DB and .VL files) named **DEMO**.

```
!LFU PCREATE DEMO 0
```

2. Create a 1000-sector contiguous database file set named **CUST**.

```
!LFU PCREATE CUST 1000
```

LFU

Continued

PDELETE

Format

!LFU PDELETE *dbname*

What It Does

PDELETE deletes both the database and the volume label files in a database file set. **PDELETE** unlinks all of the logical names pointing to the volume label of the database.

Example

Delete all of the components of database **DEMO**.

```
!LFU PDELETE DEMO
```

LFU

Continued

PLIST

Format

!LFU PLIST *dbname*

What It Does

PLIST displays tabular information about a database file set. It shows the information contained within the volume label file—all the logical files in the database and each file's type, location, size, record length, and last valid record number.

Example

PLIST displays the volume label file information on the logical files within the database file **CUST**.

```
!LFU PLIST CUSTL

DB FILE: CUST
      FILE      FILE     STARTING   # OF     RECORD     LAST      # OF
      NAME      TYPE     SECTOR   SECTORS   LENGTH   RECORD     BYTES
     CUSTL        L          0       26       128      100       12928
    CUSTI1        I         26       11       512       10        5632
    CUSTI2        I         37       11       512       10        5632
    CUSTI3        I         48       11       512       10        5632
                                    -----                        ------
          TOTAL SECTORS:              59              BYTES:     29824
```

LFU

Continued

PRENAME

Format

!LFU PRENAME *old-dbname new-dbname*

What It Does

PRENAME changes the name of the physical files that compose a database file set. **PRENAME** renames the database and volume label files and relinks all of the logical files pointing to the volume label file. It changes the names from *old-dbname* to *new-dbname*.

Example

PRENAME renames the database file set **CUST** to **CUSTOMER**.

!LFU PLIST CUSTL

```
DB FILE: CUST
       FILE      FILE     STARTING    # OF      RECORD      LAST      # OF
       NAME      TYPE     SECTOR    SECTORS     LENGTH    RECORD     BYTES
      CUSTL        L          0        26         128       100      12928
     CUSTI1        I         26        11         512        10       5632
     CUSTI2        I         37        11         512        10       5632
     CUSTI3        I         48        11         512        10       5632
                                      -----                          ------
            TOTAL SECTORS:            59                 BYTES:     29824
```

!LFU PRENAME CUST CUSTOMER

!LFU PLIST CUSTOMER

```
DB FILE: CUST
       FILE      FILE     STARTING    # OF      RECORD      LAST      # OF
       NAME      TYPE     SECTOR    SECTORS     LENGTH    RECORD     BYTES
      CUSTL        L          0        26         128       100      12928
     CUSTI1        I         26        11         512        10       5632
     CUSTI2        I         37        11         512        10       5632
     CUSTI3        I         48        11         512        10       5632
                                      -----                          ------
            TOTAL SECTORS:            59                 BYTES:     29824
```

LFU

Continued

STOP

Format

STOP

What It Does

STOP terminates **LFU** when it is in interactive mode.

STOP is valid as the last item in an **LFU** command line issued from the CLI, but it is not necessary.

Example

Since **LFU** is executed without parameters, the utility enters interactive mode and prompts you for the missing arguments. The default answers are shown in brackets. After receiving all the answers, **LFU** returns to the COMMAND question. Enter **STOP** to halt **LFU**.

```
!LFU
COMMAND: LINIT
LOGICAL FILE NAME: CUSTI3
BYTES PER KEY [0]: 4
REBUILD BLOCKING FACTOR (PERCENT) [50]:
DUPLICATE KEYS ALLOWED (Y,N) [N]: Y


COMMAND: STOP
```

LFU

Continued

LFU Examples

1. This example uses **LFU** to create a small logical file database called **NAMEDB**. The database contains one logical data file **NAME**, which can hold 100 names and identification numbers. The database has two index files: **ALPHAX** and **IDNDX**. **ALPHAX** contains six-byte alphabetic keys. **IDNDX** contains two-byte numeric keys. Since the two keys have different lengths, **INDEXCALC** must be run twice.

   Run **INDEXCALC** to determine the size needed for the logical files **ALPHAX**, **IDNDX**, and **NAME**.

```
* !INDEXCALC

BYTES PER KEY: 2
BYTES PER DATA RECORD: 24
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

84 MAXIMUM KEYS PER INDEX BLOCK
42 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
3 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
5 BLOCKS (512 BYTES EACH) IN INDEX
5 SECTORS IN INDEX
5 SECTORS IN DATA FILE

CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE? (Y OR N) [N]: Y

BYTES PER KEY: 6
BYTES PER DATA RECORD: 24
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]: Y

50 MAXIMUM KEYS PER INDEX BLOCK
25 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
5 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
7 BLOCKS (512 BYTES EACH) IN INDEX
7 SECTORS IN INDEX
5 SECTORS IN DATA FILE

CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE? (Y OR N) [N]: N

*
```

LFU

Continued

Once **INDEXCALC** finishes its calculations, use **LFU PCREATE** to create the database file **NAMEDB** as a contiguous file with enough room for the logical files **NAME**, **ALPHAX**, and **IDNDX**. Then use **LFU LCREATE** to create the three logical files. The **INDEXCALC** data tells you that the files **NAME** and **IDNDX** are each five sectors long and **ALPHAX** is seven sectors long, bringing the total sectors for the three logical files to 17. Enter 17 at the maximum file/element size prompt; since this number must be a multiple of four, **LFU** will round it up to 20 sectors. In setting up blocks for the index files, remember that **LFU** starts numbering with zero. Thus, enter the number 4 for **IDNDX** (0 - 4), which has five 512-byte blocks, and the number 6 for **ALPHAX** (0 - 6), which has seven 512-byte blocks.

```
* RUN "LFU
LFU   REV X.XX

Command: PCREATE
Physical file name: NAMEDB
Maximum file/element size (or 0 for random file): 17

Command: LCREATE
Logical file name: NAME
DB file name: NAMEDB
Type (I=index, D=direct, L=linked): L
Record length (bytes): 24
Last record number: 100
Null fill file (Y,N) [N]: <CR>

Command: LCREATE
Logical file name: IDNDX
DB file name: NAMEDB
Type (I=index, D=direct, L=linked): I
Block size (512 or 2048 bytes): 512
Last block number: 4
Bytes per key [0]: 2
Rebuild blocking factor (percent) [50]: <CR>
Duplicate keys allowed (Y,N) [N]: <CR>

Command: LCREATE
Logical file name: ALPHAX
DB file name: NAMEDB
Type (I=index, D=direct, L=linked): I
Block size (512 or 2048 bytes): 512
Last block number: 6
Bytes per key [0]: 6
Rebuild blocking factor (percent) [50]:
Duplicate keys allowed (Y,N) [N]: Y

Command: STOP
```

LFU

Continued

When you have set up your database, you can use **LFU PLIST** to display the volume label information on the files.

```
* !LFU PLIST NAMEDB

DB FILE: NAMEDB
FILE      FILE    STARTING      # OF      RECORD    LAST   # OF
NAME      TYPE    SECTOR        SECTORS   LENGTH    RECORD BYTES

NAME      L       0             5         24        100    2424
IDNDX     I       5             5         512       4      2560
ALPHAX    I       10            7         512       6      3584
                                ------                     -----
          TOTAL SECTORS:        17        TOTAL BYTES:     8568
```

2. This AOS/VS example uses the **LFU** command line to set up a logical file database structure with 2048-byte block indexes. (Only AOS/VS and UNIX systems support 2048-byte blocks.) The names are the same as those used in the first example so you can see the difference between using a 512-byte index and a 2048-byte index.

   Your responses to the **INDEXCALC** prompts are the same as the ones in the first example, except you make the index size 2048.

```
* !INDEXCALC

BYTES PER KEY: 2
BYTES PER DATA RECORD: 24
MAXIMUM NUMBER OF DATA RECORDS: 100
INDEX BLOCKING FACTOR ( % PERCENT) [50]: 50
INDEX BLOCK SIZE (512 OR 2048) [512]: 2048
DUPLICATE KEYS ALLOWED? (Y OR N) [N]: N


240 MAXIMUM KEYS PER INDEX BLOCK
170 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
1 BLOCK(S) AT LEVEL 0
2 BLOCKS (2048 BYTES EACH) IN INDEX
8 SECTORS IN INDEX
5 SECTORS IN DATA FILE


CALCULATE THE INDEX INFORMATION FOR ANOTHER FILE? (Y OR N) [N]: Y


BYTES PER KEY: 6
BYTES PER DATA RECORD: 24
```

LFU *Continued*

```
MAXIMUM NUMBER OF DATA RECORDS: 100
INDEX BLOCKING FACTOR ( % PERCENT) [50]: 50
INDEX BLOCK SIZE (512 OR 2048) [512]: 2048
DUPLICATE KEYS ALLOWED? (Y OR N) [N]: Y


170 MAXIMUM KEYS PER INDEX BLOCK
85 KEYS PER BLOCK WITH A 50 PERCENT BLOCKING FACTOR
2 BLOCK(S) AT LEVEL 1
1 BLOCK(S) AT LEVEL 0
4 BLOCKS (2048 BYTES EACH) IN INDEX
16 SECTORS IN INDEX
5 SECTORS IN DATA FILE


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

This time the file **NAME** is five sectors long, **IDNDX** is eight sectors long, and **ALPHAX** is 16 sectors long, bringing the total sectors for the three logical files to 29 sectors. However, since **IDNDX** is a 2048-byte block index, it must start on a sector boundary that is a multiple of four. If you use **LCREATE** to create the linked-available-record file **NAME** first, the three sectors it takes to reach a boundary that is a multiple of four are wasted—from block five (where **NAME** ends) to block eight (where **IDNDX** starts). This puts the total size of the physical file at 32 sectors. You can avoid wasting the sectors by executing an **LCREATE** command on your indexes first and then on your data file when you are using 2048-byte block indexes. (See the **LFU LCREATE** command for more information.)

Use the command line for **PCREATE** to create the database and volume label files for **NAMEDB**. Since this example maintains the creation order of example one and uses **LCREATE** to create the data file first, **NAMEDB** requires 32 sectors.

```
* !LFU PCREATE NAMEDB 32
```

The following command line creates the logical file **NAME** within the database file **NAMEDB** as a type L file with a record length of 24 bytes and a last usable record number of 100. The new file is not filled with nulls.

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

This command line creates the file **IDNDX** within the database file **NAMEDB** as an index file using 2048-byte blocks. The last usable block in the file is 1 in an index with two calculated 2048-byte blocks (0-1). **IDNDX** has keys that are two bytes long and a blocking factor of 50 percent. It does not allow duplicate keys.

```
* !LFU LCREATE IDNDX NAMEDB I 2048 1 2 50 N
```

LFU

Continued

The final command line creates the file ALPHAX within the database file **NAMEDB** as an index file using 2048-byte blocks. The last usable block in the file is 3 in an index with four calculated 2048-byte blocks (0-3). **ALPHAX** has keys that are six bytes long and a blocking factor of 50 percent. It allows duplicate keys.

```
* !LFU LCREATE ALPHAX NAMEDB I 2048 3 6 50 Y
```

Use the **PLIST** command to display the volume label information on the files within the database **NAMEDB**.

```
* !LFU PLIST NAMEDB

DB file: NAMEDB
       File      File     Starting   # of     Record     Last      # of
       Name      Type      Sector   Sectors   Length    Record    Bytes
       NAME        L          0        5        24        100      2424
       *DEL                   5        3       512          2      1536
      IDNDX        I          8        8      2048          1      4096
     ALPHAX        I         16       16      2048          3      8192
                                     -----                        ------
                     Total Sectors: 32                  Bytes:    16248
```

The ***DEL** shows the sectors that were skipped so that the first index could start on a boundary that is a multiple of four. These wasted sectors can be used if another logical file of the same size is created.

LIBRARY *Utility*

Displays the names of files in the library directory.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "LIBRARY

What It Does

LIBRARY gives you a quick listing of all nonpermanent files in the Business BASIC library directory (**$LIB.DR** for DG/RDOS and **$SYSLIB** for AOS/VS).

How To Use It

Execute **LIBRARY** by entering **RUN**, **CHAIN**, or **SWAP "LIBRARY**.

On AOS/VS systems, you need **$SYSLIB** and the directory where it resides on your search list.

Example

Once run, **LIBRARY** displays information about the Business BASIC system on this AOS/VS (**$SYSLIB**) system.

```
* RUN "LIBRARY

:UTIL:BBASIC:$SYSLIB                                  7/06/91   8:18:58

      BLACKJACK.SV        FM.TB          FORMIO.SL          NEWS.DB
      DATA.TB             PARAM.TB       CUST.TB            CUST
      EDIT                FILES          FILESORT           INITINDEX.SL
  .
  .
  .
Listed space = 1417 sectors, 725504 bytes
 Total space = 1225 sectors, 725504 bytes
```

LINDEXBLD *Utility*

Builds or rebuilds an index file (logical file structure).

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "LINDEXBLD

What It Does

Use **LINDEXBLD** to build or rebuild an index file for a logical file database structure. (For PARAM files, use **INDEXBLD.**) Back up your files before using this utility.

LINDEXBLD operates interactively, prompting you for information on the input file and the index file.

When the input file is an index file, **LINDEXBLD** assumes you want to rebuild the index. It reads the source index sequentially into a scratch file. Then it initializes the index file and rebuilds it from the scratch file with the specified blocking factor.

If the input file is a linked-available-record data file, **LINDEXBLD** prompts you on whether to use **IBUILD** or **KADD** (**XBUILD**) to build the index file. With **IBUILD**, **LINDEXBLD** calls **TBUILD** to build a tag file and then calls **QFILESORT** to sort the tag file. When **KADD** is selected, **LINDEXBLD** uses a blocking factor of 50%.

The input file can be a sorted tag file with the format of the intermediate file built by **TBUILD**. In this case, **LINDEXBLD** does not generate an intermediate file for sorting.

Anytime **LINDEXBLD** encounters an error, the utility returns an error message and fills block 0 of the output index file with nulls. If you are building in place (i.e., both the input file and the output file are the same), this action corrupts your original index. To recover you must have a backup copy of your index. When building in place, **LINDEXBLD** creates a tag file using **TBUILD**. If the system crashes after **LINDEXBLD** has displayed the message `TEMP FILE BUILD COMPLETED... BUILDING INDEX`, the tag file (**TAG-.TM**) may exist on the disk. Then you can use it as the input file to rebuild the index.

How To Use It

Execute **LINDEXBLD** by entering **RUN**, **CHAIN**, or **SWAP "LINDEXBLD**. This starts the **LINDEXBLD** dialog.

LINDEXBLD *Continued*

The program asks for the name of the input file and whether it is an index, data, or sorted tag file. If you specify a data file, you must choose between using **KADD** or **IBUILD**. Use **IBUILD** if you need a specific blocking factor (**KADD** uses a blocking factor of 50 percent) or if the input file is sorted.

LINDEXBLD then attempts to open the specified file using **LOPEN FILE**.

When a nondatabase index file is the input file, the system asks you for the byte offset to record 0. With a nondatabase data file, the system asks you the following questions:

```
BYTE OFFSET TO RECORD 0 [0]:
RECORD SIZE IN BYTES:
LAST RECORD NUMBER IN INPUT FILE:
```

With data files, the system asks whether to check for deleted records. This tells the program where the data begins and what format is expected. When the check is specified, deleted records are omitted from the intermediate file; otherwise, deleted records are defined as containing nulls in the first two bytes (status word) of the record.

To describe the key entry, the program asks for the number of key fields, the key field location, and the field length. **LINDEXBLD** repeats the key location and length questions for each field specified. The utility does not pad individual fields, but it does pad the total key length when necessary to make the length an even number of bytes. The first byte of the record is numbered 1 rather than 0.

If you use a data file as the input file, the key field must be in the same location in each record of the data file.

Then the system asks for the name of the output index file. If the file is a nondatabase file, the system asks the following questions:

```
BYTE OFFSET TO RECORD 0 [0]:
INDEX BLOCK SIZE (512 OR 2048):
LAST INDEX BLOCK NUMBER:
```

To determine the organization of the index, the system asks for the blocking factor and if duplicate keys are allowed. The blocking factor you specify overrides the blocking factor provided in the header block of previously created index files.

Since only AOS/VS and UNIX systems support 512- and 2048-byte block indexes, **LINDEXBLD** under DG/RDOS prints `INDEX BLOCK SIZE IS 512` instead of a size query.

LINDEXBLD *Continued*

Example

The following is an example of the **LINDEXBLD** dialog on an AOS/VS system. The default responses are shown in brackets.

```
* RUN "LINDEXBLD
LOGICAL INDEXBLD


INPUT FILE NAME: CUST
INPUT FILE IS INDEX(0), DATA(1), OR SORTED TAG(2) [0]: 1
USE IBUILD(0) OR KADD(1) [0]: 0
CHECK FOR DELETED RECORDS; YES(1),NO(0) [0]: 0
TOTAL KEY LENGTH IN BYTES: 20
NUMBER OF FIELDS IN KEY [1]: 1
KEY FIELD 1 LOCATION IN DATA RECORD (BYTE 1-N): 3
FIELD LENGTH IN BYTES: 20
INDEX FILE NAME: NEWDEX
BYTE OFFSET TO RECORD 0 [0]: 0
INDEX BLOCK SIZE (512 OR 2048): 512
LAST INDEX BLOCK NUMBER: 2
BLOCKING FACTOR ( %) [50]: 50
DUPLICATE KEYS ALLOWED: YES(1),NO(0) [0]: 0
BUILDING TEMPORARY FILE
TEMP FILE BUILD COMPLETE... BUILDING INDEX


*
```

LINITINDEX.SL *Subroutine*

Initializes an index file that was opened with the LOPEN FILE statement.

AOS/VS	DG/RDOS	UNIX

Format

GOSUB 7700

Input Variables

F%	The file number of an index file (type I) opened using the **LOPEN FILE** statement
X	The key length in bytes
Y	A flag to allow duplicate keys: 1 Duplicate keys 0 Don't allow duplicates
Z	The percentage of the file blocking factor for the index
YY	A flag to specify index block size: 1 2048-byte blocks (AOS/VS and UNIX systems only) 0 512-byte blocks (AOS/VS, DG/RDOS, and UNIX systems) If YY is not specified, it defaults to 0 or a 512-byte block index.

Scratch Variables

T9$	A string to hold a block. It must be dimensioned to the size specified in YY.
X1	Temporary variable
Y1	Temporary variable
Z1	Temporary variable
YY0	Temporary variable
YY1	Temporary variable

Line Numbers

7700	Entry point to **LINITINDEX.SL**
7700-7790	**LINITINDEX.SL** occupies these line numbers.

What It Does

LINITINDEX.SL initializes a logical index file that was not created with the **LFU** utility. This subroutine cannot initialize a data file.

LINITINDEX.SL is similar to **INITINDEX.SL**, except that **LINITINDEX.SL** is used to initialize index files in the logical file database structure while **INITINDEX.SL** initializes index files in the PARAM file database structure.

LINITINDEX.SL

Continued

LINITINDEX.SL builds block 0 of an index file from input arguments specifying the key length, the blocking factor, and whether duplicate keys are allowed. Block 0 is a header block describing the logical index file.

How To Use It

To use **LINITINDEX.SL**, perform the following steps:

1. Add the statement **ENTER "LINITINDEX.SL** to your program.
2. Dimension the string variable LFTABL$ and null fill it to a current length of at least 26 times the highest logical file number.
3. Use the **LOPEN FILE** command to get file number of the index file for the F%input variable. Set up the other input variables.
4. Include the program statement **GOSUB 7700**, which calls **LINITINDEX.SL**.

When using the **LOPEN FILE** command on an index file, you must specify a last record number less than or equal to 65535. Block 0 allows only two bytes to describe the number of bytes per entry, so the default last record number of 16777215 is too large and causes an error.

Example

This example creates the database file set **EMPFIL** with the **LFU** command **PCREATE** and the logical file **EMPDTA** with the **LFU** command **LCREATE**. The program then calls **LINITINDEX.SL** to initialize the index file **TMP**.

```
* !LFU PCREATE EMPFIL 0
* !LFU LCREATE EMPDTA EMPFIL L 85 500 N
* LIST
00010 DIM T9$[512],LFTABL$[52]          :Dimension required strings.
                                        :LFTABL$ is dimensioned to 26*2
                                        :because highest logical file
                                        :number for this program is 2.
00020 LET LFTABL$=FILL$[0]              :Initialize logical file table.
00050 LOPEN FILE[1,T9$],"EMPDTA"        :Open employee data file.
00060 OPEN FILE[15,0],"TMP"             :Use OPEN to open temporary
                                        :file for index.
00070 LOPEN FILE[2,15],"TMP","I",512,65000:Open TMP as a logical
                                        :index file.
00080 LET F%=2                          :Logical file to be initialized
                                        :has logical file number 2.
00090 LET X=8                           :Record key length of 8 bytes.
00100 LET Y=1                           :Duplicate keys allowed.
00110 LET Z=50                          :Blocking factor of 50 %.
00120 GOSUB 07700           :Call LINITINDEX.SL to initialize
                            :TMP; index is now ready to use.
                            :Use KADD, KNEXT, etc. as necessary.
```

LINK

BASIC CLI Command

Assigns an alternate name to a file.

AOS/VS	DG/RDOS	UNIX

Format

!LINK *linkname filename*

Arguments

linkname	An alias for a file
filename	The file that *linkname* points to (the resolution file). It can have a directory specifier and it can be another link entry. In AOS/VS and UNIX, *filename* can be a pathname.

What It Does

LINK establishes an alias entry in the current directory to a resolution file in the same directory or another directory.

No error message is given if the system cannot find the resolution file.

How to Use It

Execute the command by entering **LINK** from the Business BASIC CLI. The arguments are required.

On AOS/VS and UNIX systems, the resolution filename can be a complete pathname.

In DG/RDOS, initialize all directories involved in the resolution chain before using a link to access a resolution file. You can link to a file whose attributes forbid linking (the DG/RDOS **CHATR** and **CHLAT** commands), but you can't use that link.

Use **UNLINK** with *linkname* to delete the link entry without affecting the resolution file.

Examples

1. A link entry, **BLACKJACK**, is made in the current directory (AOS/VS) for **CARDS.SV**, which is also in the current directory.

```
!LINK BLACKJACK CARDS.SV
!LIST BLACKJACK

BLACKJACK        LNK  CARDS.SV
CARDS.SV               1536 BBS 01/16/91 11:11 06/26/91 [747752] 0
!
```

LINK *Continued*

2. On an DG/RDOS system, example 1 would look like this:

```
!LINK BLACKJACK CARDS.SV
!LIST BLACKJACK

BLACKJACK       GAMES:CARDS.SV
CARDS.SV              2560 S@CW 08/10/91 09:10 08/18/91 [020465] 0
!
```

3. This DG/RDOS example initializes **MYDIR** (the resolution file's directory) and then establishes a link entry, **TEMP.LK**, in the current directory for **MYFILE.SR** in directory **MYDIR**.

```
!INIT MYDIR
!LINK TEMP.LK MYDIR:MYFILE.SR
!LIST TEMP.LK
TEMPLK                MYDIR:MYFILE.SR
MYFILE.SR                 534 D
```

4. In AOS/VS, be sure you state the pathname correctly. In this example, the link entry **TEST** is in the current directory, but you need the full pathname for the resolution file **ALAN**.

```
!LINK TEST :UDD:USER:ALAN
!LIST TEST
TEST                  LNK   :UDD:USER:ALAN
```

LIST

BASIC CLI Command

Displays information about files in the current directory.

AOS/VS	DG/RDOS	UNIX

Format

!LIST [*filename*[/N] ...]

Arguments

filename[/N] The name of a file in your directory. You can use the templates allowed by your operating system in this argument (in this case, UNIX systems use AOS/VS templates). With the /N switch, files that do not match the filename or template are listed.

Global Switches

/A (DG/RDOS only) List both permanent and nonpermanent files.

/B List filenames only.

/C List creation time as *mm/dd/yy hh:mm*.

/E List all file information. (/E overrides /B, /C, /F, /O, and /U.)

/F List (in octal) logical address of first block in file (0 if unassigned). *AOS/VS and DG/RDOS only*

/K Do not list links.

/L List to the default output queue.

/N List links only.

/O List date the file was last opened as *mm/dd/yy*.

/P Sort list by date of last access.

/Q List the names only. They are printed four or eight names across.

/S Sort list alphabetically.

/U List file use count (decimal). *AOS/VS and DG/RDOS only*

/W Sort list by time of last write.

/X Sort by file size, ascending.

Local Switches

mm-dd-yy/A List only those files created on or after the date *mm-dd-yy*.

mm-dd-yy/B List only those files created before the date *mm-dd-yy*.

What It Does

LIST displays information about one or more files or link entries in the current directory. **LIST** only displays the filenames that match *filename*.

LIST *Continued*

The file sizes displayed with **LIST** indicate the relative location of the byte with the highest position ever written to that file. This is not necessarily the number of bytes physically allocated to the file. Listed space represents all file sizes listed on the screen. Total space represents all the file sizes in the current directory. The number of sectors reported is the number of sectors needed to hold all the files at the listed byte sizes. This does not have to equal the actual number of sectors in use.

For link entries on DG/RDOS systems, **LIST** displays the name of the link and resolution files, the directory specifier (if any) given when the link was created, and an @ sign to indicate that the link was made to a file in its directory's parent partition. Link access attributes, if any, are preceded by a backslash (\). They are described below.

Attribute	Meaning (DG/RDOS only)
P	Permanent file, cannot be deleted or renamed.
S	Save file (compiled, core image), or Business BASIC program file.
W	Write protected file, cannot be written to.
R	Read protected file, cannot be read.
A	Attribute protected file whose attributes cannot be changed; you cannot remove the A attribute.
N	No resolution entry allowed; links can be made but not used.
&	User-defined attribute. Use this symbol to define an attribute for a unique access specification. (See your DG/RDOS system manual for more information on user-defined attributes.)
?	User-defined attribute. Use this symbol to define an attribute for a unique access specification. (See your DG/RDOS system manual for more information on user-defined attributes.)

For link entries on AOS/VS systems, **LIST** displays the name of the link and each link name in the chain. An O following the use count indicates that the file is open, but it can be opened by other users; and an E following the use count indicates the file is exclusively open and cannot be opened by other users.

For link entries on UNIX systems, **LIST** displays the link filename. **LIST/N** lists the links only.

LIST *Continued*

On DG/RDOS systems, **LIST** also displays one of the following file characteristics.

Characteristic	Meaning (DG/RDOS only)
D	File is randomly organized.
C	File is contiguously organized.
T	File is a DG/RDOS partition.
Y	File is a directory.

On AOS/VS systems, **LIST** gives the file type, file size, and, for a link file, the name of the resolution file.

On UNIX systems, **LIST** gives the file's size, access permissions, group, and owner.

How To Use It

Execute the command by entering **LIST** from the Business BASIC CLI. The optional global switches are appended to either the command word or to another global switch, while local switches appear as separate arguments. The **/N** switch can only be used with the argument *filename*.

Examples

1. On a DG/RDOS system, this example displays information about **MAIN.SR**. Since the full filename is entered, all the information about the file is displayed, just as if a **/E** switch had been used.

   This display says that **MAIN.SR** is 820 bytes long, its resolution attributes are write protected, and the file is random. It has a link attribute of permanent. **MAIN.SR** was last written to at 2:29 p.m., Jan. 18, 1991, and was last opened Feb. 25, 1991. The first block of the file resides as disk block 2164 (octal) and is not open. The **MAIN.SR** attributes don't allow this file to be deleted via a link.

```
* !LIST MAIN.SR
MAIN.SR         820 WD/P     01/18/91 14:29  02/25/91 [002164]

*
```

LIST *Continued*

2. On a DG/RDOS system, this example lists in alphabetical order the files in the directory with an .SV extension. This listing shows only the size and attributes for each file.

```
* !LIST/S -.SV
TEST                                          02/08/91  11:08:49

ADD$RECDS.SV        2048 SCW
COLORS.SV           2560 SCW
GLSUMMARY.SV        1536 SCW
INQUIRE.SV          3072 SCW
TEST.SV             1536 SCW

LISTED SPACE = 21 SECTORS,   10752 BYTES
 TOTAL SPACE = 303 SECTORS,   155136 BYTES

*
```

3. This DG/RDOS example lists all the files in the **TEST** directory that have a .SV extension. This listing includes files that have been given the permanent attribute using the **CHATR** command. The files are sorted in ascending order by file size. **LIST** displays the file name, the file size in bytes, the file attributes, the creation date and time (or date last modified), the date the file was last opened, the disk location of first block of the file, and the file use count.

```
* !LIST/A/E/X -.SV
TEST                                            3/05/91  10:08:17

TEST.SV          1536 SCW     12/20/84 12:23  12/20/84 [012662]  0
GLSUMMARY.SV     1536 SCW     11/14/90 11:08  01/30/91 [011216]  0
GLEDGER.SV       1536 SCPW    11/30/90 12:46  01/30/91 [011546]  0
ADD$RECDS.SV     2048 SCW     10/08/89 16:48  01/28/91 [011510]  0
COLORS.SV        2560 SCW     10/01/90 13:05  01/18/91 [011475]  0
INQUIRE.SV       3072 SCW     12/07/88 11:49  02/08/91 [011502]  0

LISTED SPACE = 24 SECTORS,   12288 BYTES
 TOTAL SPACE = 267 SECTORS,   136704 BYTES

*
```

LIST

Continued

4. This DG/RDOS example lists the links in this directory and shows their resolution files. **$LPA** is linked to the secondary printer, **$LPT1**. **ALPHAX**, **IDNDX** and **NAME** are linked to the volume label file **NAMEDB.VL**.

```
* !LIST/S/N
MAIN                                    01/18/91  11:30:47

$LPA.                          $LPT1.
ALPHAX.                        MAIN:NAMEDB.VL
IDNDX.                         MAIN:NAMEDB.VL
NAME                           MAIN:NAMEDB.VL

LISTED SPACE = 0 SECTORS, 0 BYTES
 TOTAL SPACE = 50 SECTORS, 30208 BYTES

*
```

5. On an AOS/VS system, **LIST** displays the file name, the file size, file type, and the name of the resolution file for a link file.

```
* !LIST $+ +.** BASIC+

:UTIL:BBASIC                                3/14/91      8:45:07

BBCNVRT1.BA                  10194  UDF
BASICGEN3                     8192  DIR
AOSVS_BB_5.20.FL             10820  TXT
RLS2.PR                      22528  PRV
BB_CONVERT.BA                 5579  UDF
$SYSLIB3                      7168  DIR
$DOC                          4096  DIR
$SYSLIB                       7168  DIR
BASIC.PR                           LNK  BBASIC.PR
VLCONVERT.BA                  9183  TXT
BBCNVRT2.BA                   7542  TXT
BBASIC.PR                   151552  PRV
RLS2.ST                      12288  STF
BASICGEN                      8192  DIR
$MISC                         2048  DIR

Listed space = 522 sectors, 267264 bytes
 Total space = 2864 sectors, 1466368 bytes
```

LOAD

BASIC CLI Command

Loads dumped files.

AOS/VS	DG/RDOS	UNIX

Format

!LOAD *inputfile* [*filename*[**/N**] ...]

Arguments

inputfile	The name of a device or directory that contains files dumped using **DUMP**.
filename[**/N**]	The name of file(s) in *inputfile*. You can use the templates allowed by your operating system with *filename* (in this case, UNIX systems use AOS/VS templates). With the **/N** switch, files matching the filename or template are not loaded.

Global Switches

/A	(DG/RDOS only) Load permanent and nonpermanent files.
/K	Do not load links.
/L	List loaded files to the line printer. (**/L** overrides **/V** and **/P**.)
/N	Load only links.
/O	Delete the current file if it exists and replace it with the file of the same name being loaded.
/P	Do not load files; display their names at the terminal.
/R	Load the most recent version of the file. When a file to be loaded has the same name as a file in the current directory, the system checks both file creation dates. If the existing file is older, the system deletes it and replaces it with the file in *inputfile*. If the existing file is not older, the system retains it and does not load the file in *inputfile*.
/V	Verify loaded files by displaying their filenames at the terminal.

Local Switches

mm-dd-yy/**A**	Load only files created on or after the date *mm-dd-yy*.
mm-dd-yy/**B**	Load only files created before the date *mm-dd-yy*.

What It Does

LOAD places dumped files from *inputfile* into the current directory. **LOAD** cannot load files dumped by **FDUMP** (use **FLOAD**) or by **XFER**. If you omit *filename* and the switches, the system loads all nonpermanent files. Use *filename* to specify which files are loaded.

LOAD *Continued*

DUMP and **LOAD** do not change a file's attributes or characteristics. Unless you specify the **/O** or **/R** switches, files in *inputfile* must have different filenames from files in the current directory.

DG/RDOS restricts the files you can load because the **DUMP** command does not allow the dumping of partitions containing embedded directories; however, you can load directories and their embedded files.

On AOS/VS and UNIX systems, if you dump to a tape from AOS/VS or UNIX Business BASIC, you must load that tape from AOS/VS or UNIX Business BASIC, not from your AOS/VS or UNIX operating system. **LOAD** does not load links to a different directory. Links should not be transported between UNIX and AOS/VS systems.

How To Use It

Execute the command by entering **LOAD** from the Business BASIC CLI. The argument *inputfile* must follow **LOAD**, but additional arguments are optional. If you use global switches, append them either to **LOAD** or to another global switch. Local switches should appear as separate arguments; **/N** is appended to *filename*.

Example

LOAD from the file, **MYDIR.DP**, the most recent copies of all files with the .SR extension, created on or after Sept. 21, 1991, that do not begin with TMP and list their names on the line printer.

```
!LOAD/L/R MYDIR.DP -.SR 9-21-91/A TMP-.-/N
```

LOCKS *Utility*

Displays your current locks.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "LOCKS

What It Does

LOCKS displays the locks that are currently set. Each job has a unique set of lock identifiers. The **LOCKS** display includes each lock's filename and lock area. If an **UNLOCK** command is executed while **LOCKS** is running, the lock associated with that **UNLOCK** disappears from the display. (See *Commands, Statements, and Functions in Business BASIC* for more information on the **LOCK** and **UNLOCK** commands.)

On UNIX systems, you use the **LOCKS** command. See *Commands, Statements, and Functions in Business BASIC* for information on the **LOCKS** command.

How To Use It

Execute **LOCKS** by entering **RUN**, **CHAIN**, or **SWAP "LOCKS**. To end the **LOCKS** utility, press the interrupt key.

To check the locks at the terminal where you are working, use either **SWAP "LOCKS** or **CHAIN "LOCKS**, not **RUN "LOCKS** or the Business BASIC CLI. This is because when you run a program, Business BASIC sets your swap level (which the **LOCKS** utility checks) to 0, and when you use the Business BASIC CLI to execute a program, the CLI closes files. Both these actions prevent you from checking the locks at your terminal using **RUN "LOCKS**.

When you use your terminal as a monitor to check the locks that were set on another terminal, it does not matter how you execute **LOCKS**.

Examples

1. On an AOS/VS system, the following display occurs when you use **RUN** to execute **LOCKS**. The display shows the status of the lock (L for set and W for waiting), the area, the filename, the starting byte at the time **LOCKS** was executed, the ending byte, and the timeout that you set with the lock. An asterisk beside an L status code indicates that another **LOCK** command is trying to lock the same area. The area number is the identifier used in the **LOCK** statement or command. To end the display, press the Escape key.

LOCKS *Continued*

```
* RUN "LOCKS

S   PID      AREA #      FILENAME     START      END      TIMEOUT

L   22         32767     JOE.NX         0         511      -1
L   45         1         FILEB          0         511       1
L   45         2         FILEA          0        1023       1
L   45         3         FILEC          0         511      10
L   45         10        FILEA          0         511       1
L   45         11        FILEA       1024        1535       3
```

2. On a DG/RDOS system, the following display occurs when you run **LOCKS**.

```
* RUN "LOCKS

AREA  JOB    AREA#   FILENAME      START     END
  0     1        1   FILEA             0     511
  1     1        2   FILEB             0     299
  2     2       30   FILEA          1024    1535

*
```

LRELINK *Utility*

Recreates a deleted record chain for a linked-available-record file (logical file structure).

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "LRELINK

Or

!LRELINK [*filename*]

Arguments

filename The name of a linked-available-record data file in the logical file structure.

What It Does

LRELINK is an interactive utility that recreates the deleted record chain for a type L logical file. **LRELINK** performs the same function on files in the logical file database structure that **RELINK** performs on files in the PARAM file database structure.

If the filename you supply to **LRELINK** is not in the logical file database structure, the utility treats the file as a physical file. **LRELINK** prompts you for the byte offset to record zero, the record length in bytes, and the last record number of the file.

Run **LRELINK** if your system crashes while a linked-available-record file is being updated. Then check the most recently modified record to see that it is correct.

How To Use It

Execute **LRELINK** by entering **RUN**, **CHAIN**, or **SWAP "LRELINK**. This starts the utility's interactive mode. **LRELINK** prompts you for the name of the file for which you want to recreate deleted record chain. You can also use a command line where you enter **LRELINK** and the filename from the Business BASIC CLI.

LRELINK

Continued

Example

LRELINK prompts you for a filename. When **CNAM** is supplied, **LRELINK** recreates its deleted record chain. Since **CNAM** is not a logical file, **LRELINK** also prompts you for its byte offset, record size, and last record number.

```
*RUN "LRELINK
FILENAME: CNAM
BYTE OFFSET: 0
RECORD SIZE(BYTES): 200
LAST RECORD NUMBER: 1000

*
```

LSPEED *Utility*

Displays or changes the default port speeds.

DG/RDOS

Format

{ RUN / SWAP / CHAIN } "**LSPEED** [*baud port ...*]

Or

!LSPEED [*baud port ...*]
!LSPEED [*baud*] *arguments*

baud	The speed or baud rate of a terminal operating on ALM, ASLM, USAM, or ULM multiplexor lines. You can enter this argument with the command or during interactive mode.
port	The port number of a terminal. You can enter this argument with the command or during interactive mode.

Global Switches

/A	Display the multplexor lines' current settings. Ignore all arguments, and don't change any speeds.
/N	Suppresses the DONE? query. This allows speeds to be changed within CLI indirect files or STARTUP macros.

What It Does

LSPEED changes the baud rate (speed) of a port operating on an ALM, ASLM, USAM, or a ULM multiplexor line. You must reset the baud switch on the back of the port when you use **LSPEED** before the program will reset the port speed. If you are a privileged user, **LSPEED** lets you change the line speed of other users' terminals.

How To Use It

Execute the command by entering **RUN**, **CHAIN**, or **SWAP "LSPEED**. This starts the **LSPEED** dialog where **LSPEED** prompts you for information. You can also execute **LSPEED** in command line format by entering **LSPEED** and its arguments from the Business BASIC CLI.

Once executed, **LSPEED** displays the line's current characteristics and speed. If **LSPEED** is operating interactively, it prompts you for the baud rate and port number (*port-n* + 2 maps to multiplexor line *n*) for the multiplexor line you want to change.

LSPEED *Continued*

LSPEED checks this speed against a table of valid speeds. If the speed is valid and the **/N** global switch is omitted, **LSPEED** instructs you to reset the baud switch on the back of the terminal and enter **YES** to indicate that the speed has been properly selected.

The program then changes the terminal's speed and queries DONE? for 60 seconds or until a reply is received. If no reply is received, the program resets the original speed. If the response is properly received, the program displays a confirmation and then terminates.

NOTE: ALM clocks and their corresponding baud rates are dependent on the way your ALM board is hardwired. You must customize the **LSPEED** program for the ALM defining the various clocks' speeds by changing the data statements (9010-9100). You must know the baud rates for your system to do this. The clocks are set during hardware initialization. **LSPEED** assigns the following default baud rates to the clocks for an ALM:

Clock	Baud
0	9600
1	4800
2	300
3	600

If your ALM is not wired this way, make the appropriate changes in the data lines of the **LSPEED** program.

Privileged users (AA and OP accounts) can change the port speed of other users' terminals. This is useful for quickly attaching temporary terminals incapable of operating at the default baud rate. When a privileged user executes **LSPEED**, additional arguments are allowed to specify whether the user is changing his own port speed or the speed of another port. The user specifies a single baud rate and no port number to change his own port speed; otherwise, he specifies the port number of the terminal he wishes to change as the next argument. Additionally, the privileged user may specify multiple pairs of baud and port arguments to change several terminals at one time.

Examples

1. **LSPEED**, executed in command line format form from port 14 (multiplexor line 12), provides the following prompts for a nonprivileged user:

```
* !LSPEED 600
PORT 14 IS 8 BITS WITH NO PARITY AND 1 STOP BIT(S) AT 4800 BAUD.
CHANGE TERMINAL TO 600 BAUD, AND TYPE "YES" AND A RETURN.
DONE? YES
YOU'RE RIGHT!  IT'S NOW 600 BAUD

*
```

LSPEED *Continued*

2. Execute **LSPEED** from port 14 to change the speed of port 14 back to a 4800 baud rate and suppress the DONE? query.

   !LSPEED/N 4800

3. **LSPEED** provides the following prompts when run from port 14.

```
* RUN "LSPEED
PORT 14 IS 8 BITS WITH NO PARITY AND 1 STOP BIT(S) AT 4800 BAUD.
NEW BAUD RATE [4800]? 1200
FOR WHICH LINE [14]?
CHANGE TERMINAL TO 1200 BAUD, AND TYPE "YES" AND A RETURN.
DONE? YES
YOU'RE RIGHT!  IT'S NOW 1200 BAUD

*
```

4. For a privileged user, **LSPEED** changes port three to baud rate 1200 and port eight to baud rate 9600 and suppresses the DONE? query.

   !LSPEED/N 1200 3 9600 8

LSTCOM

BASIC CLI Command

Compares two program listing files.

AOS/VS	DG/RDOS

Format

!LSTCOM *listfile1 listfile2* [*outputfile*]

Arguments

listfile	The name of a text file containing Business BASIC code.
outputfile	The name of the file to receive the results of the comparison.

Global Switch

/L	Send the **LSTCOM** output to the default output queue. Use the Business BASIC CLI command **GQUE** to retrieve the default output queue.

What It Does

LSTCOM performs a line by line comparison of two text files containing Business BASIC code. You can specify *outputfile*, the line printer (**/L** switch), or the terminal (the default) to receive the lines that differ between the two program listing files. **LSTCOM** only compares lines beginning with statement numbers; it ignores all lines beginning with a colon ":" (indicating a comment) and all blank lines.

LSTCOM displays an error message when it encounters a nonblank line that does not begin with a statement number or a colon.

NOTE: Line numbers in Business BASIC can be up to five digits long. If you have two identical programs except that one program uses leading zeroes in its line numbers and the other program doesn't (for example: `00100 REM` vrs. `0100 REM`), **LSTCOM** flags these files as being different.

How To Use It

Execute the command by entering **LSTCOM** from the Business BASIC CLI. The names of two listing files must follow **LSTCOM**. If used, the **/L** switch is appended to **LSTCOM**.

LSTCOM

Continued

Example

Use **LSTCOM** to compare **NEWPROG.LS** to **OLDPROG.LS** and to put the results in the file **RESULTS**. By listing the programs before you execute **LSTCOM**, you can see that differences exist. Once **LSTCOM** ends, you can type **RESULTS** to see how **LSTCOM** indicates the differences.

```
* !TYPE NEWPROG.LS
00100 INPUT A
00110 FOR I=1 TO A
00120 PRINT I*A

00130 NEXT I
00140 STOP

* !TYPE OLDPROG.LS
00100 INPUT A,B
00110 FOR I=1 TO A
00120 PRINT I*A
00125 LET B=B*A
00127 PRINT B
00130 NEXT I
00140 STOP

* !LSTCOM NEWPROG.LS OLDPROG.LS RESULTS
* !TYPE RESULTS
NEWPROG.LS: 00100 INPUT A
OLDPROG.LS: 00100 INPUT A,B

OLDPROG.LS: 00125 LET B=B*A
OLDPROG.LS: 00127 PRINT B

*
```

LSTMERGE

BASIC CLI Command

Merges a program list file and a comment file.

AOS/VS	DG/RDOS

Format

!LSTMERGE *listfile commentfile* [*outputfile*]

Arguments

listfile	The name of a text file containing Business BASIC code.
commentfile	The name of a text file containing Business BASIC code and program comments.
outputfile	The name of an optional file to receive the combined file (*listfile* and *commentfile*).

Global Switch

/L	Send the **LSTMERGE** results to the default output queue. Use the Business BASIC CLI command **GQUE** to retrieve the default output queue.

What It Does

LSTMERGE merges each line in *listfile* with its matching line number in *commentfile* to pick up the comments for that line. It sends the merged lines to *outputfile*. If you do not specify *outputfile* or use the global **/L** switch, the lines remain in *commentfile*. Lines in *commentfile* that contain comments only are included unconditionally in the **LSTMERGE** output.

NOTE: Line numbers in Business BASIC can be up to five digits long. If *listfile* and *commentfile* use different line-numbering schemes (i.e., one uses leading zeroes such as 00100 and the other one doesn't — 0100), **LSTMERGE** is unable to merge these files correctly.

How To Use It

Execute **LSTMERGE** by entering the command from the Business BASIC CLI. The names of a program listing file and a program listing file containing comments must follow **LSTMERGE**.

Use either a text editor or the Business BASIC **LIST** command or both to create a program listing in *commentfile*. When you use **LIST** to create an uncommented listing file for the updated program, do not replace the old lines with new lines that have the same line numbers. If you do, you get an old comment on a new line. You can delete any comments you do not want by using a text editor on the *commentfile*. Do not **RENUMBER** before using **LSTMERGE**; you lose comments for lines that do not exist in *commentfile*.

LSTMERGE

Continued

Example

List **PROG.SL** on the line printer with the comments from **PROG.SR**.

```
!LSTMERGE/L PROG.SL PROG.SR
```

LXFER *Utility*

Copies one logical file to another logical file (logical file structure).

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "LXFER

Or

!LXFER *source-file destination-file*

Arguments

source-file — The name of a logical file whose contents you want transferred to *destination-file*.

destination-file — The name of an existing logical file that will receive the contents of *source-file*. **LXFER** displays an error message if *destination-file* does not exist.

Global Switches

/V — Verify the amount of data transferred.

/I — Ignore warning messages and transfer data unconditionally.

What It Does

LXFER transfers the contents of *source-file* to *destination-file*. Both logical files must be the same type in the logical file database structure. Both files must be defined. **LXFER** does not create the *destination-file*.

The number of logical records defined for *destination-file* can differ from the number defined for *source-file*. In that case the smaller number is transferred. If the logical files are type I, the last-available-block pointer in block 0 is adjusted. If you define fewer records for the *destination-file* than for the *source-file*, the system displays a warning message. It processes the transfer, but it ignores the additional source records.

The length of the *destination-file* record can differ from the length of the *source-file* record. If the *destination-file* record length is greater, **LXFER** adjusts and transfers the data. If the *destination-file* record length is less, a warning message and a query about whether to continue the transfer are displayed. If you answer **YES** to the query, the system truncates the extra data and transfers the resulting record.

When the record lengths are the same, **LXFER** uses a **BLOCK READ** and a **BLOCK WRITE** to transfer the records. When the record lengths differ, **LXFER** uses an **LREAD** and an **LWRITE** to transfer the records.

LXFER *Continued*

How To Use It

Start the **LXFER** dialog by entering **RUN**, **CHAIN**, or **SWAP "LXFER**. The dialog begins with a prompt for the source and destination files. To end **LXFER**, enter **STOP**.

You can also execute **LXFER** in a command line format from the Business BASIC CLI where you enter the name of a source file and destination file with the command.

If you want to use **LXFER** to enlarge a logical file database, perform the following steps:

1. Create a temporary database that is the size you want it to be (**LFU PCREATE**).
2. Create a temporary logical file in this database for each logical file that is in the current database (**LFU LCREATE**). The temporary logical files can be larger than the existing logical files. You must give the temporary files different filenames from the existing files.
3. Use the utility **LXFER** to copy the logical file in the current database to the corresponding logical file in the expanded temporary database. You must repeat this step for each logical file that will be in the new database.
4. Delete the old database (**LFU PDELETE**).
5. Rename the temporary database to the old database name (**LFU PRENAME**).
6. Rename the logical files to the same names that existed in the old database (**LFU LRENAME**).

Examples

1. Transfer the logical file **CLIENTS** in **NBASIC** to **CUST** in the current directory. The files are the same size so **LXFER** uses **BLOCK READ/BLOCK WRITE**. The system verifies that 13,312 bytes were transferred.

```
* !LXFER/V NBASIC:CLIENTS CUST
13312 BYTES TRANSFERRED

*
```

2. The logical file **CLIENTS** has a record length of 128 bytes while **CUST** has a record length of 200 bytes. Each has a last record number of 100. **LXFER** uses **LREAD** and **LWRITE** to transfer the data. With the **/V** switch, **LXFER** also displays the amount of data read and the amount of data written to **CUST**.

```
* !LXFER/V CLIENTS CUST
12928 BYTES READ, 20200 BYTES WRITTEN

*
```


LXFER

Continued

3. A database named **CUSTDB** needs to be enlarged so that the data file and the index file will hold 500 records instead of 100. **LFU PLIST** displays the current database file information.

```
* !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
```

After you run **INDEXCALC** to determine the number of sectors needed for the new file, use LFU to set up the temporary database file:

```
* !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
```

Use the **LXFER** utility to copy the logical files.

```
* !LXFER/V CUST TCUST       :Copies the current data file into
                            :the temporary data file.
5120 bytes transferred

* !LXFER/V CUSTNO TCUSTNO :Copies the current index file into
                          :the temporary index file. The 3072 bytes transferred
                          :last-available-block pointer in block 0 is adjusted.

* !LFU PDELETE CUSTDB :Deletes the current database.

* !LFU PRENAME TEMPDB CUSTDB      :Renames the temporary database
                                  :to the current database name.

* !LFU LRENAME TCUST CUST         :Renames the data file.

* !LFU LRENAME TCUSTNO CUSTNO     :Renames the index file.

* !LFU PLIST CUSTDB               :PLIST of the enlarged file.
```

LXFER

Continued

DB file: CUSTDB

File Name	File Type	Starting Sector	# of Sectors	Record Length	Last Record	# of Bytes
CUST	L	0	49	50	500	25050
CUSTNO	I	49	18	512	17	9216
			---			------
		Total Sectors:	67		Bytes:	34266

The following output from **INDEXVRFY** shows the **CUSTNO** index before it was copied:

```
* !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   ******
```

The next display uses **INDEXVRFY** to show the **CUSTNO** index after it was copied and renamed.

```
* !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 ******
```

MDIR

BASIC CLI Command

Displays the master directory name.

DG/RDOS

Format

!MDIR

What It Does

MDIR displays the name of the master directory.

How To Use It

Execute the command by entering **MDIR** from the Business BASIC CLI.

Example

MDIR tells you that the current master directory is **DE0**.

```
!MDIR
DE0
```

MOVE

BASIC CLI Command

Transfers files from one directory to another.

AOS/VS	DG/RDOS	UNIX

Format

!MOVE *directory* [*filename*[/N] ...]

Arguments

directory — The name of the initialized directory that receives the files. *directory* can be a partition or a subdirectory. In AOS/VS, the full pathname cannot exceed 31 characters, including any colons.

filename[/N] — The name of a file in the current directory that you want to match. You can use asterisk (*), dash (-), or plus (+) templates (in this case, UNIX systems use AOS/VS templates). *filename* cannot have directory or device specifiers as prefixes. With the /N switch, **MOVE** transfers all files except those matching the filename or template.

Global Switches

/A — (DG/RDOS only) Move permanent and nonpermanent files.

/C — Used with **/D**. Repeat each filename and wait for confirmation that the file is to be deleted. Press CR to delete the file, and any other character to keep the file.

/D — Delete the original files once all transfers are complete (i.e., after copying them).

/K — Do not move links.

/L — List moved files to the default output queue.

/N — Move only links.

/O — If a file with the same name as *filename* exists in *directory*, delete it before moving *filename*.

/P — Move files in the order of the date last opened.

/R — Move the most recent copy of *filename*. If a file with the same name is in the destination directory and has a more recent creation date, *filename* is not moved.

/S — Move files in alphabetical order.

/V — Verify moved files by displaying them at the terminal.

/W — Move files based on the time of the last write to them.

Local Switches

mm-dd-yy/**A** — Move only files created on or after this date (*mm-dd-yy*).

mm-dd-yy/**B** — Move only files created before this date (*mm-dd-yy*).

MOVE

Continued

What It Does

MOVE transfers data from one directory to another. Without *filename* or switches, **MOVE** copies all nonpermanent entries in the current directory to the directory you specify. If you specify filenames, **MOVE** searches the current directory for the filenames and moves only those files.

You can move only resolution and link entries, not directories or partitions. **MOVE** does not change the file's characteristics or information in the filename entry.

How To Use It

Execute the command by entering **MOVE** from the Business BASIC CLI. **MOVE** must be followed by the name of the source directory. You can also specify the files to be moved as arguments following directory. The optional global switches are appended to either the command word or another global switch. Local switches appear as separate arguments at the end of the command line. The **/N** switch must be appended to *filename*.

NOTE: Be careful when you use any of the global switches that delete files because any attempt to delete a link file will delete the link's resolution file instead.

Example

MOVE transfers the most recent copies of files with the .SR extension to the DG/RDOS directory **BACKUP.DR** and displays the moved filenames at your terminal.

```
!MOVE/R/V BACKUP -.SR
TEST.SR
TEST1.SR
!
```

MOVETABREC *Utility*

Copies FM table file records.

AOS/VS	DG/RDOS	UNIX

Format

```
{ RUN   }
{ SWAP  } "MOVETABREC
{ CHAIN }
```

What It Does

MOVETABREC is an interactive utility that duplicates File Maintenance (**FM**) table file records within a table file. This command is useful for defining multiple pages and/or formats where the same fields appear on more than one page or format.

How To Use It

MOVETABREC only works with records in a table file.

Execute the utility by entering **RUN**, **CHAIN**, or **SWAP "MOVETABREC**. **MOVETABREC** asks you for the information it needs; you can get this information from **FM**. Then **MOVETABREC** displays information about the file at your terminal. **MOVETABREC** continues to prompt you for information until you press the interrupt key.

Example

MOVETABREC displays the information you supply on the file and then asks you for another source record number. You stop **MOVETABREC** by pressing the interrupt key.

```
*RUN "MOVETABREC

MOVE RECORDS IN A TABLE FILE
TABLEFILENAME: CUST.TB
SOURCE RECORD NUMBER: 16
DESTINATION RECORD NUMBER: 36
NUMBER OF RECORDS: 6
VERIFY? YES
16        ADDRESS 1
17        CITY, STATE
18        BALANCE
19        PHONE NO
20        SEX
21        MISC.
SOURCE RECORD NUMBER: <ikey>

IKEY AT 0130
*
```

NEWS

Utility

Provides a logon message program.

DG/RDOS

What It Does

Since this utility can only be used on DG/RDOS systems by someone with system manager privileges, **NEWS** is explained in the *Business BASIC System Manager's Guide*.

OPCLI *Utility*

Performs system operator functions.

DG/RDOS

What It Does

Since this utility can only be used on DG/RDOS systems by someone with system manager privileges, **OPCLI** and its commands are explained in the *Business BASIC System Manager's Guide*.

OPEN *Utility*

Opens files in the PARAM file structure.

AOS/VS	DG/RDOS	UNIX

Format

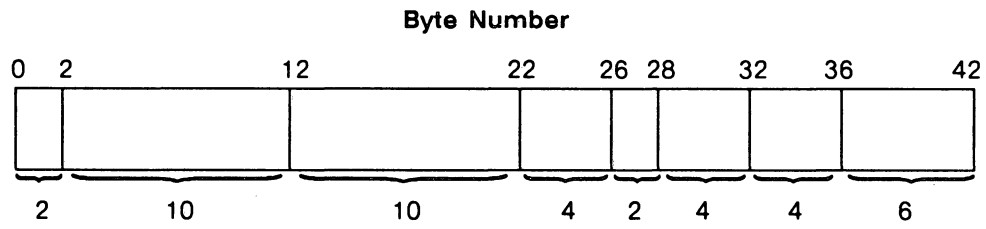
SWAP "OPEN

What It Does

OPEN opens the physical file containing the subfile that you want to use and returns the information you need to set up a C1 (file characteristics) array entry for that file.

The utility only works with files that have a PARAM file entry. You must supply the name of the subfile or physical file and the mode of open you want to use. The modes are described under the **OPEN FILE** statement in *Commands, Statements, and Functions in Business BASIC*.

OPEN searches the PARAM file for an entry describing your subfile. The utility then uses the mode you selected and opens the physical file for your subfile file on the lowest available channel.



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

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

Figure 1-3. PARAM File

OPEN

Continued

The **OPEN** utility can only open up to 16 physical files at a time. However, it can open a maximum of 32 subfiles. The limit of 32 subfiles is because every entry in the C1 array requires four elements, each four bytes long, and the C1 array must be small enough to fit into the 512-byte common area.

OPEN cannot open two subfiles in the same physical file in different modes; therefore, it chooses the least severe open mode. When more than one subfile is opened within a single physical file, use mode 5 for all subfiles.

How To Use It

Do not use **RUN** or **CHAIN** with **OPEN**, only **SWAP**. To execute **OPEN**, do the following:

1. Dimension an argument string to at least 512 bytes.
2. Assign the filenames and the access modes for each file, separated by commas, to the argument string (see **OPEN FILE** in *Commands, Statements, and Functions in Business BASIC*). The files can be subfiles and/or physical files with PARAM entries. The string must be null-filled to its dimensioned length with no embedded spaces.
3. Place the argument string in the common area with a **BLOCK WRITE**.
4. Enter the statement **SWAP "OPEN**.
5. Use **BLOCK READ** to get the Cl information from the common area and put it in the argument string.

After performing the **BLOCK READ**, check the first two bytes of the string for an error. If they equal -1, an error occurred. In that case, the next two bytes contain the error number, and the rest of the string contains the error message.

Unpredictable errors occur if you use an argument string containing errors.

When the **OPEN** is successful, the C1 array can be built from the argument string, which contains the channel number of the opened files and record information. Extract the information (four bytes per element in the C1 array) from the string using the **ASC** function. **GETREC.SL** and **POSFL.SL** can then use the C1 array to position to and get an available record from the subfile.

OPEN

Continued

The format of a a C1 array is illustrated in Figure 1-4. This C1 array uses two subfiles: **EMPIX** and **EMP**. **EMPIX** is an index file that has 6 sectors of 512 bytes each. **EMPIX** starts at byte position 0 in its physical file, **MASTER**. **EMP** is a data file with 100 records of 52 bytes each and starts at byte position 3072 in the physical file **MASTER**. **EMPIX** and **EMP** open on the same channel (0) because they are part of the same physical file.

		CHANNEL NUMBER 0	BYTE OFFSET 1	FILE SIZE 2	RECORD SIZE 3
EMPIX (subfile of MASTER)	**0**	0	0	6	512
(subfile of MASTER)	**1**	0	3072	100	52
PHYS (subfile of MASTER)	**2**	1	0	100	50

Figure 1-4. Sample C1 Array

Example

This program segment uses **OPEN** and then checks for an error. If no error has occurred, it fills in the C1 array for the opened files. Since more than one subfile is being opened within a physical file, mode 5 is used for both subfiles.

```
00010 DIM X$[512],C1[2,3],REC$[48]    :Dimension common string, C1
                                      :array, and record string.
00020 LET X$="SUBFILE1,5,SUBFILE2,5,PHYS,6",FILL$(0)
                                      :Set up the argument string
                                      :with filenames, access modes.
00030 BLOCK WRITE X$                  :Send file info to common area.
00040 SWAP "OPEN                      :Execute OPEN;
                                      :OPEN opens files on channels
                                      :and returns string to common area.
00050 BLOCK READ X$                       :Read common info into X$.
                                      :This info is for the C1 array.
00055 UNPACK "JJ",B$,ERRIN,ERRNO
00060 IF ERRIN<>-1 THEN GOTO 00100    :Check for an error;
                                          :-1 indicates an error.
00070 PRINT "ERROR # ";ERRNO;" - ";B$(5,512)
00080 END
```

OPEN

Continued

```
00100 LET K=1                          :Pointer to first element in string
00110 FOR I=0 TO 2                     :For each subfile file 0, 1, and 2
00120   FOR J=0 TO 3                   :and for each dimension to C1 array
00130     LET C1(I,J)=ASC(X$(K,K+3))
                                       :Extract element, put in C1 array.
00140     LET K=K+4                    :Bump pointer 4 bytes to
00150   NEXT J                         :next string element.
00160 NEXT I
. . .
```

PARAMCON *Utility*

Converts a PARAM file structure into a logical file structure.

AOS/VS	DG/RDOS

What It Does

PARAMCON converts a PARAM file database structure into a logical file database structure. The two structures differ in the location of the file descriptions and in terminology. The PARAM structure maintains file descriptions in the PARAM file, while the logical structure keeps file descriptions in volume label (.VL) files. Also, the PARAM structure uses the term "subfile" to refer to files within a physical file, and the logical structure uses the term "logical."

Information on this utility, like other program conversion tools, is documented in the on-line file **CONVERT.DOC**, located in the Business BASIC directory **DOC**.

PARAMPRT *Utility*

Prints the contents of the PARAM file.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "PARAMPRT

What It Does

PARAMPRT produces a report describing the PARAM file. The report includes the name of the subfile, the master file, the byte offset to record 0, the record length, the last record number, and the highest record number used for each file. The report can be printed at your terminal or at the default output queue.

How To Use It

Execute **PARAMPRT** by entering **RUN, CHAIN,** or **SWAP "PARAMPRT.** **PARAMPRT** asks whether to send the output to your terminal or to the printer (the default output queue). It then produces the PARAM file report.

Example

PARAMPRT displays PARAM file report on the terminal.

```
* RUN "PARAMPRT

Output to Printer (Y or N) [N]? N

                                PARAM FILE

Record   Sub         Master      Record              Last      Highest #
Number   File Name   File Name   Offset    Length    Record #  Used

  1      ACNTIX      ACCOUNTING       0       512        11          11
  2      ACNTS       ACCOUNTING    5632       128       200           4
  3      CUSTI1      CUSTI            0       512        10          10
  4      CUST12      CUSTI         5120       512        10          10
  5      CUSTI3      CUSTI        10240       512        10          10
  6      CUST        CUSTFILE         0       128       100          10
  7      CUST.LG     CUST.LG          0       146       200           0
  8      NAME        NAME             0       512      1000           0
  9      TAX         TAX              0       512      1000           0
 10      TAXES       TAXES            0        35      1000           0
```


PD

Utility

Displays information about a program in working storage or in a SAVE file.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "PD

What It Does

PD displays information about a Business BASIC program that is either a SAVE file or the program currently in the working storage. The **PD** display includes the variables shown by the **VAR** utility as well as:

- The last channel used
- The last statement executed
- The next statement to be executed
- The status of the **GOSUB/RETURN** stack
- The status of the **FOR/NEXT** stack
- The status of the **DO** stack (AOS/VS systems only)
- The status of **ON ERR** and **ON IKEY** traps
- The program segment sizes (the size display is similar to that of the **SIZE** utility)

NOTE: If you are on a UNIX system, use the PROGRAM DISPLAY command to get this information. For more information, see *Commands, Statements, and Functions in Business BASIC*.

You can send the **PD** information to a file, a line printer, or a terminal. Use the information in **PD** to find a program error that is nested in loops and subroutines. When a program uses multiple **ON ERR**, **ON IKEY**, and **DEF** statements, **PD** shows the statements that are currently in effect.

Executing **PD** is like taking a snapshot of the current state of the program. Thus, if you modify the program, you need to run the program again before you swap to **PD**.

How To Use It

Execute **PD** by entering **RUN**, **CHAIN**, or **SWAP "PD**. Use **RUN "PD** to refer to a SAVE file and **SWAP "PD** to refer to the program in working storage. Once executed, **PD** prompts you for the output file. You can specify the line printer or any output file, or press CR or New Line to have **PD** display the output at your terminal.

PD

Continued

If you execute **PD** using **CHAIN** or **RUN**, the utility also asks for the name of the program. It then asks you whether you want to list the program variables.

Examples

1. The **PD DEMO** program is listed and then run. This leaves the program in working storage, so to use **PD** on it you must enter **SWAP "PD**. This example is for an AOS/VS system and uses DO loops.

```
* LIST

00010 REM - PD Demo program showing assorted info
00020 ON IKEY THEN GOTO 09000            : IKEY trap definition.
00030 ON ERR THEN GOTO 09000             : Error trap definition.
00040 DEF FNA(X)=OR(X,-AND(X,256))       : User-defined functions.
00050 DEF FNB(X)=OR(X,-AND(X,32768))
00060 LET I=1                            : Start loops.
00070 DO                                 : \
00080   LET J=I                          :  \
00090   DO WHILE J<=I*2-1                :   \
00100     GOSUB 00160                    :   Nested DO loops & GOSUBs
00110     LET J=J+1                      :
00120   END LOOP                         :   /
00130   LET I=I+1                        :  /
00140 UNTIL I>20                         : /
00150 END                                : Ends the program.
00160 LET K=FNA(I)
00170 DO WHILE K<=J
00180   GOSUB 00220 : THIS PROGRAM DEMONSTRATES
00190   LET K=K+1
00200 END LOOP
00210 RETURN
00220 REM THIS PROGRAM DEMONSTRATES
00230 REM THE FEATURES OF PD
00240 STOP
00250 RETURN
09000 REM - BAIL OUT
09010 STOP

* RUN

STOP at 240

* SWAP "PD

OUTPUT FILE:
SCRATCH         3 VARIABLES - List them? (Y/N) [N] Y
I =    1
```

PD

Continued

```
J =   1
K =   1

GOSUB/RETURN stack: 00110 [ 00190 ]

Last executed: 00250;   Next: 00010;   Last err: # 0 @00000;   Last
file: 0

ON ERR defined at: 00030    ON IKEY defined at: 00020

DEFined functions:          FNA @00040          FNB @00050

FOR/NEXT stack:  Empty

DO stack:
  DO loop       @ 00080  Control expression undefined
  DO WHILE loop @ 00100  Control expression defined
  DO WHILE loop @ 00180  Control expression defined

For current page size of : 2048, (Alternate page size of :  512)
     Segment       In Pages          In Bytes
     Program:       1 (  1)           490 [ 1558 left in page]
     Data   :       1 (  2)           866 [ 1182 left in page]
     Total  :       2 (  3)          1356 [ 4096 effectively]
     Maximum:     127              262140
     Left   :     125              260784

*
```

2. This example is like the previous one except that it is for an RDOS system and uses FOR/NEXT loops.

*** LIST**

```
00010 REM - PD Demo program showing assorted info
00015 ON IKEY THEN GOTO 09000          :IKEY trap definition.
00017 ON ERR THEN GOTO 09000           :Error trap definition.
00020 DEF FNA(X)=OR(X,-AND(X,256))     :User-defined functions.
00030 DEF FNB(X)=OR(X,-AND(X,32768))
00040 FOR I=1 TO 20                              :Start loops.
00050   FOR J=I TO I*2-1               :\
00060     GOSUB 00100                  :>Nested FOR loops and GOSUBs.
00070   NEXT J                         : /
00080 NEXT I                           :/
00090 END                              :Ends the program.
00100 FOR K=FNA(I) TO J
00120   GOSUB 00200 : THIS PROGRAM DEMONSTRATES
00150 NEXT K
00180 RETURN
00200 REM THIS PROGRAM DEMONSTRATES
```

PD

Continued

```
00210 REM THE FEATURES OF PD
00220 STOP
00250 RETURN
09000 REM - BAIL OUT
09010 STOP

* RUN

STOP AT 00220

* SWAP "PD

OUTPUT FILE:
SCRATCH      3 VARIABLES - List them? (Y/N) [N] Y
I = 1
J = 1
K = 1

GOSUB/RETURN stack: 0070 [ 0150 ]

Last executed: 00220;  Next: none;  Last err: # 0 @00000;  Last file
: 0

ON ERR defined at: 00017    ON IKEY defined at: 00015

DEFined functions:          FNA @00020                  FNB @00030


FOR/NEXT stack:    * = Active
* Index var: I     begins @ 00050  Inc: 1      End value: 20
* Index var: J     begins @ 00060  Inc: 1      End value: 1
* Index var: K     begins @ 00120  Inc: 1      End value: 1

For current page size of : 2048, (Alternate page size of :  512)
     Segment      In Pages          In Bytes
     Program:      1 (  1)        436 [ 1612 left in page]
     Data   :      1 (  1)        270 [ 1778 left in page]
     Total  :      2 (  2)        706 [ 4096 effectively]
     Maximum:     15             30720
     Left   :     13             30014

*
```

PED

Utility

Displays the system status.

DG/RDOS

Format

```
{ RUN  }
{ SWAP } "PED
{ CHAIN}
```

What It Does

PED is a continuous program that displays the status of all active jobs on the system that were started on a type 6 terminal. The display is similar to the **STAT** display (see **STAT**), but **PED** incurs less overhead. However, **PED** works with type 6 terminals only; all other terminal types must use **STAT**.

How To Use It

Enter **RUN**, **CHAIN**, or **SWAP "PED** to start the utility. **PED** clears the screen before starting its display. To stop **PED**, press the Esc key. To get the time of the last refresh and the current date, press the Erase-page key. **PED** clears the screen and redisplays the system status. The default refresh rate is three seconds. You can set this refresh rate to any number of seconds by entering the refresh period, in seconds, and pressing CR during the DELAY period. Note that **PED** discards input entered during the refresh cycle so the data must be entered while the program is idle. You also invoke **PED** when you execute **STAT** with the **/R** global switch at a type 6 terminal.

Example

PED provides continuous information on the system status. Before beginning the display, **PED** clears the screen and places the time in the left corner of the screen and the date in the right corner. To end the **PED** utility, press the Esc key. The following example shows the **PED** display and explains the information in the different columns.

PED

Continued

```
* RUN "PED

04:56:41                                                        07/10/91
00    R 19  NBASIC    PED    AAAAA6  00  0  3190  1016   90.4  78 00:01
a  b c d    e         f      g       h   i  j     k      l     m  n
```

The **PED** columns contain:

a	Job number
b	I/O status (A blank indicates it has been satisfied.)
c	Program status: R Running C Compiling B Compiling and running
d	The program's task priority
e	Directory
f	Program's name
g	Account code used at logon time
h	The job's terminal number (-1 for detached jobs)
i	Job's push level
j	Program size in bytes
k	The data size in bytes
l	The CPU time in seconds used since logon
m	Number of calls made since logon
n	The length of time the job has been logged on

PLB

Utility

Builds system and user program libraries.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "PLB

What It Does

The Program Library Builder (**PLB**) utility lets you create system and user program libraries. Program libraries are files that contain program SAVE file images with a hashed index to the images. The system normally keeps libraries open. Thus, you can execute a program using the libraries without the overhead of executing an operating system open. This helps you maximize system performance.

Business BASIC does not supply any libraries, so if you want to take advantage of this feature, you must use **PLB** to build them. However, you can use the file **LIB.CM**, which comes with Business BASIC, to build the system program library **BASIC.PL** (see the section "System Program Library").

How To Use It

To create a program library, you (or your system manager) must log on to Business BASIC in the **$SYSLIB** (**$SYSLIB3**) directory on AOS/VS or the **$SYS** (**$SY3**) directory on DG/RDOS and execute **PLB** by entering **RUN**, **CHAIN**, or **SWAP** "**PLB**. Once **PLB** starts, it prompts you for the name of the library, the name of the input file that lists the programs you want in the library, and the frame size of the library.

Your input file can be any text file that contains the names of the programs that you want in the library. You supply the name of your input file at the prompt `"INPUT FILE [$TRI]:?"` If you press the New Line key without entering an input filename, Business BASIC prompts you for the program names. (The default, **$TRI**, tells Business BASIC that the input will come from the terminal.) If you are going to refer to a program by an alias (such as a link or the lowercase version of the program name), then include that alias, either in your input file or at the Business BASIC prompt `NAME(S):`, when you enter the program name. All the program aliases must appear on the same line as the program name to which they refer. Separate the program names by commas.

To maximize performance, create a contiguous library file before you start **PLB**. **PLB**, however, will create the library file if it does not exist.

For the frame size, select a prime number that is large enough to contain all of the program and alias names. The minimum frame size is 3. You can have 8 entries per frame; however, the effect of hashing usually prevents you from achieving that density. The rule of thumb for picking a frame size is to select a prime number that is 1 less

PLB

Continued

than a multiple of 4 and greater than one quarter of the number of programs. For instance, the **BASIC.PL** system program library that is created using **LIB.CM** as the input file contains approximately 80 programs and a frame size of 23 suffices (see the section "System Program Library" for information on **BASIC.PL** and **LIB.CM**).

The optimum frame sizes are the prime numbers 3, 7, 11, 19, and 23. These sizes are most efficient at using the sectors that are assigned to the library index. When you run **PLB** later to add entries to an existing library, **PLB** does not ask you for a frame size. If you want to create a larger frame size for an existing library, you must delete the library file, recreate it, and then run **PLB** again, selecting a larger frame size this time.

After you have built a program library, you must terminate Business BASIC and then execute it again before you can use the programs in the library. You can execute any of the programs in a program library using **CHAIN**, **SWAP**, or **RUN** simply by placing the library prefix before the program name. Use a pound sign (#) if the program is in the system program library and a percent sign (%) if the program is in a user program library. For example, to execute the CLI from the system program library rather than from **$SYSLIB** (**$SYSLIB3**) or **$SYS** (**$SY3**), you can use any of the following commands:

*** RUN "#CLI**

*** CHAIN "#CLI**

*** SWAP "#CLI**

You can use the Business BASIC CLI command **XFER** to copy and reload a system or user program library while the Business BASIC is running. However, make sure that you do not overwrite the library while users are reading from it. If you change the frame size, the hashing algorithm cannot find entries for users who have already opened the library. Thus, you should shut down Business BASIC before you recreate the system program library (**BASIC.PL**).

The sections "System Program Library" and "User Program Library" contain more information on program libraries and examples of them.

System Program Library

The system program library is called **BASIC.PL**. While Business BASIC does not supply this file, it does supply an input file, **LIB.CM**, that you can use to create the system library. **LIB.CM** contains the default list of program names used to build the system program library. You can create your own input file, modify **LIB.CM** and use it, or use **LIB.CM** as it comes with Business BASIC.

Example

This example sets up the system program library as a random AOS/VS file with a file element size of 700 blocks, which should be adequate for the default list of program names in **LIB.CM**. (You need more space and a frame size greater than 23 if you plan to use many user modules.)

PLB

Continued

```
* !DIR $SYSLIB
* !CCONT BASIC.PL 700


* !PLB
PROGRAM LIBRARY FILE BUILDER - REV X.XX
LIBRARY NAME [BASIC.PL]?
INPUT FILE [$TRI]? LIB.CM
FRAME SIZE: 23
.
.
.
```

The system displays a verification listing of the modules in the library and then returns you to the Business BASIC prompt for keyboard mode. Now you must terminate Business BASIC and re-execute it. This enables you use the programs in the library.

User Program Library

User program libraries are like system program libraries except that they are usually smaller. You can have one user program library per user. With a user program library you need to create an input file that contains the names of the programs you want in the library as well as their aliases. This can be a file that you supply at the `INPUT FILE [$TRI]?` prompt or you can enter the program names and their aliases interactively.

To use a user program library, open it for reading. Use **STMA 20** to pass the channel number to Business BASIC. This can be done in a program or with statements similar to the following:

```
00010 OPEN FILE (1,4),"MYLIB.PL"      :Open the library.
00020 STMA 20,1                 :Indicate that it is a user library.
00030 SWAP "%ADVENTURE"         :Now use it.
```

This maps the program library to your user channel number. Now you can use channel 1 in a subsequent **OPEN FILE** statement to use the library.

Example

The following example creates a user library called **TMPLIB**. **TMPLIB** is a random file that has a frame size of 7. Note that the programs use aliases (i.e., links and case-sensitive names). The program **ADVENTURE** has an alias link name of **ADVENT**. Pressing the New Line key without entering a filename at the `NAME(S):` prompt ends the **PLB** session.

PLB *Continued*

```
* !DIR $SYSLIB
* RUN "PLB
PROGRAM LIBRARY FILE BUILDER - REV X.XX
LIBRARY NAME [BASIC.PL]? TMPLIB
INPUT FILE [$TRI]:?
CREATING NEW LIBRARY
FRAME SIZE: 7


NAME(S): WUMPUS,wumpus
PROGRAM ADDED:  WUMPUS
LINK ADDED: wumpus
NAME(S): ADVENTURE,ADVENT,adventure,advent
PROGRAM ADDED: ADVENTURE
LINK ADDED: ADVENT
LINK ADDED: adventure
LINK ADDED: advent
NAME(S): TREK,trek
PROGRAM ADDED: TREK
LINK ADDED: trek
NAME(S): TREK.OL,trek.ol
PROGRAM ADDED: TREK.OL
LINK ADDED: trek.ol
NAME(S): CUBIC,cubic
PROGRAM ADDED: CUBIC
LINK ADDED: cubic
NAME(S):


72 BLOCKS IN USE.
*
```

By building **TMPLIB** as a random file, you can determine the size needed for the contiguous library. In the next example, the library is made contiguous for speed and extra size.

```
* !CLI
! LIST TMPLIB
TMPLIB                  36864 UDF 11/11/91 13:51 11/11/91 [003725] 0


! CCONT MYLIB.PL 120
! XFER TMPLIB MYLIB.PL/N
36864 BYTES TRANSFERRED
! DELETE TMPLIB
```

POP

BASIC CLI Command

Terminates the Business BASIC CLI and clears the common area.

AOS/VS	DG/RDOS	UNIX

Format

!POP

What It Does

POP terminates Business BASIC's CLI program and returns you to your previous level. **POP** sets the first word of the common area to <255> and the remainder to <0>. The common area is used to pass Business BASIC CLI commands.

How To Use It

Execute **POP** by entering it from the Business BASIC CLI or by entering it in a command line that you are passing to the Business BASIC CLI (see **CLI**).

Examples

1. You execute the Business BASIC CLI program and then pop back to Business BASIC keyboard mode (indicated by an * prompt).

```
* RUN "CLI
CLI REV.  X.XX
!POP


*
```

2. The following two-line program puts you in the CLI, where it waits until you enter **POP**. That returns you to the program, and the message `CLI POPPED OUT` appears.

```
* 00010 SWAP "CLI
* 00020 PRINT "CLI POPPED OUT"
* RUN
CLI REV. X.XX
!POP
CLI POPPED OUT


*
```

PORTS *Utility*

Displays the processes on the system.

AOS/VS

Format

{ RUN / SWAP / CHAIN } "PORTS

What It Does

PORTS shows you the the processes active on the system, the process ID (PID) of each process, what the processes are running, and the terminal associated with each process.

How To Use It

Execute **PORTS** by entering **RUN**, **CHAIN**, or **SWAP "PORTS**.

Example

Business BASIC displays processes in a tree structure showing the parent/child relationships. Processes with an arrow (-->) are those that are currently active. They are displayed in high-intensity.

PORTS

Continued

```
* RUN "PORTS
:/d
1 (PMGR:PMGR) ..................... :PMGR <321.417 Sec>
2 (OP:OP) ......................... :CLI <4.765 Sec>
   |   3 (OP:EXEC) ................. :UTIL:EXEC <4.405 Sec>
   |   |  4 (OP:LPB) ............... :UTIL:XLPT <10.855 Sec>
   |   |  5 (LINUS:CON2) ........... :CLI <1.000 Sec>
   |   |  13 (SHEILA:CON7) ......... :CLI <1.013 Sec>
   |   |   \__15 (SHEILA:015) ...... -->:UTIL:BBASIC <0.466 Sec>
   |   |       \_17 (SHEILA:017) ... -->:UTIL:$LIB:PORTS<0.123 Sec>
   |   |  14 (CARL:CON13) .......... :CLI <0.471 Sec>
   |   |   \_23 (CARL:023) ......... -->:UTIL:SED <4.978 Sec>
   |   |  16 (MEDLIN:CON9) ......... :CLI <1.024 Sec>
   |   |  19 (KIM:CON26) ........... :CLI <1.376 Sec>
   |   |  20 (SETH:CON3) ........... -->:CLI <4.049 Sec>
   |   |   \_12(SETH:012) .......... -->:UTIL:BBASIC <1.375 Sec>
   |   |  22 (OP:CON24) ............ :UTIL:XLPT <0.128 Sec>
   |___5 (RLS2:RLS2) ............... :UTIL:RLS2 <0.130 Sec>
   |___6 (OP:INFOS_II) ............. :INFOS_II <0.678 Sec>
    \__7 (OP:NETOP) ................ :NET:NETOP <0.369 Sec>
       |  8 (OP:X25) ............... :NET:X25 <30.062 Sec>
       |  9 (OP:RMA) ............... :NET:RMA <1.329 Sec>
       |  10 (OP:SVTA) ............. :NET:SVTA <6.081 Sec>
        \_11 (OP:FTA) ............... :NET:FTA <0.172 Sec>
```

POSFL.SL *Subroutine*

Positions the file pointer to a record in a data file (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

Format

GOSUB { 9610 / 9612 }

Input Variables

C1	The file characteristics array.
F%	The number of the subfile in the C1 array (row in the C1 array).
R1	A record number.
V%	A byte offset into the record (if using entry line 9612).

Output Variables

C%	The physical file's channel number (from C1 array).
R9	The byte position of record R1 in the physical file (useful for rereading or rewriting).
R8	The byte position of the record in the subfile (useful in **LOCK** statements).

Line Numbers

9610	Entry point to position the file pointer to the beginning of record R1.
9612	Entry point to position the file pointer to the byte offset in record R1 specified in V% (0 is first byte in record).
9610-9645	**POSFL.SL** occupies these line numbers.

What It Does

POSFL.SL positions a file pointer to a record or a byte within a record in a PARAM file database structure. Also, you need **POSFL.SL** if you use **GETREC.SL** or **DELREC.SL** (see **GETREC.SL** and **DELREC.SL**).

POSFL.SL gets information from the C1 array. Your file number, F%, corresponds to the row in the C1 array where the information on the file is kept. **POSFL.SL** uses R1 to find the record. If you specify **GOSUB 9612**, **POSFL.SL** positions the file pointer using V% as the offset into the record; otherwise, entry point 9610 positions the pointer to the beginning of record R1.

NOTE: If you use the logical file database structure, use the **LREAD FILE** and **LWRITE FILE** statements instead of **POSFL.SL**. **LREAD FILE** and **LWRITE FILE** combine the positioning and the input/output into a single statement, thus

POSFL.SL

Continued

freeing the code space normally occupied by **POSFL.SL**. Also, these statements perform error checking that is not part of the **READ** and **WRITE** statements used with **POSFL.SL**. (See **LREAD FILE** and **LWRITE FILE** in *Commands, Statements, and Functions in Business BASIC*.)

How To Use It

To use **POSFL.SL**, perform the following steps:

1. Enter "**POSFL.SL** into your program.
2. Build a C1 array and assign the subfile number (the row in the C1 array) to F%.
3. Assign the byte offset to V%.
4. Include either the program statement **GOSUB 9612** to position the file pointer to the byte offset in R1 specified by V% or the program statement **GOSUB 9610** to position the file pointer to the beginning of record R1.

When you use the V% offset, R8 is relative to V% but not to the beginning of the subfile, so be careful about using it in **LOCK** statements.

Example

This segment of code from the program **TEST** uses **POSFL.SL** to position to record R1 with V% as an offset.

```
* LIST
.                              :Code to open files and fill C1 array.
.
.
00130 LET F%=1                 :Subfile 1 (row 1 in C1 array).
00140 LET V%=2                 :Offset 2 bytes from byte 0 (third
                               :byte).
00150 INPUT "RECORD NUMBER; ",R1
00160 GOSUB 09612              :Position to record R1 (using offset
                               :V%).
00170 READ FILE (C%),REC$      :C% returned by POSFL.SL.
.
.
.
* ENTER "POSFL.SL
* SAVE "TEST
```

PRINT

BASIC CLI Command

Prints a text (ASCII) file on the default output queue.

AOS/VS	DG/RDOS

Format

!PRINT *filename ...*

Argument

filename	The name of a text (ASCII) file.

Global Switches

/A	Include permanent symbols in the cross-reference list. Use this switch with **/X**.
/D	Allow duplicate symbols in the cross-reference list. Use this switch with **/X**.
/H	Print a heading at the top of each page.
/O	Override print suppression of library subroutines; used with **/H** or **/X**.
/R	Print a blank line before remark lines in the format "*nnnn* **REM +**" or "*nnnn* **REM **" for easier identification of routines.
/X	Print a symbol cross-reference list at the end of the listing (see the Business BASIC CLI command **TABLE**).

What It Does

PRINT copies the contents of text files to the default output queue where the files are printed. You can use **STMA 9,3** or the Business BASIC CLI command **GQUE** to retrieve the default output queue. **PRINT** works with text files only, so use **FPRINT** on SAVE files.

If your file is a source program with comments, you can also print a cross-reference list of all variables and their corresponding statement numbers using the **/X** switch. This switch executes a **TABLE** command (see **TABLE**). When the **TABLE** command encounters a statement in the form:

REM + *comment*
or
REM*comment*

it does not print the lines following the statement until it encounters another **REM** statement in the same form. This suppresses the printing of library subroutines. Use the **/O** switch to override this suppression.

How To Use It

Execute the command by entering **PRINT** from the Business BASIC CLI. At least one filename must follow **PRINT**.

PRINT

Continued

Example

This command line prints the source file **MAIN.SR** with a heading on each page (**/H**). The **/O** switch allows subroutines that begin with a **REM***comment* statement to be printed also.

```
!PRINT/H/O MAIN.SR
```

PROGPRT

BASIC CLI Command

Prints reference information for a program.

AOS/VS	DG/RDOS	UNIX

Format

!PROGPRT [*outputfile1*/**L**] *filename1* [[*outputfile2*/**L**] *filename2*] ...

Arguments

filename — The name of a program list file created by a Business BASIC **LIST** command.

outputfile/**L** — The name of the output file to which you want to append the listing. This argument overrides the global /**L** switch. Always append /**L** to *outputfile*. If *outputfile* does not exist, Business BASIC creates it. If no *outputfile* argument is given, then the output is directed to the default output queue.

Global Switch

/X — Do not print the variable cross-reference table.

What It Does

PROGPRT takes a program listing file (created by the Business BASIC **LIST** command) and generates reference information that you can use in debugging your Business BASIC program. You can direct this information to a file by using the *outputfile*/**L** argument or to a printer, which is the default output queue. When you use **PROGPRT** without the /**X** switch, the new listing also includes a cross-reference of all variables, channel numbers, some commonly used Business BASIC statements, and their statement numbers.

The information generated by **PROGPRT** is set up so that a blank line precedes each **REM** (remark) statement in the program. Each statement number shows all the **GOTO** and **GOSUB** references for that statement. This way you can follow the flow of the Business BASIC program and find unused code in the program. Statements that do not exist but are referred to by another line are highlighted. In addition, **PROGPRT** prints the program name, date, and time at the beginning of the report, so that you can easily identify your listings.

When you execute **PROGPRT** without the /**X** switch, it includes the statements **BLOCK, CHAIN, DATA, DEF, DIM, ERR, FILE, IKEY, RESTORE, STMA, STMB, STMC, STMD, STME, STMU, STOP,** and **SWAP** in its cross-reference table.

PROGPRT

Continued

How To Use It

Execute the command by entering **PROGPRT** from the Business BASIC CLI. **PROGPRT** must be followed by the name of at least one program. You can also specify an output file to receive **PROGPRT**'s result. If you use the global switch /**X**, append the switch to the command word.

Example

The output for **DEMO.LS**, a list file created with the Business BASIC command **LIST**, is placed in the file **DEMO.FL** (where FL stands for formatted listing). This file can be typed or printed. Entering

!PROGPRT DEMO.FL/L DEMO.LS

produces:

```
00010 REM - PROGPRT Demo program to show PROGPRT's features
00015 ON ERR THEN GOTO 09000
00020 ON IKEY THEN GOTO 09000
00100 REM * SETUP
00110 DEF FNA(X)=OR(X,-AND(X,256))
00120 DEF FNB(X)=OR(X,-AND(X,32768))
00130 LET LOOP1=0
00500 REM
00510 FOR I=1 TO 20
00530   GOSUB 01000 : FUNCTION LOOP
00550 NEXT I
00600 END
01000 REM FUNCTION LOOP
01010 LET K=FNA(I)
01020 GOSUB 02000 : PROGPRT DEMO
01040 RETURN
02000 REM PROGPRT DEMO
02010 PRINT "PROGPRT DEMO"
02020 RETURN
03000 REM * FNB LOOP1
03010 FOR X=1 TO 5
03020   PRINT "FNB = ";FNB(1)
03030 NEXT X
03050 GOTO 00300
09000 REM - Bail out
09010 STOP
```

The output file, **DEMO.FL**, which was produced by **PROGPRT**, contains the following information. Note that the code from line 03000 through 03050 never gets executed and no reference lines are indicated anywhere between 03000-03050.

PROGPRT

Continued

```
!TYPE DEMO.FL

Program ID:  DEMO.LS                                  3/08/91  10:22:08

00010 REM - PROGPRT Demo program to show PROGPRT's features
00015 ON ERR THEN GOTO 09000
00020 ON IKEY THEN GOTO 09000

00100 REM * SETUP
00110 DEF FNA(X)=OR(X,-AND(X,256))
00120 DEF FNB(X)=OR(X,-AND(X,32768))
00130 LET LOOP1=0

****************************************

                                  Referenced at:  03050
00300          STATEMENT NOT FOUND!

****************************************

00500 REM
00510 FOR I=1 TO 20
00530   GOSUB 01000                                : FUNCTION LOOP
00550 NEXT I
00600 END
                                  Referenced at:  00530
01000 REM FUNCTION LOOP
01010 LET K=FNA(I)
01020 GOSUB 02000                                  : PROGPRT DEMO
01040 RETURN

                                  Referenced at:  01020
02000 REM PROGPRT DEMO
02010 PRINT "PROGPRT DEMO"
02020 RETURN

03000 REM * FNB LOOP1
03010 FOR X=1 TO 5
03020   PRINT "FNB = ";FNB(1)
03030 NEXT X
03050 GOTO 00300

                                  Referenced at:  00015 00020
09000 REM - Bail out
09010 STOP
```

PROGPRT

Continued

```
************************************************************

DEMO.LS contains 1 missing statement(s)

************************************************************

=========================================================================

Command Usage

DEF         00110 00120

ERR         00015

IKEY        00020

STOP        09010

Variable References

I           00510 00550 01010

K           01010

LOOP1       00130

X           00110 00120 03010 03030
```

PROTECT *Utility*

Protects Business BASIC SAVE files (programs).

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } **"PROTECT**

What It Does

PROTECT modifies a SAVE file so that you cannot use the **LIST** command to display the file.

UNIX users can use the Business BASIC command **PROTECT** to protect SAVE files. See *Commands, Statements, and Functions in Business BASIC.*

How To Use It

Execute **PROTECT** by entering **RUN**, **CHAIN**, or **SWAP "PROTECT**. The utility then prompts you for the name of the SAVE file that you want to protect. Next the utility asks if you want to change the line numbers as an extra level of protection. Keep the current line numbers if this program is called by another program using a statement that includes the phrase **GOTO** *linenumber*.

To exit **PROTECT**, press the New Line key when the utility prompts you for a filename. On DG/RDOS, the **PROTECT** and **PROTECT.OL** programs are in the **$SYS** and **$SY3** directories. To use **PROTECT**, create links to these programs from the directory that contains the programs you want to protect. If you want all users to be able to use the **PROTECT** utility, create the links in the **$LIB** and **$LIB3** directories.

Example

This example uses the utility to protect the program **PROG1** and to change its line numbers. **PROTECT** halts when you press New Line at the second filename prompt.

```
* RUN "PROTECT

FILE TO BE 'PROTECTED': PROG1
DO YOU WANT TO CHANGE LINE NUMBERS? (Y or N): Y

YOUR FILE IS NOW PROTECTED.

FILE TO BE 'PROTECTED':
*
```

PROTFORM.SL *Subroutine*

Makes an SM formatted screen field protected or unprotected.

AOS/VS	DG/RDOS	UNIX

What It Does

PROTFORM.SL is used with the Screen Maintenance utility. It is explained under **SM**.

Even though UNIX systems do not support **SM**, you can use this subroutine on UNIX systems. If you are using Business BASIC DG mode (specified by including the **-D** option on the command line to execute Business BASIC), you must use 7-bit mode. **SM** screens do not support 8-bit mode. If you are using Business BASIC in non-DG mode, you must specify the **-C** option when you use **SM** screens. This is because the screens contain embedded DG characters.

PRTCOM

BASIC CLI Command

Prints a BLDCOM documentation file.

AOS/VS	DG/RDOS

Format

!PRTCOM *filename* ... [*"symbols"*/**S**]

Arguments

filename	The name of a documentation file produced by **BLDCOM**.
"symbols"/**S**	An optional symbol within quotation marks that is used with the /S switch. This symbol specifies the heading to be printed when it is encountered in the text. You can list several symbols separated by commas within the same set of quotation marks. If you don't use this argument, all the symbols are printed.

Global Switches

/F	Print each module's comments on a new page.
/L	Print comments on the line printer (overrides /O).

Local Switch

/O	When appended to a filename, print comments to this outputfile. This switch is used only with *filename*.

What It Does

PRTCOM prints a documentation file created by **BLDCOM**. When you have one of the following symbols as the second character of a comment symbol (e.g., :!, ;!, C!, etc.), it marks the beginning of specific sections of comments and the following headings are printed:

Symbol	Heading
!	(Module name and optional short description.)
>	CALLING SEQUENCE
#	DESCRIPTION
$	EXTERNAL STORAGE
&	EXTERNAL ROUTINES
%	EXTERNAL PROGRAMS
@	FILES USED

You can specify one or more of these symbols in the *"symbols"*/**S** argument so that you print only selected portions of the documentation file. If you don't use the *"symbols"*/**S** argument, your entire file is printed.

How To Use It

Execute the command by entering **PRTCOM** from the Business BASIC CLI. At least one filename must follow **PRTCOM**, but the argument *"symbols/***S**" is optional. Append global switches either to the command word or to another global switch.

PRTCOM

Continued

To use the *"symbols"*/S argument, place the special characters in your comment file as the second character after the comment character (: for Business BASIC comments, ; for assembly language comments, and C for FORTRAN comments). Use a text editor to enter these comments and symbols in your listing file.

Example

First, use the Business BASIC CLI command **BLDCOM** to create a documentation file containing the library subroutines. Then use **PRTCOM** to display the documentation file.

```
* !BLDCOM/V -.SL
DOCUMENTATION FILE: TMP
GETCM.SL
GETLAST.SL
INITINDEX.SL

* !PRTCOM TMP
GETCM          (GETCM.SL)

D E S C R I P T I O N
      INITCM reads the common area into T9$ and sets Q9 to point
      to the first argument (Command name and global switches).
      Each GETCM returns the next argument in X$ and switches in
      S.  S is returned as -1 after all arguments have been read.
      This routine will also parse COM.CM generated by RDOS CLI.
      T9$ and Q9 must be preserved between calls to GETCM.

C A L L I N G   S E Q U E N C E
      GOSUB 7500 : READ NEXT COMMAND ARGUMENT
      X$ - CONTAINS FIELD READ
      S  - SWITCHES FOR THIS FIELD (=-1 IF END OF COMMAND LINE)

E X T E R N A L   S T O R A G E
      T9$ - 512 BYTE STRING FOR HOLDING COMMON (PERMANANT)
      Q9 - POINTER INTO T9$ (PERMANANT)

GETLAST.SL     (GETLAST.SL)

D E S C R I P T I O N
      Returns the last record number from record 0 of a linked file.
```

PRTCOM

Continued

```
C A L L I N G   S E Q U E N C E
      F%     - Logical file number

      GOSUB 09950

      LSTREC - Last record number
      ACTREC - Number of active records currently in file (only if
               record length is greater than or equal to 14 bytes)
      E      - Error flag as follows:
               E  =  0     Normal return
               E  <> 0     Appropriate error code, indicates one of

E X T E R N A L   S T O R A G E

      X      - Scratch variable

      INITCM    (GETCM.SL)

D E S C R I P T I O N
      INITCM initializes T9$ and Q9 for GETCM.

C A L L I N G   S E Q U E N C E
      GOSUB 07550

      INITINDEX    (INITINDEX.SL)

D E S C R I P T I O N
      (Modified for BBIV double & triple)
      INITINDEX creates block 0 and block 1 of an index file.

C A L L I N G   S E Q U E N C E
      Y$  - File descriptor string (as described in SEC 9.3.2)
      T9$ - Scratch string (MUST BE DIM'ED TO AT LEAST 512 BYTES)
      X   - Key length in bytes
      Y   - Duplicate-key-flag (=1 if duplicate keys are allowed)
      YY  - Set to 1 for 2048 size blocks or 0 for 512.
      Z   - Blocking factor
      X1  - Size of file (In blocks)
      GOSUB 7700

E X T E R N A L   S T O R A G E
      Y1  - Temporary
      YY0 - Temporary
      YY1 - Temporary
      Z1  - Temporary

*
```

QFILESORT *Utility*

Quickly sorts a data file.

AOS/VS	DG/RDOS	UNIX

Format

SWAP "QFILESORT

What It Does

QFILESORT quickly sorts a data file in ascending order on a single unsigned alphanumeric key. Unlike **FILESORT**, **QFILESORT** does not sort on multiple keys or allow other kinds of collating sequences (see **FILESORT**). Also, **QFILESORT** does not check for deleted records.

How To Use It

Do not use **RUN** or **CHAIN** with **QFILESORT**, only **SWAP**. To execute **QFILESORT**, perform the following steps in your program:

1. Open your data file to assign it a channel number.
2. Set up an argument string and use a **BLOCK WRITE** statement to place it in the common area (see Table 1-14). The argument string is the same as for **FILESORT**, except that substring locations 12,12 (sort modifier to check for deleted records) and 18,18 (key type to sort in a different order) must have zero values.
3. Include the program statement **SWAP "QFILESORT**.
4. Use a **BLOCK READ** to get the string from the common area and check for an error. The error code is returned in the first two bytes of the string, and the rest of the string is unused. A zero indicates a successful sort, a negative value means an operating system or I/O error, and a positive value is a Business BASIC error (see Appendix B).

All arguments are considered binary. The **QFILESORT** Argument String table explains the arguments and their locations in the string.

QFILESORT

Continued

Table 1-14. QFILESORT Argument String

Location	Size	Contents
1,1	1	Channel number of file you want to sort
2,5	4	Number of records you want to sort. Records must exist in the file.
6,7	2	Record size in bytes
8,11	4	Byte offset to first record if data file is a subfile
12,12	1	Must have a zero value (does not check for deleted records)
13,13	1	Number of key fields in record (always 1)
14,15	2	First byte of key (1 is first byte of record)
16,17	2	Last byte of key
18,18	1	Sorting order. Must have a zero value since **QFILESORT** only sorts in ascending order.

Example

This program segment uses X$ to hold the argument string. The string is filled in lines 100 through 140 and then placed in the common area with a **BLOCK WRITE** in line 150. The program then swaps to **QFILESORT**. When the utility finishes, the program gets X$ from the common area and, in line 180, checks the string to see if the sort was successful.

Since **QFILESORT** does not check for deleted records, the number you enter for records in file should be the number of records actually written to the file instead of the maximum number of records for that file. In this case, the file could have a record limit of 100 records, but only 45 have been written to the file so only 45 can be sorted.

QFILESORT

Continued

```
00010 DIM X$[512]                        :X$ to get argument string for
                                         :common.
.                                        :Put code to open the file
00100 LET X$=CHR$(0),CHR$(45,4)          :Channel number; # of records
                                         :in file.
00110 LET X$[0]=CHR$(32,2),CHR$(32,4) :Bytes per record; offset-skip
                                         :record 0.
00120 LET X$[0]=CHR$(0),CHR$(1)          :Always zero; # of key fields
                                         :- always 1
00130 LET X$[0]=CHR$(3,2),CHR$(6,2)      :Key field bytes 3 to 6
00140 LET X$[0]=FILL$(0)                 :No more arguments
00150 BLOCK WRITE X$                     :Send into common area
00160 SWAP "QFILESORT                    :Execute QFILESORT and return
00170 BLOCK READ X$                      :Read common area
00180 PRINT ASC(X$[1,2])                 :Check for errors
.
.
.
```

QUICKILL *Utility*

Terminates Business BASIC without warning messages.

DG/RDOS

What It Does

Since this utility can only be used on DG/RDOS systems by someone with system manager privileges, **QUICKILL** is explained in the *Business BASIC System Manager's Guide*.

QUIT

BASIC CLI Command

Ends the BASIC CLI program but retains the common area.

AOS/VS	DG/RDOS	UNIX

Format

!QUIT

What It Does

QUIT performs a **POP** but does not clear the common area (see **POP**). Use **QUIT** to terminate the Business BASIC CLI without affecting the contents of the common area so you can examine it when debugging your own CLI commands. You can use **GETCM.SL** or **BLOCK READ** to get information from the common area.

How To Use It

Execute the command by entering **QUIT** from the Business BASIC CLI.

Example

Once in the CLI, you issue a **QUIT** to return to Business BASIC keyboard mode.

```
*RUN "CLI
CLI REV X.XX
!QUIT
*
```

RELEASE

BASIC CLI Command

Releases a device or directory.

DG/RDOS

Format

!RELEASE *item*

Argument

item Either the name of a disk, tape drive, or other suitable device or the name of any subdirectory or partition except the master directory.

What It Does

RELEASE closes any disk or tape drive that has been initialized. You must release a disk before you remove it from the disk unit. **RELEASE** automatically rewinds a magnetic tape. After a **RELEASE** command, the system prevents access to the directory or device until you initialize it with an **INIT** or **DIR** command.

When used on a partition, the command also releases all subordinate directories. You can release a partition or directory to prevent input/output access to it.

RELEASE does not accept the master directory as a valid argument. When you release your current directory, the system asks you to verify that action. If you respond **Y** to this query, the master directory becomes your current directory.

How To Use It

Execute the command by entering **RELEASE** from the Business BASIC CLI. **RELEASE** must be followed by the argument *item*.

Examples

1. Disk unit **DP1** is released, so that you can remove the disk pack from drive 1.

   !RELEASE DP1

2. Magnetic tape unit 0 is released, and the tape is rewound.

   !RELEASE MT0

RELEASE

Continued

3. **RELEASE** closes the partition **USER.DR** and its subdirectories to I/O access. You do not need to enter the .DR extension when you use a partition or directory name with **RELEASE**.

   !RELEASE USER

4. The current directory (**MYDIR**) is released, so the system puts you in the system directory (**DE0**).

   !RELEASE MYDIR
   Do you really want to RELEASE the current directory (Y or N)? **Y**
   Your directory is now: DE0
   !

RELINK *Utility*

Recreates the deleted record chain for a linked-available-record file (PARAM file structure).

AOS/VS	DG/RDOS	UNIX

Format

```
{ RUN   }
{ SWAP  } "RELINK
{ CHAIN }
```

Or

!RELINK *filename*

Arguments

filename The name of a subfile or a physical data file.

What It Does

RELINK is an interactive utility that recreates the deleted record chain for a linked-available-record file. **RELINK** first searches the PARAM file for name of the data file you specified. If there is no PARAM entry for the file (i.e., it's a physical file), **RELINK** asks you for the byte offset to the file's record 0, the record size, and the maximum number of records. **RELINK** then searches for deleted records (indicated by a 0 in the first two bytes of the record) and puts them on the new deleted-record chain.

Run **RELINK** if your system crashes while a linked-available-record file is being updated. Then check the record you modified last to be sure you did not lose any information during the crash.

How To Use It

Execute **RELINK** by entering **RUN**, **CHAIN**, or **SWAP "RELINK**. This starts the interactive mode of **RELINK**. You can also use a command line format where you enter **RELINK** and a filename from the Business BASIC CLI.

Examples

1. When **RELINK** is run, it enters interactive mode and prompts you for information about the file. In this case, since the file does not have a PARAM entry, **RELINK** also asks for the byte offset, the record size, and the last record number in the file.

```
* RUN "RELINK
FILENAME: CUST
```

RELINK *Continued*

```
BYTE OFFSET: 0
RECORD SIZE(BYTES): 200
LAST RECORD NUMBER: 1000


*
```

2. **RELINK** is executed in command line format from the Business BASIC CLI to recreate the deleted record chain for the data file **EMPLOYEE**. **EMPLOYEE** has a PARAM file entry, so **RELINK** does not prompt you for any information.

   !RELINK EMPLOYEE

RENAME

BASIC CLI Command

Changes the names of files.

AOS/VS	DG/RDOS	UNIX

Format

!RENAME *oldname1 newname1* [*oldname2 newname2*] ...

Arguments

oldname	The name of a file in the current directory.
newname	A new name for the file.

What It Does

RENAME changes *oldname* to *newname* and deletes *oldname* so that it no longer exists in the current directory.

On DG/RDOS systems, **RENAME** fails if the **P** (permanent) attribute is set on the file.

On AOS/VS systems, having PERMANENCE turned on does not block a **RENAME** command; however, improper access control lists (ACLs) can block a **RENAME**.

How To Use It

Execute the command by entering **RENAME** from the Business BASIC CLI. At least two arguments, first the old name of the file and then the new name, must follow **RENAME**. You can **RENAME** more than one file at a time as long as you separate the *oldname/newname* argument pairs by spaces.

Example

RENAME the file **MIKE.SR** (in the current directory) to **MARK.SR**. When **RENAME** finishes, only **MARK.SR** is listed in the directory.

```
!RENAME MIKE.SR MARK.SR
```

RENUM *Utility*

Renumbers selective lines of a program listing file and removes REM statements.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "RENUM

What It Does

RENUM lets you change (renumber) specified line numbers within your program listing file. **RENUM** also adjusts the **GOTO** and the **GOSUB** statements when it renumbers these lines. Because you specify the line numbers to be changed, the utility **RENUM** differs from the command **RENUMBER**, which always renumbers entire programs and sets any unresolved references to 00000.

RENUM preserves the original file in case you number statements so that they overlap in the new file. The utility warns you when this happens. You can deliberately overlap statements to relocate code segments; however, this is valid only when the new file is entered into Business BASIC.

If you renumber files with unresolved references (i.e., references to code that is not part of that file), **RENUM** leaves those statement numbers unmodified. **RENUM** warns you when this happens.

You can use **RENUM** to remove all **REM** statements from the program. The utility adjusts the **GOTO** and **GOSUB** statements as necessary to compress the program. The remarks and the original form of any modified **GOTO** or **GOSUB** statements are copied into another file so that you can re-enter them later to reproduce the program in its original form but with new numbers.

Although **RENUM** resolves statement numbers in all the usual Business BASIC statements, it ignores the contents of **REM** statements. Thus, **RENUM** does not resolve **IOBUILD** statements with **RECORD**, **ERR=** and **EOF=** clauses.

How To Use It

Execute **RENUM** by entering **RUN**, **CHAIN**, or **SWAP "RENUM**. This starts the **RENUM** dialog. The name of the output file must differ from the name of the input file. Also, the **RENUM** dialog abbreviates "statement" number as "stno."

You can continue renumbering segments of a program until you enter **N** in response to the prompt `Additional renumbering (Y,N) [N]?`

RENUM *Continued*

Example

Before using **RENUM**, the program file for **MENU** looks this way:

```
* LIST
00200 REM ** DISPLAY MENU    :listing of original program, MENU
00205 PRINT @(-30);
00220 PRINT @(1,30);"MASTER MENU"
00225 PRINT @(2,30);"==========="
00228 PRINT @(5,18);"OPTION";@(5,27),"PROGRAM DESCRIPTION"
00229 REM
00230 PRINT @(8,20);"1    --   CREATE FILES"
00240 PRINT @(9,20);"2    --   INQUIRE ON BADGE NUMBER"
00260 PRINT @(10,20);"3    --   INQUIRE ON PRODUCT"
00270 PRINT @(15,20);"8    --   PURGE FILES PRIOR TO CREATING"
00280 PRINT @(16,20);"9    --   END"
00300 INPUT USING "",@(22,20),"ENTER OPTION #:  ",X
```

Since **RENUM** needs an input file, list the above program to the file **MENU.LS**. When you execute **RENUM**, it also asks you for the output file name, the **REM** file name, where to start and end renumbering, and what increment to use. This example uses an input file name of **MENU.LS**, an output file of **MENU.RN**, and a **REM** file **MENU.RM**. The renumbering begins with statement number 200 and continues to statement number 260. The first new statement number is 200 and the increment from that number is by 5 until the old statement number 260 is reached. Since **Y** is entered in response to the additional renumbering query, the dialog starts again. Once you end **RENUM**, you can type output file to check the changes. You can also type the **REM** file. That file contains all the **REM** statements that were removed from the output file.

```
* LIST "MENU.LS
* RUN "RENUM
RENUM - REV X.XX
 Input file: MENU.LS
Output file: MENU.RN
REM file (if you want output stripped) MENU.RM
From old stno [00001] 200 to old stno [99999] 260
New stno [00010] 200 increment [+10] 5
Pass 1
 12 Statements found
Highest statement #  300
Last statement renumbered: 260 -> 240
Additional renumbering (Y,N) [N]? Y
From old stno [00001] 270 to old stno [99999] 300
New stno [00010] 290 increment [+10] 5
Last statement renumbered: 280 -> 295
Additional renumbering (Y,N) [N]? N
PASS 2
```

RENUM *Continued*

```
* !TYPE MENU.RN
00205 PRINT @(-30);
00210 PRINT @(1,30);"MASTER MENU"
00215 PRINT @(2,30);"==========="
00220 PRINT @(5,18);"OPTION";@(5,27),"PROGRAM DESCRIPTION"
00230 PRINT @(8,20);"1    --    CREATE FILES"
00235 PRINT @(9,20);"2    --    INQUIRE ON BADGE NUMBER"
00240 PRINT @(10,20);"3    --     INQUIRE ON PRODUCT"
00290 PRINT @(15,20);"8    --     PURGE FILES PRIOR TO CREATING"
00295 PRINT @(16,20);"9    --     END"
00300 INPUT USING "",@(22,20),"ENTER OPTION #:   ",X

* !TYPE MENU.RM
00200 REM ** DISPLAY MENU    :listing of original program, MENU
00225 REM
```

NOTE: UNIX systems support numbers up to 99999. AOS/VS and DG/RDOS systems, however, support line numbers as high as 32767. Thus, if you had executed this example on an AOS/VS or DG/RDOS system, the seventh and fourteenth lines would have looked like:

```
From old stno [00001] 200 to old stno [32767] 260

....

From old stno [00001] 270 to old stno [32767] 300
```

where the user enters the numbers 260 and 300.

RNAM *Utility*

Changes the names of program variables.

AOS/VS	DG/RDOS

Format

```
{ RUN  }
{ SWAP } "RNAM
{ CHAIN}
```

What It Does

RNAM is an interactive utility that assigns new names to program variables in working storage or in a SAVE file. It also tells you the number of variables and the program name (you must give it the name of the program file).

NOTE: On UNIX systems, you can use the **VAR RENAME** command to change the names of program variables. See *Commands, Statements, and Functions in Business BASIC* for more information this command.

How To Use It

Execute **RNAM** by entering **RUN**, **CHAIN**, or **SWAP "RNAM**. To rename program variables in a SAVE file, use either **RUN** or **CHAIN "RNAM** because **SWAP "RNAM** operates on the program in working storage. Once called, the utility queries you for information.

RNAM displays the program name and the number of variables in your working storage. After this, **RNAM** asks for an old variable name. Enter an existing variable name. **RNAM** then asks for the new name. Enter the new name for the variable.

These questions are repeated until you press CR without entering a name in response to the old variable name question. Optionally, you can **RUN** or **CHAIN "RNAM** and change the names of variables in a SAVE file. **RNAM** asks for the program name before entering the same interactive dialog it uses when you swap to it.

When you run a program with **DIM** statements followed by a **STOP** and then delete the **DIM** statements without saving them in the program, **RNAM** does not function properly. In that case, **RNAM** renames the variables, but the space allocated for them is still referred to under the old name, generating "undimensioned variable" errors at runtime. Then, if you add new **DIM** statements to the program, you waste the space that was allocated for the old variables. To save space and avoid this error, list the program, re-enter it, and then save it again before using **RNAM**.

RNAM *Continued*

Example

The following program is in working storage.

```
* LIST
00005 DIM F$[80]
00008 LET F$="D3.0,2X,'POWER OF 2:',D11.0,5X,"
00009 LET F$[0]="'BIT-WORD',D3.0,'  BIT-BYTE',D2.0"
00010 FOR I=0 TO 31
00020     LET J=2^I
00030     LET K=MOD(I,16)
00040     LET L=MOD(I,8)
00050     PRINT USING F$,I,J,K,L
00060 NEXT I
```

To change the variable names for a program in working storage, swap to **RNAM**. **RNAM** responds with the program name and the number of variables. It then prompts you for the old variable name. Once you enter that, **RNAM** prompts you for the new variable name. This continues until you press CR without supplying an old variable name.

```
* SWAP "RNAM
SCRATCH              5 VARIABLES
OLD NAME: I          NEW NAME: BIT
OLD NAME: J          NEW NAME: POWER2
OLD NAME: K          NEW NAME: WDBIT
OLD NAME: L          NEW NAME: BYTBIT
OLD NAME: F$         NEW NAME: FMT$
OLD NAME:
```

To confirm the changes, list the program again.

```
* LIST
00005 DIM FMT$[80]
00008 LET FMT$="D3.0,2X,'POWER OF 2:',D11.0,5X,"
00009 LET FMT$[0]="'BIT-WORD',D3.0,'  BIT-BYTE',D2.0"
00010 FOR BIT=0 TO 31
00020     LET POWER2=2^BIT
00030     LET WDBIT=MOD(BIT,16)
00040     LET BYTBIT=MOD(BIT,8)
00050     PRINT USING FMT$,BIT,POWER2,WDBIT,BYTBIT
00060 NEXT BIT

*
```

SCHANS *Utility*

Displays the system channel assignments.

AOS/VS	DG/RDOS

Format

```
{ RUN  }
{ SWAP } "SCHANS
{ CHAIN}
```

What It Does

SCHANS displays the current system channel assignments for the Business BASIC system.

On DG/RDOS systems, the **SCHANS** display takes up four columns: the first is the channel number; the second, the user status table address; the third, the filename of the physical file opened on that channel; and the fourth, the user ID of the job associated with the file. Channels in use by the Business BASIC system do not have an ID.

On AOS/VS systems, **SCHANS** displays only the channel number and the filename of the Business BASIC process that is executing.

How To Use It

Execute **SCHANS** by entering **RUN**, **CHAIN**, or **SWAP "SCHANS**.

Examples

1. On an DG/RDOS system, **SCHANS** provides a four-column display. Since the Business BASIC system is using channel numbers 0 through 2 (shown in the first column), no user ID for those files appears in the fourth column. The status table address shows up in the second column, and the filename is given in the third column. Since the file **JUNK** was opened before **SCHANS** was run, it appears on channel 3 with the user ID AAJWC6.

```
* OPEN FILE(0,0),"JUNK"
* RUN "SCHANS

0          32768 BBASIC.ER
1          32768 BASIC.PS
2          32768 .
3          9405  JUNK          AAJWC6

*
```

SCHANS *Continued*

2. On an AOS/VS system, **SCHANS** gives the channel number and the filename of the current channel assignments.

```
* OPEN FILE(0,0),"JUNK"
* RUN "SCHANS
34          :UTIL:BBASIC:BBASIC.OL
35          :UTIL:RLS2.COMM
36          :PER:CON2
37          :PER:CON2
38          :UTIL:BBASIC:$SYSLIB:BASIC.ER
40          :UTIL:BBASIC:$SYSLIB:BASIC.PR
41          :UDD:DELAMAR:JUNK

*
```

SCRNIO.SL

Subroutine

Enables edited screen input/output for CSM screens.

AOS/VS	DG/RDOS	UNIX

What It Does

SCRNIO.SL is part of the Conversational Screen Maintenance utility. It is explained under **CSM**.

SDIR

BASIC CLI Command

Sets the system directory.

DG/RDOS

Format

!SDIR *directory*

Argument

directory The name of a directory without the .DR extension. This is the directory that you want to set as the system directory.

What It Does

SDIR changes the Business BASIC system directory to the directory specified. The system directory is the directory from which you executed Business BASIC. It is the directory where your Business BASIC system or links to it reside. The Business BASIC CLI command **GSDIR** displays the name of your system directory.

NOTE: The system directory can also be changed if you use the DG/RDOS **DIR** command. In that case, the system directory becomes the directory you change to with. To avoid this, use the Business BASIC CLI **DIR** command to change directories.

How To Use It

Execute the command by entering **SDIR** from the Business BASIC CLI. The name of a directory (without the .DR extension) must follow **SDIR**.

Example

Change the current system directory to **BBASIC.DR**.

```
!GSDIR
GILES
!SDIR BBASIC
!GSDIR
BBASIC
```

SFORM.SL

Subroutine

Provides formatted handling of fields for the CSM screens.

AOS/VS	DG/RDOS	UNIX

What It Does

SFORM.SL is part of the Conversational Screen Maintenance utility. It is explained under **CSM**.

SIZE *Utility*

Displays the working storage space allocations.

AOS/VS	DG/RDOS

Format

SWAP "SIZE

What It Does

The **SIZE** program displays the working storage space allocations for the job swapping to **SIZE**. The data's size appears in both byte and page quantities. The page size is determined by whether or not the system is window mapped. In a window mapped environment, the size is 2048 bytes; otherwise, the page size is 512 bytes. Due to Business BASIC's page allocation method, a working storage without anything in it has either 1024 or 4096 bytes of memory assigned to it.

NOTE: On UNIX systems, you can use the **SIZE** command to display the working storage space allocations. See *Commands, Statements, and Functions in Business BASIC* for more information this command.

How To Use It

Execute the utility by entering **SWAP "SIZE**. Do not use **RUN** or **CHAIN** with this utility or execute it from the Business BASIC CLI. To display the working storage for a SAVE file, load the file into working storage before you execute **SIZE**.

Examples

1. On an DG/RDOS system, **SIZE** displays the space allocations for the program **CSM**, which has been loaded into working storage.

```
* LOAD "CSM
* SWAP "SIZE

PROGRAM NAME: CSM
PAGE SIZE:              512   BYTES
PROGRAM SIZE:            11   PAGE(S),     564   bytes
DATA SIZE:                1   PAGE(S),     490   bytes
TOTAL SIZE:              12   PAGE(S),    6054   bytes
MAXIMUM:                 48   PAGE(S),   24576   bytes
REMAINING:               36   PAGE(S),   18522   bytes
```

SIZE

Continued

2. This **SIZE** example displays the space allocated for an AOS/VS Business BASIC system.

```
* LOAD "CSM
* SWAP "SIZE

PROGRAM NAME: CSM
PAGE SIZE:              2048   BYTES
PROGRAM SIZE:              3   PAGE(S),    5564   bytes
DATA SIZE:                 1   PAGE(S),     490   bytes
TOTAL SIZE:                4   PAGE(S),    6054   bytes
MAXIMUM:                  14   PAGE(S),   28672   bytes
REMAINING:                10   PAGE(S),   22618   bytes
```

SLINE *BASIC CLI Command*

Selects a port and attaches job to it.

DG/RDOS

Format

!SLINE *port*/**L** [*width*/**W**]

Arguments

port/**L**	The number of the port (terminal or printer line) that you want the current job to attach to. The **/L** switch is required with this argument.
width/**W**	The page width that you want on the destination port. The default is 80. This argument is optional, but when it is used it requires the **/W** switch.

Global Switch

/P	Issue a form feed on the attached port.

What It Does

SLINE attaches the current job to the port number indicated by *port*. You can set the page width for the destination port and send a form feed to the port. Normally, **SLINE** is used in conjunction with the Business BASIC CLI command **START** to execute jobs on other terminals.

How To Use It

Execute the command by entering **SLINE** from the Business BASIC CLI. **SLINE** must be followed by the *port*/**L** argument, but the *width*/**W** argument is optional. If you use the global switch, append it to **SLINE**. The specified port must not have another job running on it. Find out which ports are in use by running the **STAT** CLI command.

Example

1. This starts the program **MYPROG.SR** running on the terminal at line 10. First, the command line starts the CLI, then it attaches the job to the terminal on line 10 with a page width of 132 characters so that after a form feed the program is displayed at the terminal. Then the command issues a **BYE** to return to its previous state.

   !START CLI/N "SLINE/P 10/L 132/W;TYPE/H MYPROG.SR;BYE"

2. This starts the Business BASIC CLI, runs **MYPROG.SV** on the terminal on line three, and then logs off Business BASIC.

   !START CLI/N "SLINE 3/L;MYPROG.SV;BYE"

SM

Utility

Creates and maintains a screen file.

AOS/VS	DG/RDOS

Format

```
{ RUN  }
{ SWAP } "SM
{ CHAIN}
```

What It Does

The Screen Maintenance (**SM**) utility is used to create and edit screen files on Data General terminals that have function keys. The screen files can contain multiple screen records. Each record represents one screen.

These screen files can be used with user-written programs or with the Business BASIC File Maintenance (**FM**) utility. If you want to use an **SM** screen with **FM**, enter the screen filename (with no extensions) in record 1 of the **FM** table file. Otherwise, **FM** uses the default **FM** screen.

With **SM**, you specify each character position on a screen as either a prompt field or an input/output field. The prompt fields are screen literals. When the screen is used by a program, the prompt fields appear exactly as you typed them. Use these fields to prompt the user for information or to explain something to the user. These fields are considered protected fields because you can use cursor placement routines to keep a user from entering data in them. Prompt fields appear in low-intensity on your screen.

The input/output fields contain field definition characters to indicate the field format (for example, numeric with a floating decimal point). You use function keys and field definition characters to set up these fields. These fields receive input from the user and/or display the output. These fields are considered unprotected fields because the user can input data in them. Input/output fields appear in high-intensity on your screen.

The Business BASIC subroutines **FORM.SL**, **PROTFORM.SL**, and **UNFORM.SL** access **SM** screens. These subroutines control the cursor placement and input requests. Use these subroutines in your program to display the **SM** screens, solicit input from the user, and wait for the user to press certain function keys to enter the input (see **FORM.SL**, **PROTFORM.SL**, and **UNFORM.SL** in this section).

The screens created by **SM** are not compatible with the screens created by the Conversational Screen Maintenance utility (**CSM**).

SM does not support 8-bit mode because **SM** screen files contain embedded control characters. (**CSM** does support 8-bit mode.)

How To Use It

Since you use **SM** function keys, commands, field definition characters, and the subroutines **FORM.SL**, **PROTFORM.SL**, and **UNFORM.SL** with this utility, more detailed information on these subjects follows the general overview of executing **SM**.

SM

Continued

Overview

Execute **SM** by entering **RUN, CHAIN,** or **SWAP "SM**. This starts the **SM** dialog. **SM** first asks for a screen filename. Enter the name of the file you are creating or editing. If you are creating a screen to use with **FM**, the name of the screen file must have an .S*n* extension, where *n* is the terminal type. (With **FM**, you omit the .S*n* extension when you add the filename to the table file. **FM** appends the extension to the filename when it reads the file from the table.)

After you enter the filename, **SM** comes up in command mode. The utility clears the screen and displays two angle brackets (>) with the cursor positioned next to them at the bottom line. This is the **SM** command line.

To set up a screen, use cursor positioning keys such as up-arrow to go to the correct field position. You can enter an input/output field by typing the field definition characters on the screen without pressing any function keys (see the section on **SM** field definition characters). For a prompt field, press the START PROTECT function key (F6), type in the information, and then press END PROTECT function key (F5) (see the section on **SM** function keys). You can define a field at any place on the screen except the bottom line, which **SM** reserves for command entry and error messages.

When you finish setting up the screen, press the ENTER COMMAND function key (F4). This puts you on the command line. Enter a command and then press the PROCESS COMMAND function key (F11). **SM** ignores the command unless you press F11 (see the section on **SM** Commands). Each time you create or change a screen, you must enter a command to save the screen; otherwise, no changes are entered into the screen file.

To edit an existing file, enter the appropriate **SM** command; then press F11 to process the command. Since **SM** comes up in command mode, you don't need to press F4 first. If a file contains multiple screens, use the screen number in your commands to refer to specific screens. The first screen number is always zero.

SM Commands

SM uses five commands: **READ, WRITE, DISPLAY, PRINT,** and **STOP**. You execute these commands from command mode by typing the command and then pressing the PROCESS COMMAND function key (F11). If you are not in command mode, press the ENTER COMMAND function key (F4). This returns you to command mode.

If the screen file contains more than one screen, enter the screen-number with the command. The default *screen-number* is the last screen referred to. You don't need to enter data for the documentation fields *system-name*, *project-name*, and *program-name*.

DISPLAY [*screen-number*]

Displays the screen specified by *screen-number* without the format characters. The default screen number is the last screen referred to.

SM

Continued

PRINT [*filename*]

Writes the current screen to *filename*. If no filename is specified, the current screen is printed at the default output queue.

READ [*screen-number*]

Displays the screen specified by *screen-number*. The default screen number is the last screen referred to.

STOP

Closes the screen file and terminates an **SM** session.

WRITE [*screen-number*] [*system-name,project-name,program-name*]

Writes the screen displayed on the terminal to the record specified by *screen-number*. The default screen number is the last screen referred to. *system-name*, *project-name*, and *program-name* are optional arguments.

SM Function Keys

SM uses function keys F1 through F11. It also uses the functions associated with pressing the Shift key and a function key (F1 through F4). This section explains the function keys.

DELETE CHARACTER (Shift-F2)

Deletes a character at the current cursor location.

DELETE LINE (Shift-F4)

Deletes the current line.

DUPLICATE TO EOL (F1)

Duplicates the previous line.

END PROTECT (F5)

Ends a prompt field. Characters following this key appear in high-intensity and indicate the type of data expected for the field. This field is not protected.

ENTER COMMAND (F4)

Positions the cursor to the command line next to the double angle brackets so that **SM** commands can be entered.

INSERT CHARACTER (Shift-F1)

Inserts a space at the current cursor location.

INSERT LINE (Shift-F3)

Inserts a blank line below the current cursor location.

PROCESS COMMAND (F11)

Executes or processes commands entered on the **SM** command line. If you are not in **SM** command mode, the F4 key positions you to the command line where you can specify an **SM** command such as **READ**, **WRITE**, **PRINT**, **DISPLAY**, or **STOP**. Pressing F11 initiates processing of the command.

SM

Continued

PROTECT TO EOL (F2)

Defines all positions from the cursor to the end of the line as a display-only field when used after F6 (the start protected field function key).

SCALE TO EOL (F3)

Fills the line from the current cursor location to the end of line with a repeating scale of the digits 1234567890... The cursor then returns to the first number so that text can be entered over the scale.

START PROTECT (F6)

Starts a prompt field. Characters following this key appear in low-intensity. This field is protected.

TAB TO NEXT

Tabs to the next input/output field.

VERIFY SCREEN

Displays what was last shown on the screen so you can verify the information. After a **DISPLAY** command, this key shows only prompt fields.

Table 1-15. SM Function Key Summary

Function Key	Operation
F1	DUPLICATE TO EOL
F2	PROTECT TO EOL
F3	SCALE TO EOL
F4	ENTER COMMAND
F5	END PROTECT
F6	START PROTECT
F9	TAB TO NEXT
F10	VERIFY SCREEN
F11	PROCESS COMMAND
Shift-F1	INSERT CHARACTER
Shift-F2	DELETE CHARACTER
Shift-F3	INSERT LINE
Shift-F4	DELETE LINE

SM

Continued

SM Field Definition Characters

Field definition characters specify the type of data a user can enter in an input/output field and/or data that is output by that field (see **SM** example 1). The user cannot enter data into an output-only field.

Field definition for input/output fields and screen composition usually are done simultaneously; however, you can edit fields later if you prefer. The field definition characters determine the field's format. Use these field definition characters if you are going to use **UNFORM.SL** or **FM**.

Table 1-16. Field Definition Characters Summary

Field Code	Meaning
8,9	Numeric field: 8 Input/output fields. 9 Output-only fields.
-	Field allowing negative numeric entries.
.	Decimal point location in a numeric field requiring a fixed number of decimal places on input.
:	Implied location of a floating decimal point. On input, the decimal point keyed in is scaled to the specified implied location.
>	Right-justified field. The > must appear in the first column of the field.
L,M	Lowercase alphanumeric field: L Input/output fields. M Output-only fields.
X,Y	Alphanumeric field: X Input/output fields. Y Output-only fields.
C,D	Alphanumeric crammed field: C Input/output fields. Input is crammed after it is entered, and output is uncrammed before it is printed. D Display-only fields. Output is uncrammed before it is printed.

SM

Continued

You can define display-only numeric and alphanumeric fields by using a special set of field characters. For numeric fields, use the character 8 to indicate an input/output field or 9 to indicate an output-only field. If the field is negative, enter a hyphen (-) in the first position of the field. A period (.) or a colon (:) marks the location of an assumed decimal point.

The period (.) indicates that you must enter exactly the number of digits specified to the right of the decimal point. Thus, for a field defined as 8888.88, an input value of 12.34 is valid and returns a value of 1234, while an input value of 1.234, 123.4, or 1234 results in an error.

The colon (:) means that the number of digits after the decimal point can vary. Thus, for a field defined as 8888:88, an input value of 12.3 returns 1230 while an input value of 12.345 or 12.34567 returns a value of 1234. Note that an input of 1234 returns a value of 123400.

You define an alphanumeric field by entering Xs for input/output fields and Ys for output-only fields. An alphanumeric field that allows lowercase characters is defined by Ls for input/output fields and Ms for output-only fields. If a field contains a C, input is crammed after it is entered and output is uncrammed before it is printed. Use a D to cram a output-only field; the output is uncrammed before it is printed.

To indicate a field that should be right-justified, type an angle bracket (>) in the first column of the field.

SM Subroutines

The subroutines **FORM.SL**, **PROTFORM.SL**, and **UNFORM.SL** work in conjunction with **SM** as follows:

FORM.SL	Handles formatted fields, data, and error messages.
PROTFORM.SL	Makes an **SM** formatted screen field protected or unprotected.
UNFORM.SL	Positions the cursor to the input/output fields and handles unformatted fields.

The subroutines share scratch variables, which are summarized below. The input and output variables that are specific to a subroutine are listed as arguments to the subroutines. Each of the subroutines is explained in this section.

SM

Continued

Scratch Variable List

I	Temporary
I%	Temporary
LINE$	String variable used for input and output to the screen. Must be dimensioned to at least 132 bytes.
N	Temporary
POINT	Temporary
SCRN$	String variable that describes each field in the screen file created by **SM**. Must be dimensioned to at least 48 bytes plus three times the number of fields on the screen.
UPND1	Primary unpend key
UPND2	Secondary unpend key. Set this to 255.
X%	The sign of the output variable X
XCOL	The column number of the current field
XDEC	The number of digits past the decimal point (used only in the formatted I/O provided by **FORM.SL**)
XFLD	This variable is used to compute the variable XPOS.
XFLGS	Bit flags describing the field as follows:

Bit	Meaning When Set
128	A numeric field; if not set, a string.
64	A protected field; if not set, an unprotected field.
32	A left-justified field
16	Numeric field can have varying numbers of decimals on input (when bit 128 is also set).
16	String field can have lowercase characters on input (when bit 128 is not set).
8	Numeric field can be negative on input (when bit 128 is also set).
8	String field is crammed (when bit 128 is not set).4,2,1 Set to XDEC if field is numeric.

XLROW	The number of the last line read from the screen (indicates the contents of LINE$)
XPOS	This variable is used to compute the positions of the columns, field width, and so on.
XROW	The line number of the current field
XWID	The width of the current field
Y%	This variable indicates an error on a **VALUE** statement.
Z%	The implied decimal point location of a numeric input variable

SM

Continued

FORM.SL

Format

GOSUB 9300

Input Variables

X, X$	The value to be sent to a screen field. Use either X for a numeric value or X$ for a string value; don't use both X and X$.

This subroutine uses **UNFORM.SL**, so check the **UNFORM.SL** input variables also.

Scratch Variables

See "Scratch Variable List" earlier in this section.

Output Variables

X, X$	The value input from a screen field. Either X for a numeric value or X$ for a string value. X and X$ are not both used at the same time.

This subroutine uses **UNFORM.SL**, so check the **UNFORM** output variables also.

Line Numbers

9100-9895	Line numbers used by **FORM.SL**, **UNFORM.SL**, and **PROTFORM.SL**.
9300	Entry point for placing the screen field in either X$ (a string field) or X (a numeric field).
9500	Entry point for sending either X$ (a string field) or X (a numeric field) to a screen field.

What It Does

FORM.SL inputs and outputs fields for **SM** screens. The subroutine uses **UNFORM.SL** to read data from input screen fields and to place data in output screen fields. **FORM.SL** also checks numeric fields for errors. The subroutine returns in the output variable E either 0 when the conversion is successful or 128 when the numeric value input from the screen contains an error. String input is converted to uppercase and crammed. Numeric output to a screen field conforms to the field's specifications, and string fields are uncrammed if necessary.

NOTE: Even though UNIX systems do not support **SM**, you can use this subroutine on UNIX systems. If you are using Business BASIC DG mode (specified by including the **-D** option on the command line to execute Business BASIC), you must use 7-bit mode. **SM** screens do not support 8-bit mode. If you are using Business BASIC in non-DG mode, you must specify the **-C** option when you use **SM** screens. This is because the screens contain embedded DG characters.

SM

Continued

How To Use It

To execute **FORM.SL**, you must perform the steps necessary to execute **UNFORM.SL** (see **UNFORM.SL**). You also need to do the following:

1. Enter the statements **"FORM.SL** and **"UNFORM.SL** into your program.
2. Initialize either X$ (string values) or X (numeric values).
3. Position the cursor to the correct screen field (**UNFORM.SL** does this).
4. Include either the program statement **GOSUB 9300** (to receive a value from a screen field) or the program statement **GOSUB 9500** (to send a value to a screen field).

SM

Continued

PROTFORM.SL

Format

GOSUB { 9200 / 9250 }

Input Variables

This subroutine uses the **FORM.SL** and **UNFORM.SL** subroutines, so check their input variables.

Scratch Variables

See "Scratch Variable List" earlier in this section.

Output Variables

This subroutine uses **FORM.SL** and **UNFORM.SL**, so check their output variables.

Line Numbers

9100-9895	Line numbers used by **FORM.SL, UNFORM.SL** and **PROTFORM.SL**
9200	Entry point to convert a field to an unprotected field. Either the **FORM.SL** output variable X (a numeric field) or X$ (a string field) is returned using formatted output.
9250	Entry point to convert the field to a protected field. Either the **FORM.SL** output variable X (a numeric field) or X$ (a string field) is returned using formatted output.

What It Does

PROTFORM.SL works with **SM** screen files either to convert a protected field to an unprotected field or to convert an unprotected field to a protected field, based on the entry point you use. **PROTFORM.SL** also places **FORM.SL** variable X (a numeric field) or X$ (a string field) in the field you specify.

NOTE: Even though UNIX systems do not support **SM**, you can use this subroutine on UNIX systems. If you are using Business BASIC DG mode (specified by including the **-D** option on the command line to execute Business BASIC), you must use 7-bit mode. **SM** screens do not support 8-bit mode. If you are using Business BASIC in non-DG mode, you must specify the **-C** option when you use **SM** screens. This is because the screens contain embedded DG characters.

SM

Continued

How To Use It

To use **PROTFORM.SL**, follow these steps:

1. Enter the statements **"PROTFORM.SL**, **"FORM.SL**, and **"UNFORM.SL** in your program.
2. Initialize the variables required by **FORM.SL** and **UNFORM.SL** (see **FORM.SL** and **UNFORM.SL**).
3. Include the program statement **GOSUB 9200** to convert a field to an unprotected field and the statement **GOSUB 9250** to convert a field to a protected field.

SM

Continued

UNFORM.SL

Format

GOSUB *linenumber*

Arguments

linenumber	The line number for the subroutine entry point. **UNFORM.SL** uses several subroutine entry points, so you need several **GOSUB** *linenumber* statements in your program. The entry points are listed under "Line Numbers."

Input Variables

E	An error code, initialized to 0.
F	A numeric expression for the field number. F equals the field number if R equals the row, or F equals 100 * row number + the field number.
R	A numeric expression for row number if F < 100.
X, X$	The value to be input from a screen field or output to a screen field. Use either X (a numeric field) or X$ (a string field). Do not use both. If used, X$ must be dimensioned to at least 80 bytes.
SCRN	A numeric expression for the screen number in the screen file.

Scratch Variables

See "Scratch Variable List" earlier in this section.

Output Variables

E	An error code.
FUNC	The function number for a function key.
X, X$	The value to receive input or send output to screen fields. Use X for a numeric field and X$ for a string field.

Line Numbers

9100-9895	**UNFORM.SL**, **FORM.SL**, and **PROTFORM.SL** occupy these line numbers.
9100	Clears all unprotected fields.
9150	Sets the necessary **STMA** statements, sets block mode, and locks the keyboard; always used.
9155	Same as 9150, but used when lowercase is not needed.
9175	Resets the **STMA** statements set by 9150 or 9155, turns interactive mode back on, and clears the terminal screen.
9400	Reads a field from the screen and returns it in X$. Use 9650 and F (or F and R) to position to the screen field first.
9550	Sends X$ to the field requested, left justifies, and handles screen overflow.

SM

Continued

9650	Positions the cursor to the requested field.
9700	Displays screen number SCRN from a screen file opened on channel 15 and loads SCRN$.
9735	Similar to 9700 except the screen is not displayed (i.e., reread SCRN$ only).
9750	Reads a function key input from the keyboard and returns a value in FUNC: 1-8 for F1 to F8, 14-16 for F9 to F11, -1 if input was not 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	Places an error message at row 24 of the screen. If E<>0 then it uses the error code in E with the **BASIC.ER** file to return the appropriate error message. If E=0, then your own message in X$ is output.
9890	Clears the error message from the screen.

What It Does

UNFORM.SL displays the **SM** screen specified in SCRN, positions the cursor at a field using F (or F and R), reads a field from the screen into X$, places X$ in a field, sends error messages, and returns a value for a function key input. **UNFORM.SL** also locks the keyboard and issues the necessary **STMA** statements to control cursor positioning.

NOTE: Even though UNIX systems do not support **SM**, you can use this subroutine on UNIX systems. If you are using Business BASIC DG mode (specified by including the **-D** option on the command line to execute Business BASIC), you must use 7-bit mode. **SM** screens do not support 8-bit mode. If you are using Business BASIC in non-DG mode, you must specify the **-C** option when you use **SM** screens. This is because the screens contain embedded DG characters.

How To Use It

To use **UNFORM.SL**, follow these steps:

1. Enter "**UNFORM.SL** into your program.
2. Open your screen file on channel 15.
3. Assign the necessary input variables. You can specify a field in two ways: either set R to the row number and F to the field number, or set F to the following: row_number * 100 + field_number.
4. Use **GOSUB** statements for the specified entry points listed in Line Numbers to perform the functions you need.

When you use **GOSUB 9400**, if the field is already in LINE$ (known by XLROW), the subroutine does not read the field. 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.

SM

Continued

Screen File Layout

SM builds and maintains a screen file for you to use with **FORM.SL**, **UNFORM.SL**, **PROTFORM.SL**, **FM**, or **SM** itself. When you follow the routines, you never have to look at the screen file layout. But if you want to do special things, you should be familiar with the screen maintenance commands.

The screen file has a record size of 5000 bytes; each screen makes up one record. The first record number is zero. Section A of a screen record is read into SCRN$ when you input/output to screen files. There are 24 line pointers, and each pointer is followed by the field definitions (XCOL, XWID, and XFLGS, all used by **UNFORM.SL**) for each line. Section B's record count tells how many screen literals (LINE$ used by **UNFORM.SL**) follow. Screen literals are data-sensitive lines that **FORM.SL** and **UNFORM.SL** read using **INPUT USING**. You only need to refer to section C when you want screen label verification.

Table 1-17. Screen File Record Format

Field Contents	Location	Size	Type
Section A: Field definitions (SCRN$)	0	variable	string
1. Line pointers (24; 1 per line)	0	48 (24*2)	numeric
2. Field definitions (1 per field)	48	# of fields*3	--
2A. Field column (XCOL)	--	1	numeric
2B. Field width (XWID)	--	1	numeric
2C. Field flags (XFLGS)	--	1	numeric
Section B: Screen literals	1600	variable	--
1. Screen literal record count	1600	4	numeric
2. Screen literal record (LINE$)	--	variable	string
Section C: Screen label information	4872	128	--
1. Status (active record = 1)	4872	2	numeric
2. Time of last modification (*hhmmss*)	4874	4	numeric
3. Date of last modification (*mmddyy*)	4878	4	numeric
4. Last modifier's account number	4882	6	string
5. Project identification	4888	30	string
6. System identification	4918	30	string
7. Program identification	4948	30	string

In Table 1-17, locations are relative to zero for file positioning purposes. When you use string subscripts, add one to all entries in the locations column.

SM *Continued*

SM Examples

1. The following **SM** screen was printed using the **SM PRINT** command:

```
PROJECT:


SYSTEM:                                    PROGRAM:


SCREEN ID: SMSCREEN[0]     LAST MODIFIED BY: AAAAA8 AT 09:05:35 ON 03/06/91


          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
  :.....................................................................:
 1:                       EMPLOYEE FILE MAINTENANCE                      : 1
 2:                                                                      : 2
 3:                                                                      : 3
 4:                                                                      : 4
 5:                                                                      : 5
 6:EMPLOYEE LAST NAME:  LLLLLLLLLL                                       : 6
 7:                                                                      : 7
 8:EMPLOYEE FIRST NAME:  LLLLLLLLLL                                      : 8
 9:                                                                      : 9
10:EMPLOYEE EXTENSION:  8888                                            :10
11:                                                                     :11
12:EMPLOYEE ID:  99-888                                                 :12
13:                                                                     :13
14:                                                                     :14
15:                                                                     :15
16:                                                                     :16
17:                                                                     :17
18:                                                                     :18
19:                                                                     :19
20:                                                                     :20
21:                                                                     :21
22:                                                                     :22
23:                                                                     :23
  :.....................................................................:
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

You can create this screen by executing **SM**.

SM

Continued

```
* RUN "SM
```

SM then clears the screen and begins its dialog. First it asks you for the name of a screen file. Enter the name of the file you want to create. (If you are going to use this file with **FM**, use an *.Sn* extension where *n* is your terminal type. You can use **STMA 1,0** to determine your terminal type.)

```
DG SCREEN FILE MAINTENANCE  REV X.XX
SCREEN FILENAME:
```

Once you enter a filename, **SM** displays a blank screen with the command line prompt at the bottom. To go to the top of the screen, press the Home key or the Up-arrow key. In the bottom right corner of the screen, **SM** displays the line and column number where the cursor is positioned.

```
>>                                                    24  3 F
```

To create the employee file screen, press F6 (the START PROTECT key), space to column 25, and type the screen heading, EMPLOYEE FILE MAINTENANCE. Press the PROTECT TO EOL function key (F2). Continue pressing it until you reach line 6 column 1.

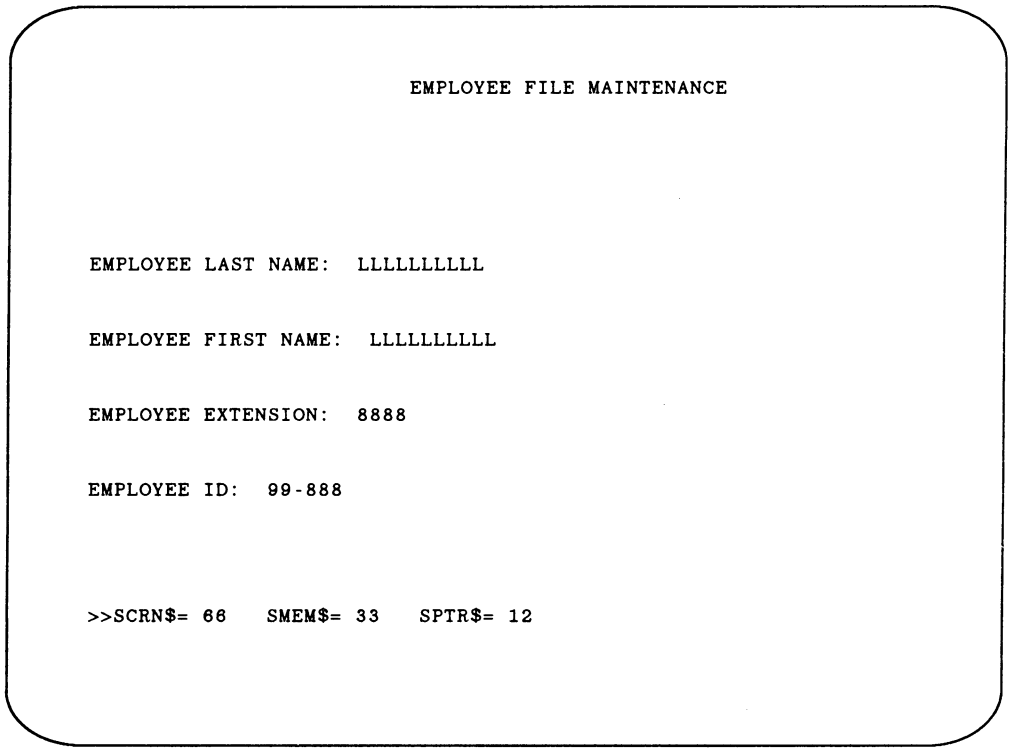
To enter the screen prompts, do the following:

1. Type the prompt.
2. Press F5 (the END PROTECT key) to start an input/output field.
3. Type the field definition characters for the input and output field.
4. Press F6 (the START PROTECT key).
5. Position the cursor to the next prompt line.
6. Return to step 1 above if you want to enter another prompt.

When you finish setting up your screen, press F4 (the ENTER CMND key). This moves the cursor to the **SM** command line. Type **WRITE 0**, and press F11 (the PROCESS CMND key) to execute the command. If you keyed the screen in correctly, the following is displayed:

SM

Continued



To leave **SM** and return to Business BASIC, press F4 (the ENTER CMND key) again, enter STOP, and press F11 (the PROCESS CMND key).

2. To use your screen, key in the following program. Before running the program, enter the **FORM.SL**, **UNFORM.SL**, **POSFL.SL**, and **GETREC.SL** subroutines, and save the program as **SMTEST**.

```
* LIST
00010 STMA 6,5                          :Do not allow interrupts
00020 DIM C1[2,3],B$[544],LX$[18],FX$[18]   :Scratch variable for
                                            :opening files
00030 DIM LAST$[10],FIRST$[10],REC$[64]     :Variables for data
                                            :file records
00040 DIM X$[80],LINE$[132],SCRN$[66] :Variables for screen file
00050 LET F%,R1,Y%,E,X=0                :Scratch variables
:
00060 REM ** Routine to OPEN files and set up the C1 array
:
00070 LET B$="EMP,5,LAST,5,FIRST,5",FILL$(0)   :Data and Index files
00080 BLOCK WRITE B$
00090 SWAP "OPEN"                       :Swap to the OPEN utility
00100 BLOCK READ B$
00105 UNPACK "JJ",X$,ERRIN,ERRNO
00110 IF ERRIN<>-1 THEN GOTO 00140          :Check errors in OPEN
00120 PRINT "ERROR # ";ERRNO;" - ";X$[5,512]
```

SM

Continued

```
00130 END
00140 LET K=1
00150 FOR I=0 TO 2                    :Build or assign C1 array
00160   FOR J=0 TO 3
00170     LET C1[I,J]=ASC(B$[K,K+3])
00180     LET K=K+4
00190   NEXT J
00200 NEXT I
00210 LET LX$=CHR$(C1[1,0],2),CHR$(C1[1,1],4),CHR$(0,2),"LAST",
FILL$(0)
00220 LET FX$=CHR$(C1[2,0],2),CHR$(C1[2,1],4),CHR$(0,2),"FIRST",
FILL$(0)
00230 OPEN FILE[15,4],"SMSCREEN"   :Open screen file
00240 GOSUB 09150            :Set keyboard and STMAs (UNFORM.SL)
00250 LET SCRN=0             :Screen number
00260 GOSUB 09700            :Display screen in SCRN (UNFORM.SL)
:
00270 REM ** Display INPUT/OUTPUT fields of the screen
:
00280 LET F=601                     :Row 6, field 1
00290 LET X$=""                     :Initialize X$
00300 LET X$[1,10]=FILL$(32)        :Fill string with
                                    :spaces for output
00310 GOSUB 09550                   :Output string to
                                    :field (UNFORM.SL)
00330 LET F=801                     :Row 8,field 1
00340 LET X$=""                     :Initialize X$
00350 LET X$[1,10]=FILL$(32)        :Fill string with
        0                           :spaces for output
00360 GOSUB 09550                   :Output string to
                                    :Field (UNFORM.SL)
00380 LET F=1001                    :Row 10, field 1
00390 LET X=0                       :Initialize X
00400 GOSUB 09500 : \ WRITEFORM     :Output X to Field (UNFORM.SL)
00420 LET F=1201                    :Row 12, field 1
00430 LET X=20                      :Set output-only field
00440 GOSUB 09500 : \ WRITEFORM     :Output X to Field (UNFORM.SL)
00450 LET F=1202                    :Row 12, field 2
00460 LET X=0                       :Initialize X
00470 GOSUB 09500 : \ WRITEFORM     :Output X to Field (UNFORM.SL)
:
00480 REM ** Request input into INPUT/OUTPUT fields
:
00490 LET F=601                     :Row 6, field 1
00500 GOSUB 09650            :Position cursor to F (UNFORM.SL)
00510 GOSUB 09300 : \ READFORM       :Read formatted input (FORM.SL)
00520 LET LAST$=X$,FILL$(32)        :Assign input to record
                                    :variable
00530 LET F=801                     :Row 8, field 1
```

SM

Continued

```
00540 GOSUB 09650            :Position cursor to F (UNFORM.SL)
00550 GOSUB 09300 : \ READFORM       :Read formatted input (FORM.SL)
00560 LET FIRST$=X$,FILL$(32)     :Assign input to record variable
00580 LET F=1001                  :Row 10, field 1
00590 GOSUB 09650            :Position cursor to F (UNFORM.SL)
00600 GOSUB 09300 : \ READFORM       :Read formatted input (FORM.SL)
00610 LET EXT=VAL(X$,Y%)     :Assign input to record variable
00620 IF Y%=0 THEN GOTO 00700        :Check for VAL funtion error
00630 LET E=128                   :If error, make it error 128
00640 GOSUB 09850                 :Output error (UNFORM.SL)
00650 GOSUB 09750                 :Read function key hit
                                  :by operator
00660 IF FUNC<>16 THEN GOTO 00640 :If not F11(value 16), enter
                                  :again
00670 GOSUB 09890            :FK was F11-clear error (UNFORM.SL)
00680 GOTO 00580                  :Go back and read field again
00700 LET F=1202                  :Row 12, field 2
00710 GOSUB0 09650           :Position cursor to F (UNFORM.SL)
00720 GOSUB 09300 : \ READFORM       :Read formatted input (FORM.SL)
00730 LET ID=VAL(X$,Y%)      :Assign input to record variable
00740 LET ID=ID+20000             :Add output variable
00750 IF Y%=0 THEN GOTO 00830        :Check for VAL function error
00760 LET E=128                   If error, make it error 128
00770 GOSUB 09850                 :Output error (UNFORM.SL)
00780 GOSUB 09750            :Read function key hit by operator
00790 IF FUNC<>16 THEN GOTO 00770 :If not F11(value 16), enter
                                  :again
00800 GOSUB 09890            :FK was F11-clear error (UNFORM.SL)
00810 GOTO 00700             :Go back and read field again
00830 PACK "ZJA10A10LJ",REC$,1,LAST$,FIRST$,EXT,ID
                                     :Build record string
00840 LOCK 1,"EMP",0,C1[F%,3]        :Lock record 0 of data file
00850 GOSUB 08400 : \ GETREC.SL   :Get next available
                                  :record in R1
00860 GOSUB 09610 : \ POSFL.SL       :Position to record R1
00870 WRITE FILE[C%],REC$         :Write record
00880 KADD LX$,B$,LAST$,R1        :Add key to LAST index using R1
00890 IF R1<=0 THEN GOTO 01010 : ** INDEX ERROR ROUTINE
                                           :Check for error
00900 KADD FX$,B$,FIRST$,R1       :Add key to FIRST index
                                  :using R1
00910 IF R1<=0 THEN GOTO 01010 : ** INDEX ERROR ROUTINE
                                           :Check for error
00920 UNLOCK                         :Unlock data record
00930 LET X$="F10=STOP   F8=CONTINUE" :Operator choices
00940 LET E=0                        :Let error code equal 0
00950 GOSUB 09850           :Output choice message (UNFORM.SL)
00960 PRINT @(24,30);              :Position cursor to receive
                                   :function key
```

SM

Continued

```
00970 GOSUB 9750                    :Read function key (UNFORM.SL)
00980 IF FUNC=15 THEN GOTO 01040    :Look for F10 (value 15)
00990 IF FUNC=8 THEN GOTO 00280     :Look for F8
01000 GOTO 00940                    :If neither, go back and try
                                    :again
01010 REM ** INDEX ERROR ROUTINE
01020 LET X$="<7>ERROR IN ADD<7> - F10=STOP"   :Assign error message
01030 GOTO 00940                    :Go display error message.
:
01040 REM ** Routine to End and print data records
:
01040 GOSUB 09175            :Reset terminal for normal use.
01050 STMA 7,5                      :Allow interrupts
01060 LET LAST$=""                  :Null key
01070 KFIND LX$,B$,LAST$,R1         :Find 1st key
01080 LET R1=ABS(R1)                :Expecting a negative return
01090 IF R1<=0 THEN GOTO 01180         :Check for end of file
01100 GOSUB 09610 : \ POSFL.SL         :Position to record
01110 READ FILE[C%],REC$            :Read record
01120 UNPACK "JA10A10LJ",REC$,X,LAST$,FIRST$,EXT,ID
                                    :UNPACK record
01130 PRINT USING "A10,3X,A10,3X,D5.0,3X,D6.0",LAST$,FIRST$,EXT,I
                                    :Print
01140 KNEXT LX$,B$,LAST$,R1         :Get next key
01150 GOTO 01110                    :Go read until end of file
01160 CLOSE
01170 END
* ENTER "FORM.SL
* ENTER "UNFORM.SL
* ENTER "POSFL.SL
* ENTER "GETREC.SL
* SAVE "SMTEST
*
```

3. The following screen was printed using the **SM PRINT** command. After you add the screen filename to the table file **TAXES.TB** using the **FM** utility, you can use this screen with the database created by **DBGEN**.

SM

Continued

```
PROJECT:

SYSTEM:                                      PROGRAM:

SCREEN ID: TAX.S8[0]     LAST MODIFIED BY: AAAAA8 AT 15:47:09 ON 10/04/91



 12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
 ................................................................................
 1:
 2:                            Taxes Screen
 3:
 4:Taxes Record Number:  8888
 5:
 6:     Index Files:  NAME.IX
 7:          Name Key:  LLLLLLLLLL
 8:
 9:     Index File:  TAX.IX
10:           Tax Key:  8888888888
11:
12:
13:     Data File:  TAXES
14:              Name:  LLLLLLLLLL
15:           Address:  LLLLLLLLLLLLLLL
16:            Income:  8888888888
17:               Tax:  8888888888
18:
19:
20:
21:
22:
23:
  :.............................................................................
            1         2         3         4         5         6         7
  12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
```

SPDIS

BASIC CLI Command

Disables DG/RDOS spooling on a device.

DG/RDOS

What It Does

Since this command can only be used with the DG/RDOS Business BASIC **Spooler**, it, like the **Spooler**, is explained in the *Business BASIC System Manager's Guide*.

SPEBL

BASIC CLI Command

Enables DG/RDOS spooling on a device.

DG/RDOS

What It Does

Since this command can only be used with the DG/RDOS Business BASIC **Spooler**, it, like the **Spooler**, is explained in the *Business BASIC System Manager's Guide*.

SPKILL

BASIC CLI Command

Deletes the DG/RDOS spool queue for a device.

DG/RDOS

What It Does

Since this command can only be used with the DG/RDOS Business BASIC **Spooler**, it, like the **Spooler**, is explained in the *Business BASIC System Manager's Guide*.

Spooler

DG/RDOS Utilities

Lets you run multiple jobs that generate print-image output to a limited resource.

DG/RDOS

What It Does

Since this utility can only be used on DG/RDOS systems by someone with system manager privileges, the DG/RDOS Business BASIC **Spooler** and the utilities it uses are explained in the *Business BASIC System Manager's Guide*.

SQUE

BASIC CLI Command

Sets the default output queue.

AOS/VS	DG/RDOS	UNIX

Format

!SQUE *filename*

Argument

filename A device or file that serves as the default output queue. *filename* can be no more than six characters long.

What It Does

The **SQUE** command sets the default output queue to filename. When the **/L** switch is used with other commands, Business BASIC sends the output of that command to the default output queue set by **SQUE**.

On AOS/VS systems, the default output queue can also be set by using the **/Q** switch when you execute Business BASIC.

On UNIX systems, the default output queue can also be set by using the **-q** global switch or by editing the first entry in the file **DEVICE_MAP**, which resides in the directory **SYSLIB**.

How To Use It

Execute the command by entering **SQUE** from the Business BASIC CLI. The argument *filename* must follow **SQUE**.

Example

SQUE sets out as the default output queue. You can verify this by using **GQUE** to display the default output queue after you use **SQUE**.

```
* !GQUE
@LPT


* !SQUE OUT


* !GQUE
OUT


*
```

START

BASIC CLI Command

Starts a detached job.

DG/RDOS

Format

!START [*program*/**N**] [*account*/**A**] [*password*/**P**] [*dir*/**D**]
[*"input1"*] [*"input2"*...]

Arguments

program/**N**	The name of program to be run as detached job. Program must be followed by a **/N** switch.
account/**A**	The detached job logs on with account as the account ID. Any program specified in the ACCNTS record overrides any program specified by the *program*/**N** option. Account must by followed by **/A**.
password/**P**	The password for a specified account; this is required when *account*/**A** is used. Password must be followed by **/P**.
dir/**D**	The directory from which the detached job is run. *dir* must be followed by **/D**.
input	A message within quotation marks that you supply as input to the job.

Global Switch

/V	Verify the start-up by displaying the job dialog as it is transmitted.

Local Switch

/I	Pass this file line by line to the started job.

What It Does

START executes a program as a detached job. If no program is specified, then nothing happens. When you specify *account*/**A** and *password*/**P**, the job is logged on accordingly; otherwise, the detached job's account is the account executing **START**. If you specify *dir*/**D**, the detached job runs in that directory; otherwise, the job runs from the current directory. Optionally, one or more messages can be supplied as input to the detached job.

A detached job is a job running independently of a terminal. You can begin a job's execution at your terminal and then detach the job by using the detach key. The default detach key is Ctrl-D. Use **STMA 4,0** or the **TERM** program to change the detach key. You can also use the **ATTACH** utility to assign a detached job to a terminal (see **ATTACH**). The system discards any terminal output from a detached job. If a detached job executes an **INPUT** statement, the job waits ten minutes for input. If nothing is entered, the job logs off. Use **START** to connect to a job and supply data for the **INPUT** statement before the job logs off.

START

Continued

How To Use It

Execute the command by entering **START** from the Business BASIC CLI. You can follow the command with one or more arguments. If the started job does not issue input requests for the supplied input, control does not return to your terminal. Started jobs normally run at a lower priority than the job executing the **START**.

Example

Start a fast dump of the **TDATA** directory. You can proceed with your work while the dump is happening without having to log back on. Note that **BYE** is used as the last command so the job is logged off the system when it finishes.

```
!START CLI/N TDATA/D "FDUMP/A/L MT0:0;BYE"
```

STAT *Utility*

Displays the status of all jobs.

AOS/VS	DG/RDOS

Formats

{ RUN / SWAP / CHAIN } "STAT

Or

!STAT

Global Switches (DG/RDOS only)

/A	Include the user account number.
/D	Include the user directory specifier.
/I	Include the I/O usage (number of system calls).
/L	Include the job logon time.
/N	Include the program name.
/P	Include the program push level.
/Q	Include the amount of CPU time used.
/R	Repeat the **STAT** output until interrupted. This is the same as **PED** in DG/RDOS.
/S	Include the size in bytes of the program and data.
/T	Include the terminal line number.
/X	Include the job task priority.

What It Does

Use **STAT** to display information on the status of your system. Executing **STAT** without any switches is the same as executing **STAT** with all switches except the **/R** switch. In both cases, **STAT** displays all of its information (columns a through n in the example).

On AOS/VS systems, you must have **PED.PR** on your search path to use **STAT**. **STAT** invokes **PED.PR**. In addition, you need the ability to create a son process.

NOTE: On UNIX systems, you can use the **BBSTAT** command to get information on the status of Business BASIC jobs. This command also works on AOS/VS systems. See *Commands, Statements and Functions in Business BASIC* for more information.

How To Use It

Execute **STAT** by entering **RUN**, **CHAIN**, or **SWAP "STAT**. No switches are allowed when you execute **STAT** using one of these three methods. To use global switches

STAT

Continued

with **STAT**, execute the command by entering **STAT** from the Business BASIC CLI. If used, global switches are appended to either the command word or another global switch.

Example (DG/RDOS)

STAT is executed from Business BASIC CLI. To explain the **STAT** output, its display has been divided into sections a through n. Each section is described after the display.

```
* !STAT

00   R 19 DZ0      STAT    AAEBG6 00 2 2286  1526 10081.7 1130 07:31
01 O R 19 ACCOUNT  ACCNTS  USIGF6 05 2 9350   724  3298.5  319 11:08
02 I C 19 TESTS    SCRATCH ABJMF6 02 0 1382 12898  1185.3  608 11:19

a. b.c.d.   e.      f.      g.   h. i.  j.   k.     l.     m.    n.
```

Column	Description
a.	The job number
b.	Indicator showing the current I/O status:
	I — Waiting on input
	O — Waiting on output
	S — On system call
	D — On DELAY
	L — On LOCK
	Q — For control (^) Q
	blank — I/O satisfied
c.	Indicator showing what the current program is doing:
	R — Running
	C — In compile state
	B — Running and compiling
d.	The current program's task priority
e.	The job's directory specifier
f.	The name of the current program
g.	The account code used at logon time
h.	The job's terminal number; -1 if detached
i.	The program push level for this job
j.	The program's size in bytes
k.	The data's size in bytes
l.	The CPU time (in seconds) used since logon time
m.	The number of system calls made since logon time (I/O usage)
n.	The time when you logged on

TABBUILD *Utility*

Quickly defines arrays for FM table files.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "TABBUILD

What It Does

TABBUILD creates field descriptor records (type 5 table file records) for **FM** table files. The field descriptor records have recurring fields. Usually, these fields are part of an array in a data record.

How To Use It

Execute **TABBUILD** by entering **RUN**, **CHAIN**, or **SWAP "TABBUILD**. **TABBUILD** starts a dialog where it asks you for the name of the table file and the information described under the **FM** utility's section on "Type 5 Records" (see **FM**). **TABBUILD** also asks for a starting record number in the table file and a record increment. This lets you leave a space between table records. In addition, **TABBUILD** requests a starting position and an increment within the data record. To make it easier to identify recurring fields, **TABBUILD** automatically appends an element number to the DESCRIPTION and ID.

After you enter the ID, **TABBUILD** displays the records added to the table file, the next available record number in the table file, and the next byte position in the data record. **TABBUILD** then prompts you for the next starting record number in the table file so that you can continue adding table records of the same type to the table file. If you do not enter a starting record number, **TABBUILD** prompts you for a new format in case you want to add records of a different type.

To end **TABBUILD**, press New Line at the FORMAT prompt without entering any data. Once you have entered the TABLE FILE NAME, you cannot use the Escape key to exit this program.

TABBUILD

Continued

Example

This example illustrates the **TABBUILD** dialog. **TABBUILD** displays its output to the terminal. **TABBUILD** returns you to keyboard mode when New Line is pressed without entering a format.

```
* RUN "TABBUILD
TABLE FILE NAME: EMP.TB
FORMAT: F11
TYPE: D
EDIT CODE: N
MINIMUM: 0       MAXIMUM: 2147483647
RECORD INC: 1    POSITION INC: 4
STARTING TABLE RECORD: 21
STARTING POSITION: 52
NUMBER OF RECORDS TO BE CREATED: 12
FIRST NUMBER TO APPEND TO FIELD NAMES: 1
DESCRIPTION: CALL FOR MONTH-
ID: CPM
21        52        CALLS FOR MONTH-1     CPM1
22        56        CALLS FOR MONTH-2     CPM2
23        60        CALLS FOR MONTH-3     CPM3
24        64        CALLS FOR MONTH-4     CPM4
25        68        CALLS FOR MONTH-5     CPM5
26        72        CALLS FOR MONTH-6     CPM6
27        76        CALLS FOR MONTH-7     CPM7
28        80        CALLS FOR MONTH-8     CPM8
29        84        CALLS FOR MONTH-9     CPM9
30        88        CALLS FOR MONTH-10    CPM10
31        92        CALLS FOR MONTH-11    CPM11
32        96        CALLS FOR MONTH-12    CPM12
NEXT RECORD IS 33           NEXT POSITION IS 100
STARTING TABLE RECORD: <NL>
FORMAT: <NL>
```

TABLE

BASIC CLI Command

Prints a cross reference for a listing file.

AOS/VS	DG/RDOS

Format

!TABLE [*outputfile-1*/**L**] *filename-1* [[*outputfile-2*/**L**] *filename-2*] ...

Arguments

filename	The name of a program listing file created by a Business BASIC **LIST** command or a text editor.
outputfile/**L**	The name of the output file to which you want to append the listing. This argument overrides global **/L**. Always append **/L** to *outputfile*. If *outputfile* does not exist, Business BASIC creates it.

Global Switches

/A	Include permanent symbols in the cross-reference list (**LET**, **IF**, etc.).
/D	Allow duplicate entries in the cross-reference list.
/H	Print a heading at the top of each page.
/L	Print output on the default output queue.
/O	Print library subroutines. Must be used with **/P**.
/P	Include a program listing before the cross-reference list.
/W	Print output in 132-column page width. Must be used with **/L**.

What It Does

TABLE prints a cross-reference table of all variables and their statement numbers in the program specified by filename. **TABLE** also prints a listing of the program like the Business BASIC CLI **PRINT** command. When **TABLE** encounters a statement in the form:

REM \ *comment*

it does not print the lines following that statement until it encounters another **REM** comment statement. This suppresses the printing of library subroutines. The **/O** switch overrides this suppression. The first **REM** must have a backslash (\) in front of the comment for **TABLE** to recognize this as the beginning of a subroutine. The **REM** that ends the subroutine does not need any special characters. If the first **REM** does not have the backslash (\), then the **REM** statements and the code between them are not printed regardless of whether the **/O** switch is used because **TABLE** does not consider this a subroutine.

TABLE

Continued

How To Use It

Execute the command by entering **TABLE** from the Business BASIC CLI. **TABLE** must be followed by the name of at least one program. You can also specify an output file to receive the results. If you use global switches, append them either to the command word or to another global switch.

Example

TABDEMO.LS, a listing program, is displayed at the terminal with cross references and remark lines but without subroutines.

!TABLE/P TABDEMO.LS

```
00010 REM - TABLE Demo program showing assorted info
00015 ON IKEY THEN GOTO 09000
00017 ON ERR THEN GOTO 09000
00020 DEF FNA(X)=OR(X,-AND(X,256))
00030 DEF FNB(X)=OR(X,-AND(X,32768))
00040 FOR I=1 TO 20
00050   FOR J=I TO I*2-1
00060     GOSUB 00100
00070   NEXT J
00080 NEXT I
00090 END
00100 FOR K=FNA(I) TO J
00120   GOSUB 00200 : THIS PROGRAM DEMONSTRATES
00150 NEXT K
00180 RETURN
00200 REM THIS PROGRAM DEMONSTRATES
00210 REM THE FEATURES OF TABLE
00220 STOP
00250 RETURN
09000 REM - BAIL OUT
09010 STOP



00100     00060
00200     00120
09000     00015     00017
I         00040     00050     00080     00100
J         00050     00070     00100
K         00100     00150
X         00020     00030
```

TBUILD *Utility*

Builds a tag file.

AOS/VS	DG/RDOS	UNIX

Format

SWAP "TBUILD

What It Does

TBUILD constructs a tag file with each record consisting of a key and a pointer to a data record. The input file to **TBUILD** can be an index file or a linked-available-record data file that has the key embedded in a data record. Keys can have multiple fields.

When your input file is a data file, **TBUILD** skips record 0 and checks the status (first two bytes) of each record for a value of 0 (indicating deleted records). This keeps deleted records out of the tag file. **TBUILD** computes the record number based on the record accessed relative to 0.

How To Use It

Do not use **RUN** or **CHAIN** with **TBUILD**; only use **SWAP**. To execute **TBUILD**, do the following:

1. Set up an argument string.
2. Use **BLOCK WRITE** to put the argument string in the common area.
3. Enter the statement **SWAP "TBUILD**.
4. Check **STMA 1,1** to see if an error occurred. A 0 indicates no error occurred. In case an error occurred in an **ON ERR** trap, check **STMA 1,2** to retrieve the line number of the error.

The following error codes can occur with **TBUILD**:

Error Code	Meaning
45	Illegal record length.
68	Index file full.
77	Illegal record number.
146	Key already exists and duplicates are not allowed.
148	File not on sector boundary.
149	Record out of sequence.
150	Illegal blocking factor.
151	Illegal key length.

TBUILD

Continued

Table 1-18. TBUILD Argument String Contents

Substring Location	Size In Bytes	Contents
1,4	4	Channel number of the open input file
5,8	4	Byte offset to record 0 of input file
9,12	4	Maximum number of records in input file, including record 0 of a linked-available-record file
13,16	4	Number of bytes per record in the input file
17,20	4	Channel number of the opened tag file
21,24	4	Byte offset to record 0 of the tag file
25,28	4	Maximum number of records in the output tag file (not counting record 0 if the input file is a linked-available-record file)
29,32	4	Number of bytes per record in tag file, plus 4 bytes for the pointer (number of bytes in substring location 39, 40 + 4)
33,34	2	Not used
35,36	2	Flag to allow duplicate keys: 1 Allow duplicate keys 0 Do not allow duplicate keys If your input file is a data file, set flag to 1.
37,38	2	Flag to check for deleted records: 1 Check for deleted records 0 Do not check for deleted records
39,40	2	Total key field length in bytes.
41,42	2	Number of key fields (-2 for index file input)
*43,44	2	First byte of key (1 is first byte of record)
*45,46	2	Last byte of key
47,*	4	Next key field's descriptors. Repeat the sequence of entering the first and last byte of the key. Since you can have multiple keys, you can enter this information several times.

If you use a data file as the input file, then the key field must be in the same location in each record in the data file. Table 1-18 explains the substring locations and their contents.

TBUILD

Continued

Example

TBUILD creates a tag file using a data file as the input file. The arguments are placed in the string X$, which is passed to the common area. The program then swaps to **TBUILD**. Lines 220 and 230 check for errors when **TBUILD** finishes.

```
00010  DIM X$ [512]           :X$ will hold arguments sent to common.
00020  LET ERCODE = 0         :Initialize ERCODE and LINENO for
00030  LET LINENO = 0         :use with STMAs.
00040  OPEN FILE(0,5),"EMP"  :OPEN the input data file EMP.
00050  OPEN FILE(1,0),"TEMP":OPEN the tag file TEMP.
.
.
.
00100  LET X$=CHR$(0,4),CHR$(0,4)          :Channel # of data file.
                                           :Data file has 0 offset,
00110  LET X$[0]=CHR$(100,4),CHR$(100,4)   :100 records, and 100 bytes
                                           :per record.
00120  LET X$[0]=CHR$(1,4),CHR$(0,4)       :Channel # of tag file.
                                           :Tag file has 0 offset,
00130  LET X$[0]=CHR$(99,4),CHR$(14,4)     :99 records, 14 bytes
00140  LET X$[0]=CHR$(0,2)                 :Not used.
00150  LET X$[0]=CHR$(1,2),CHR$(1,2)       :Allow duplicates, check
                                           :deleted recs.
00160  LET X$[0]=CHR$(10,2),CHR$(2,2)      :Total key 10 bytes, 2
                                           :fields.
00170  LET X$[0]=CHR$(3,2),CHR$(6,2)       :First field, bytes 3 to 6
00180  LET X$[0]=CHR$(15,2),CHR$(20,2)     :Second field, bytes 15 to
                                           :20.
00190  LET X$[0]=FILL$(0)                  :No more fields.
00200  BLOCK WRITE X$                      :Send to common area.
00210  SWAP "TBUILD                        :Execute TBUILD and return.
00220  STMA 1,1,ERCODE                     :Get error code from TBUILD.
00230  STMA 1,2,LINENO                     :Get line of error if ON ERR
                                           :trap.
00240  IF ERCODE THEN GOTO 00700           :If TBUILD not successful,
                                           :go to 700.
.
.
.
```

TCOPY

BASIC CLI Command

Copies files from tape to tape.

AOS/VS	DG/RDOS

Format

!TCOPY *inputfile* [*outputfile*] [*filename*/**L**] [*number*/**N**]

Arguments

inputfile	The name of a file on a magnetic tape or intermediate disk file that you use as input for **TCOPY**.
outputfile	The name of a file on a magnetic tape or an intermediate disk that receives the **TCOPY** output. If the global **/R** switch is used, this argument is optional; otherwise, it is required.
filename/**L**	The name of the file that receives verification information. Filename must be followed by a **/L** switch. This argument is optional.
number/**N**	The number of consecutive files to be copied. Number must be followed by a **/N** switch. This argument is optional.

Global Switches

/E	Extended verification; report the number of records found for each record size found.
/I	Ignore any tape read errors.
/L	Verify the copied file to the default output queue. This switch overrides the **/V** and **/E** switches.
/R	Read the input file only. This switch allows **/E** or **/V** without actual copying.
/V	Verify copied files by listing them at the terminal.

What It Does

TCOPY copies one or more files from one magnetic tape to another magnetic tape or, if you have only one tape drive, to an intermediate disk file. By default, **TCOPY** copies all files on a tape (or disk file) beginning with the inputfile through the last file on tape. You can use the *number*/**N** argument to specify the number of files that you want copied.

How To Use It

Execute **TCOPY** by entering it from the Business BASIC CLI. **TCOPY** must be followed by the name of the file you want to copy. You must specify an output file to receive the information unless you use the **/R** switch. If you use global switches, append them either to the command word or to another global switch.

TCOPY *Continued*

The files on the tape (or disk file) can have records varying in length up to a maximum record size of 8192 bytes. Records with an odd number of bytes cannot be duplicated properly due to hardware restrictions.

Examples

1. On a DG/RDOS system, **TCOPY** copies file 3 on tape unit 0 to file 0 of tape unit 1. The **1/N** switch means that only file 3 is copied.

   !TCOPY MT0:3 MT1:0 1/N

2. On an AOS/VS system, **TCOPY** reads all files on tape unit 0 beginning with file 0 and reports the number of records of each size found. Since the **/R** switch is used, no outputfile argument is required.

   !TCOPY/E/R @MTA0:0

TERM

Utility

Changes certain terminal key functions.

AOS/VS	DG/RDOS

Format

```
{ RUN   }
{ SWAP  } "TERM
{ CHAIN }
```

What It Does

TERM changes the keys used to perform a detach function (DG/RDOS only), a line cancel, an interrupt, a character delete, and a character echo during a delete. This allows you to specify which key sequence you want to use to perform each of those functions.

On AOS/VS systems, **TERM** works only on terminal type 6.

How To Use It

Execute **TERM** by entering **RUN**, **CHAIN**, or **SWAP "TERM**. This starts the **TERM** dialog. **TERM** asks you if you want to change the character. If you enter **Y**, then **TERM** prompts you for a value for each key. You must press the actual key that you want to change the function to. **TERM** then echoes the ASCII value of that key.

Example

In this DG/RDOS example, the Ctrl-D key is pressed for the detach key, Ctrl-X for the line cancel key, Ctrl-H for the delete key and the delete character echo, and New Line for the interrupt key. The angle brackets < > set off the keys and key combinations. Do not enter the angle brackets. The ASCII value of each key you enter is displayed after the key.

```
* RUN "TERM

DO YOU WANT TO CHANGE THE DE-ATTACH CHAR? Y
DE-ATTACH CHAR: <CTRL-D> 4
DO YOU WANT TO CHANGE THE LINE CANCEL CHAR ? Y
LINE CANCEL CHAR: <CTRL-X> 24
DO YOU WANT TO CHANGE THE CHARACTER DELETE CHAR ? Y
CHARACTER DELETE CHAR: <CTRL-H> 8
DO YOU WANT TO CHANGE THE CHARACTER DELETE ECHO ? Y
CHARACTER DELETE ECHO: <CTRL-H> 8
DO YOU WANT TO CHANGE THE IKEY CHAR ? Y
IKEY CHAR: <NEWLINE>10
```

TFER

BASIC CLI Command

Copies a file between a tape and a disk.

AOS/VS	DG/RDOS

Format

!TFER *inputfile outputfile recordsize*/**R** [*blocksize*/**B**]

Arguments

inputfile, *outputfile* The names of tape or disk files. Tape files are indicated by the colon (:) separating the drive from the file. Disk files must be in the current directory, since no directory specifier is allowed.

recordsize/**R** The size of the record in bytes. The **/R** switch must be appended to *recordsize*. To read DG/RDOS format tapes, set recordsize to 510.

blocksize/**B** The number of records in a block; the default is 1. The **/B** switch must be appended to blocksize. The *blocksize*/**B** argument is optional.

Global Switches

/A Convert EBCDIC code to ASCII code.

/E Convert ASCII code to EBCDIC code.

/V Allow variable length records (for ASCII text files).

/P Give the user even parity.

What It Does

TFER copies blocks of up to 8192 bytes from tape to disk or from disk to tape. You can also use **TFER** to perform ASCII/EBCDIC conversions during the transfer. **TFER** accepts variable length ASCII records. **TFER** does not produce an DG/RDOS dump format tape. (All tape I/O is done via **MTDIO**—see *Commands, Statements, and Functions in Business BASIC*.)

How To Use It

Execute **TFER** by entering it from the Business BASIC CLI. The arguments *inputfile*, *outputfile*, and *recordsize*/**R** must follow **TFER**. If you don't use *blocksize*/**B**, **TFER** assumes that you want one record per block. If you use global switches, append them either to the command word or another global switch.

For the *input/output* file argument, you can use a **LINK** or **EQUIV** (DG/RDOS only) command for the tape drive, but not for the drive and the file. Don't use a directory specifier since **TFER** interprets anything with a colon as a tape drive.

TFER

Continued

To read a DG/RDOS format tape, you must specify 510 for *recordsize*/**R**. If you use the /**V** switch to specify variable length records, **TFER** pads the records with spaces to the specified record size or truncates characters that go beyond the specified record size. If you want to change fixed length records to variable length records, **TFER** truncates trailing spaces. A file cannot span two reels of tape.

Examples

1. **TFER** copies an EBCDIC tape file with 80-byte records (20 records per block) to the disk file **SOURCE.SR**, converting to ASCII and dropping trailing spaces in the process.

   !TFER/A/V MT0:0 SOURCE.SR 80/R 20/B

2. **TFER** copies an ASCII source file on disk in current directory to the tape file **MT0:0**. **TFER** allows for 80-byte records (1 record per block), converts the file to EBCDIC, and pads the records with spaces.

   !TFER/E/V MYFILE.SR MT0:0 80/R

TYPE

BASIC CLI Command

Displays a file at your terminal.

AOS/VS	DG/RDOS	UNIX

Format

!**TYPE** *filename1* [*filename2* ...]

Arguments

filename	The name of a text file.

Global Switches (AOS/VS and DG/RDOS only)

/A	Include permanent symbols in the cross-reference list (**LET**, **IF**, etc). Must be used with **/X**.
/D	Allow duplicate symbols in the cross-reference list. Must be used with **/X**.
/H	Print a heading at the top of each screen page.
/O	Print library subroutines. Must be used with **/H** or **/X**.
/X	Print a symbol cross-reference list at the end of the listing (see **TABLE**).

What It Does

TYPE displays a text file at your terminal. To display a binary file, use **FPRINT**. Use the global switches with **TYPE** if you want to display a cross-reference table for listing files (see the Business BASIC CLI command **TABLE**).

How To Use It

Execute **TYPE** by entering it from the Business BASIC CLI. **TYPE** must be followed by at least one filename. If you use global switches, append them either to the command word or to another global switch.

Example

This command line displays the contents of the subroutine **POSFL.SL** on the terminal.

TYPE

Continued

```
* !TYPE POSFL.SL
:! POSFL - POSITION A LOGICAL SUBFILE
:#
:     POSFL positions the file on logical channel F% to record R1.
:     It returns in C% the physical channel that the file is open on.
:     An ERROR 16 occurs at statement 9615 if R1 is less than zero.
:     An ERROR 16 occurs at statement 9620 if R1 is beyond the end of
:     the file.
:> CALLING SEQUENCE
:     F%   - Logical file number
:     R1   - Logical record number
:     GOSUB 09610
:     C%   - Physical channel number
:     R8   - Logical byte position (for LOCK)
:
:> ALTERNATE CALLING SEQUENCE
:     F%   - Logical file number
:     R1   - logical record number
:     V%   - Beginning byte within the record
:     GOSUB 09612
:     C%   - Physical channel number
:     R8   - Logical byte position (for LOCK)
:$ STORAGE USED
:     V%   - OFFSET INTO RECORD
:     R9   - BYTE POSITION OF RECORD R1
:     C1   - FILE CHARACTERISTICS ARRAY
:            C1[F%,0] - PHYSICAL CHANNEL #
:            C1[F%,1] - BYTE OFFSET TO REC 0
:            C1[F%,2] - MAX # OF RECORDS
:            C1[F%,3] - # BYTES/RECORD
::
09610 REM \ POSFL.SL
09611 LET V%=0                :FOR POSITION TO BEGINNING OF RECORD
09612 REM \ POSFL WITH OFFSET V%
09613 LET C%=C1[F%,0]         :CHANNEL #
09615 IF R1<0 THEN LET V%=1/0          :INVALID REC #
09620 IF R1>C1[F%,2] THEN LET V%=1/0   :ILLEGAL REC # - PANIC ! ! !
09625 LET R8=R1*C1[F%,3]      :LOGICAL FILE BYTE POINTER
09630 LET R9=C1[F%,1]+R8+V%   :BYTE POSITION OF R1 IN PHY FILE
09640 POSITION FILE[C%,R9]
09645 RETURN
09649 REM * END POSFL.SL


*
```

UCHANS *Utility*

Displays your channel assignments.

AOS/VS	DG/RDOS	UNIX

Format

{ RUN / SWAP / CHAIN } "UCHANS

What It Does

UCHANS tells you which files are open on which channels. The first column of the **UCHANS** display is the channel number, and the second is the physical filename opened on that channel.

How To Use It

Execute **UCHANS** by entering **RUN**, **CHAIN**, or **SWAP "UCHANS**. Don't execute **UCHANS** from the Business BASIC CLI because the Business BASIC CLI performs a **CLOSE**.

Example

UCHANS shows that six files are open on a DG/RDOS system and that they are using channels 0, 1, 2, 5, 7, and 8.

```
* SWAP "UCHANS

0       CUSTLOG.LG
1       CUST
2       $LPT
5       CUST13
7       CUST12
8       CUST.TB
```

UNFORM.SL *Subroutine*

Positions the cursor to the input/output fields and handles unformatted fields for SM screen files.

AOS/VS	DG/RDOS	UNIX

What It Does

UNFORM.SL is part of the Screen Maintenance utility. It is explained under **SM**.

Even though UNIX systems do not support **SM**, you can use this subroutine on UNIX systems. If you are using Business BASIC DG mode (specified by including the **-D** option on the command line to execute Business BASIC), you must use 7-bit mode. **SM** screens do not support 8-bit mode. If you are using Business BASIC in non-DG mode, you must specify the **-C** option when you use **SM** screens. This is because the screens contain embedded DG characters.

UNLINK

BASIC CLI Command

Deletes a link entry.

AOS/VS	DG/RDOS	UNIX

Format

!UNLINK *linkname1* [*linkname2* ...]

Argument

linkname — The name of a link entry established by a **LINK** command. You can use dash (-), asterisk (*), and plus (+) templates. (In this case, UNIX systems use AOS/VS templates.)

Global Switches

/C — Confirm each *linkname* before its deletion. The system echoes the *linkname* on the terminal. You approve the deletion of that *linkname* by pressing CR or New Line. If you press any other key, the *linkname* is not deleted.

/L — List deleted entries on the default output queue (overrides /V).

/V — Display the names of the deleted link entries at your terminal.

Local Switch

/N — Do not delete links matching the name in the *linkname* argument to which this switch is appended. Templates can be used; they function as they do on your operating system. If you use this switch, it must be appended to *linkname*.

What It Does

UNLINK deletes the link entries from the current directory without affecting the resolution file.

How To Use It

Execute **UNLINK** by entering it from the Business BASIC CLI. **UNLINK** must be followed by the name of at least one link entry. You can unlink several files with a single **UNLINK** command. If used, global switches are appended either to the command or to another global switch. When the /N local switch is used, it must be appended to *linkname*.

UNLINK

Continued

Example

Delete and list on the terminal all link entries that begin with TEMP and have the .SR extension, as well as the links **BARRY.LS** and **TEMP.LK**.

```
!UNLINK/V BARRY.LS TEMP-.SR TEMP.LK
BARRY.LS
TEMP0202.SR
TEMPORARY.SR
TEMP.LK
```

VACUUM *Utility*

Creates DG/RDOS CLI macros to work with subdirectories and partitions.

DG/RDOS

What It Does

Since this utility can only be used on DG/RDOS systems by someone with system manager privileges, **VACUUM** is explained in the *Business BASIC System Manager's Guide*.

VAR

Utility

Lists the variables in a SAVE file or a program in working storage.

AOS/VS	DG/RDOS

Format

{ RUN / SWAP / CHAIN } "VAR

What It Does

VAR supplies an alphabetical list of the variables in a SAVE file or in a program in working storage. **VAR** displays all the variables defined in that program's symbol table.

NOTE: On UNIX systems, use the **VAR DISPLAY** command to display variables. See *Commands, Statements, and Functions in Business BASIC* for more information.

How To Use It

Execute **VAR** by entering **RUN**, **CHAIN**, or **SWAP "VAR**. To use **VAR** with a program in working storage, enter **SWAP "VAR**. Use **RUN** or **CHAIN "VAR** with a SAVE file. Once executed, **VAR** asks for the output file. You can specify the line printer or any output file, or press CR/New Line to have **VAR** display the list at your terminal. If **VAR** is executed using **RUN** or **CHAIN**, it asks for the program name. You must specify the name of a program file (i.e., the name you used to save the program). The name of the program file cannot be the same as the name of your output file. If you enter the same name for your input and output files, **VAR** asks you for a different output filename.

Example

VAR lists the variables for **PROG1**. Since no output file is specified, **VAR** displays the five variables at the terminal.

```
* RUN "VAR

OUTPUT FILE:

PROGRAM NAME:PROG1

PROG1 5 VARIABLES

A$    C%    C1    REC$     R1
```

VFU

BASIC CLI Command

Edits a forms control file for a data channel printer.

AOS/VS	DG/RDOS

This documentation applies to DG/RDOS only. On AOS/VS systems, **VFU** runs the AOS/VS CLI utility **FCU.PR** and takes no arguments or switches. To use **VFU** with AOS/VS Business BASIC, you need to be able to create a son process, and you must have **FCU.PR** on your search path. For documentation on how to use **VFU** on AOS/VS systems, see the AOS/VS CLI manual.

Format

!VFU *vfu-filename*

Argument

vfu-filename	The name of a **VFU** file, indicated by a .VF extension, that you are creating or accessing.

Global Switches

/C	Create a new **VFU** file.
/E	Edit an existing **VFU** file.
/F	Allow uppercase and lowercase characters. Use **/F** when the line printer has a full character set. Always use this switch with **/X**.
/L	List the **VFU** file to the line printer.
/N	Line printer has non-DMA controller; use this switch with **/X**.
/P	Print the alignment file after loading **VFU** (default file: **ALIGNVFU**).
/T	Display and/or alter the tab settings (DMA printer only).
/S	Direct the global printer action (**/L** and **/X**) to the secondary line printer, **$LP2** (**$LP2** should be linked to **$LPTl**).
/V	Verify channel and tab settings at the terminal.
/X	Transfer the **VFU** file to the printer's **VFU** unit.

Local Switches

/L	The listing file to receive the **VFU** file.
/P	The alignment pattern file.

What It Does

VFU works with forms control files for printers. The Business BASIC CLI command:

- Creates the file you specify with *vfu-filename* and appends .VF to the filename.

VFU

Continued

- Displays files.
- Edits existing .VF files.
- Loads existing .VF files into the printer's memory.

You can specify three settings for the printer forms in each *vfu-filename*: tab stops, form size in lines, and multiple line-number/channel-number pairs.

How To Use It

Before using **VFU**, you must enable access to the printer's memory in Business BASIC by executing **VFU/A** from the DG/RDOS CLI. Do not execute **VFU/X** from the DG/RDOS CLI because that command overrides **VFU/A**. All enabling or disabling of transfers to a printer's memory in Business BASIC is done through the DG/RDOS CLI. Each enabling or disabling command must be entered separately.

Execute **VFU** from the Business BASIC CLI. Use a global switch to specify the **VFU** function you want. You can enter multiple commands with multiple switches. In a multiple command line, the create and edit commands are processed first, followed by load commands, then display commands. You do not need to enter the .VF extension to access **VFU** files.

The **VFU** commands and their prompts are explained below.

Create a .VF File

When you enter **VFU/C** *filename*, the **VFU** program displays its banner and the new filename and begins the following dialog:

1. TAB CONTROL:
   WANT STANDARD TABS (EVERY 8 COLUMNS), ENTER Y/N

   If you want standard tab stops (at columns 0, 7, 15...127), enter **Y**; **VFU** then skips to question 3. To set your own tabs, answer **N**; **VFU** asks:

2. COLUMN NUMBER OR <CR>

   This lets you set the number of line tab stops you want. Enter each column number at which you want a tab stop in this file; press CR after entering each number. **VFU** repeats this question until you enter CR without a number.

3. VFU CONTROL:
   WANT STANDARD (11 INCH), ENTER Y/N

   Once entered, you cannot change your answers to questions 3 and 4 in this file. A standard form is 11 inches (66 lines) long, has channel 1 set for line 1, and channel 12 set for line 63. If you want to use the standard forms, enter **Y**; **VFU** then skips to question 5. To specify a different form length, enter **N**; **VFU** then displays question 4.

VFU

Continued

4. FORM LENGTH IN LINES (1-143)

   Enter the number of vertical lines you want on a printed page.

5. LINE NUMBER OR <CR>

   Specify a vertical line in which you want to set a channel hole; then press CR. **VFU** asks:

   CHANNEL NUMBER:

   Specify the channel number that you want associated with the line number you gave in the last question. **VFU** repeats questions 5 and 6 until you enter CR in response to question 5.

Once these questions have been answered, **VFU** creates *vfu-filename*.VF and returns you to the Business BASIC CLI.

Display a File

To display the **VFU** tab stops and channel settings, enter either **VFU/L** to send the file to the first line printer or **VFU/V** to display the file at your terminal. You must enter the name of the file with these commands. You can also place the information in an output file instead of displaying it by appending the local **/L** switch to the name of an output file.

When you use the **VFU** display commands, **VFU** presents the line number/channel settings in the form 1–*c*, where 1 is the line number of a hole and *c* is the channel number of a hole. If you enter the default value for question 3, (11-inch forms), **VFU** channels are shown as:

1-1 63-12

Edit a File

To edit a file, append the **/E** switch to the **VFU** command and specify the *vfu-filename* you want to edit. If you also use a display switch with **/E** (e.g., **VFU/V/E**), the *vfu-filename* is displayed and the following dialog begins:

1. COLUMN NUMBER OR <CR>:

   Enter the number of the column where you want to add a tab or to clear an existing tab. Press CR to enter information.

2. SET (S) OR CLEAR (C):

   To set a tab at the column number specified in step 1, enter **S**; to clear a tab at this column, enter **C**. **VFU** repeats the sequence until you press CR without a column number in response to question 1.

3. FORM LENGTH IN LINES(1-143):

   Enter the number of vertical lines you want to display.

VFU

Continued

4. LINE NUMBER OR <CR>

   Enter the line number of the channel hole that you want to set or clear.

5. CHANNEL NUMBER:

   Enter the channel number that you want associated with line number (for set) or that is already associated with the line number (for clear).

6. SET (S) OR CLEAR (C):

   To set a new channel hole, enter **S**; to clear an existing hole, enter **C**. **VFU** then repeats questions 4, 5, and 6 until you press CR without any text in response to question 4.

Once you finish the **/E** (edit) dialog, **VFU** updates the *vfu-filename* with the new settings and displays the new settings on the display file you specified (with **/V**, etc.).

Load a File into the Printer's Memory

For the first data channel line printer, enter **VFU** *vfu-filename* or **VFU/X** *vfu-filename*; for the second data channel line printer, enter **VFU/S** *vfu-filename*. After you enter the load command, **VFU** displays a ready prompt. Then, when you press a key, **VFU** halts spooling and uses **XFER** to copy the file into the printer's memory, allowing you to print files on forms that need the format control contained in the **VFU** file.

Examples

1. **VFU** is used to create the file **PAYROLL1.VF**.

```
!VFU/C PAYROLL1

CONTROL PROGRAM
CREATING PAYROLL1.VF
TAB CONTROL:
WANT STANDARD TABS (EVERY 8 COLUMNS), ENTER Y/N: N
COLUMN NUMBER OR <CR>: 3
COLUMN NUMBER OR <CR>: 9
COLUMN NUMBER OR <CR>: 16
COLUMN NUMBER OR <CR>: 28
COLUMN NUMBER OR <CR>: 50
COLUMN NUMBER OR <CR>:
VFU CONTROL:
WANT STANDARD (11 INCH), ENTER Y/N: N
FORM LENGTH IN LINES (1-143): 44
LINE NUMBER OR <CR>:  1
CHANNEL NUMBER: 1
LINE NUMBER OR <CR>:  4
CHANNEL NUMBER: 2
```

VFU

Continued

```
LINE NUMBER OR <CR>:  9
CHANNEL NUMBER: 3
LINE NUMBER OR <CR>:  41
CHANNEL NUMBER: 12
LINE NUMBER OR <CR>:
```

2. **VFU** is used for displaying and editing **PAYROLL1.VF**, which was created in example one.

   !VFU/V/E PAYROLL1

```
CONTROL PROGRAM
PAYROLL1.VF 01/16/85 14:22:16
TAB STOPS:
3, 9, 16, 28, 50
VFU CHANNELS:
1-1, 4-2, 9-3, 41-12

EDITING PAYROLL1.VF
TAB CONTROL:
COLUMN NUMBER OR <CR>: 8
SET (S) OR CLEAR (C):  S
COLUMN NUMBER OR <CR>: 9
SET (S) OR CLEAR (C):  C
COLUMN NUMBER OR <CR>:
VFU CONTROL:
LINE NUMBER OR <CR>:  14
CHANNEL NUMBER: 4
SET (S) OR CLEAR (C): S
LINE NUMBER OR <CR>:
PAYROLL1.VF 06/15/83 14:28:25
TAB STOPS:
3, 8, 16, 28, 50
VFU CHANNELS:
1-1, 4-2, 9-3, 14-4, 41-12
```

   Now, **VFU** is used to transfer **PAYROLL1.VF** to the first data channel line printer. Prior to executing this command, you must have enabled the transfer to the printer in Business BASIC by entering **VFU/A** from the DG/RDOS CLI.

   !VFU PAYROLL1

```
CONTROL PROGRAM
PREPARE TO LOAD PAYROLL1.VF
WAIT UNTIL OUTPUT TO THE PRINTER HAS
COMPLETED. MAKE SURE PRINTER IS READY
AND ON-LINE.
STRIKE ANY KEY WHEN READY
```

VLCONVERT *Utility*

Converts volume label files to current revision of Business BASIC

AOS/VS	DG/RDOS	UNIX

What It DOES

You use the **VLCONVERT** utility to convert volume label files that were produced under previous revisions of Business BASIC to the current revision of Business BASIC. Information on this utility, like other program conversion tools, is documented in the on-line file **CONVERT.DOC**, located in the Business BASIC directory **DOC**.

VLPRINT

BASIC CLI Command

Displays the contents of a volume label (.VL) file.

AOS/VS	DG/RDOS	UNIX

Format

!VLPRINT *physical-file*

Arguments

physical-file The name of a physical database file without the .VL extension that is associated with a .VL file.

Global Switches

/L Send the listing to the default output file for this job.

What It Does

The **VLPRINT** Business BASIC CLI command supplements the **LFU** utility. Using **VLPRINT**, you can display information contained in the volume label (.VL) file of the specified physical file.

How To Use It

Execute **VLPRINT** by entering it from the Business BASIC CLI. The name of a physical database file must be entered with **VLPRINT**. If used, the global switch is appended to **VLPRINT**.

Example

VLPRINT displays the information contained on the volume label file for the physical database file **CUSTOMER**.

```
!VLPRINT CUSTOMER

DB FILE: CUSTOMER    TABLE OF CONTENTS      PAGE NO. 1

      FILE       FILE     STARTING   # OF     RECORD     LAST      # OF
      NAME       TYPE      SECTOR   SECTORS   LENGTH    RECORD     BYTES

      CUST        L           0        26       128       100      12928
    CUSTI1        I          26        11       512        10       5632
    CUSTI2        I          37        11       512        10       5632
    CUSTI3        I          48        11       512        10       5632
                                      ------                       ------
                    TOTAL SECTORS: 59                BYTES:       29824
```

XBUILD *Utility*

Builds an index from a data file.

AOS/VS	DG/RDOS	UNIX

Format

SWAP "XBUILD

What It Does

XBUILD uses the **KADD** command to build an index file with a blocking factor of approximately 50 percent for a linked-available-record file.

How To Use It

Do not use **RUN** or **CHAIN** with **XBUILD**; use only **SWAP**. To execute **XBUILD**, do the following:

1. Open your input and index files.
2. Set up an argument string that includes the channel numbers for the input and index files. This string is like the argument string for **IBUILD** except that **XBUILD** ignores the blocking factor.
3. Use the **BLOCK WRITE** command to put the argument string in the common area.
4. Enter the statement **SWAP "XBUILD**.
5. Check **STMA 1,1** to see if an error occurred. The utility returns either a 0 for a successful **XBUILD** or the appropriate Business BASIC error code. If the error occurred in an **ON ERR** trap, the line number of the error is returned in **STMA 1,2**.

The following error codes can occur when you are using **XBUILD**:

Error Code	Meaning
45	Illegal record length.
68	Index file full.
77	Illegal record number.
146	Key already exists and duplicates are not allowed.
148	File not on sector boundary.
149	Record out of sequence.
150	Illegal blocking factor.
151	Illegal key length.

XBUILD

Continued

XBUILD looks for a key in the data file based on the locations you specify; the key can have multiple fields. **XBUILD** then computes the record pointer based on the record accessed relative to record 0. In building the index file, **XBUILD** skips record 0. It checks each record's status (first two bytes) so that deleted records (indicated by a status value of 0) are omitted from the index. Table 1–19 explains the substring locations and contents descriptions for the **XBUILD** argument string.

Table 1–19. XBUILD Argument String

Substring Location	Size In Bytes	Contents
1,4	4	Channel number of the open input file
5,8	4	Byte offset to record 0 of input file
9,12	4	Maximum number of records in input file, including record 0 of a linked-available-record file
13,16	4	Number of bytes per record in input file
17,20	4	Channel number of opened index file
21,24	4	Byte offset to record 0 of index file
25,28	4	Maximum number of blocks in index file (use **INDEXCALC** to calculate this)
29,32	4	Number of bytes per block in index file, either 512 bytes (DG/RDOS, AOS/VS, UNIX) or 2048 bytes (AOS/VS, UNIX)
33,34	2	Keys per block at the specified blocking factor (use the **INDEXCALC** utility)
35,36	2	Flag to allow duplicate keys: 1 Allow duplicate keys. 0 Do not allow duplicate keys.
37,38	2	Flag to check for deleted records: 1 Check for deleted records. 0 Do not check for deleted records.
39,40	2	Total key field length in bytes
41,42	2	Number of key fields
*43,44	2	First byte of the key (1 is the first byte of the record)
*45,46	2	Last byte of the key
47,*	4	The next key field's descriptors. Repeat the sequence of entering the first and last byte of the key. Since you can have multiple keys, you can enter this information several times.

XBUILD

Continued

Example

This segment of code sets up an argument string and then swaps to **XBUILD**.

```
00010 DIM X$ [512]              :DIMension the argument string X$
00020 LET ERCODE=0              :Initialize ERCOD and LINENO for
00030 LET LINENO=0              :use with STMAs.
.                               :Put code to open both files here.
.                               :Channel of input file is C%,
.                               :CHANNEL OF NEW INDEX FILE IS CT%.
00100 LET X$=CHR$(C%,4),CHR$(0,4)     :Input file has 0 offset.
00110 LET X$[0]=CHR$(100,4),CHR$(100,4)
                                      :100 records and 100 bytes per
                                      :record.
00120 LET X$[0]=CHR$(CT%,4),CHR$(0,4) :Index file (CT%) has 0 byte
                                      :offset.
00130 LET X$[0]=CHR$(8,4),CHR$(512,4) :8 blocks and 512 bytes per
                                      :block.
00140 LET X$[0]=CHR$(36,2)   :Blocking factor is 36 entries/blocks.
00150 LET X$[0]=CHR$(1,2),CHR$(1,2)   :Allow duplicates, check
                                      :deleted recs.
00160 LET X$[0]=CHR$(10,2),CHR$(2,2)  :Total key 10 bytes, 2 fields.
00170 LET X$[0]=CHR$(3,2),CHR$(6,2)   :First field, bytes 3 to 6.
00180 LET X$[0]=CHR$(15,2),CHR$(20,2) :Second field, bytes 15 to 20.
00190 LET X$[0]=FILL$(0)              :No more fields.
00200 BLOCK WRITE X$                  :Send into common area.
00210 SWAP "XBUILD"                   :Execute XBUILD and return.
00220 STMA 1,1,ERCODE                 :Get error code from XBUILD.
00230 STMA 1,2,LINENO                 :Get line of error if ON ERR
                                      :trap.
00240 IF ERCODE THEN GOTO 00700       :If XBUILD not successful,
                                      :go to 700.
.
.
.
```

XFER

BASIC CLI Command

Copies one file to another file.

AOS/VS	DG/RDOS

Format

!XFER *sourcefile destinationfile*[*switches*]

Arguments

sourcefile, destinationfile	The names of any device or disk files. Each can include directory specifiers, but neither can be a directory.
switches	Any of the local switches. If used, the switches are appended to *destinationfile*.
	/C Organize *destinationfile* contiguously (both files must be disk files).
	/R (DG/RDOS only). Organize *destinationfile* randomly.
	/N Do not create *destinationfile*; it must already exist.

Global Switches

/A	Perform an ASCII transfer. Transfer the file line by line, taking appropriate read/write actions, such as inserting line feeds after carriage returns, etc.
/B	Append *sourcefile* to *destinationfile*; this is required if you are copying to an existing disk or tape file.

What It Does

XFER copies *sourcefile* to *destinationfile*. When you use **XFER** to copy a file to a disk file, **XFER** creates *destinationfile* unless you specify the **/N** local switch. When you use **XFER** to copy a file to an existing disk or tape file, you must use the **/B** global switch. **XFER** does not copy the old filename, attributes, creation date, etc.

How To Use It

Execute the command by entering **XFER** from the Business BASIC CLI. You must enter a *sourcefile* name and a *destinationfile* name with **XFER**. If you use global switches, append them either to the command word or to another global switch. Local switches are appended to *destinationfile*.

With **XFER**, you must use **$TRI** to refer to your terminal for input. To refer to your current terminal as the output buffer, use **$TRO**. Never use a terminal device that the Business BASIC interpreter uses for your input or output. In DG/RDOS, for example, do not refer to **$TTI** and **$TTO** if Business BASIC has been brought up in the background or to **$TTI1** and **$TTO1** if Business BASIC has been brought up in the foreground.

XFER

Continued

In DG/RDOS, if you omit switches, **XFER** organizes *destinationfile* sequentially. To organize it randomly, you must use the **/R** local switch. To organize it contiguously, both files must be disk files, and you need to use the **/C** local switch. If you use **XFER** to a SAVE file, be sure to use **CHATR** to add an **S** attribute to *destinationfile*.

Examples

1. Business BASIC appends the file **SUB.SR** to **MAIN.SR**.

   !XFER/B SUB.SR MAIN.SR

2. Business BASIC copies the file **MAIN.SR** in ASCII format to the file **LISTFILE**.

   !XFER/A MAIN.SR LISTFILE

3. Business BASIC copies the file **TEXT** in ASCII format to the tape file 0 on magnetic tape drive 0.

   !XFER/A/B TEXT MT0:0

4. In DG/RDOS, **XFER** moves the file **JOELCLINE** to the printer.

   !XFER/A JOELCLINE $LPT

End of Chapter

Appendix A
Business BASIC Subroutines and Utilities

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

Table A-1 Business BASIC Subroutines

Subroutine	Line Numbers	Function
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 logical file.
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).

Table A-1. Business BASIC Subroutines *(continued)*

Subroutine	Line Numbers	Function
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.
	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 calls 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.
	9850	ERROUT outputs an error message on the command line.
	9890	ERRCLEAR clears the error message from the screen.

Table A-1. Business BASIC Subroutines *(concluded)*

Subroutine	Line Numbers	Function
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	For lowercase, sets the necessary STMAs and block mode. Also, 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$.
	9735	Rereads SCRN$.
	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.

This table lists the Business BASIC utilities and summarizes their functions.

Table A-2. Business BASIC Utilities

Utility	Function
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.

Table A-2. Business BASIC Utilities (*concluded*)

Utility	Function
PARAMCON	Converts a PARAM file structure into a logical file structure.
PARAMPRT	Prints the contents of the PARAM file.
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 a 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.

End of Appendix

Appendix B
Business BASIC PARAM and Logical File Database Routines

This table describes the Business BASIC subroutines and utilities that work with the PARAM file database structure.

Table B-1 PARAM Routines

Routine	Type	Function
DBGEN	Utility	Builds files for the PARAM file structure.
DELREC.SL	Subroutine	Deletes a record in a linked-available-record file and places it on the deleted-record chain.
FILESORT	Utility	Sorts a data file.
FINDFILE.SL	Subroutine	Finds a subfile and builds a C1 array.
FM	Utility	Provides file maintenance functions for data files and table files.
FMLOG	Utility	Prints an FM log file.
FMPRINT	Utility	Prints a file maintained by FM.
GETREC.SL	Subroutine	Gets the number of the next available record in a linked-available-record chain.
IBUILD	Utility	Builds an index file.
INDEXBLD	Utility	Builds or rebuilds index files.
INDEXCALC	Utility	Calculates the size of an index file.
INDEXPRT	Utility	Prints an index file.
INDEXVRFY	Utility	Verifies the structure of an index file.
INITFILE	Utility	Creates and/or initializes index or data files.
INITINDEX.SL	Subroutine	Initializes an index file.
OPEN	Utility	Opens physical files and subfiles.
PARAMCON	Utility	Converts a PARAM file structure into a logical file structure.
PARAMPRT	Utility	Prints the contents of the PARAM file.
POSFL.SL	Subroutine	Positions the file pointer to a record in a data file.
QFILESORT	Utility	Quickly sorts a data file.
RELINK	Utility	Recreates the deleted record chain of a linked-available-record file.
TBUILD	Utility	Builds a tag file.
XBUILD	Utility	Builds an index file.

This table describes the Business BASIC subroutines and utilities that work with the logical file database structure.

Table B-2 Logical Routines

Routine	Type	Function
DBFIX	BASIC CLI command	Adjusts the characteristics of logical database files.
DBMOVE	Utility	Moves logical file structures from one directory to another.
FILESORT	Utility	Sorts a data file.
GETLAST.SL	Subroutine	Retrieves the number of active records and the highest record in use in a linked-available-record file.
IBUILD	Utility	Builds an index file.
INDEXCALC	Utility	Calculates the size of an index file.
INDEXPRT	Utility	Prints an index file.
INDEXVRFY	Utility	Verifies the structure of an index file.
IREBLD	Utility	Rebuilds an index file.
LFDATA.SL	Subroutine	Gets the file description.
LFM	Utility	Provides file maintenance functions.
LFU	Utility	Creates and manipulates files.
LINDEXBLD	Utility	Builds or rebuilds index files.
LINITINDEX.SL	Subroutine	Initializes an index file that was opened with the LOPEN file statement.
LRELINK	Utility	Recreates the deleted record chain of a linked-available-record file.
LXFER	Utility	Copies one logical file to another.
QFILESORT	Utility	Quickly sorts a data file.
TBUILD	Utility	Builds a tag file.
VLPRINT	Utility	Displays the contents of a volume-label file.
XBUILD	Utility	Builds an index file.

End of Appendix

Index

A

B

C

D

E

F

G

I

J

K

L

M

O

P

Q

R

S

T

U

V

W

X

Related Documents

Commands, Statements, and Functions in Business BASIC 093-000351

Defines each Business BASIC command, statement and function. It is an alphabetical reference manual for programmers.

Using Business BASIC on DG/UX™ and INTERACTIVE UNIX Systems 093-000685

Describes how to load and generate Business BASIC on DG/UX™ and INTERACTIVE UNIX systems. It is intended for the system manager or system operator.

Business BASIC System Manager's Guide 093-000388

Describes how to load and generate Business BASIC on AOS, AOS/VS, AOS/VS II, RDOS, and DG/RDOS systems. It is intended for the system manager or system operator.

Learning Business BASIC 093-000684

Acquaints experienced programmers with Business BASIC operations and programming procedures for DG/UX and INTERACTIVE UNIX systems. It provides an overview of the commands, functions, subroutines, and utilities available to programmers.

Programming with Business BASIC 093-000480

Acquaints experienced programmers with Business BASIC operations and programming procedures for AOS, AOS/VS, RDOS, and DG/RDOS systems. It provides an overview of the commands, functions, subroutines, and utilities available to programmers.

AOS INFOS® II System User's Manual 093-000152

Provides information on using the AOS INFOS II file management system.

AOS/VS INFOS® II System User's Manual 093-000299

Provides information on using the AOS/VS INFOS II file management system.

Business BASIC Summary 069-000263

Summarizes Business BASIC commands, statements, functions, and utilities in a pocket-sized book for programmers using AOS, AOS/VS, RDOS, and DG/RDOS systems.

DASHER® D2 File Maintenance and Screen Maintenance Template 093-000212

DASHER® D200 File Maintenance and Screen Maintenance Template 093-000265

DASHER® D210/211 D410/D460 CFM and CSM Template 093-000409

DASHER® D210/211 D410/D460 SM and FM Template 093-000410

TIPS ORDERING PROCEDURES

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a) MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

      Send your order form with payment to:
      Data General Corporation
      ATTN: Educational Services/TIPS G155
      4400 Computer Drive
      Westboro, MA 01581-9973

   b) TELEPHONE – Call TIPS at (508) 870-1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a) Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.
   b) Check or Money Order – Make payable to Data General Corporation.
   c) Credit Card – A minimum order of $20 is required for Mastercard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	Shipping & Handling Charge
1-4 Units	$5.00
5-10 Units	$8.00
11-40 Units	$10.00
41-200 Units	$30.00
Over 200 Units	$100.00

   If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	Discount
$1-$149.99	0%
$150-$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

6. Allow at least two weeks for delivery.

RETURNS

7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870-1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	SHIP TO: (No P.O. Boxes - Complete Only If Different Address)
COMPANY NAME _____	COMPANY NAME _____
ATTN: _____	ATTN: _____
ADDRESS _____	ADDRESS (NO PO BOXES) _____
CITY _____	CITY _____
STATE _____ ZIP _____	STATE _____ ZIP _____

Priority Code _____ (See label on back of catalog)

_____ _____ _____ _____ _____
Authorized Signature of Buyer (Agrees to terms & conditions on reverse side) | Title | Date | Phone (Area Code) | Ext.

ORDER #	QTY	DESCRIPTION	UNIT PRICE	TOTAL PRICE

A SHIPPING & HANDLING

☐ UPS	ADD
1-4 Items	$ 5.00
5-10 Items	$ 8.00
11-40 Items	$ 10.00
41-200 Items	$ 30.00
200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0 - $149.99	0%
$150 - $499.99	10%
Over $500.00	20%

Tax Exempt # or Sales Tax (if applicable) _____

ORDER TOTAL	
Less Discount See B	-
SUB TOTAL	
Your local* sales tax	+
Shipping and handling - See A	+
TOTAL - See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is _____. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number _____ Expiration Date _____

_____
Authorized Signature
(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508-870-1600.

Form 702
Rev. 8/87

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. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. 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.

2. TAXES
Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. 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.

4. LIMITED MEDIA WARRANTY
DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY
EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, 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, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY
A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. 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, excluding its conflict of law rules. 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. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)
Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision-locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

Cut here and insert in binder spine pocket

DataGeneral
Data General Corporation, Westboro, Massachusetts 01580