IBM

IBM BASIC/VM System Services

Program Product

B B B B B

A A A A

S S S S S

I I I I I

C C C C C

Order Number:

Program Number:

Release Number
2

IBM

IBM BASIC/VM System Services

Program Product

Order Number:
SC26-4028-2

Program Number:
5668-996

Release Number:
2

Third Edition (February 1986)

This is a major revision of, and makes obsolete, SC26-4028-1.

This edition applies to Release 2 of IBM BASIC/VM, Program Product 5668-996, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the preface. Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent republication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This manual is intended for application programmers who need *guidance and reference* material to run BASIC programs under VM/SP CMS. You should use this manual in conjunction with the operating system-independent information presented in *IBM BASIC Programming Guide* and *IBM BASIC Language Reference*.

To use this manual you should be familiar with BASIC statements, commands, and functions, as well as CMS commands.

IBM BASIC/VM is a program product that translates BASIC source programs into machine language for processing both inside and outside the BASIC environment. IBM BASIC/VM runs under:

- Virtual Machine/System Product—Conversational Monitor System (VM/SP CMS) interactive and batch, Release 3 or 4

Throughout this manual, the name of the product, IBM BASIC/VM, and the shortened names IBM BASIC and BASIC, are used interchangeably. BASIC is the name of the language this product uses.

Manual Organization

This manual is organized into three sections.

The first section gives guidance information on how to use BASIC under CMS, with explanations and examples of CMS commands useful with BASIC.

The second section gives you information on using BASIC under IBM Virtual Machine/Personal Computer (VM/PC).

The third section gives reference information; it lists CP and CMS commands used most often by the BASIC programmer. Options relevant to BASIC are discussed.

Note: Examples in this manual assume that you are using certain models of the IBM 327x display terminal. If you are not, you may need to check with your system administrator to adapt these examples for your terminal.

Industry Standards

The IBM BASIC/VM program product is designed according to the specifications of the following industry standards, as understood by IBM as of December, 1984:

- American National Standard for minimal BASIC, ANSI X3.60-1978
- International Organization for Standardization proposed standard ISO Minimal BASIC dp ISO-6373
- European Computer Manufacturers Association, Standard ECMA-55 Minimal BASIC, January 1978

These standards are technically equivalent.

In addition, IBM BASIC has many features not contained in the above standards.

IBM BASIC Publications

The IBM BASIC publications are designed to help you develop your programs with minimum difficulty.

This manual, *IBM BASIC/VM System Services*, provides you with IBM BASIC/VM-specific guide and reference documentation for running BASIC programs under CMS.

The other IBM BASIC programming publications are:

- *IBM BASIC Programming Guide*, SC26-4027, contains guidance information on designing, coding, debugging, testing, and executing BASIC programs.
- *IBM BASIC Language Reference*, SC26-4026, gives the semantic rules for coding BASIC programs.
- *IBM BASIC Language Reference Summary*, SX26-3736, is a pocket-size card designed for quick reference to BASIC.

VM/SP CMS Publications

Detailed system operating information has been omitted from the IBM BASIC publications. Therefore, to use BASIC effectively, you should have the following manuals available:

- *IBM Virtual Machine/System Product: CMS Command and Macro Reference*, SC19-6209
- *IBM Virtual Machine/System Product: CMS User's Guide*, SC19-6210
- *IBM Virtual Machine/System Product: Terminal Reference*, GC19-6206
- *IBM Virtual Machine/System Product: CP Command Reference for General Users*, SC19-6211

- *IBM Virtual Machine/System Product: System Product Editor Command and Macro Reference*, SC24-5221
- *IBM Virtual Machine/System Product: System Product Editor User's Guide*, SC24-5220
- *IBM Virtual Machine/System Product: System Messages and Codes*, SC19-6204
- *VSE/VSAM Programmer's Reference*, SC24-5145
- *Using VSE/VSAM Commands and Macros*, SC24-5144
- *VSE/VSAM Messages and Codes*, SC24-5146

If your BASIC programs communicate with the Graphical Data Display Manager (GDDM), you will find the following manuals useful:

- *Graphical Data Display Manager Base: Programming Reference*, SC33-0101
- *Graphical Data Display Manager Presentation Graphics Feature: Programming Reference*, SC33-0102

If your BASIC programs communicate with SQL/DS (Structured Query Language/Data System), you will find the following manuals useful:

- *SQL/Data System Concepts and Facilities*, GH24-5013
- *SQL/Data System Application Programming*, SH24-5018
- *SQL/Data System Terminal User's Guide—VM/SP*, SH24-5045

For calls to other languages, you will find the following manuals useful:

- *IBM OS/VS COBOL Compiler and Library Programmer's Guide*, SC28-6483
- *VS FORTRAN Programming Guide*, SC26-4118
- *OS PL/I Optimizing Compiler: Programmer's Guide*, SC33-0006
- *OS PL/I Optimizing Compiler: CMS User's Guide*, SC33-0037

Summary of Amendments

IBM BASIC/VM Release 2, February 1986

New title

The title of this manual has changed to *IBM BASIC/VM System Services*. The content of the manual is unchanged, except as noted in the following paragraphs.

Calling Programs Written in Other Languages

- Information has been added for the new SQL Interface and a new section has been added for calling SQL/DS.
- The section on calling GDDM (Graphical Data Display Manager) has been updated.

Using the COMPILE Command

Information has been added on specifying an external file.

Using the Set Log Command

This section has been updated to include the new APPEND option.

File Specification in BASIC Statements and Commands

Information has been added to indicate that the term 'file-spec' now, optionally, includes filetype and filemode in addition to filename.

Accessing Tapes and FILEDEF Command

These sections have been updated to indicate the support for multifile tapes.

Edit Command and Edit Subcommands

These sections have been deleted because XEDIT is the preferred editor to use outside the BASIC environment.

New Topics

- Using BASIC under VM/PC
- Using the INITIALIZE Command
- Using the PROFILE Command
- Using the RENAME Command
- Changing the Virtual Console
- Using the Console Stack
- Accessing a SQL/DS Relational Data Base
- Using the CMS FILEDEF Command
- Appendix C—IBM BASIC/VM Release 1 Library
- Appendix D—Shared Segment Starter

Appendix A and Appendix B

These appendixes have been updated to include new module names.

Service Changes

Several technical and minor editorial changes have been made throughout.

IBM BASIC/VM Release 1.1, December 1983

New title

The title of this manual has changed to *IBM BASIC/VM Application Programming: System Services*. The content of the manual is unchanged, except as noted in the following paragraphs.

Calling Programs Written in Assembler

Information has been added on calling programs written in Assembler language from your BASIC program.

Service Changes

Several technical and minor editorial changes have been made throughout the manual.

Contents

Figures

A Brief Description of IBM BASIC under VM/SP CMS

IBM BASIC/VM is a program product that runs under VM/SP CMS, Release 3 or 4.

IBM BASIC/VM is both source and object code compatible with IBM BASIC/MVS. Valid source programs and compiled object code can run on either VM or MVS. The only changes necessary are for any system-dependent functions invoked in the CALL SYSTEM statement, for any system-dependent file specifications in the OPEN or CHAIN statements, and for system-dependent exception codes.

The information provided in this manual is system dependent. This manual tells you how to use IBM BASIC/VM both inside and outside of the BASIC environment.

The Processor

The IBM BASIC Processor (hereafter called "the Processor") processes programs, written in the BASIC language, either inside or outside the BASIC environment.

When inside the BASIC environment, the Processor provides full screen and line editing facilities for creating and modifying programs, immediate syntax checking and error diagnostics, and producing machine-language "code packets," which are executed under the control of the Processor on a statement-by-statement basis. Also, with the COMPILE command, all facilities outside the BASIC environment are made available from inside the BASIC environment.

Outside the BASIC environment, the Processor acts as a compiler that accepts a source program and produces an object module and a listing data set.

Using BASIC either inside or outside the BASIC environment requires the IBM BASIC Library (see below).

For information on using the facilities of the Processor, see *IBM BASIC Programming Guide.*

The Library

The IBM BASIC Library (hereafter called "the Library") contains numeric functions, advanced mathematical functions, character-handling functions, input/output and error-handling functions, and system functions such as time and date. You have to use the Library to execute programs and to use BASIC inside or outside the BASIC environment.

With the Library, you can execute a BASIC object module and you can execute a BASIC load module created by the CMS GENMOD command. These object modules and load modules are suitable to be loaded for execution on IBM System/370, IBM 30xx processors, and IBM 43xx processors.

For complete information regarding the contents and use of the Library, see *IBM BASIC Language Reference*.

System Services

This manual gives you guidance information specific to BASIC on developing and running programs under VM/SP CMS, including:

- Operating inside the BASIC environment
- Operating outside the BASIC environment
- Preparing programs for execution
- Executing programs
- File handling
- Interactive operation
- Batch operation

In addition, reference documentation is provided on CMS commands especially useful with BASIC.

Note: Examples in this manual assume that you are using certain models of the IBM 327x display terminal. You may need to check with your system administrator to adapt these examples for your terminal.

The following 327x models are supported by BASIC:

2, 2A, 2B, 2C, 3, 3B, 4, 5

Other models will operate in line mode.

CMS System Services

This section gives you task-oriented guidance information on why and how to use CMS services to develop and process BASIC programs.

You can use BASIC in four ways:

- Inside the BASIC environment where all the BASIC features are available
- As a compiler that accepts a source program file and produces an object program (text file) and a listing file
- To execute a BASIC object program (text file)
- To execute a BASIC module created by the CMS GENMOD command

BASIC communicates with you through the CMS "virtual console" mechanism. This means that you can communicate with BASIC interactively through a display type of terminal (such as the 327x family of terminals), or through other terminals as well. You can also run BASIC in disconnected mode, or using the CMS Batch Facility.

Logging On and Off CMS

Before you can begin using BASIC, you must log on CMS. After you have completed a terminal session, you must log off CMS.

Logging On CMS

To use BASIC under CMS, you must first connect with VM/SP CMS using the LOGON command. For example, if your logon id is USER3:

```
LOGON USER3
```

connects you with the VM system after you supply your password.

Your organization may have other logon procedures that you must use. Check with your system administrator.

After you are running under VM, you will see a message in the lower right-hand corner of the screen. This message tells you which subsystem is in control of your virtual machine, and the state of that subsystem. For example:

RUNNING Indicates CMS is executing. If you press the enter key twice and the word CMS appears on your screen, there are no programs executing; this means you can enter CMS commands. If the word CMS does not appear, there is a program currently executing.

CP READ Indicates that there is a CP interrupt and you are communicating directly with the Control Program. At this point, you may enter any valid CP command.

VM READ Indicates that VM is waiting for input.

MORE... Indicates that your screen is full and there is more data to be displayed. If you do nothing, the screen will clear automatically and display the remaining data in approximately 15 seconds. If you press the enter key, the word HOLDING will appear in the lower right-hand corner and the screen will not be cleared until you clear it manually.

To clear the screen manually, press either the PA2 key or the CLEAR key.

HOLDING Indicates that the current screen of data will remain until you clear the screen by pressing either the PA2 key or the CLEAR key.

Logging Off CMS

To leave CMS and drop the connection with VM/SP, issue the LOGOFF command to CMS or CP:

```
LOGOFF
```

The BASIC Environment under CMS

The BASIC environment under CMS provides you with all the language features, commands, immediate statements, and editing facilities described in *IBM BASIC Language Reference*. This mode of operation uses both the IBM BASIC Processor and the Library.

The term file-spec will be used in this section. For more information on file-specs, see "File Specification in BASIC Statements and Commands" on page 46.

Invoking the BASIC Environment under CMS

After you are connected with VM/SP CMS, you can invoke the BASIC environment by issuing the following command:

```
BASIC [(option[)]]
```

where:

option

```
PROFILE [(file-spec[)]] or NOPROF.
```

The default **file-spec** is an installation option (the IBM-supplied value is BASPROF). The default filetype is an installation option (the IBM-supplied value is BASPROF). The default option is PROFILE if the profile exists and NOPROF if the profile does not exist. For more information about files and file-specs, see "Using Files" on page 45.

This starts the IBM BASIC Processor in the BASIC environment. (Your organization may have another invocation procedure; check with your system administrator.) The profile, as determined above, is processed.

BASIC then displays the following message:

```
IBM BASIC/VM VERSION x RELEASE y.z  yyyy/mm/dd  hh:mm
  (c) Copyright IBM Corporation 1982, 1985
 *
   _
```

to indicate it is waiting for your input. (*x* is the version, and *y.z* is the release of IBM BASIC/VM you are using. The *yyyy/mm/dd* is the current date, and *hh:mm* is the current time.) The asterisk prompt is BASIC's indication that you are expected to enter a BASIC command, immediate statement, or statement beginning with a line number.

Using BASIC Commands

All the BASIC commands described in *IBM BASIC Language Reference* are available for your use.

Many of these commands operate independently of the operating system environment, but the semantics of some commands are dependent on CMS features; in particular, those commands that use files (specify a file-spec).

The BASIC commands that have rules unique to CMS are discussed in this section. Most of these rules have to do with specifying the file-spec in a command. For more information on specifying a file-spec, see "Using Files" on page 45.

Using the COMPILE Command

The COMPILE command translates the BASIC program in your workspace into machine-executable code.

Two files are created during compilation: a file with default filetype TEXT, containing the object program and a file with default filetype LISTING, containing the printable output. Both are written on the first writable disk, in normal CMS

search order if you do not specify a filemode. You can specify the filename (and, optionally, the filetype and filemode) for either or both of these files in the COMPILE command.

Example

```
COMPILE OBJECT (ALPHA) OUT (BETA)
```

compiles the program in the workspace and sends the TEXT file to ALPHA and the LISTING file to BETA.

The COMPILE command may also include a file-spec which specifies a file to be loaded into the workspace and then compiled. The default filetype is BASIC. It will function as if a LOAD command had been issued for that file (except that syntax messages for the LOAD are not displayed), followed by a COMPILE command. Syntax messages will be displayed by the compilation. The compilation is not done if you interrupt the load. The filename or ddname specified becomes the name of the workspace. For more information on how files are loaded, see "Using the LOAD Command" on page 9.

If you do not specify an OBJECT (an object is to be generated) or an OUT clause, the file-spec is taken to be the same as the name of your workspace. (BASIC requests a file-spec if your workspace does not have a name.)

Example

If the workspace name is MYPROG,

```
COMPILE
```

writes the object program to MYPROG TEXT A and the listing to MYPROG LISTING A.

You may override the default filetype and filemode by specifying them in the command.

Example

```
COMPILE TESTP BASIC C OBJECT(TEMP TEXT C) OUT(TEMP LISTING C)
```

You can override the default BASIC file definitions using CMS FILEDEF and DLBL commands (see "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65). However, because the COMPILE command may produce two files with the same ddname as the source file, the override applies only to those ddnames explicitly stated in the COMPILE command (in the OBJECT or OUT clauses).

Example

If the workspace name is MYPROG,

```
SYSTEM "FILEDEF MYFILE PRINTER"
COMPILE OUT(MYFILE)
```

causes the listing file to be sent directly to the printer; the object file MYPROG TEXT, is written to disk (see "Using the SYSTEM Command" on page 18).

Example

If the preceding example is changed to:

```
SYSTEM "FILEDEF MYFILE PRINTER"
COMPILE
```

Both files are written to disk: MYPROG TEXT and MYPROG LISTING. The FILEDEF command had no effect because MYFILE was not included as a ddname in an OUT clause.

For your convenience, with this command you can also override the default file definitions for the text and listing files with FILEDEF commands that specify a ddname of TEXT (to redirect the TEXT file) or LISTING (to redirect the LISTING file).

Example

```
SYSTEM "FILEDEF LISTING PRINTER"
COMPILE
```

directs the listing file to the printer.

However, if you specify a name in the OBJECT or OUT clauses of the COMPILE command, the FILEDEF command is effective only if it refers to that name.

Example

```
SYSTEM "FILEDEF LISTING PRINTER"
COMPILE OUT(ABC)
```

routes the listing output to ABC LISTING and *not* to the printer since ABC is not the same name as LISTING.

Using the FETCH Command

The FETCH command reads a previously stored workspace (see "Using the STORE Command" on page 17) into the workspace. You should specify the file-spec in the FETCH command. If a filetype is not specified, the default filetype is BASTEXT (which is the filetype that STORE normally uses). Unless you indicate otherwise, the normal CMS search order is used for the filemode.

The filename or ddname specified in the file-spec becomes the name of the workspace. If the file is a CMS disk file, the filename, filetype, and filemode of the file are remembered for use in a subsequent STORE command. The file specification is forgotten if a LOAD, RUN, or COMPILE command, or a CHAIN statement loads a file, or if a FETCH, INITIALIZE, or RENAME command is used to change the workspace name.

Example

```
FETCH CALC
```

reads the file CALC BASTEXT into the workspace.

You may override the default filetype and filemode by specifying them in the command (see "Using CMS Disk Files" on page 49).

The default file definitions can be overridden through the CMS FILEDEF and DLBL commands. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Using the HELP Command

When you are in BASIC's HELP mode, one of the "actions" you may choose is to "print" one or more help panels (see *IBM BASIC Language Reference*). This causes the indicated panel(s) to be written to a file with filename BASHELP and filetype LISTING.

Example

```
HELP
PRT
```

prints a listing of the displayed help panel to BASHELP LISTING on the first writable disk.

This file can be directed elsewhere, for instance, to a printer, by CMS commands. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Example

The following commands:

```
SYSTEM "FILEDEF BASHELP PRINTER"
HELP
PRT MESSAGES
```

prints a listing of all the BASIC messages on the printer. The ddname BASHELP is the only name effective for printed output from HELP.

Printing of help panels can be interrupted. See "Generating an Attention Interrupt" on page 44.

Using the INITIALIZE Command

The INITIALIZE command clears the workspace and, optionally, sets the workspace name.

If no file-spec is specified, the workspace does not have a name.

If you specify a file-spec, the filename or ddname in the file-spec becomes the name of the workspace. The file-spec is also remembered for a subsequent SAVE command (see "Using the SAVE Command" on page 15). The file-spec is forgotten if a LOAD, RUN, or COMPILE command, or a CHAIN statement loads another file, or if a FETCH, INITIALIZE, or RENAME command changes the workspace name.

Example

```
INITIALIZE ACCNTREC
```

sets the workspace name to ACCNTREC.

Example

```
INITIALIZE PAYMENT VSBASIC
```

sets the workspace name to PAYMENT and PAYMENT VSBASIC is remembered as the file-spec for a subsequent SAVE command. If you want the SAVE to go to a specific CMS disk, you may specify a filemode.

Example

```
INITIALIZE PAYMENT VSBASIC C1
```

Using the LIST Command

The LIST command includes an optional OUT clause to direct the listing to a file rather than the terminal. The file-spec must be included in the OUT clause. BASIC automatically assigns a filetype of LISTING. The normal CMS search order is used to find the first writable disk.

Example

```
LIST XREF OUT(CROSSREF)
```

writes a cross-reference listing of the program in your workspace to the file CROSSREF LISTING A.

You may override the default filetype and filemode by specifying them in the command (see "Using CMS Disk Files" on page 49).

Example

```
LIST 10 TO 20 OUT(FRAGMENT LIST C1)
```

The default file definition can be overridden with the CMS FILEDEF and DLBL commands. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Example

```
SYSTEM 'FILEDEF CROSSREF PRINTER'
LIST XREF OUT(CROSSREF)
```

sends the cross-reference listing directly to the printer (see "Using the SYSTEM Command" on page 18).

Using the LOAD Command

The LOAD command reads a BASIC source program file into your workspace. A file-spec is specified in the LOAD command. If a filetype is not specified, a default filetype of BASIC is assumed. Unless you indicate otherwise, the normal CMS search order is used for the filemode.

Example

```
LOAD LOANS
```

reads the file LOANS BASIC into the workspace.

You may override the default filetype and filemode by specifying them in the command (see "Using CMS Disk Files" on page 49).

Example

```
LOAD LOANS VSBASIC B
```

The default file definition can be overridden with the CMS FILEDEF and DLBL commands. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Example

If LOANS has filetype VSBASIC:

```
SYSTEM 'FILEDEF LOANS DISK LOANS VSBASIC B'
LOAD LOANS
```

reads the file VSBASIC LOANS B file into your workspace (see "Using the SYSTEM Command" on page 18).

The filename or ddname specified in the file-spec becomes the name of the workspace. If the file is a CMS disk file, the filename, filetype, and filemode of the file are remembered for use in a subsequent SAVE command. The file specification is forgotten if a LOAD, RUN, or COMPILE command, or a CHAIN statement loads a file, or if a FETCH, INITIALIZE, or RENAME command is used to change the workspace name.

Using the MERGE Command

The MERGE command merges a portion of a BASIC source file with the contents of your workspace. A file-spec is specified in the MERGE command. If a filetype is not specified, a default filetype of BASIC is assumed. The normal CMS search order is used for the filemode unless you indicate otherwise.

Example

```
MERGE SUBRTNS 2000 TO 2500
```

merges the BASIC source lines having line numbers between 2000 and 2500 from the file SUBRTNS BASIC.

You may override the default filetype and filemode by specifying them in the command (see "Using CMS Disk Files" on page 49).

Example

```
MERGE SUBS VSBASIC
```

The default file definition can be overridden with the CMS FILEDEF and DLBL commands. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Using the PROFILE Command

The PROFILE command processes a profile. A file-spec is specified in the PROFILE command. The default filetype is an installation option (the IBM-supplied default is BASPROF). The CMS search order is used if a filemode is not specified. For more information on specifying the file-spec, see "Using Files" on page 45.

Example

```
PROFILE SETUP BASPROF
```

processes SETUP BASPROF profile.

Example

```
PROFILE TEST BASLINES
```

processes TEST BASLINES profile.

Example

```
PROFILE RUN COMMANDS D
```

processes RUN COMMANDS D profile.

You may override the default file definition using the CMS FILEDEF and DLBL commands. For more information on how to do this, see "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.

Using the PURGE Command

The PURGE command deletes *all* files on your A-disk having the specified filename and the filetype BASIC, BASDATA, BASTEXT, LISTING, or TEXT. You may also delete a specific file by giving the filetype and filemode (the default is A) of the file to be deleted. The file-spec will not be treated as a ddname even if a FILEDEF has been specified.

Example

If you have a source file named MYFILE you had previously compiled, your A-disk could contain:

```
MYFILE BASIC
MYFILE TEXT
MYFILE LISTING
```

If so,

```
PURGE MYFILE
```

would delete all three files.

You may also delete a specific file by giving the filetype and filemode (the default is A) of the file to be deleted.

Example

```
PURGE A4000 BASLOG
```

would delete A4000 BASLOG on your A-disk.

Example

```
PURGE ATEST VSBASIC B
```

would delete the file ATEST VSBASIC B.

You may use CMS commands to delete files. For more information on how to do this, see "Using the SYSTEM Command" on page 18.

Example

```
SYSTEM 'ERASE MYFILE TEXT'
```

deletes only MYFILE TEXT from your A-disk.

You cannot use CMS commands to override the file definition for a PURGE command; that is, a FILEDEF or DLBL command specifying MYFILE would not be effective.

OS and VSAM files cannot be purged.

Using the QUERY Command

The QUERY command gives you information about CMS files. (This information is the same as that presented by the CMS LISTFILE command.) The file-spec will not be treated as a ddname even if a FILEDEF has been specified. As in a LISTFILE command, asterisks may occur in the filename and filetype. The filemode may be an asterisk. The filemode defaults to A.

The QUERY command allows you to inquire about different types of files. The keywords used to indicate the various types of files correspond to the CMS filetypes, as shown in Figure 1.

QUERY Keyword	CMS Filetype
FILE[S]	BASDATA
PROG[RAM[S]]	BASIC
WORK[SPACE[S]]	BASTEXT
ALL	any

Figure 1. Correspondence between QUERY Keywords and CMS Filetypes

Example

```
QUERY PROG
```

Lists all the files stored on your A-disk that have the filetype BASIC.

Example

```
QUERY AA
```

displays information about the file(s) whose name is AA. Remember, you can have several different files such as source, workspace, object, and data files that all have the same filename but a different filetype, and QUERY will display them all.

Figure 2 shows the results of the command QUERY AA.

```
* QUERY AA
FILENAME  FILETYPE  FM  F/V  LRECL  RECORDS  BLOCKS     DATE      TIME     VOLSER
AA        BASIC     A1  V       65       13       1  11/19/81  11:03:05  USER91
AA        BASTEXT   A1  V     1024       17       2  11/19/81  11:05:52  USER91
```

Figure 2. QUERY Command — Example of Display

In the display, file AA with filetype BASIC is a source file that was created by the SAVE command, and file AA with filetype BASTEXT is a workspace file that was created by the STORE command.

You may also specify the filetype and filemode as part of the file-spec.

Example

```
QUERY PROG TEST
```

and

```
QUERY TEST BASIC
```

produce the same results.

An asterisk can be used in the file-spec as in a CMS LISTFILE.

Example

```
QUERY TEST BAS*
```

would list all files with filename TEST and with a filetype beginning with BAS that are on your A-disk.

The QUERY command includes an optional OUT clause to direct the information to a file rather than to the terminal. You must include a file-spec in the OUT clause. BASIC uses a default filetype of LISTING. The filemode letter defaults to that of the first read/write disk located using the CMS search order; the filemode number defaults to 1.

Example

```
QUERY OUT(MYFILES)
```

writes information about all your CMS A-disk files to the file MYFILES LISTING A1, if your A-disk is read/write.

You can override the default filetype and filemode by specifying them in the OUT clause (see "Using CMS Disk Files" on page 49).

Example

```
QUERY OUT(MYFILES LIST B2)
```

You can override the default file definition for the OUT clause by using the CMS FILEDEF and DLBL commands. (See "Using Files" on page 45.)

```
SYSTEM 'FILEDEF MYFILES DISK MYFILES MYTYPE T'
QUERY OUT(MYFILES)
```

writes the information about all your CMS A-disk files to the file MYFILES MYTYPE T1.

Note: The QUERY command creates a temporary file named CMS EXEC A1 on your A-disk. If you do not have a read/write A-disk, you will get an error. If a CMS EXEC file already exists on your A-disk, entering a QUERY command will cause it to be overwritten and then erased.

Using the RENAME Command

The RENAME command gives you information about the workspace name and allows the workspace name to be changed.

If you specify a file-spec in the RENAME command or in response to the RENAME command, the filename or ddname in the file-spec becomes the name of the workspace. The file-spec is also remembered for a subsequent SAVE command. The file-spec is forgotten if a LOAD, RUN, or COMPILE command, or a CHAIN statement loads another file, or if a FETCH, INITIALIZE, or RENAME command is used to change the workspace name.

Example

```
RENAME PROG BASIC2
```

sets the workspace name to PROG and remembers PROG BASIC2 for a subsequent SAVE.

Using the RUN Command

The RUN command normally executes the program in the workspace but, if the command includes an optional file-spec, one of the following is done:

- A BASIC source program file, with the default filetype BASIC, is loaded (see "Using the LOAD Command" on page 9) into the workspace and executed.
- A BASIC object program, with the filetype TEXT, is loaded and executed.

- A BASIC object program is loaded from a TXTLIB and executed.

The normal CMS search order is used for filemode.

The file definition associated with the OBJECT option cannot be overridden by a FILEDEF or DLBL command. BASIC will search for a file with the specified filename and TEXT filetype. If the file is not found, BASIC will search TXTLIBs that have been established by a CMS GLOBAL command for the object program. An error will occur if the object program can not be found.

Example

```
RUN CALC
```

causes the source program in file CALC BASIC to be loaded into the workspace and executed.

Example

```
RUN CUSTOMER (OBJECT)
```

causes the object program in file CUSTOMER TEXT to be loaded and executed. If CUSTOMER TEXT is not found, TXTLIBs that have been established with a CMS GLOBAL command are searched for CUSTOMER.

To RUN source files that do not have filetype of BASIC, you must specify the filetype or use CMS commands to override (see "Using the LOAD Command" on page 9.)

When the OBJECT option is specified, it should *not* be in quotes and you should not specify a filetype or filemode; if you do, an error will occur.

Using the SAVE Command

The SAVE command writes a BASIC source program from your workspace to the specified file. A filetype of BASIC is the default.

Example

```
SAVE NEWPROG
```

writes the program in your workspace to NEWPROG BASIC. The file is written to your first read/write disk (in CMS search order) and the filemode number defaults to 1.

You may override the default filetype and filemode by specifying them in the command (see "Using CMS Disk Files" on page 49).

Example

```
SAVE NEWTEST VSBASIC C2
```

The default file definition can be overridden with the CMS FILEDEF and DLBL commands. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Example

```
SYSTEM "FILEDEF NEW DISK NEWPROG BASIC C"
SAVE NEW
```

writes the program to the file NEWPROG BASIC C1 (see "Using the SYSTEM Command" on page 18).

If you enter the SAVE command without a file-spec, BASIC will prompt you with the file-spec remembered for saving the file (see "Using the LOAD Command" on page 9). If there isn't a file-spec remembered, then it will prompt you with the workspace name. If there isn't a workspace name, a file-spec is requested.

Using the SET LOG Command

The SET LOG command causes your terminal dialog with BASIC to be logged to a file. The filename defaults to BASLOG, or a file-spec may be specified by an OUT clause in the SET LOG command. In either case, the default filetype is BASLOG. The file will be written to your first read/write disk (in CMS search order) with a filemode number of 1.

If you specify OUT, an existing file of the same name will be replaced. If you do not specify OUT, an existing file of the same name will be replaced the first time during the session but will be added to on subsequent requests.

If the APPEND option is specified, the designated file or the default file (if there is no OUT clause) is not erased and subsequent logging is added to the file. This is useful if the default BASLOG file or the file named in the OUT clause already exists. If the APPEND option is not specified, the current contents of the data set will be erased when the OUT option is specified or when the SET LOG without an OUT clause is entered the first time.

Example

```
SET LOG ON
```

initiates logging to filename BASLOG with filetype BASLOG. This file remains open until you issue a SET LOG OFF command or leave BASIC (using the QUIT command).

Example

```
SET LOG OUT (SESSION)
```

initiates logging to filename SESSION with filetype BASLOG.

You may override the default filetype and filemode by specifying them in the command (see "Using CMS Disk Files" on page 49).

Example

```
SET LOG OUT (SESSION LOG D2)
```

You can override the default file definition through the CMS FILEDEF and DLBL command. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Example

```
SYSTEM "FILEDEF BASLOG PRINTER"
SET LOG ON
```

causes the terminal dialog to be logged on the printer (see "Using the SYSTEM Command" on page 18).

Using the SET TERM Command

If you are logged on to a 327x-type terminal, you can use the SET TERM command with the LINE option to put BASIC into VM line-by-line mode. (Line-by-line mode is faster than scrolling mode on remote terminals.)

When a BASIC program is executing in line-by-line mode, the PRINT FIELDS and INPUT FIELDS statements can be executed. However, we recommend that you do not intermix those statements with PRINT, INPUT, or LINPUT statements because the screen may be erased.

For details on clearing your screen when it is full, see "VM Screen Messages" on page 4.

Note: In line-by-line mode, Format 2 of the CHANGE command is not available. You may use the SET TERM SCROLL command to switch back to scroll mode to use Format 2 of the CHANGE command on 327x-type display terminals.

Note: If LIST FORWARD and BACKWARD are used in line-by-line mode, a *"MORE..."* message (see "VM Screen Messages" on page 4) may occur during the display of a page.

The SET TERM LINE command may be executed from a BASIC program by using the CALL BASIC statement (see *IBM BASIC Language Reference*).

If you change your virtual console, you may need to enter a SET TERM command (see "Changing the Virtual Console" on page 30 for details).

Note: If you use the LINE option you should make sure that the quote character (") is not the default escape character in CP. If it is, then you will not be able to enter a quote when you use the line option. This also applies to the number sign (#) because it is often used as the default line end symbol in CP. You can change the default escape character and default line end by using the CP TERM command.

Using the STORE Command

The STORE command writes your workspace to the specified file. A filetype of BASTEXT is the default.

Example

```
STORE MYPROG
```

writes your workspace (the source program in the workspace) along with the BASIC internal representation of the program to the file MYPROG BASTEXT. The file is written to your first read/write disk (in CMS search order) and the filemode number defaults to 1.

You may override the default filetype and filemode by specifying them in the command (see "Using CMS Disk Files" on page 49).

Example

```
STORE TRYIT BASTEXT D2
```

The default file definition can be overridden with the CMS FILEDEF and DLBL commands. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Example

```
System "Filedef XYZ disk SFILE BASOCJ C2"
Store XYZ
```

Writes your workspace to the file SFILE BASOCJ C2.

If you enter the STORE command without a file-spec, BASIC will prompt you with the file-spec remembered from a previous FETCH. If there isn't a file-spec remembered, then it will prompt you with the workspace name. If there isn't a workspace name, a file-spec will be requested.

Using the SYSTEM Command

The SYSTEM command allows you to temporarily communicate with CMS using CMS commands and then return to BASIC, while preserving the workspace and the information on your terminal screen.

The CMS commands you can use are those that are "nucleus resident" or that execute in the "transient area" (For definitions of these terms and complete descriptions of commands, see *IBM Virtual Machine/System Product: CMS Command and Macro Reference*.)

These commands are known as the CMS subset. (Figure 3 on page 19 lists the CMS subset commands.) The CMS subset installed for your system may not include all these; check with your system administrator.

You should be careful when using these commands; some of them can adversely affect your BASIC terminal session.

There are two ways to use the SYSTEM command:

- Execute a single CMS command
- Enter CMS subset mode and communicate directly with CMS

Using the SYSTEM Command to Execute a Single CMS Subset Command: You can include a single CMS command as a quoted string in the SYSTEM command. In this case, the CMS command is executed and control is immediately returned to BASIC, which then prompts you for the next BASIC command.

CMS Subset Commands

ACCESS[1]	FILEDEF[1]	MODMAP	START[2]
ASSGN	GENDIRT	OPTION	STATE
COMPARE	GENMOD	PRINT	STATEW
CP[2]	GLOBAL[1]	PUNCH	SVCTRACE
DEBUG[1]	HELP	QUERY	SYNONYM
DISK	HX[2]	READCARD	TAPE
DLBL[1]	INCLUDE[2]	RELEASE[1]	TYPE
ERASE[1]	LISTFILE	RENAME	
EXEC[2]	LOAD[2]	RETURN	
FETCH	LOADMOD[2]	SET	

CP Commands

ADSTOP	DISCONN[1]	LINK	STORE[2]
BEGIN[2]	IPL[2]	LOGOFF[2]	SYSTEM[2]
QUERY	SPOOL	TERM[1]	SET

1 Indicates commands that can alter your BASIC session.

2 Indicates commands that could adversely affect your BASIC session or your CMS session.

Figure 3. CMS Subset Commands and CP Commands

Example

```
SYSTEM "FILEDEF BASHELP PRINTER"
```

causes the CMS FILEDEF to be executed and continues by prompting for the next BASIC command (with the standard BASIC command prompt: an asterisk).

Note that some CMS commands may display information at your terminal (for example, LISTFILE, QUERY). If you are using a 327x terminal, your BASIC screen will disappear and be replaced by the appropriate CMS screen. To restore your BASIC screen, press PA2 or CLEAR, or wait for CMS to do its normal timeout of a display.

You can also execute a CMS EXEC file from BASIC using the SYSTEM command.

Example

If you want to execute the EXEC file TEST EXEC, you can enter the following:

```
SYSTEM 'EXEC TEST'
```

Note: The word EXEC must precede the name of the CMS EXEC file you want to execute.

To issue a CP command from BASIC, use the SYSTEM command and code the characters CP at the beginning of the character string.

Example

You can use the SYSTEM command to spool your printed output to class A:

```
SYSTEM "CP SPOOL PRINT CLASS A"
```

There may be instances when you must respond to a prompt from a CMS subset command, an EXEC procedure, or a CP command that is executed from the SYSTEM command.

Some CMS and CP commands can destroy or alter your BASIC or CMS session. These commands are listed in Figure 3 on page 19. You should be careful when using these commands; they can sometimes affect your BASIC session.

Using the SYSTEM Command to Enter CMS Subset Mode: If you enter only the word SYSTEM (or any abbreviation of at least two characters), you enter the CMS subset mode and are put in direct communication with CMS. You can then enter a sequence of CMS subset commands as though BASIC were not present. When you want to return to BASIC, you merely enter the subset command RETURN.

On a 327x terminal, when you enter CMS subset mode, you are notified by "CMS SUBSET" at the top of the screen. When you enter RETURN, CMS's "MORE..." prompt appears at the lower right of the screen. You can then restore the BASIC screen by pressing PA2 or CLEAR, or by waiting for CMS to do its normal timeout of a display.

Using CALL SYSTEM Statements in BASIC Programs

You can use the CALL SYSTEM statement to execute CMS subset commands from a BASIC program. For a complete list of subset commands, see Figure 3 on page 19.

To execute a CMS subset command from a BASIC program, you need only issue a CALL SYSTEM statement, with the command as the value of a character string parameter.

Example

If you have a data file with a CMS fileid of PAYROLL MASTER B you can use the CALL SYSTEM statement to issue a CMS FILEDEF command for the data file:

```
100 CALL SYSTEM ('FILEDEF PAYMAST DISK PAYROLL MASTER B')
110 OPEN #1: 'PAYMAST', OUTIN, NATIVE
```

After these statements are executed, the PAYMAST file in your program is equated with disk file PAYROLL MASTER B.

Some CMS commands use nonzero return codes for "information content" and not just to indicate command failure. For example, the STATE command is used to verify the existence of a file. If the file exists, CMS sets the return code to 0. If the file does not exist, the return code is 28. When issuing such commands using

the CALL SYSTEM statement, it is often necessary to look at the CMS return code.

The following example illustrates a technique for testing the CMS return code after performing a CALL SYSTEM with the STATE command. A variable named "rc" is set to the return code that is tested by the program after CALL SYSTEM.

```
100 Rem:  Program to demonstrate return code testing from CALL SYSTEM
110 On error goto assign_rc
120 DO
130   Input "Enter name of file you wish checked:  ": fn$
140   Exit if fn$ = ""
150   Call system ("STATE " & fn$)
160   If rc = 0 then
170     Print "*** Hooray! " & fn$ & " exists."
180   Else
190     Print "*** Sorry, but the file " & fn$ & " does not exist."
200     rc=0
210   End if
220   Print
230 Loop
240 Stop
250 assign_rc: If err <> -11002 then &
  & Print "Not a CALL SYSTEM ERROR":stop
260 Print "    CODE = ";code: rc=code: continue
270 End
```

Note in particular lines 110 (ON ERROR), 250 and 260 (error handling with CODE function and CONTINUE), and that line 200 reinitialized rc (the return code variable).

You can use the CALL SYSTEM statement to do everything that the SYSTEM command, as described under "Using the SYSTEM Command to Execute a Single CMS Subset Command" on page 18, can do.

Running Programs in the BASIC Environment

You can execute programs inside the BASIC environment by using the RUN command. In the BASIC environment, you can control the execution of your program using BASIC commands, immediate statements, and debugging statements as described in *IBM BASIC Language Reference* and *IBM BASIC Programming Guide*.

If your program encounters a problem while running (such as an I/O error) or comes to a breakpoint, control is passed back to BASIC. The error is reported by BASIC, and BASIC remains in control. This allows you to investigate the problem using BASIC's debugging features.

Except for these additional facilities, programs run inside the BASIC environment behave the same as programs run outside the BASIC environment. (See "Running Precompiled BASIC Programs under CMS (BASICRUN)" on page 25 and "Creating and Running Self-Contained Application Load Modules" on page 26.)

Using BASIC as a Compiler under CMS

BASIC can be used as a stand-alone compiler. The compiler reads a source program file and writes an object program (text file) and a listing file.

The compiler is normally invoked as follows (however, check with your system administrator to determine if your organization uses this invocation):

```
BASIC file-spec [([option[[,] option]...][)]]
```

where:

file-spec specifies the source program file. A default filetype of BASIC is assumed if the file-spec is not a ddname and does not include a filetype. (See "Using Files" on page 45.)

option is one of the compiler option keywords. These are mostly the same as for the COMPILE command (see *IBM BASIC Language Reference*) and are summarized in Figure 4 on page 23. (If you compile any program without specifying any options, the OPTIONS IN EFFECT section of the compiler output shows you the defaults in force for your organization.)

Option keywords must be separated by at least one space. The separation may include more than one space as well as a comma.

BASIC may be invoked to do a compilation in the following ways:

- From a CMS terminal session
- From an EXEC
- While disconnected
  - Using the console stack
  - From an EXEC that was started before you disconnected
- Using the CMS batch facility

In the absence of any overriding FILEDEF or DLBL commands, the compilation writes the object and listing files to your first read/write disk with a filemode number of 1. The filename of each file is the same as the filename or ddname used in your invocation of the compiler. The filetypes are TEXT for the object program file and LISTING for the listing file.

You may override the default filetype and filemode by specifying them in the file-spec of the source program file (see "Using CMS Disk Files" on page 49).

Option	Meaning
SOURCE | NOSOURCE	List (or do not list) the source program.
OBJECT | NOOBJECT	Generate (or do not generate) an object (text) file.
MAP | NOMAP	Include (or do not include) in the listing an allocation map of variables, arrays, and subprograms.
XREF | NOXREF	Generate (or do not generate) a cross-reference listing.
LIST | NOLIST	Generate (or do not generate) a listing of the generated machine instructions.
FLAG ({I | W | E | S})	Display and list diagnostic messages at or above the following level: I = informational W = warning E = errors S = severe errors
FIPS | NOFIPS	Diagnose (or do not diagnose) deviations from ANSI Minimal Standard BASIC.
SPREC | LPREC	Compile unformatted PRINT statements so they print short precision (or long precision). Compile with the default for real data as short precision (or long precision).
PROFILE [(file-spec)] | NOPROF	Read (or do not read) the designated profile. The default file-spec is an installation option (the IBM-supplied value is BASPROF). The default filetype is an installation option (the IBM-supplied value is BASPROF). The default option is PROFILE if the profile exists and NOPROF if the profile does not exist. For more information about files and file-specs, see "Using Files" on page 45.

Figure 4. BASIC Compiler Options

Example

```
BASIC TEST BAS D
```

compiles the source file TEST BAS D, sends the object to TEST TEXT, and sends the listing to TEST LISTING on the first read/write disk with filemode number 1.

You can override the default file names for the text and listing. A FILEDEF or DLBL command that specifies a ddname of TEXT redirects the text file. A FILEDEF or DLBL command that specifies a ddname of LISTING redirects the listing file. (See "Using the CMS FILEDEF Command" on page 53 and "Using VSAM Files under CMS" on page 65.)

Example

```
FILEDEF TEXT PUNCH
FILEDEF LISTING PRINTER
BASIC CALC (SOURCE OBJECT)
```

compiles the source file CALC BASIC, sends the object program to the punch, and sends the listing to the printer.

Unless they were specified with the PERM option, all FILEDEFs are cleared after the compilation.

When you invoke BASIC for a compilation, BASIC first identifies itself:

```
IBM BASIC/VM VERSION x RELEASE y.z  yyyy/mm/dd  hh:mm
  (c) Copyright IBM Corporation 1982, 1985
```

Then it displays errors in the compilation, either at the default FLAG option level or at the level you have explicitly requested. (See FLAG option in Figure 4 on page 23.) This is followed by a summary of the number of errors encountered at each level in the FLAG option. Finally, if FLAG(I) is in effect, the message

```
COMPILATION COMPLETED.
```

should appear, and control is returned to CMS. Figure 5 shows a sample compilation that contains an error. Figure 6 shows a compilation without errors.

```
R;
basic sample (source object flag(i))
IBM BASIC/VM VERSION x RELEASE y.z  yyyy/mm/dd  hh:mm
  (c) Copyright IBM Corporation 1982, 1985
140 PRNIT A$, B$
        1
###1) BAS30164S PRECEDING IDENTIFIER IS NOT A STATEMENT
      KEYWORD  '='  '('  OR  ',' EXPECTED FOR ASSIGNMENT
      STATEMENT.
SEVERE ERRORS                 1
COMPILATION COMPLETED.
R(00012);
```

Figure 5. Example of Compilation Containing an Error

```
R;
basic sample (source object flag(i)
IBM BASIC/VM VERSION x RELEASE y.z  yyyy/mm/dd  hh:mm
  (c) Copyright IBM Corporation 1982, 1985
NO STATEMENTS FLAGGED IN THIS COMPILATION.
COMPILATION COMPLETED.
R;
```

Figure 6. Example of Compilation without Errors

Running Precompiled BASIC Programs under CMS (BASICRUN)

After you have compiled a BASIC program, you can invoke the text file (load it and start it executing) without entering the BASIC environment or performing any further CMS operations on the text file. All you need is your object program (text) file and the IBM BASIC Library.

The object program is a text file and cannot be executed directly by CMS. It must be loaded and linked to BASIC runtime library routines. This can easily be done by invoking the module BASICRUN and passing it the name of the object program file. BASICRUN does the initial loading and dynamic loading and linking of runtime routines. However, see your system administrator to find out whether or not this invocation is different for your organization.

The named file is required to have filetype TEXT. The normal CMS search order is used for filemode. (You should *not* specify a filetype or a filemode, and the filename should *not* be in quotes.) If the file is not found, BASIC will search TXTLIBs that have been established by a CMS GLOBAL command for object programs.

Example

```
BASICRUN SAMPLE
```

executes the program compiled in Figure 6.

BASICRUN may be invoked in the following ways:

- From a CMS terminal session
- From an EXEC
- While disconnected
  - Using the console stack
  - From an EXEC that was started before you disconnected
- Using the CMS batch facility

Arguments may be passed to the program.

Example

```
BASICRUN SAMPLE QQ LIST NO
```

The tokens (truncated to 8 characters, if necessary) following the program name are concatenated together and separated by spaces. If invoked from a terminal, tokens will be converted to uppercase. This value can be retrieved in the program by using the PARM$ intrinsic function.

Creating and Running Self-Contained Application Load Modules

You can use the CMS commands GENMOD, INCLUDE, and LOAD to build self-contained, nonrelocatable MODULE files of BASIC object programs. You can then execute the module file by merely entering its filename as a command to CMS.

The MAP option of the BASIC compiler gives you a list of the IBM BASIC Library routines to which your program refers directly. Many Library routines call upon other Library routines. Appendix B, "IBM BASIC/VM Library Module Cross-Reference" on page 105, gives the interdependencies among Library routines. Thus, using your compilation listing and the appendix, you can determine all the Library routines needed by your program. (For the minimum list of required library parts, see Appendix A, "Minimum List of IBM BASIC/VM Library Modules" on page 103.) If you compiled your program using Release 1 of BASIC, see also Appendix C, "IBM BASIC/VM Release 1 Library" on page 111.

The Library object file, BLIOCRT, *must* be included in your program module. If you include additional Library files, your program module *must* also contain BLIDIRZ.

If you have specified the AUTO option or defaulted to it, BLIDIRZ also includes BLIDIRA, BLIDIRC, BLIDIRI, BLIDIRL, DLIDIRM, BLIDIRN, BLIDIRO, BLIDIRS, BLIDIR2, and BLIDIRX in your program module. If you specify the NOAUTO option, you *must* explicitly include these additional TEXT files if BLIDIRZ is included.

As long as the IBM BASIC Library is available, other Library routines do not have to be included; they are dynamically loaded if necessary.

If the Library is available in the BLISEG discontiguous shared segment and BLIDIRZ is not included in the load module, the library routines in the shared segment BLISEG will be used when the program is executed.

The procedure for generating BASIC load modules is:

1. LOAD BLIOCRT.
2. INCLUDE your program and any other pertinent Library TEXT files.
3. Use GENMOD to build the module.

LOAD, INCLUDE, and GENMOD are CMS commands. These commands should be executed outside the BASIC environment.

Note: BLIOCRT should be the entry point for the module. Unless some other language (FORTRAN, COBOL, PL/I, or Assembler) program is linked in, BLIOCRT will automatically be the entry point. In other cases it may be necessary to use the RESET option specifying BLIOCRT on the last LOAD or INCLUDE preceding the GENMOD in order to guarantee that BLIOCRT is the entry point. Also, if BLIOCRT does not occur first in the module, the FROM option specifying the first entry may be needed for the GENMOD.

Note: Do not use any Release 1 Library routines when using Release 2 Library routines. BASIC program units compiled under Release 2 require the Release 2 Library. BASIC program units compiled under Release 1 may be linked with either the Release 1 Library or the Release 2 Library. If you link a program compiled under Release 1 with the Release 2 Library, see Appendix C, "IBM BASIC/VM Release 1 Library" on page 111.

Example

```
LOAD BLIOCRT
INCLUDE SAMPLE
GENMOD SAMPLE
```

generates a MODULE file (MODULE is the filetype) with the filename SAMPLE on your A-disk (the filename is given in the GENMOD command). You can now enter:

```
SAMPLE
```

and the program is executed.

The load module may be invoked in the following ways:

- From a CMS terminal session
- From an EXEC
- While disconnected
  - Using the console stack
  - From an EXEC that was started before you disconnected
- Using the CMS batch facility

Arguments may be passed to the program.

Example

```
SAMPLE QQ LIST NO
```

The tokens (truncated to 8 characters, if necessary) following the module name are concatenated together and separated by spaces. This value can be retrieved in the program by using the PARM$ intrinsic function. If invoked from a terminal, tokens will be converted to uppercase.

For information on how to start a compiled BASIC program which has been placed in a discontiguous shared segment, see Appendix D, "Shared Segment Starter" on page 113.

BASIC Return Codes under CMS

Whenever BASIC relinquishes control to CMS, a return code is issued. This can occur in several situations:

- BASIC terminates, because of an error or a QUIT command.
- The BASIC compiler terminates because of an error or because it has completed a compilation.
- A BASIC program, invoked from CMS (interactive, disconnected, or batch) or from BASICRUN, terminates.

BASIC has a set of predefined return codes. These are shown in Figure 7 on page 29.

You may specify the return codes to be returned from BASIC programs by coding numeric expressions on your STOP and END statements. When you execute the program and it terminates because of a STOP or an END statement, the value of the numeric expression is the return code.

For CMS return codes, see *VM/SP System Messages and Codes*.

The Virtual Console

All terminal input and output in BASIC are done using the CMS virtual console. In addition, BASIC determines the characteristics of the real terminal, if any (see "Running in Disconnected Mode" on page 30), and uses those characteristics to determine how the terminal is to be handled.

If you are logged on to a 327x-type display terminal, BASIC takes control of the display and operates in full-screen scrolling mode.

You can be in line-by-line mode as a result of the way the product was installed, or because a SET TERM LINE command was issued either directly (see "Using the SET TERM Command" on page 17) or indirectly using the PROFILE command (see "Using the PROFILE Command" on page 11).

If you are logged on using a hardcopy terminal or remote Teletype[1] type terminal (either hardcopy or display), or in SET TERM line mode, BASIC operates in line-by-line mode. In most cases, it is permissible to disconnect from one type of terminal and reconnect to another type while in BASIC (see "Changing the Virtual Console" on page 30).

---

1 Registered trademark of the Teletype Corp.

Value	Meaning
0	No errors
4	Warning-level errors detected in BASIC program compilation
8	Errors detected in BASIC program compilation
12	Severe errors detected in BASIC program compilation
16	Unrecoverable compilation or command error
20	Insufficient memory
24	Unable to load BASIC Processor
28	No user program specified (BASICRUN only)
32	Unrecoverable error (Library only)
36	Incorrect call to Library routine
40	Unable to load user program
44	Master Library directory is invalid
48	Error freeing memory at termination
52	Unable to load BASIC Library

Figure 7. BASIC Return Codes

Using the Console Stack

In either full-screen or line-by-line mode, the CMS console stack can be used to provide terminal input to BASIC.

In all cases, when the IBM BASIC Processor requests input with the "*" prompt, or when an INPUT statement is executed, the contents of the console stack, if any, are read.

If no lines are stacked, prompting to the terminal occurs normally. Repeated input prompts continue to read lines from the stack until it is empty.

For further information about using the console stack, see *Virtual Machine / System Product: CMS User's Guide*.

Running in Disconnected Mode

All the facilities of BASIC are available to the CMS user running in disconnected mode. In this mode, BASIC detects the absence of a real terminal and processes console I/O in line-by-line mode.

If any I/O is done by BASIC before being disconnected, you should SET TERM LINE before doing any further I/O while disconnected. See "Changing the Virtual Console" for details.

Console output is recorded on the CMS console spool (if started with the CP SPOOL command), and, optionally, at the BASIC terminal log (using the SET LOG command).

Terminal input can be read from the console stack. In order to prevent your userid from being forced off the system when running in disconnected mode, take care to ensure that stacked input is available if requested by BASIC.

When running in disconnected mode, you should not use the INPUT FIELDS and PRINT FIELDS statements.

Changing the Virtual Console

If you change the virtual console while in BASIC, BASIC may not be aware of the change and may continue to use the characteristics of the previous virtual console.

If you encounter problems after changing the console, press the ENTER key and then enter a SET TERM {SCROLL | LINE} command to establish the correct terminal characteristics. However, this does not work if you have a program running.

If you become disconnected from a 327x-type terminal while in SCROLL mode, BASIC will wait until you can log back on. When you log back on, you should press the ENTER key to cause BASIC to resume. BASIC will determine what kind of terminal you are now using.

The SET TERM command and the PAUSE statement force BASIC to determine the current virtual console characteristics. If you are running in LINE mode or on a LINE mode terminal and you change to a 327x-type terminal, or change from a smaller screen to a larger screen, BASIC may not be aware of the change unless you enter a SET TERM command.

If you want to run a program while disconnected, you should make sure you are running in LINE mode prior to any I/O to the terminal (otherwise, BASIC will wait until you log back on). You may enter a SET TERM LINE command prior to becoming disconnected to accomplish this.

Note: The SET TERM command can be executed from a program by using the CALL BASIC statement.

You should be aware of the following changes from Release 1:

- In Release 1, if a user becomes disconnected while in SCROLL mode, terminal output goes to the console spool; whereas, in Release 2, BASIC will wait until the user logs back on and presses the ENTER key.
- In Release 1, if a user becomes disconnected while in SCROLL mode, terminal input will be read from the console stack; whereas, in Release 2, BASIC will wait until the user logs back on and presses the ENTER key.
- In Release 1, if the terminal is changed between two terminal I/Os to a larger screen while in SCROLL mode, BASIC will clear the screen and then do the I/O; whereas, on Release 2, the I/O may be done as if the screen has not changed (if this occurs the user must enter a SET TERM command to have data displayed and input formatted properly on the screen).
- In Release 1, if a user is disconnected from a line-mode terminal and then reconnects to a 327x-type terminal, BASIC will change to full screen mode; whereas, in Release 2, BASIC will stay in line mode until the user enters a SET TERM command.

Running under the CMS Batch Facility

You can run BASIC under the control of the CMS batch facility. This is similar to running in disconnected mode.

Console input and output are handled as they are when running in disconnected mode.

The only restriction imposed by the CMS Batch Facility is that you cannot issue a FILEDEF command for the virtual card reader.

For further information on using the CMS batch facility, see *Virtual Machine / System Product CMS User's Guide*.

Using I/O Statements in BASIC Programs

Your BASIC programs communicate with other systems by one of two methods:

1. Terminal I/O.

   Several BASIC statements are used to read from and write to the terminal:

   INPUT
   INPUT FIELDS
   LINE INPUT
   PRINT
   PRINT FIELDS

2. File I/O.

   BASIC contains many statements that access files:

CLOSE	PUT
*DEBUG ON	READ File
DELETE	REREAD
GET	RESET
*INPUT File	REWRITE
*LINE INPUT File	SCRATCH
OPEN	*TRACE ON
*PRINT File	WRITE

   The asterisk (*) indicates statements which, when they refer to a file reference number of 0, access the terminal rather than a file.

Using Terminal I/O Statements

All BASIC terminal I/O is performed through the virtual console, as described under "The Virtual Console" on page 28. If your virtual console is not a display-type terminal, you should not use the INPUT FIELDS and PRINT FIELDS statements.

Using File I/O Statements

BASIC programs access files by means of the OPEN statement. The OPEN statement for any file establishes a connection between the file-spec and the file reference number. The file-spec identifies the file to CMS. The file reference number is a number used to identify the file for other file I/O statements within the BASIC program.

Before running a program, you can (in some cases, must) use CMS commands to connect the names used in the program to actual files. This is discussed in "Using CMS Disk Files" on page 49.

If there are no such overriding CMS commands when your program opens a file, BASIC attempts to attach to a file having the same filename as that used in your OPEN statement and a filetype of BASDATA. The disk search order and rules for creation of new files are given beginning with "Using CMS Disk Files" on page 49.

There are several things to remember:

- Files accessed with keyed organization (the keyword KEYED in the OPEN statement) must be VSAM files.
- VSAM files must be defined outside BASIC and must be identified to BASIC with CMS DLBL commands.
- Files opened as DEVICE PRINTER have a printer carriage control character prefixed to each record.

- Files opened as DEVICE 3800 have a carriage control character followed by a font control (table reference) character prefixed to each record.
- The maximum record length is 32756.

Calling Subprograms from BASIC

A BASIC program unit can:

- Call BASIC subprograms
- Call programs written in other languages
- Access a SQL/DS relational data base

The called subprogram or programs written in other languages may be prelinked with a calling program unit or can be dynamically loaded as needed.

These subjects are discussed in the following sections.

Calling BASIC Subprograms

When you are running inside the BASIC environment and the CALL statement is a source statement in the workspace, BASIC first checks to see if the subprogram is also present in source form in the workspace. If it is, the workspace subprogram is used.

Otherwise, BASIC attempts to find and load a file with the same filename as the subprogram and a filetype of TEXT. This file definition cannot be overridden by a FILEDEF command. If the file is not found, BASIC will search TXTLIBs established by a CMS GLOBAL command for the object program.

If the calling main program or subprogram is itself compiled, the subprogram called must also be compiled. TEXT files are dynamically loaded when they are called during execution. Note that the subprogram in a TEXT file is used, regardless of whether the subprogram exists as source in the workspace.

After a subprogram is loaded, BASIC remembers it; thus, subsequent calls do not cause reloading during execution of the program.

All dynamically loaded subprograms are released when the program terminates.

Note: The following calls are valid:

- A call from a main program or subprogram in the workspace to another subprogram in the workspace
- A call from a main program or subprogram in the workspace to a compiled TEXT file
- A call from a compiled TEXT file to another compiled TEXT file

You *cannot* call from a TEXT file to a program in the workspace.

Usage of Compiled BASIC Subprograms

When compiled BASIC subprograms are dynamically loaded, certain considerations apply. To avoid unnecessary complications, use one of the following two procedures:

1. Compile all subprograms together with the main program. This will produce a single text file that will be loaded when the program is run.

2. Alternatively, compile each subprogram separately. The name of the source file should be the same as the subprogram name; this will produce a text file for each subprogram with the appropriate name. Each subprogram can then be dynamically loaded only when needed.

If you do not apply either of the above procedures, the applicable considerations are as follows:

1. If more than one subprogram is contained in a dynamically loaded text file (as a result of compiling them together), the name of the first subprogram must be the same as the name of the text file. Subprograms other than the first can only be called by others in the same text file (only the first name will be known outside at run time).

2. CALLs to subprograms that were compiled together with the calling program unit will be resolved when the text file is dynamically loaded. CALLs to a subprogram that is not in the same text file will cause a dynamic load of the called name when that call is executed.

Calling Programs Written in Assembler

The CALL statement to an Assembler language program from a BASIC program follows the same rules as the CALL from a program unit to another BASIC subprogram.

For details on the format of the CALL statement, see *IBM BASIC Language Reference*. For details on writing Assembler language programs that can be called from BASIC, see *IBM BASIC Programming Guide*.

You must make the called Assembler language program available at run time as a preassembled TEXT file or by including it in a BASIC program load module. When a CALL to an Assembler language program executes, control passes to the named program. When the Assembler language program completes execution, control returns to the calling program which continues at the next executable statement.

Calling Programs Written in Other Languages

The interlanguage communication (ILC) facility provides the means to call programs compiled by:

- The OS/VS COBOL Compiler.
- The VS FORTRAN Compiler (Release 3.0 or later is required for passing character data).
- The OS PL/I Optimizing Compiler.

You can also execute Graphical Data Display Manager (GDDM) functions from your BASIC program through this facility.

If the called programs load dynamically, you must have read/write access to a disk in order to create temporary files. Temporary files have a filename of BLIT*nnnn* (where *nnnn* is a 4-digit number), have a filetype of TEXT, and are erased after loading. An existing file with such a name will not be overwritten.

For more information, see *IBM BASIC Programming Guide* and *IBM BASIC Language Reference*.

Calling COBOL Subprograms

Before dynamically calling a COBOL subprogram (or called program), BASIC must be told which COBOL subprograms are to be called. You can do this through the following statement:

```
CALL CLINK [(character-expression [,character-expression]...)]
```

where the value of each character expression names a COBOL subprogram you will be calling with subsequent CALL COBOL statements. In your program, the CALL CLINK statement must be executed before the first CALL COBOL statement.

A CALL CLINK statement with no arguments clears any previous CLINK statement definitions. (The dynamically loaded COBOL subprograms are removed and the storage can be reused.) A CALL CLINK statement with arguments clears any previous CLINK statements and allows you to call the new set of COBOL subprograms named by the arguments.

After your BASIC program has identified the COBOL subprograms to be called, the program can issue the following statement:

```
CALL COBOL (character-expression [,argument]...)
```

This statement calls a precompiled COBOL subprogram named by the value of the character expression.

For information on passing arguments to subprograms, see *IBM BASIC Language Reference*.

You must make the COBOL subprograms available as precompiled TEXT files. The COBOL subprograms may be in a TXTLIB if the appropriate GLOBAL has been established. The COBOL library must also be accessible (by means of the

appropriate GLOBAL command). Check with your system administrator for the TXTLIB and/or LOADLIB libraries needed.

Note: If the COBOL subprograms are prelinked with your BASIC program, the CALL CLINK statement is not needed and is ignored. For a discussion of prelinking BASIC programs with COBOL subprograms, see "Prelinking BASIC Programs with Other Language Programs" on page 39.

Example

In the following example, you are calling the COBOL subprogram INVENT and passing it an item number and price.

```
100 CALL CLINK ('INVENT')  ! Identify INVENT
110 CALL COBOL ('INVENT',ITEM_NUMBER%, PRICE) ! Call INVENT
```

Calling FORTRAN Subroutines

Before dynamically calling a FORTRAN subroutine, BASIC must be told which FORTRAN subroutines are to be called. You can do this through the following statement:

```
CALL FLINK [(character-expression [,character-expression]...)]
```

where the value of each character expression names a FORTRAN subroutine your BASIC program will be calling. In your BASIC program, the CALL FLINK statement must be executed before the first CALL FORTRAN statement.

A CALL FLINK statement with no arguments clears any previous FLINK statement definitions. (The dynamically loaded FORTRAN subroutines are removed and the storage can be reused.) A CALL FLINK statement with arguments clears any previous FLINK statements and allows you to call the new set of FORTRAN subroutines named by the arguments.

After your BASIC program has identified the FORTRAN subroutines to be called, the BASIC program can issue the following statement:

```
CALL FORTRAN (character-expression [,argument]...)
```

This calls a precompiled FORTRAN subroutine named by the value of the character expression.

For information on passing arguments to subroutines, see *IBM BASIC Language Reference*.

Note: VS FORTRAN Release 3 or later is required for passing character data.

You must make the FORTRAN subroutines available as precompiled TEXT files. The FORTRAN subroutines may be in a TXTLIB if the appropriate GLOBAL has been established. The FORTRAN library must also be accessible (by means of the appropriate GLOBAL command). Check with your system administrator for the TXTLIB and/or LOADLIB libraries needed.

Note: If the FORTRAN subroutines are prelinked with your BASIC program, the CALL FLINK statement is not needed and is ignored. For a discussion of how to

prelink BASIC programs with FORTRAN subroutines, see "Prelinking BASIC Programs with Other Language Programs" on page 39.

Example

In the following example, you are calling the FORTRAN subroutine FACTOR, which computes the factorial of 9.0 and returns the result in FACTORIAL%:

```
100 CALL FLINK ('FACTOR')  ! Identify FACTOR
110 CALL FORTRAN ('FACTOR',9.0,FACTORIAL%) ! Call FACTOR
120 PRINT "The factorial of 9.0 is: ",FACTORIAL% ! Print result
```

Calling PL/I Procedures

Before dynamically calling a PL/I procedure, BASIC must be told which PL/I procedures are to be called. You can do this through the following statement:

```
CALL PLINK [(character-expression [,character-expression]...)]
```

where the value of each character expression names a PL/I procedure your BASIC program will be calling with subsequent CALL PLI statements. Your program must execute the CALL PLINK statement before the first CALL PLI statement.

A CALL PLINK statement with no arguments clears any previous PLINK statement definitions. (The dynamically loaded PL/I procedures are removed and the storage can be reused.) A CALL PLINK statement with arguments clears any previous PLINK statements and allows you to call the new set of PL/I procedures named by the arguments.

After your BASIC program has identified the PL/I procedures to be called, the BASIC program can issue the following statement:

```
CALL PLI (character-expression [,argument]...)
```

which calls a precompiled PL/I procedure named by the value of the character expression.

For information on passing arguments to procedures, see *IBM BASIC Language Reference*.

You must make the PL/I procedures available as precompiled TEXT files. The PL/I procedures may be in a TXTLIB if the appropriate GLOBAL has been established. The PL/I libraries must also be accessible (by means of the appropriate GLOBAL command). Check with your system administrator for the TXTLIB libraries needed.

Note: If the PL/I procedures are prelinked with your BASIC program, the CALL PLINK statement is not needed and is ignored. For a discussion of how to prelink BASIC programs with PL/I procedures, see "Prelinking BASIC Programs with Other Language Programs" on page 39.

Example

In the following example, you are calling the PL/I procedure VENDOR and passing a vendor name and number to process.

```
100 CALL PLINK ('VENDOR')  ! Identify VENDOR
110 CALL PLI ('VENDOR',VENDOR_NAME$,VENDOR_NUMBER) ! Call VENDOR
```

If the called PL/I procedure is not found, a message indicates the name not found and also the name PLICALLA. This does *not* mean that a procedure named PLICALLA is required; the name is merely an entry point that BASIC uses. When you rerun the program and supply the named missing procedure, this reference will be resolved.

Calling GDDM (Graphical Data Display Manager)

Your programs can call Graphical Data Display Manager (GDDM) (Release 2 or later) functions through the following BASIC statement:

```
CALL GDDM(func [,argument]...)
```

where **func** is a character expression or a numeric expression. The statement calls GDDM functions and PGF functions, if the PGF feature is available to your organization.

For more information on GDDM, see *IBM BASIC Language Reference*, *GDDM Base: Programming Reference*, and *GDDM Presentation Graphics Feature: Programming Reference*.

You must make GDDM available by using the appropriate CMS GLOBAL command. Check with your system administrator for the TXTLIB libraries needed.

Example

```
GLOBAL TXTLIB ADMGLIB
```

Your programs cannot use the following GDDM functions:

```
FSINIT, SPINIT, FSEXIT, FSRNIT, FSTERM
```

An exception (-4103) occurs if the value of the numeric expression is the code or the character expression is the name for one of these functions.

On the first call to GDDM, the GDDM interface calls SPINIT to initialize the system programmer interface to GDDM. The SPIBFLAG settings SPIBPVCF, SPIBSTGF, SPIBARBF, and SPIBCIRF are set to 1 with all others set to 0. When the BASIC program terminates, FSTERM is called automatically.

Example

In the following example, you are calling the GDDM function page create (FSPCRT). The arguments being passed are page-id, depth, width, and type.

```
100 CALL SYSTEM ("GLOBAL TXTLIB ADMGLIB")
110 CALL GDDM ("FSPCRT",1,24,80,0) ! Invoke GDDM
```

Accessing a SQL/DS Relational Data Base

Your programs can access and operate on existing SQL/DS (Structured Query Language/Data System) data base by means of the special BASIC statement whose general format is:

```
CALL SQL (SQL-stmt, status-info(), message-var [,value-list])
```

Example:

```
CALL SQL ("SELECT NAME,YEARS FROM STAFF",SQLRC(),MSG$,NAM$,YRS%)
```

which will retrieve the two specified columns (NAME and YEARS) from a row of the table named STAFF, and place their values in your NAM$ and YRS% program variables. Exception conditions (if there are any) will be indicated in SQLRC() and the error messages will be indicated in MSG$.

See *IBM BASIC Language Reference* for information about the kinds of parameter requirements, special restrictions, and SQL statements supported by BASIC. For an overview of the SQL language itself, see *SQL/DS Concepts and Facilities* and *SQL/DS Application Programming*. Note, however, that ISQL and the various data base utilities referred to in these manuals are not accessible using the CALL SQL statement.

Check with you system administrator before you attempt to use the CALL SQL statement. Make sure that:

- Your organization has installed SQL/DS.
- The file BLIOC06 ASMSQL (provided by BASIC) has been preprocessed for the SQL/DS data base you will be accessing.
- You have CONNECT authority on that data base and have been granted appropriate privileges on the tables you will access.

Prelinking BASIC Programs with Other Language Programs

The following sections show you how to prelink BASIC programs with programs written in other languages. You will need to do this only if you want to create a load module that contains all the BASIC programs and the other language programs. For a discussion of how to create a load module for BASIC programs, see "Creating and Running Self-Contained Application Load Modules" on page 26.

Declaring Other Language Subprograms

When your program dynamically calls other language subprograms, the BASIC interlanguage communication (ILC) interface creates a table consisting of the names of the subprograms to be called and their entry point addresses. However, when you create a load module and include the other language called routines, you must create this table yourself and also include it in the load module.

Calls to Assembler, GDDM, and SQL language subprograms do *not* require this table.

You only need one ILC table for each language in which programs are written that you want to prelink with your BASIC program.

To create and link the ILC interface tables, do the following:

1. Use XEDIT or the editor of your choice with filetype ASSEMBLE and filemode A. The filename should not start with BLI.

   The file must be record length 80 and record format fixed. The content of the file must follow this format:

```
BLISxxx  CSECT
         EXTRN  ilc
         DC     F'n'
         DC     CL8'subprog1',V(subprog1)
         DC     CL8'subprog2',V(subprog2)
         .
         .
         .
         DC     CL8'subprogn',V(subprogn)
         END
```

   where:

   xxx is specified as follows:

   COT for COBOL subprograms
   FOT for FORTRAN subroutines
   PLT for PL/I procedures

   ilc is specified as follows:

   ILBOSTB0 for COBOL subprograms
   VSCOM# for FORTRAN subroutines
   PLICALLA, PLIMAIN for PL/I procedures

   n indicates the number of subprograms included in this table

   subprog1 through **subprogn**

   are the names of each subprogram. Each name can be from one to eight characters long, but must be padded with blanks on the right so that the length is equal to eight bytes.

   When you have created the file, save it and proceed with step 2.

2. Assemble the file you have just created using the CMS assembler as shown:

```
ASSEMBLE filename
```

   where:

   filename is the same name used in step 1.

   This creates a relocatable TEXT file named filename.

3. Build the load module for the BASIC program. (See "Creating and Running Self-Contained Application Load Modules" on page 26.) Include the TEXT file generated in step 2 and the IBM BASIC Library routine BLISILC. Remember to include BLIDIRZ, because an IBM BASIC Library routine is included. Include library routines needed by the other language subprograms, if not automatically linked in. BLIOCRT should be the entry point for the module. To do this, use the RESET option specifying BLIOCRT on the last LOAD or INCLUDE command preceding the GENMOD.

   Note: Because BLIDIRZ is included, the shared segment BLISEG will not be used for IBM BASIC Library routines even if it is available.

Example

The following illustrates how to create a load module, FACTORL, using a BASIC program, BASFAC, calling a FORTRAN subroutine, FACTOR. The FORTRAN subroutine has been precompiled, and its TEXT file is made available to you. The BASIC program is as follows:

```
100 REM - BASFAC:  This program calls the FORTRAN
110 REM -          subroutine FACTOR passing it a
120 REM -          decimal value.  The subroutine
130 REM -          computes the factorial of the decimal
140 REM -          value and returns the result in the
150 REM -          integer variable FACTORIAL.
160 INTEGER FACTORIAL
170 CALL FORTRAN ("FACTOR",9.0,FACTORIAL)
180 PRINT "THE FACTORIAL OF 9.0 IS: ",FACTORIAL
190 END
```

The steps to take in creating the load module are:

1. Compile the BASIC program:

   ```
   BASIC BASFAC (MAP)
   ```

   This step creates files BASFAC TEXT and BASFAC LISTING.

2. Examine the LISTING file to get the names of IBM BASIC Library routines needed for your module. The reference you are looking for will appear as follows:

   ```
   ***** LIBRARY ROUTINES REQUIRED *****

   BLIIEND  BLIIIOL  BLIIPR   BLISILC  BLISINT  BLISTRM
   ```

3. Create the ILC table FACTLNK ASSEMBLE A:

   ```
   BLISFOT  CSECT
            EXTRN VSCOM#
            DC    F'1'                    ONLY 1 ROUTINE.
            DC    CL8'FACTOR',V(FACTOR)  "FACTOR"
            END
   ```

4. Assemble the ILC table:

   ```
   ASSEMBLE FACTLNK
   ```

   This step creates the files FACTLNK TEXT A and FACTLNK LISTING A.

5. Ensure that you have access to the FORTRAN library routines by using the CMS GLOBAL command:

```
GLOBAL TXTLIB VFORTLIB
```

Note: Check with your system administrator to obtain the correct name for the VS FORTRAN TXTLIB or any TXTLIB you may require.

6. Load the TEXT files. By using the CMS LOAD and INCLUDE commands, you load the TEXT files into your CMS workspace:

```
LOAD BLIOCRT
INCLUDE BLIOCOO
INCLUDE BLIDIRZ
INCLUDE BLIIEND
INCLUDE BLIIIOL
INCLUDE BLIIPR
INCLUDE BLISILC
INCLUDE BLISINT
INCLUDE BLISTRM
(plus other Library routines required by above routines)
INCLUDE BASFAC
INCLUDE FACTLNK (RESET BLIOCRT)
```

7. Create the module FACTORL, using the CMS GENMOD command:

```
GENMOD FACTORL
```

At this point, you can directly execute your module by merely entering:

```
FACTORL
```

Prelinking GDDM Support Routines with BASIC Programs

Calls to GDDM functions do not use an ILC table to prelink with BASIC. Instead, you can use the following procedure to link GDDM with a BASIC program:

1. Compile your BASIC program with the MAP option.

2. Examine the listing of your compiled BASIC program to obtain the names of the necessary IBM BASIC Library routines. You must include the routines BLISGDD and BLIDIRZ. Note that because BLIDIRZ is included, the shared segment BLISEG will not be used for IBM BASIC Library routines even if it is available.

3. Ensure you have access to the GDDM library routines when you load the program. (Check with your system administrator about any TXTLIB libraries you must make available.) For example:

```
GLOBAL TXTLIB ADMRLIB
```

4. Use the CMS LOAD and INCLUDE commands to load the necessary TEXT file into your CMS workspace. After you have included all the necessary BASIC TEXT files and the TEXT file for your program, you must include the GDDM library routines by issuing an INCLUDE command:

```
INCLUDE ADMASP
```

5. Create your module using the CMS GENMOD command.

6. Before invoking the program, ensure that you have access to the GDDM library routines. (Check with your system administrator about any TXTLIB libraries you must make available.) For example:

```
GLOBAL TXTLIB ADMGLIB
```

Note: For specific details of each step, refer to the example in "Creating and Linking the Interface Tables" on page 40.

You are now ready to execute your module merely by entering its name on the terminal.

Chaining to BASIC Programs

Main programs that are the targets of CHAIN statements are dynamically loaded. They can be compiled TEXT files or BASIC source program files.

When a CHAIN statement is encountered within the BASIC environment, BASIC first attempts to find a TEXT file of the indicated name. If a TEXT file is found, it is loaded and used. If no TEXT file is available, BASIC then attempts to find a source file, with the filetype of BASIC, and reload the workspace.

For a source file, the default filetype and filemode may be overridden by specifying them in the CHAIN statement (see "Using CMS Disk Files" on page 49). You may *not* specify a filetype or a filemode for TEXT files.

Example

```
10 CHAIN "T3 BASIC D"
```

You can override the default file definition for a source file using the FILEDEF command. To do this, see "Using Files" on page 45.

In programs running outside the BASIC environment, CHAIN statements can refer only to compiled TEXT files. The program that caused the CHAIN to occur must have been dynamically loaded.

Note: For a CHAIN statement, BASIC will *not* search TXTLIBs for the object program.

The following assembler language program can be used to cause the first program to be dynamically loaded by a linked module:

```
          TITLE   'BASIC CHAIN STARTER - BEGCHAIN'
* This program can be linked with BLIOCRT to start a chain of BASIC
* programs.  It must get control before BLIOCRT.  It starts the
* chain by altering the parameter list to include the name of the
* first program in the chain.  Control is then transferred to
* BLIOCRT to begin execution of the program.
*
* To build the chain starter:
*     Edit this program to invoke the first program in the chain
*         (coded below as 'CHAIN1')
*     ASSEMBLE BEGCHAIN
*     LOAD BEGCHAIN BLIOCRT (CLEAR RESET BEGCHAIN
*     (Optionally include BLIDIRZ and other library routines desired)
*     (Do not include any compiled BASIC program units.  They must be
*     dynamically loaded.)
*     GENMOD CHAINRUN
* To execute the programs, enter
*     CHAINRUN parms
*
* Where parms are values to be passed to the program (program may
* retrieve values by using the intrinsic function PARM$.
* Note: Use desired module name instead of 'CHAINRUN'
*
BEGCHAIN  CSECT
          EXTRN    BLIOCRT
          BALR     15,0
          USING    *,15
          ST       2,SAVER2       SAVE REGISTER 2
*
* COPY INCOMING PARAMETERS TO PARMBUFF
          LA       2,PARMBUFF+16
LOOP      MVC      0(8,2),8(1)    MOVE A PARAMETER
          LA       1,8(,1)        BUMP POINTERS TO NEXT PARAMETER
          LA       2,8(,2)
          CLI      0(1),X'FF'     CHECK FOR END OF PARM LIST
          BNE      LOOP
*
* TRANSFER TO BLIOCRT WITH MODIFIED PARAMETER LIST
          L        2,SAVER2       RESTORE REGISTER 2
          LA       1,PARMBUFF     USE NEW PARM LIST
          L        15,ABLIOCRT    TRANSFER CONTROL TO BLIOCRT
          BR       15
*
ABLIOCRT  DC       A(BLIOCRT)
SAVER2    DC       F'0'
PARMBUFF  DS       0F
          DC       CL8'BASICRUN'
          DC       CL8'CHAIN1'      PUT NAME OF FIRST PROGRAM HERE
          DC       200X'FFFFFFFF'
PARMEND   DS       0F
          END      BEGCHAIN
```

Generating an Attention Interrupt

Attention interrupts are usually recognized by BASIC except while running a program that has ON ATTN IGNORE specified. For information on when BASIC commands and programs will recognize an attention interrupt, see *IBM BASIC Programming Guide.*

You can generate attention interrupts from the keyboard in several different ways, depending upon the type of terminal you are using. Most commonly, on a 327x

display terminal, or equivalent, you can generate an attention interrupt by doing the following:

1. Press the RESET key.
2. Press ENTER.

If your program is in an input loop, where you are endlessly prompted for input, you interrupt it by doing the following:

1. Press the PA1 key. (This clears the screen, and the system should respond with CP READ.)
2. Press the PA1 key again. (The system should restore the BASIC screen and bring you back to where you were in the program.)
3. Enter a slash (/) to satisfy the system's input request.
4. Press the ENTER key. (The system should respond with ATTENTION INTERRUPT AT LINE x.)

For more detailed information and for information regarding other types of terminals, see *IBM Virtual Machine/System Product: Terminal User's Guide.*

Note: It is not possible to signal attention to BASIC when running in disconnected mode, while under the control of the CMS Batch Facility, or while executing BASIC as a compiler.

Halting Execution with the HX Command

In CMS command mode, you can use the HX immediate command to halt the execution of a command, of an EXEC procedure, or of a program.

You can use the HX command to halt a compilation when BASIC is invoked as a compiler.

When your terminal session is under the control of BASIC in the BASIC environment, the HX command is not a valid command.

Note: The HX command should normally *not* be used while in the CMS subset mode, nor should it be used to halt the execution of a CMS command initiated by the SYSTEM command or the CALL SYSTEM statement. Doing so terminates the execution of BASIC.

Using Files

BASIC under CMS uses two file systems: the CMS file system and VSAM (Virtual Storage Access Method).

The CMS file system is used for files on CMS minidisks; it also allows you to read and write tapes, to write to the virtual printer and virtual punch, to read from the virtual card reader, and to read and write the virtual console (terminal).

BASIC programs can access CMS files on disk as sequential, stream, or relative files, but not as keyed files.

VSAM files reside on disk. VSAM files can be accessed as sequential, stream, relative, or keyed files.

File Specification in BASIC Statements and Commands

The file-spec specified in an OPEN statement, CHAIN statement, BASIC command, or BASIC invocation can have the form:

1. `filename[/password]`
2. `filename filetype [filemode]`
3. `"filename [filetype [filemode]]"[/password]`
4. `ddname[/password]`

Where:

filename is one to eight characters and may include letters A through Z, digits 0 through 9, and the national characters @, #, and $. A QUERY command may include asterisks (*). (For more information on the use of asterisks in the file-spec for a QUERY command, see "Using the QUERY Command" on page 12.) A filename specifies the name of a CMS disk file or, if loading object, the name of a member of a TXTLIB.

filetype is one to eight characters and may include letters A through Z, digits 0 through 9, and the national characters @, #, and $. A QUERY command may include asterisks (*). (For more information on the use of asterisks in the file-spec for a QUERY command, see "Using the QUERY Command" on page 12.) A filetype specifies the filetype of a CMS disk file. If the filetype is omitted when referencing a CMS disk file, a default filetype is assumed as shown in Figure 8 on page 47.

filemode is made up of two characters: the filemode letter and the filemode number. The filemode letter specifies the virtual disk on which the file resides, A through Z. The filemode number is a number from 0 to 5. The filemode may be an asterisk (*) in a QUERY command (see "Using the QUERY Command" on page 12). If the filemode number is left off, a default is used. The filemode is optional and if you omit it, a default is assumed. The defaults are determined as described in "Using CMS Disk Files" on page 49.

ddname is one to eight characters and may include letters A through Z, digits 0 through 9, and the national characters @, #, and $. A ddname specifies a ddname of a CMS FILEDEF or DLBL command. For

more information on this, see "Using the CMS FILEDEF Command" on page 53 and "Identifying VSAM Files to BASIC" on page 67.

password is one to eight characters and may include letters A through Z, digits 0 through 9, and the national characters @, #, and $.

The following figure shows the CMS filetypes used by BASIC.

Filetype	BASIC Command or Statement
BASDATA	OPEN Statement (and File I/O Statements) PURGE Command QUERY Command (FILE option)
BASIC	CHAIN Statement (second choice) COMPILE Command LOAD Command MERGE Command PURGE Command QUERY Command (PROGRAM option) RUN Command (SOURCE option) SAVE Command
BASLOG	SET LOG Command
BASPROF	PROFILE Command BASIC Invocation (This may be different for your installation; see your system administrator.)
BASTEXT	FETCH Command PURGE Command QUERY Command (WORKSPACE option) STORE Command
LISTING	COMPILE Command (OUT option) HELP Command (PRT action) LIST Command (OUT option) PURGE Command QUERY Command (OUT option)
TEXT	CALL Statement (for dynamic load) CHAIN Statement (first choice) COMPILE Command (OBJECT option) PURGE Command RUN Command (OBJECT option)

Figure 8. CMS Filetypes Used by BASIC

All leading and trailing spaces in a file-spec are ignored.

The first three forms may only be used to specify CMS disk files. The fourth form requires a CMS FILEDEF or DLBL command that specifies the same ddname. If there is a ddname, the fourth form is assumed instead of the first.

In the third form,

```
"filename [filetype [filemode]]"[/password]
```

apostrophes may be used in place of quotation marks. Note that the quotation marks or apostrophes must be part of the value of a character expression or character constant specified for a CHAIN statement to a source program or specified for the fileid of an OPEN statement.

Example

```
10 CALL SYSTEM ("FILEDEF TEST DISK TEST DATA A1")
20 OPEN #4: "TEST", OUTPUT, BEGIN
```

would cause the CMS disk file TEST DATA A1 to be accessed while

```
10 CALL SYSTEM ("FILEDEF TEST DISK TEST DATA A")
20 OPEN #4: "'TEST'", OUTPUT, BEGIN
```

would cause the CMS disk file TEST BASDATA with a default filemode (see "Using CMS Disk Files" on page 49) to be accessed.

If the closing quotation mark or apostrophe appears at the end of the command line, it may be omitted.

In certain commands, the second form,

```
filename filetype [filemode]
```

may cause ambiguous syntax. This can occur when using a filetype that is the same as an option keyword. The ambiguity is resolved by treating the word as an option if the rest of the command is valid. Otherwise, the word is treated as a filetype.

Example

```
RUN PROG STEP
```

would treat STEP as the STEP option and not as the filetype STEP.

```
RUN PROG STEP STEP
```

would run PROG STEP in STEP mode because the first STEP cannot be the STEP option.

When you want to use a filetype that is the same as a keyword, you may use form three of a file-spec (that is, use quotation marks or apostrophes). For example, if you want to RUN the program PROG with filetype STEP, you should enter:

```
RUN "PROG STEP"
```

If a password is specified for a CMS file, it is accepted but ignored.

If a password is specified for a VSAM file, it must match the password defined in the VSAM catalog. See *Using VSE/VSAM Commands and Macros* and *VSE/VSAM Programmer's Reference.*

Only a filename may be specified as the file-spec for a CHAIN statement to a text file and the RUN command when the OBJECT option is specified. It may *not* be in quotes (that is, the third form of a file-spec may *not* be used).

Using CMS Disk Files

CMS disk files may be specified by not including a ddname in the file-spec of an OPEN statement, CHAIN statement, command, or BASIC invocation. If a ddname is included, a CMS disk file may be specified by using a CMS FILEDEF command (see "Accessing CMS Disk Files" on page 55).

Specifying CMS Disk Files in OPEN Statements

When the file-spec in an OPEN statement is not a ddname, the referenced file is assumed to be a CMS file.

The file is found in the following way:

1. If the file-spec includes a filename, filetype, filemode letter and filemode number, the designated file is accessed. The specified filemode number is ignored when searching for an existing file. The specified filemode number does not override the number for an existing file except for OUTPUT with BEGIN.

   INPUT If the file does not exist, an exception occurs when the OPEN statement is executed.

   OUTPUT If the specified disk is read-only, an exception is generated when the OPEN statement is executed. If a new file is created (either because it did not exist or BEGIN was specified), the specified filemode is assigned to the file.

   OUTIN If the specified disk is read-only, an exception occurs. If the file did not exist, the specified filemode is assigned to the file.

2. If the filemode number is omitted, the filemode number is assumed as shown below:

   INPUT The filemode number of an existing file with the same filename, filetype, and filemode letter as specified is used. An exception occurs if there is no such file.

   OUTPUT If it is an existing file and BEGIN is not specified, the filemode number of the file is used. If a new file is created (either because it did not exist or BEGIN was specified), a filemode number of 1 is assumed. An exception occurs if the disk is read only.

   OUTIN If it is an existing file, the filemode number of the file is used. If the file does not exist, a filemode number of 1 is assumed. An exception occurs if the disk is read-only.

3. If the filemode letter is also omitted, the filemode is determined in the following way:

   INPUT Using the CMS search order, the filemode letter of the first disk on which the file exists is used. An exception occurs if there is no such file. The same filemode number is used.

   OUTPUT Using the CMS search order, the filemode letter of the first disk on which the file exists is used. The same filemode number is used if BEGIN is not specified. The filemode number is set to 1 if BEGIN is specified.

   If not found, a new file is created with a filemode letter equal to the first virtual disk in search order that is read/write, and the filemode number defaults to 1.

   If the disk for an existing file is read-only, an exception is generated when the OPEN statement is executed.

   OUTIN Same as OUTPUT except that if BEGIN is specified for an existing file, the filemode number is not changed.

4. If the filetype is omitted, a filetype of BASDATA is assumed and the filemode is determined as in number 3 above.

Record Attributes for Files Specified in OPEN Statements

The record attributes (record format and record length) of the file are determined in the following way:

1. If opening for OUTIN or OUTPUT to a nonexistent file, or if opening OUTPUT BEGIN, the record attributes specified in the RECORDS option of the OPEN statement will be used. If the RECORDS option has been specified "VARIABLE 0" then the actual record length assigned to the file will be 32756. If a record length is not specified, the default described below is used.

   When the RECORDS option is not specified the file will be assigned default record attributes. The default record format is fixed for relative files; the default for nonrelative files is variable (this is an IBM supplied default that may be changed by your system administrator). The default record lengths are shown below: (These are IBM supplied defaults that may be changed by your system administrator.)

Type	Default Record Length
DISPLAY	133
INTERNAL	255
NATIVE	255

Figure 9. Default Record Lengths for DISPLAY, INTERNAL, and NATIVE Files

2. If it has been determined that an existing file is being referenced (other than with OUTPUT BEGIN) then the record length attributes of the file must be checked against the attributes specified in the OPEN statement. The record format is verified as follows:

   If the record format provided by the OPEN statement does not match the record format of the existing files, an exception will be reported. If the RECORDS option is not specified in the OPEN statement, the record format provided for RELATIVE files is fixed; for non-RELATIVE files it is variable (this is an IBM supplied default that may be changed by your system administrator).

   The record length is verified as follows:

   - If the file has fixed-length records and the record length is not specified in the RECORDS option of the OPEN statement, then the record length of the existing file will be used.

   - If the file has variable-length records and the record length is not specified in the RECORDS option of the OPEN statement (or is specified as "VARIABLE 0"), then the record length of the existing file is used for INPUT files, while for non-INPUT files the maximum of the default (32756 if specified as "VARIABLE 0") and the file's current record length will be used.

   - If the file has fixed-length records and the record length specified in the RECORDS option of the OPEN statement does not equal the record length of the existing file, then an exception will be reported.

   - If the file has variable-length records and the record length specified in the RECORDS option of the OPEN statement is not greater than or equal to the current maximum record length of the existing file, then an exception will be reported.

Specifying CMS Disk Files in CHAIN Statements, BASIC Commands and BASIC Invocations

When the file-spec in a CHAIN statement, in a BASIC command, or in a BASIC invocation is not a ddname, the referenced file is assumed to be a CMS file. The file is found as follows:

1. If the file-spec includes a filename, filetype, filemode letter, and a filemode number, the designated file is accessed. For output files, if a file exists with a different filemode number, it is overwritten with the new filemode number. (If APPEND is specified on a SET LOG command, the filemode number is *not* changed.) The specified filemode number is ignored for input files and by the PURGE command. For QUERY, only files with the specified filemode numbers are listed. For output files and the PURGE command, the disk must be read/write.

2. If the filemode number is omitted, the filemode number is assumed as shown below:

   Input The filemode number of an existing file with the same filename, filetype, and filemode letter as specified is used. An exception or error occurs if there is no such file.

   Output A filemode number of 1 is assigned. (If APPEND is specified on a SET LOG command, the filemode number of an existing file is not changed.) The disk must be read/write.

   PURGE Same as input files. The disk must be read/write.

   QUERY Same as input files unless the filemode is an asterisk; then all the disks on which the file exists are included. For more information on using asterisks, see "Using the QUERY Command" on page 12.

3. If the filemode letter is also omitted, the filemode is determined in the following way:

   Input Using the CMS search order, the filemode letter of the first disk on which the file exists is used. An error occurs if there is no such file.

   Output The filemode letter of the first read/write CMS disk in the CMS search order with a filemode number of 1 is used. (If APPEND is specified on a SET LOG command, the filemode number of an existing file is not changed.)

   PURGE The filemode defaults to A. An error occurs if there is no such file or the disk is read-only.

   QUERY The filemode defaults to A.

4. If the filetype is omitted, the appropriate default is used as given in Figure 8 on page 47. The filemode is determined as described in number 3 above.

   Note: The filetype must be omitted when referencing TEXT files using a CHAIN statement, or in a RUN command with the OBJECT option.

Input files may be specified in the commands COMPILE, FETCH, LOAD, MERGE, PROFILE, and RUN, in the BASIC invocation, or by the CHAIN statement.

Output files may be generated by the compiler and the commands HELP (subcommand PRT), LIST with OUT option, QUERY with OUT option, SAVE, SET LOG, and STORE.

Record Attributes for Files Specified in CHAIN Statements, BASIC Commands, and BASIC Invocation

The record attributes of input and output files must be determined. These files will be assigned default attributes according to the type of file being referenced. A file's type corresponds to the default CMS filetype assigned to the file as described in Figure 8 on page 47. The IBM supplied default record length attributes for the different file types are shown in the following figure:

Filetype	Record Length	Record Format
BASIC	156	V
BASLOG	160	V
BASPROF	156	V
BASTEXT	1024	V
LISTING	133	V
TEXT	80	F

Figure 10. Record Lengths and Record Formats for BASIC Files

The record length default for LISTING files may be changed by your system administrator.

Given these defaults the record length attributes are determined as follows:

1. For output files (but not for appending to an existing file for SET LOG) the appropriate default attributes shown above will be used (an existing file is overwritten).

2. Otherwise, the attributes of the existing file will be used. Note that TEXT (object) files referenced by CHAIN statements, the RUN command with the OBJECT option, or a BASIC invocation must have 80 character fixed-length records or system errors will result.

Using the CMS FILEDEF Command

The CMS FILEDEF command can be used to access CMS disk files, the virtual printer spool file, tapes, the virtual punch, the virtual reader, the virtual console, and a dummy file.

The FILEDEF command can be used to change the file accessed externally to BASIC without changing your BASIC program.

In order to use a FILEDEF, the file-spec must specify a ddname. For more information on using the FILEDEF command, see "FILEDEF Command" on page 87 and *IBM Virtual Machine/System Product: CMS Command and Macro Reference*.

There are relatively few restrictions imposed by BASIC on the use of the FILEDEF command as described in *IBM Virtual Machine/System Product: CMS Command and Macro Reference*. You should be aware of the following:

- The CMS FILEDEF command may be issued before entering BASIC, from the BASIC environment using the SYSTEM command, or from a BASIC program using the CALL SYSTEM statement.
- All FILEDEF commands that are not specified with the PERM option are cleared when BASIC terminates using the QUIT command, when BASICRUN terminates, or a linked BASIC program terminates.
- FILEDEF commands may not specify the virtual card reader when running in CMS batch mode.
- When a FILEDEF command specifies any device other than DISK, the file must be processed sequentially; that is, the file may not be opened RELATIVE or KEYED.
- The BASIC statements REREAD and REWRITE are allowed only for files on disk devices.
- File positioning on the OPEN statement (other than BEGIN) is not permitted on any device except tape or disk.
- The RESET statement may be specified only for files on disk and tape devices; for tape devices, the positioning may only be BEGIN or END/APPEND.
- Because the FILEDEF command overrides the default file conventions, inadvertent overwriting of program or data files is more probable than usual. Therefore, you should take appropriate care in processing such files.
- Obvious usage conflicts (such as attempting to send output to the virtual card reader, attempting to read from the printer) result in error messages.
- The FILEDEF command may not be used to override the default file name, filetype, and filemode for TEXT (object) files and will be ignored if attempted. This applies to the CALL and CHAIN statements, the RUN command with the OBJECT option, and BASICRUN invocation.
- See "Accessing Tapes" on page 59 for tape considerations.
- The tape options SUL and DISP MOD are not supported.
- The tape option LABOFF is treated differently by BASIC than as described in *IBM Virtual Machine/System Product: CMS Command and Macro Reference.*

The general form of the FILEDEF command for accessing a CMS disk file is:

```
FILEDEF ddname DISK filename filetype [filemode]
```

where:

ddname Tells CMS that ddname is synonymous with the file identifier: filename, filetype [filemode]. When BASIC refers to a file named ddname, the reference is routed to the CMS disk file specified in the FILEDEF. In this case, the BASIC default filename, filetype, and filemode may be overridden.

Example

```
10  CALL SYSTEM ("FILEDEF RESULTS DISK AUGUST RESULTS A1")
20  OPEN #12:  "RESULTS",OUTPUT, NATIVE, BEGIN
```

will cause the output to go to AUGUST RESULTS A1.

The FILEDEF options supported by BASIC when referencing CMS disk files are: CHANGE, NOCHANGE, LRECL, PERM, and RECFM. The RECFM option can be coded as F to specify fixed-length records, or V to specify variable-length records. All other attributes of the RECFM option will be ignored.

When you use a FILEDEF to reference a CMS disk file, BASIC must verify the record length attributes specified by the RECFM and LRECL options and assign the proper attributes to the file.

If the record format (RECFM) is not specified in the FILEDEF, BASIC determines it as described in "Record Attributes for Files Specified in OPEN Statements" on page 50 and "Record Attributes for Files Specified in CHAIN Statements, BASIC Commands, and BASIC Invocation" on page 53. If specified in the FILEDEF, the record format is verified as follows:

- If the file is referenced by a CHAIN statement, a BASIC command, or a BASIC invocation, BASIC uses the following procedure:
  - For output files (but not for appending to an existing file for SET LOG), the record format specified by the RECFM option of the FILEDEF will be used (an existing file is overwritten).
  - Otherwise, if the record format specified by the RECFM of the FILEDEF does not match the record format of the existing file then an error will be reported.
- If the file is referenced by an OPEN statement, BASIC uses the following procedure:
  - If opening OUTIN or OUTPUT for a nonexistent file, or if opening OUTPUT BEGIN, and the record format provided by the OPEN statement does not match the record format specified by the RECFM option of the FILEDEF, then an exception will be reported. If the RECORDS option is not specified in the OPEN statement, the record format provided for RELATIVE files is fixed; for non-RELATIVE files it

is variable (this is an IBM supplied default that may be changed by your system administrator).

- If opening an existing file (for other than OUTPUT BEGIN) and the record format provided by the OPEN statement, the FILEDEF, and the existing file are all not the same, then an exception will be reported. If the RECORDS option is not specified in the OPEN statement, the record format provided for RELATIVE files is fixed; for non-RELATIVE files it is variable (this is an IBM supplied default that may be changed by your system administrator).

- If the record length (LRECL) is not specified in the FILEDEF, it is determined as described in "Record Attributes for Files Specified in OPEN Statements" on page 50 and "Record Attributes for Files Specified in CHAIN Statements, BASIC Commands, and BASIC Invocation" on page 53. If specified in the FILEDEF, the record length is verified as follows:

  - If the file is referenced by a CHAIN statement, a BASIC command, or a BASIC invocation, BASIC uses the following procedure:

    — For output files (but not for appending to an existing file for SET LOG), the record length specified by the LRECL option of the FILEDEF will be assigned to the file.

    — Otherwise, the record length specified by the LRECL option of the FILEDEF must be equal to the record length of the file if it has fixed-length records or must be greater than or equal to the current maximum record length of the file if it has variable-length records. If this is true, then the record length of the LRECL will be assigned to the file; otherwise, an error will be reported for the file.

    — If opening OUTIN or OUTPUT for a nonexistent file, or if opening OUTPUT BEGIN, and a nonzero record length is specified in the OPEN statement and it does not match the record length specified by the RECL option of the FILEDEF, then an exception will be reported.

    — If opening an existing file (for other than OUTPUT BEGIN) and "VARIABLE 0" is specified, or the record length is not specified in the OPEN statement, then the record length specified by the LRECL of the FILEDEF must equal the record length of the file if it has fixed-length records or must be greater than or equal to the current maximum record length of the file if it has variable-length records. If this is true, then the record length of the LRECL will be assigned to the file; otherwise, an exception will be reported.

    — If opening an existing file (for other than OUTPUT BEGIN) and a nonzero record length is specified in the OPEN statement, then this value must be equal to the record length specified by the LRECL option of the FILEDEF. If this is not true, an exception will be reported. If it is true, then the record length of the RECORDS option must equal the record length of the file if it has fixed-length records or must be greater than or equal to the current maximum record length of the file if it has variable-length records. If this is true, then this record length will be assigned to the file; otherwise, an exception will be reported.

You can route output files directly to the virtual printer spool file. (Check with your system administrator if you don't know where your printer output is printed.) For a file that has carriage control as the first character in each record, use the FILEDEF command to route the file to the virtual printer, as follows:

```
FILEDEF ddname PRINTER
```

Only the following types of files have a carriage control character as the first character in each record:

- BASIC Processor-produced files that have a default filetype of LISTING (see Figure 8 on page 47).
- BASIC program-generated data files (default filetype BASDATA) that are opened with a DEVICE PRINTER or DEVICE 3800 clause.
- BASIC program-generated data files (default filetype BASDATA) that are not opened with a DEVICE PRINTER or DEVICE 3800 clause but for which the program explicitly places a carriage control character at the beginning of each record. In this case, you should specify the RECFM FA or RECFM FM option in the FILEDEF.

  For other files, a space character is prefixed to each output record for use as the carriage control character.

Example

Your program creates a file MYPRINT which can be directed to the printer:

```
100 CALL SYSTEM ('FILEDEF MYPRINT PRINTER')
110 OPEN #1: "MYPRINT, DEVICE PRINTER", DISPLAY, OUTPUT
     .
     .
     .
250 CLOSE #1:
```

Example

```
SYSTEM 'FILEDEF BASLOG PRINTER'
SET LOG ON
```

will print the log.

The FILEDEF options supported by BASIC when referencing printer files are: CHANGE, NOCHANGE, LRECL, PERM, and RECFM. The only RECFM option that is supported is M to specify machine print control characters and A to specify ASA control characters. All other RECFM codes are ignored.

The only record attribute that is important for PRINTER files is the record length; the record format is fixed (specification of variable in the OPEN statement or V in the FILEDEF will be ignored).

BASIC verifies the record length as follows:

- If the file is referenced by a BASIC command or BASIC invocation, BASIC uses the following procedure:
  - If the LRECL option of the FILEDEF is coded and the value is less than or equal to 132 (133 if it is a LISTING type file), then the LRECL value will be used; otherwise, an error will be reported.
  - If the LRECL option is not coded in the FILEDEF, then the file will be assigned a default record length according to the filetype of the file being referenced (see Figure 10 on page 53 for a list of the filetypes and default record lengths). For a non-LISTING type file, if the default record length is greater than 132, then 132 will be used instead. For LISTING type files, if the default record length is greater than 133, then 133 will be used instead.
- If the file is referenced by an OPEN statement the following procedure is used:
  - If the LRECL option of the FILEDEF is coded and the value is greater than 132 (133 if DEVICE PRINTER/3800 is specified in the OPEN statement or A or M is specified in the RECFM of the FILEDEF), then an exception will be reported.
  - If the LRECL option of the FILEDEF is coded and the value is less than or equal to 132 (133 if DEVICE PRINTER/3800 is specified in the OPEN statement or A or M is specified in the RECFM of the FILEDEF) and the RECORDS option is not specified in the OPEN statement (or the record length is omitted or is zero), then the record length specified by the LRECL is assigned to the file.
  - If the LRECL option of the FILEDEF is not coded and the RECORDS option of the OPEN statement is not specified (or the record length is omitted or is specified as zero), then the file will be assigned a default record length. (See Figure 9 on page 50 for a list of the default record lengths.) When "VARIABLE 0" is specified by the RECORDS option or the default record length is greater than 132 (133 if DEVICE PRINTER/3800 is specified in the OPEN statement or A or M is specified in the RECFM of the FILEDEF), then 132 or (or 133) will be used for the record length.
  - If the LRECL option of the FILEDEF is not coded and the RECORDS option of the OPEN statement specifies a nonzero value and this value is less than or equal to 132 (133 if DEVICE PRINTER/3800 is specified in the OPEN statement or A or M is specified in the RECFM of the FILEDEF), then the file will be assigned the record length specified by the RECORDS option; otherwise, it will be assigned 132 (or 133).

An additional attribute of importance to printer files is print control (A or M):

1. When the file is referenced by a BASIC command or BASIC invocation, the print control specified by the RECFM option of the FILEDEF is ignored. ASA control will be used if the file is a LISTING filetype (see Figure 8 on page 47 for a list of the filetypes).

2. If the DEVICE PRINTER or DEVICE 3800 options are specified on the OPEN statement, the control specified by the RECFM option of the FILEDEF is ignored and ASA print control will be used.

3. For an OPEN statement without DEVICE PRINTER or DEVICE 3800, the print control specified in the FILEDEF will be used. If print control is specified, the program must provide the necessary control characters in the first column of each record.

Accessing Tapes

You can connect a file with a tape unit by means of:

```
FILEDEF ddname TAPn
```

where:

n Is the symbolic number of the tape drive. *n* can be 1, 2, 3, or 4, representing virtual addresses 181, 182, 183, and 184, respectively. For more details, see *IBM BASIC Virtual Machine/System Product: CMS Command and Macro Reference.*

Example

To save a BASIC program from your workspace on magnetic tape (attached as virtual address 181):

```
SYSTEM 'FILEDEF MYTAPE TAP1'
SAVE MYTAPE
```

BASIC will allow you to access multifile tapes and to specify tape label processing. To do this, you use a second operand after TAPn. For further explanation of these operands, see "FILEDEF Command" on page 87 and *IBM Virtual Machine/System Product: CMS Command and Macro Reference.*

When a tape file is opened with BEGIN and the corresponding FILEDEF for the tape specifies LABOFF (the default), the tape is rewound. If the FILEDEF specifies BLP, SL, NSL, or NL after TAPn, the tape is positioned to the beginning of the indicated file.

When a tape is opened with END or APPEND and the corresponding FILEDEF for the tape specifies LABOFF (the default), the tape is rewound and then positioned at the next tapemark. If the FILEDEF specifies BLP, SL, NSL, or NL, the tape is positioned at the end of the indicated file.

If a tape is RESET (or RESTOREd), the tape is repositioned as described above. If the file is opened for OUTPUT, a tapemark is written before repositioning.

If the file is opened for OUTPUT, a tapemark is written when the file is closed. If you then mount a blank tape and specify NL, the tape will run off the end of the reel. Write a tape mark to prevent this from occurring.

Tapes may not be opened for OUTIN access.

The FILEDEF options SUL and DISP MOD are not supported.

Here are some examples of positioning multifile tapes:

```
FILEDEF FILEA TAP2 BLP 2
```

positions the tape to the second file but does not check to see if the tape has a label.

```
FILEDEF FILEA TAP2 SL 2 VOLID DEPT10
```

specifies the second file on TAP2 with standard labels. When this tape is opened, CMS checks to see that it has a VOL1 label with a volume serial number of DEPT10.

The FILEDEF options supported by BASIC when referencing tape files are: BLOCK, BLKSIZE, CHANGE, NOCHANGE, DEN, LEAVE, LRECL, NOEOV, PERM, RECFM, TRTCH, 7TRACK, and 9TRACK. All RECFM options are supported with the exception of U (undefined record format).

When a tape file is referenced, BASIC must verify that the record attributes have been specified correctly and select the attributes that should be assigned to the file. BASIC is not provided with the attributes of the existing tape files. Therefore, BASIC can only examine the attributes provided by the FILEDEF and the attributes provided by BASIC command, statement, or invocation referencing the file. BASIC will select the attributes to be used from one of these two sources and pass them to CMS. If any of these attributes is incompatible with the current attributes of the existing file, unpredictable results may occur.

BASIC verifies the record format as follows:

- If the file is referenced by a CHAIN statement (source only), a BASIC command, or a BASIC invocation, the following procedure is used:
  - If the RECFM option of the FILEDEF is not coded, the file will be assigned a default record format according to the filetype of the file being referenced (see Figure 10 on page 53 for a list of the filetypes and default attributes).
  - Otherwise, the file will be assigned the record format specified by the RECFM option of the FILEDEF.
- If the file is referenced by an OPEN statement, the following procedure is used:
  - If the RECFM option is not coded in the FILEDEF and the RECORDS option is not specified in the OPEN statement, then the file will be assigned a default record format. The default for relative files is fixed; for nonrelative files it is variable (this is an IBM-supplied default that may be changed by your system administrator).
  - If the RECFM option is not coded in the FILEDEF and the RECORDS option is specified in the OPEN statement, then the file is assigned the record format specified by the RECORDS option.
  - If the RECFM option of FILEDEF does not match the record format provided by the OPEN statement, then an exception will be reported. If

the RECORDS option is not specified in the OPEN statement, the record format provided for RELATIVE files is fixed; for non-RELATIVE files it is variable (this is an IBM supplied default that may be changed by your system administrator).

- When you verify the record length, pay special attention to the record length of files with variable-length records. The record length specified by an OPEN statement defines the length of the data portion of the record while the value specified by the FILEDEF defines the "physical record length." The "physical record length" is equal to four plus the length of the data portion of the record.

BASIC verifies the record length as follows:

- If the file is referenced by a CHAIN statement (source only), a BASIC command, or a BASIC invocation, the following procedure is used:

  - If the LRECL option of the FILEDEF is not coded, the file will be assigned a default record length according to the filetype of the file being referenced (see Figure 10 on page 53 for a list of the filetypes and default attributes). If the file has variable length records, 4 is added to the default record length to obtain the "physical record length."

  - Otherwise, the file will be assigned the record length specified by the LRECL. If the file has variable-length records, the value specified by the LRECL is the "physical record length."

- If the file is referenced by an OPEN statement, the following procedure is used:

  - If the LRECL option is not coded in the FILEDEF and the RECORDS option of the OPEN statement is not specified (or the record length is omitted), then the file will be assigned a default record length (see Figure 9 on page 50 for a list of the default record lengths). If the file has variable-length records, 4 is added to the default record length to obtain the "physical record length."

  - If the LRECL option is not coded in the FILEDEF and the RECORDS option of the OPEN statement specifies "VARIABLE 0," then the file will be assigned a record length of 32752 (the physical record length will be 32756).

  - If the LRECL option is not coded in the FILEDEF and the RECORDS option in the OPEN statement specifies a nonzero value, then the file is assigned this value. If the file has variable-length records, 4 is added to the value specified by the RECORDS option to obtain the "physical record length."

  - If the file has fixed-length records and the record length specified by the LRECL option of the FILEDEF does not equal the nonzero record length specified by the RECORDS option of the OPEN statement, then an exception will be reported.

  - If the file has variable-length records and the record length specified by the LRECL option of the FILEDEF is not equal to four plus the nonzero

record length specified by the RECORDS option of the OPEN statement, then an exception will be reported.

- Block size is an additional file attribute that is important only for tape files and only when the RECFM option of the FILEDEF specifies blocking. If RECFM FB or RECFM VB is specified, but no BLOCK or BLKSIZE option is coded in the FILEDEF, one of the following default block sizes will be used:
  - For a variable record length file, the block size will be four plus the physical record length.
  - For a fixed record length file, the block size will equal the record length.
- If the block size is provided by the FILEDEF, it will be used unless one of the following error conditions exists:
  - The file has variable-length records and the block size is not greater than or equal to four plus the physical record length.
  - The file has fixed-length records and the block size is not a multiple of the record length.
- If neither RECFM FB nor RECFM VB is specified, the tape file will not be blocked and a BLOCK or BLKSIZE option in the FILEDEF will be ignored.

Writing to the Punch

Output files can be directed to the CMS virtual punch.

```
FILEDEF ddname PUNCH
```

Example

To send the BASIC program in your workspace to the virtual punch:

```
SYSTEM 'FILEDEF DECK PUNCH'
SAVE DECK
```

Example

To direct the text output of a compilation to the punch:

```
SYSTEM 'FILEDEF TEXT PUNCH'
COMPILE
```

The FILEDEF options supported by BASIC when referencing virtual punch files are: CHANGE, NOCHANGE, LRECL, and PERM.

The only record attribute that is important for PUNCH files is the record length; the record format is fixed (specification of variable in the OPEN statement or V in the FILEDEF will be ignored).

BASIC verifies the record length as follows:

- If the file is referenced by a BASIC command or BASIC invocation, the following procedure is used:
  - If the LRECL option of the FILEDEF is coded and the value is less than or equal to 80, then the LRECL value will be used; otherwise, an error will be reported.
  - If the LRECL option is not coded in the FILEDEF, then the file will be assigned a default record length according to the filetype of the file being referenced (see Figure 10 on page 53 for a list of the filetypes and default record lengths). If the default record length is greater than 80, then 80 will be used instead.
- If the file is referenced by an OPEN statement the following procedure is used:
  - If the LRECL option of the FILEDEF is coded and the value is greater than 80, then an exception will be reported.
  - If the LRECL option of the FILEDEF is coded, the value is less than or equal to 80, and the RECORDS option is not specified in the OPEN statement (or the record length is omitted or is zero), then the record length specified by the LRECL is assigned to the file.
  - If the LRECL option of the FILEDEF is not coded and the RECORDS option of the OPEN statement is not specified (or the record length is omitted or is specified as zero), then the file will be assigned a default record length or 80 if the default is greater than 80. For a list of the default record lengths, see Figure 9 on page 50. When "VARIABLE 0" is specified by the RECORDS option, 80 will be used for the record length.
  - If the LRECL option of the FILEDEF is not coded, the RECORDS option of the OPEN statement specifies a nonzero value, and this value is less than or equal to 80, then the file will be assigned the record length specified by the RECORDS option; otherwise, it will be assigned 80.

Reading from the Reader

The CMS virtual card reader is accessed by:

```
FILEDEF ddname READER
```

Example

To read a BASIC source program from the reader into your workspace:

```
SYSTEM 'FILEDEF X READER'
LOAD X
```

Note: The LOAD command names the workspace X, and because you cannot write to the reader, unless you rename the workspace before you issue it or clear the FILEDEF, a subsequent SAVE command will fail. You can rename the workspace in either of the following ways:

- Issue a RENAME command before you issue a SAVE command
- Specify a different name on the SAVE command itself

The FILEDEF options supported by BASIC when referencing virtual reader files are: CHANGE, NOCHANGE, LRECL, and PERM.

The only record attribute that is important for READER files is the record length; the record format is fixed (specification of variable in the OPEN statement or V in the FILEDEF will be ignored).

The record length is verified in the same manner as virtual punch files as described in "Writing to the Punch" on page 62. (A CHAIN statement for a source file is handled in the same manner as a BASIC command.)

Accessing the Terminal

The CMS virtual console (the terminal) is accessed by:

FILEDEF ddname TERMINAL

Example

```
10 CALL SYSTEM ('FILEDEF TERMIN TERMINAL')
20 OPEN #10: 'TERMIN',INPUT
30 CALL SYSTEM ('FILEDEF TERMOUT TERMINAL')
40 OPEN #11: 'TERMOUT',OUTPUT
```

will cause input statement referencing file #10 to read from the terminal and output statements referencing file #11 to write to the terminal.

The virtual console may not be opened for OUTIN access.

The FILEDEF options supported by BASIC when referencing virtual console files are: CHANGE, NOCHANGE, LRECL, PERM, and RECFM. The A code of the RECFM option (to specify ASA print control) is usually ignored; however, for output files the rules are the same as for printer files, as described on page 58. All other attributes of the RECFM option will be ignored.

The only record attribute that is important for console files is the record length; the record format is fixed (specification of variable in the OPEN statement or V in the FILEDEF will be ignored).

The record length is verified as follows:

- If the file is referenced by a CHAIN statement, a BASIC command, or BASIC invocation, the following procedure is used:
  - If the LRECL option of the FILEDEF is coded and the value is less than or equal to the current CP LINESIZE value (or the current CP LINESIZE value plus one if LISTING type file), then the LRECL value will be used; otherwise, an error will be reported.
  - If the LRECL option is not coded in the FILEDEF, then the file will be assigned a default record length according to the filetype of the file being referenced (see Figure 10 on page 53 for a list of the filetypes and default

record lengths). If the default record length is greater than the current CP LINESIZE value (or the current CP LINESIZE value plus one if LISTING type file), then LINESIZE (or LINESIZE plus one for LISTING type files) will be used instead.

- If the file is referenced by an OPEN statement, the following procedure is used:

  - If the LRECL option of the FILEDEF is coded and the value is greater than the current CP LINESIZE value (or the current CP LINESIZE value plus one if DEVICE PRINTER/3800 is specified in the OPEN statement or A is specified in the RECFM of the FILEDEF for an output file), then an exception will be reported.

  - If the LRECL option of the FILEDEF is coded, the value is less than or equal to the current CP LINESIZE value (or the current CP LINESIZE value plus one if DEVICE PRINTER/3800 is specified in the OPEN statement or A is specified in the RECFM of the FILEDEF for an output file) and the RECORDS option is not specified in the OPEN statement (or the record length is omitted or is zero), then the record length specified by the LRECL is assigned to the file.

  - If the LRECL option of the FILEDEF is not coded and the RECORDS option of the OPEN statement is not specified (or the record length is omitted or is specified as zero), then the file will be assigned a default record length or the current CP LINESIZE value (or the current CP LINESIZE value plus one if DEVICE PRINTER/3800 is specified in the OPEN statement or A is specified in the RECFM of the FILEDEF for an OUTPUT file) if the default is greater than the current CP LINESIZE value (or the current CP LINESIZE value plus one). See Figure 9 on page 50 for a list of the default record lengths. When "VARIABLE 0" is specified by the RECORDS option, the current CP LINESIZE value will be used for the record length.

  - If the LRECL option of the FILEDEF is not coded and the RECORDS option of the OPEN statement specifies a nonzero value and this value is less than or equal to the current CP LINESIZE value (or the current CP LINESIZE value plus one if DEVICE PRINTER/3800 is specified in the OPEN statement or A is specified in the RECFM of the FILEDEF for an OUTPUT file), then the file will be assigned the record length specified by the RECORDS option; otherwise, it will be assigned the current CP LINESIZE value (or the current CP LINESIZE value plus one).

Using VSAM Files under CMS

Unless you make explicit arrangements for VSAM access, all files handled within BASIC are assumed to have sequential, stream, or relative organization, with CMS performing the access. You must use VSAM for any keyed file; you can also use it for sequential or relative files.

VSAM must be installed on your system before you can use VSAM files. See your system administrator to find out how to establish linkage to VSAM.

Prior to any use within BASIC, a VSAM file must have been created, and the file and catalog must be identified to CMS.

Creating VSAM Files under CMS

The creation of VSAM files can include several steps and many parameters. For details, see *VSE/VSAM Programmer's Reference* and *IBM Virtual Machine/System Product: CMS User's Guide.*

First, your system must have a VSAM data space and a VSAM master catalog defined. Your organization probably has established procedures for creating and maintaining these. The VSAM data space is a minidisk in DOS format and is where the VSAM files and catalogs reside. The master catalog has a filename of IJSYSCT and must also reside on a DOS-formatted minidisk (possibly on the same one as the data space).

You can then create a VSAM file by:

- Executing a DLBL command telling CMS on which minidisk to find the VSAM master catalog, IJSYSCT
- Executing an AMSERV command to create the file

Note that this must be done prior to entering BASIC; AMSERV is not one of the CMS subset commands that can be used under the BASIC SYSTEM command.

Example

In *IBM BASIC Programming Guide*, Sample Program 4 works with a keyed file it expects to have 80 characters per record and with the first 15 characters of each record used as the key field.

This file could be created by:

```
DLBL IJSYSCT  K  (VSAM PERM)
AMSERV DEF15
```

where K is the filemode of the minidisk with the master catalog and DEF15 is the filename of a file having filetype AMSERV and containing the access method command:

```
DEFINE CLUSTER            -
  (NAME (KEY15)           -
  VOL (DOSDSK)            -
  TRACKS (20)             -
  INDEXED                 -
  KEYS (15 0)             -
  RECORD SIZE (80  800)   -
  REUSE)
```

This defines an indexed file whose name in the catalog is KEY15. It has 20 tracks on the disk whose volume identification is DOSDSK. The key begins in the first position of each record and is 15 characters long. The records are variable in size, with an average length of 80 and a maximum length of 800.

Note: There must be at least one space preceding the keyword DEFINE in the file DEF15 AMSERV.

Identifying VSAM Files to BASIC

In order to access an existing VSAM file in BASIC, two DLBL commands are required, one to identify the VSAM master catalog and one to identify the file.

Example

From within the BASIC environment, you can issue:

```
SYSTEM 'DLBL IJSYSCT  K  (VSAM)'
SYSTEM 'DLBL CUSTOMR K  DSN KEY15 (VSAM)'
```

to identify the catalog and the file defined in the preceding example (assuming the K-disk had the volume DOSDSK). Sample Program 4 in *IBM BASIC Programming Guide* contains an OPEN statement that specifies the ddname CUSTOMR. That OPEN statement will now access the VSAM file KEY15.

The DLBL commands stay in effect until they are reset, either explicitly or by some system action that clears them. The status of DLBL commands can be checked by entering the DLBL command with no operands.

Note: BASIC programs intended to access VSAM files with sequential or relative organization will access CMS files by default if the the DLBL commands are not in effect; a file defined as KEYED in BASIC ("indexed" in VSAM) cannot be opened unless the DLBL commands are in effect.

The file as defined above has no records in it until the BASIC program creates them. The file is never defined again, unless you delete it entirely; you would delete it if you wanted to change any of its characteristics.

The AMSERV MYDEL command deletes the file when the file MYDEL AMSERV contains:

```
DELETE KEY15 PURGE
```

Note: There must be at least one space preceding the keyword DELETE in the file MYDEL AMSERV.

The record attributes (record format and record length) of a VSAM file are determined as follows:

- If the file is referenced by a CHAIN (source only), a BASIC command, or a BASIC invocation, then all attributes are determined from the cluster definition of the file.
- If the file is referenced by an OPEN statement, the following procedure is used:
  - If the file is opened as RELATIVE and the OPEN statement specifies variable-length records, then an exception will be reported.
  - If the RECORDS option of the OPEN statement specifies "VARIABLE 0" or does not specify a record length, then the maximum record length specified in the cluster definition will be used.

- If the RECORDS option of the OPEN statement specifies a nonzero record length, then the following rules apply:
  - If the file is RELATIVE then the record length specified by the OPEN must be equal to the maximum record length in the cluster definition of the VSAM file.
  - If the file is not RELATIVE and is opened for INPUT, then the record length specified by the OPEN must be greater than or equal to the maximum record length in the cluster definition of the VSAM file.
  - If the file is not RELATIVE and is opened for OUTPUT, then the record length specified by the OPEN must be less than or equal to the maximum record length in the cluster definition of the VSAM file.
  - If the file is not RELATIVE and is opened for OUTIN, then the record length specified by the OPEN must be equal to the maximum record length in the cluster definition of the VSAM file.

Using the CODE Function to Interpret CMS or VSAM Errors

You can determine that the operating system will detect an error by using the BASIC statement ON ERROR GOTO or by using an exception-recovery clause in your I/O statements. If you want to identify the error, you then use the CODE function to get the value forwarded by the operating system. For CP, CMS, or VSAM errors, the CODE function returns a hexadecimal value whose meaning is as follows:

- First (high-order) byte

  00 = CMS (bytes 2 through 4 contain the CP or CMS return code)

  04 = VSAM (bytes 2 through 4 as follows)

- Second byte for VSAM

  04 = VSAM OPEN error

  08 = VSAM CLOSE error

  0C = VSAM REQUEST error

  10 = VSAM MODCB error

  14 = VSAM TESTCB error

  18 = VSAM SHOWCB error

- Third byte for VSAM

  Low order byte of register 15 as returned by VSAM

- Fourth byte for VSAM

  VSAM return code

Example

```
100  ON ERROR GOTO DET_If_SYS_ERR
     .
     .
     .
500  DET_If_SYS_ERR:If INT(ABS(ERR)/1000) = 11 THEN SYSTEM_ERROR
     .
     .
     .
handle non-system error
     .
     .
     .
600  SYSTEM_ERROR:  REM  Determine system error
610  If INT(CODE/(2**24)) = 4 THEN VSAM_ERROR
620  CMS_CP_CODE = MOD(CODE,2**24)
     .
     .
     .
handle CMS or CP code
     .
     .
     .

650  VSAM_ERROR:  REM Determine VSAM error, split up bytes
660  VSAM_TYPE_CODE = MOD(CODE/(2**16), 256)
670  VSAM_REG_15 = MOD(CODE/(2**8), 256)
680  VSAM_RETURN_CODE = MOD(CODE,256)
     .
     .
     .
handle VSAM codes
     .
     .
     .
```

For more information on return codes from CMS commands, see *VM/SP: CMS Command and Macro Reference*. For more information on CP return codes and other CMS return codes, see *VM/SP: System Messages and Codes*.

For more information on VSAM error and return codes, see *VSE/VSAM Messages and Codes*.

Using IBM BASIC under VM/PC

IBM Virtual Machine/Personal Computer (VM/PC) is an IBM licensed program that runs on the IBM Personal Computer XT/370 or AT/370. VM/PC gives you an interactive system that has the characteristics of a VM/SP system.

There are three different methods you can follow to use IBM BASIC under VM/PC. (You may want your system administrator to do these tasks for you.)

- Create the BASIC module on the VM/PC. Then load the module into a nucleus extension with the NUCXLOAD command. Creating the module is necessary only when you first access BASIC, or after a new release has been installed on the host system. However, you must issue the NUCXLOAD command after every Initial Program Load (IPL) of CMS.

- Copy (download) the BASIC module onto local disk files and then invoke BASIC in local sessions. You need to download only when you first access BASIC, or when a new release has been installed on the host system.

- Link to the host-system minidisk containing BASIC and then access BASIC from the local session as a remote minidisk. You must do this after every Initial Program Load (IPL) of CMS, and whenever the link to the host system fails.

  Depending on your link with the system and on the system load, this often is not an efficient way to operate. Therefore, it is not further described in this section.

You can then invoke the Processor either inside or outside the BASIC environment.

Note: If you frequently get a message telling you that the loader table has overflowed, you should add the following command to your PROFILE EXEC procedure:

```
SET LDRTBLS 5
```

This command should correct the problem.

VM/PC Processing Restrictions on IBM BASIC

Any BASIC restrictions on CMS processing apply for VM/PC as well.

The following processing capabilities are *not* available when you are executing an object program in a local session:

- VSAM file processing
- Magnetic tape processing
- The Graphical Data Display Manager (GDDM)
- SQL/DS

In addition, in the SYSTEM command and the CALL SYSTEM statement, you cannot use any CMS subset commands not supported under VM/PC.

Using NUCXLOAD With IBM BASIC

You can decrease the processing time needed to access BASIC repeatedly by installing IBM BASIC as a nucleus extension.

You must first create the BASIC and/or BASICRUN modules on your VM/PC system. The BASIC module contains all the Processor and Library components for operating within the BASIC environment. The BASICRUN module contains the components needed to execute a compiled BASIC program outside the BASIC environment.

After you have created these modules, you can upload them into the host system; they can then be downloaded to other VM/PC stations.

To create the modules, follow these steps. (You may be able to have your system administrator do this for you.)

1. Make sure you have the BASIC text files available on the host system. (You may need to read them from the installation tape and copy them onto a CMS minidisk.)

2. Modify the following line from the BGEN EXEC on the installation tape. Add the RLDSAVE option at the end of the following LOAD command:

   ```
   LOAD BLIOCRT (CLEAR NOAUTO NOLIBE RESET BLIOCRT
   ```

   as follows:

   ```
   LOAD BLIOCRT (CLEAR NOAUTO NOLIBE RESET BLIOCRT RLDSAVE
   ```

3. Create the modules during your VM/PC local session by entering one or both of the following:

```
BGEN (ALL
```

This command creates BASIC MODULE A.

or

```
BGEN L (ALL
```

This command creates BASICRUN MODULE A.

For more information on the LOAD, INCLUDE, NUCXLOAD, NUCXDROP, and NUCXMAP commands, see *VM/PC User's Guide*.

After these modules have been created, you can upload them into the host system; they can then be downloaded to other VM/PC stations. You may want to rename these modules first; this prevents inadvertent overwriting of the host system disks.

After the BASIC and/or BASICRUN module has been created, you can load them into your nucleus by entering the following commands:

```
NUCXLOAD BASIC
```

or

```
NUCXLOAD BASICRUN
```

You must issue these commands each time you Initial Program Load (IPL) CMS. You can put these commands into your PROFILE EXEC, which will issue them for you.

After this procedure is completed, you can invoke BASIC by entering:

```
BASIC
```

and the IBM BASIC Processor is ready for your use. For more information on invocation and using BASIC in the BASIC environment, see "The BASIC Environment under CMS" on page 4.

Downloading IBM BASIC to VM/PC

First, check with your system administrator to find out how the IBM BASIC Processor is stored on the host system:

- If the IBM BASIC Processor is a single load module, you can immediately download the Processor as described in the following paragraphs.

  For VM/PC usage, it is usually most efficient to use this module, which is created with the SPACE installation option. If BASIC was installed using the SPEED option, see the next step.

- If the IBM BASIC Processor is a group of text files, or is a combination of text files and load modules, or was installed with the SPEED option, ask your system administrator to create a Processor in a single load module with the SPACE option.

You can then download BASIC, using the commands shown in Figure 11 on page 75. The procedure is as follows:

1. Link (if necessary) and access the local minidisk that is the target minidisk for the copy operation. If the target minidisk is your own minidisk, the link is not required.

2. Link and access the host minidisk that contains the BASIC modules.

3. Copy the BASIC modules from the host minidisk to the local minidisk. If you are using the BASIC reserved word file, be sure to copy it. The default name is BASRESW BASOPTS. Be sure to copy any other BASIC options file you may be using. (Ask your system administrator about this.) This is known as downloading. Downloading is necessary only when you first wish to access BASIC, or after a new release has been installed on the host system.

4. Release the host BASIC minidisk; it is no longer required.

The BASIC module must include all the Processor and Library text files.

1) Link and access the target minidisk.

```
CP LINK vm/pc-id ttt aaa W write-password
ACCESS aaa filemode1
```

2) Link and access the host minidisk that contains the BASIC modules.

```
CP LINK host-id hhh bbb RR read-password REMOTE
ACCESS bbb filemode2
```

3) Copy the files you want (one or more of the following)

```
COPYFILE BASIC MODULE filemode2 = = filemode1
COPYFILE BASICRUN MODULE filemode2 = = filemode1
COPYFILE BASHELP MACLIB filemode2 = = filemode1
```

(If you are using the reserved word files, be sure to copy it.
The default name, shown below, is BASRESW BASOPTS.)

```
COPYFILE BASRESW BASOPTS filemode2 = = filemode1
```

4) Release the host BASIC minidisk

```
RELEASE filemode2 (DET
```

Where:

ttt - is the virtual address of the local target minidiskat will store the BASIC modules.
aaa - is an unused virtual address on the local VM/PC mache.
hhh - is the virtual address of the host minidisk that conins the BASIC modules.
bbb - is the virtual disk address you use to refer to the st disk.
filemode1 - is the filemode of the target minidisk.
filemode2 - is the filemode of the host minidisk that contains the BASIC modules.

Figure 11. CMS Commands to Download BASIC from the Host System to VM/PC

If you only want to execute BASIC programs from previously compiled text, you only need to copy the BASICRUN module. However, if you want to develop and process BASIC programs completely, you should copy both the BASIC and BASICRUN modules.

If space is available, you can also download the BASIC HELP panels into local storage. To do this, copy the file BASHELP MACLIB.

There are approximately 1200 HELP panels, and you can choose to download any or all of them. For example, you might want to download only those files associated with BASIC messages. (The filenames of BASIC message HELP panels all begin with the prefix BAS or BLI, followed by digits. For example: BAS00034 BASHELP.) Use the CMS MOVEFILE command to move a panel from the maclib. You can either create a new maclib and download that or create individual

files with filetype BASHELP. For information on the CMS commands MACLIB and MOVEFILE, see *VM/SP Command and Macro Reference.*

HELP panels containing syntax information corresponding to that in *IBM BASIC Language Reference* are identified with filenames of the statements and commands. (For example, the OPEN statement file is BASHOPEN BASHELP.)

If you don't have the space available to download the HELP panels, you can access them for your BASIC session by linking to the appropriate host system disk.

Before you download the modules and/or HELP panels, be sure you have enough space. See *IBM BASIC/VM Installation and Customization* for storage requirements (or look at the space on the host disk where BASIC resides).

Invoking BASIC

If BASIC is not on a minidisk you normally access, you must first make it available.

Example

```
CP LINK vm/pc-id ttt aaa RR read-password
ACCESS aaa filemode1
```

See Figure 11 on page 75 for definition of variables.

If BASIC is stored on your A-disk, or another disk you can access, you can omit these commands.

If the BASIC HELP panels are not on the same VM/PC minidisk as the Processor, you must also link to the minidisk on which they reside before using the HELP facility.

Example

```
CP LINK vm/pc-id ttt aaa RR read-password
ACCESS aaa filemode1
```

If you have not downloaded the BASIC HELP panels and you want to use the HELP facility, you must access the appropriate disk on the host system:

```
CP LINK host-id hhh bbb RR read-password REMOTE
ACCESS bbb filemode2
```

The host system HELP panels are then available for your use.

If you must issue the LINK and ACCESS commands each time you log on to VM/PC, you can put them into your PROFILE EXEC, which issues them for you.

Invoking the BASIC Environment

After you've made the IBM BASIC Processor available to VM/PC, you can invoke it by entering:

```
BASIC
```

and the BASIC commands, immediate statements, and source statements are available for your use, with the restrictions noted in "VM/PC Processing Restrictions on IBM BASIC" on page 72. For more information on invocation and using BASIC inside the BASIC environment, see "The BASIC Environment under CMS" on page 4.

You can compile programs with the COMPILE command. You can execute programs, either directly from the source or from compiled text, with the RUN command.

Compiling and Executing BASIC Programs Outside the BASIC Environment

You can compile and execute BASIC programs outside the BASIC environment. However, this is usually not the most efficient way to use the Processor. For tips on saving processing time, see "BASIC Programming Tips" on page 78.

After you have entered a source program and saved it, you can tell BASIC to compile the program by entering:

```
BASIC SAMPL
```

Where SAMPL is the name of your saved program. (Its filetype is BASIC.) This compiles the program, using the default compiler options in force for your organization. For more information, see "Using BASIC as a Compiler under CMS" on page 22.

If you want, you can override one or more default compiler options, as follows:

```
BASIC SAMPL (options
```

which compiles SAMPL, using the Processor options you choose.

After you have compiled a BASIC program, you can load and execute the TEXT file without entering the BASIC environment and without performing any CMS operations on the text file. Instead, you can use the BASICRUN routine, as follows:

```
BASICRUN SAMPL
```

which executes the program you compiled using the BASIC command.

BASIC Programming Tips

Use BASIC inside the BASIC environment. This saves processing time, because in this way you invoke the IBM BASIC Processor only once and run under its control to create, compile, and run your programs. Using BASIC in batch mode takes extra processing time, because there is a separate invocation of BASIC for each program you want to process.

You can improve processing time if you specify IBM BASIC Processor options that do not request printed listings: NOSOURCE, NOLIST, NOMAP, and NOXREF.

CMS System Services Reference

This chapter contains reference information for the CMS and CP commands used most often by the BASIC programmer. Each command description gives the format, required operands, and options needed for BASIC programs. The majority of the commands have options in addition to those discussed here.

Note: For a more complete description of the commands that follow, see *IBM Virtual Machine/System Product CMS Command and Macro Reference* and *IBM Virtual Machine/System Product CP Command Reference for General Users.*

Format Notation

How job control statement and command formats are to be interpreted in this publication is discussed in the following paragraphs.

- **Truncations and abbreviations of commands**: Where truncation or abbreviation of a command name is allowed, the shortest acceptable version is in uppercase letters. However, the CMS commands can be entered with any combination of uppercase or lowercase letters.

- **Uppercase letters, words, and numbers**: Must be coded in the statement exactly as shown.

- **Lowercase letters and words**: Represent variables, for which programmer-supplied information is substituted.

- **Symbols** in the following list must be coded exactly as shown:

apostrophe	'
asterisk	*
comma	,
equal sign	=
parentheses	( )
period	.
slash	/

- **Logical OR symbols (|)**: Represent alternative choices from which one choice can be made.

- **Braces ({ })**: Represent alternative related items (such as alternative choices) from which one choice *must* be made.

- **Square brackets ([ ])**: Group optional related items (such as alternative choices) from which one choice can optionally be made.

- **Ellipses (...)**: Specify that the preceding syntactical unit can optionally be repeated. (A *syntactical unit* is a single syntactical item, or a group of syntactical items, enclosed in square brackets.)

- **Blanks**: Are used to improve the readability of the control statement definitions. They must not appear in an operand field, unless a definition explicitly states otherwise.

AMSERV Command

The AMSERV command invokes Access Method Services to:

- Define VSAM catalogs, data spaces, or clusters
- Alter, list, copy, delete, export, or import VSAM catalogs and data sets

Format

COMMAND	OPERANDS
AMserv	fn fn2 (options...)

where:

fn1 Specifies the filename of a CMS file with a filetype of AMSERV that contains the Access Method Services control statements to be executed. CMS searches all your accessed disks, using the standard search order, to locate the file.

fn2 Specifies the filename of the CMS file that is to contain the Access Method Services listing; the filetype is always LISTING.

If fn2 is not specified, the LISTING file has the same name as the AMSERV input file (fn1).

options The following options may be specified:

PRINT Spools output listing to the virtual printer.

TAPIN Specifies that tape input is on the tape at address 18n or TAPn, where n is 1, 2, 3, or 4.

TAPOUT Specifies that tape output should be written to the tape at address 18n or TAPn, where n is 1, 2, 3, or 4.

For documentation on using the AMSERV command, see "Using VSAM Files under CMS" on page 65.

CP Command

The CP command is used to send commands to the VM/SP control program environment without leaving the CMS environment.

Format

COMMAND	OPERANDS
CP	commandline

where:

commandline Is any CP command valid for your CP command privilege class. If this field is omitted, you are placed in the CP environment and may enter CP commands without preceding each command with CP. To return to CMS, issue the CP command BEGIN.

When you are communicating directly with CMS (for example, in the CMS subset as described in "Using the SYSTEM Command to Execute a Single CMS Subset Command" on page 18), you can execute CP commands without preceding each command with CP. But, if you are using the SYSTEM command or CALL SYSTEM statement to execute a single CP command, the CP is required (see "Using the SYSTEM Command" on page 18 and "Using CALL SYSTEM Statements in BASIC Programs" on page 20).

Frequently Used CP Commands

The CP commands listed below are those most frequently used by the BASIC programmer. These commands are fully described in *VM/SP CP Command Reference for General Users*.

CP Command	Function
Begin	Continue or resume execution of the virtual machine at either a specific storage location or at the address in the current PSW.
LINK	Provide access to a specific DASD by a virtual machine.
LOGoff	Disable access to CP. Terminate the session.
Logon	Provide access to CP. Initiate the session.
Query	Request information about machine configuration and system status.
SET	Control various functions within the virtual machine.

SPool	**Alter spooling control options; direct a file to another virtual machine or to a remote location.**
TERMinal	**Define or redefine the input and attention handling characteristics of your virtual console.**

DLBL Command

In CMS, the DLBL command defines and identifies VSAM catalogs, clusters, and data spaces. DLBL also identifies VSAM files used for program input/output, and identifies input/output files for AMSERV.

Format

COMMAND	OPERANDS
DLBL	ddname{DUMMY | filemode} DSN fileid (options...)

For a description of all the options that can be specified and for further information on the use of this command, see *IBM Virtual Machine Facility/SP: CMS Command and Macro Reference.*

where:

ddname Specifies a 1- to 7-character filename. This is the filename used in your BASIC program or command.

DUMMY Indicates that no real input/output is to be performed. A read results in an end-of-file condition, and a write results in a successful return code.

filemode Specifies a valid CMS disk mode letter and, optionally, a filemode number.

DSN Indicates non-CMS files.

fileid Specifies a VSE fileid.

options The following options may be specified:

VSAM indicates a VSAM file. This option must be specified for VSAM functions, except when the EXTENT, CAT, or BUFSP option is specified.

PERM indicates that this DLBL definition can be cleared only with an explicit CLEAR request.

For documentation on using the DLBL command, see "Using VSAM Files under CMS" on page 65.

ERASE Command

The ERASE command deletes CMS files from any CMS disk directory that you are linked to in read/write mode.

Format

COMMAND	OPERANDS
ERASE	fn ft fm (TYPE)

where:

fn Specifies the filename of the file(s) to be erased. An asterisk coded in this position indicates that all filenames are to be used.

ft Specifies the filetype of the file(s) to be erased. An asterisk coded in this position indicates that all filetypes are to be used.

fm Specifies the filemode of the file(s) to be erased. An asterisk coded in this position indicates that all filemodes are to be used to which you have read/write access.

TYPE Displays the file identifier of each file erased.

The ERASE command is useful when you want to delete only one of several files having the same filename; see "Using the PURGE Command" on page 11.

EXEC Command

The EXEC command executes one or more CMS commands or EXEC control statements contained in a specified EXEC file.

Format

COMMAND	OPERANDS
EXec	fn args...

where:

[EXec] Indicates that the EXEC command may be omitted if executing the EXEC procedure from the CMS command environment and the SET IMPEX OFF command has not been issued.

However, if the EXEC command is issued from the BASIC environment (through a SYSTEM command or CALL SYSTEM statement), the EXEC command *must* be specified.

fn Indicates the filename of a file containing one or more CMS commands and/or EXEC control statements to be executed. The filetype of the file must be EXEC. The file may include CMS commands (not BASIC commands) which cause a BASIC compilation and/or execution.

args Indicates any arguments you may pass to the EXEC.

FILEDEF Command

The FILEDEF command overrides default file definitions made by BASIC.

Format

COMMAND	OPERANDS
Filedef	ddname device | DISK fn ft fm (PERM)

where:

ddname The data definition name provides the link between your BASIC input and output statements and commands, and the file that you want to process or create.

device Specifies the input/output device on which the file will reside. The following devices may be specified:

TERMINAL
Specifies that your file is to be read or written at the terminal.

Terminal lines are accepted or displayed by CMS's terminal screen handling method. This means that if you are in the BASIC environment (where BASIC is controlling a scrolling screen), your BASIC screen is temporarily replaced by a CMS screen.

PRINTER [(RECFM option[])]
Indicates that your file is to be written on a virtual printer. Files defined as PRINTER may be used only for output.

For details on RECFM options, see the FILEDEF command in *IBM Virtual Machine/System Product: CMS Command and Macro Reference.*

PUNCH
Indicates that your file is to be punched by a virtual card punch. Files defined as PUNCH may be used only for output. See "Writing to the Punch" on page 62.

READER
Indicates that you want to read a file from a virtual card reader. Files defined as READER may be used only for input. See "Reading from the Reader" on page 63.

DUMMY
Indicates that no real input or output operation is to be performed. For input, this causes end-of-file to be returned.

TAPn

Indicates that you want to use an existing tape file or create a new one. The variable n represents the tape unit number and can be in the range from 1 through 4. The numbers correspond to tape units attached to your virtual machine addresses 181 through 184. See "Accessing Tapes" on page 59.

When you define a tape file with the FILEDEF command, you can also specify the type of label processing to be done for the file. To do this, use a second operand after the word TAPn. This allows access to multifile tapes. For more information on how to use these operands, see "Using the FILEDEF TAPn Operands" below, and "Accessing Tapes" on page 59.

DISK Indicates that you want to use a disk file. See "Accessing CMS Disk Files" on page 55.

fn ft [fm] Is the CMS file identifier: filename, filetype, and optional filemode.

PERM Specifies that you do not want the FILEDEF cleared when BASIC terminates.

Using the FILEDEF TAPn Operands

You define a tape file with the FILEDEF command by entering:

Format

Command	Operands
Filedef	ddname TAPn LABOFF | BLP n | SL n VOLID volid | NL n | NSL filename (PERM)

where:

LABOFF Indicates that there is no CMS tape label processing for this tape file. LABOFF is the default. The tape is not positioned if you specify this operand. (*Note*: The OPEN statement will position the tape to either the beginning or end of the first file on the tape as determined by the BEGIN or END/APEND positioning clause.)

BLP Indicates that the system is to bypass label processing but that the tape is to be positioned before the file is processed.

SL Indicates that you are using IBM standard labels.

NL Indicates that your tape has no IBM standard labels. (Do not use this operand if your tape has a VOL1 label. A file on it will not be opened.)

NSL Indicates that you are using nonstandard labels.

For the operands BLP, SL, and NL:

n Indicates the position of the file on a multifile volume. When you don't specify n, the default is 1.

For SL files:

volid Is a 1 to 6 character volume serial number to be verified by reading the VOL1 label on the tape. (For more information, see *CMS Users Guide*.) VOLID is only valid for SL tape files. If you don't specify VOLID, the volume label on the tape is not checked.

For the NSL operand:

filename Is the filename of a file that contains a routine for processing nonstandard labels. This routine is required.

You can not use the FILEDEF options SUL, LEAVE, NOEOV, and DISP MOD.

For more information on how to use the FILEDEF command, see *IBM Virtual Machine/System Product: CMS Command and Macro Reference.*

GENMOD Command

The GENMOD command is used to generate a nonrelocatable MODULE file on a CMS disk.

Format

COMMAND	OPERANDS
Genmod	filename MODULE

where:

filename Is the name of the MODULE file being created. If filename is not specified, the file created has a filename equal to the first entry point in the module.

The GENMOD command is used following the LOAD command and possibly the INCLUDE command. See "Creating and Running Self-Contained Application Load Modules" on page 26.

GLOBAL Command

The GLOBAL command specifies the text libraries to be searched in resolving external references.

Format

COMMAND	OPERANDS
GLobal	TXTLIB libname ...

where:

TXTLIB Precedes the specification of text libraries to be searched for missing non-BASIC subprograms when the LOAD or INCLUDE command is issued, or when a dynamic load occurs.

libname

Specifies the filename(s) of the text libraries to be searched. As many as eight libraries may be specified. For non-BASIC subprograms, filetypes must be TXTLIB. The libraries are searched in the order in which they are named. If no library names are specified, the command cancels the effect of any previous GLOBAL command.

Check with your system administrator for the text libraries available.

HX Command

The HX command is an immediate command that is used to halt the execution of a program executing under CMS. See "Halting Execution with the HX Command" on page 45.

Format

COMMAND	OPERANDS
HX	

The HX command can be used to stop BASIC during the compilation of a BASIC program outside the BASIC environment.

Normally, the HX command should *not* be used while in CMS subset mode, nor should it be used to stop execution of a command initiated by the SYSTEM command or CALL SYSTEM statement, unless the user wants to terminate the execution of BASIC.

When a terminal session is under the control of the BASIC environment, the HX command is invalid.

The HX command works only when BASIC has been invoked as a compiler or when you are in CMS SUBSET mode after entering a SYSTEM command.

INCLUDE Command

The INCLUDE command reads one or more TEXT files (containing relocatable object code) from disk and loads them into virtual storage, establishing the proper linkages between the files. In order for the INCLUDE command to produce the desired results, a LOAD command must have been previously issued.

Format

COMMAND	OPERANDS
INclude	fn ... (AUTO | NOAUTO | RESET name)

where:

fn Is the name of a file to be loaded into storage. Each file must have a filetype of TEXT and consist of relocatable object code.

AUTO Searches your accessed virtual disks for TEXT file names to resolve undefined references. It is the default option.

NOAUTO Suppresses automatic resolution of undefined references as TEXT file names.

RESET name Specifies the entry point for the module.

The INCLUDE and CMS LOAD commands are used preceding a GENMOD command to create a load module from one or more object program text modules (plus any needed IBM BASIC Library text modules) when loading and executing your program under CMS. See "Creating and Running Self-Contained Application Load Modules" on page 26. The RESET option may be needed to make sure that BLIOCRT is the entry point to the module.

LOAD Command

The LOAD command reads one or more CMS TEXT files from disk and loads them into virtual storage, establishing the linkage between the files for execution.

Note: This command is a CMS command that differs from BASIC's LOAD command.

Format

COMMAND	OPERANDS
LOAD	fn ... (AUTO | NOAUTO | RESET name)

where:

fn Specifies the name of a file to be loaded into storage. Each file must have a filetype of TEXT and consist of relocatable object code.

AUTO Searches your accessed virtual disks for TEXT file names to resolve undefined references. It is the default option.

NOAUTO Suppresses automatic resolution of undefined references as TEXT file names.

RESET name Specifies the entry point for the module.

The INCLUDE and LOAD commands are used preceding a GENMOD command to create a load module from one or more object program text modules (plus any needed IBM BASIC Library text modules) when loading and executing your program under CMS. See "Creating and Running Self-Contained Application Load Modules" on page 26. The RESET option may be needed to make sure BLIOCRT is the entry point for the module. For example:

```
LOAD files (RESET BLIOCRT)
```

LOGOFF command

The CP LOGOFF command ends your VM terminal session.

FORMAT

COMMAND	OPERANDS
LOGOFF	

LOGON Command

The CP LOGON Command starts a VM terminal session.

Format

COMMAND	OPERANDS
LOGON	userid

where:

userid Specifies your user identification.

After you've typed in the LOGON command and your userid and then pressed ENTER, VM responds by asking you for your password:

```
ENTER PASSWORD:
```

You then enter your password, which may not show on the screen, depending on your terminal.

PRINT Command

The PRINT command prints a CMS file on the virtual printer.

Note: This command is a CMS command that differs from BASIC's PRINT command.

Format

COMMAND	OPERANDS
PRint	fn ft fm (options...)

where:

fn Specifies the filename of the file to be printed.

ft Specifies the filetype of the file to be printed.

fm Specifies the filemode of the file to be printed, indicating the disk on which the file resides.

If *fm* is specified as an asterisk (*), all accessed disks are searched for the specified file.

options The following options may be specified:

CC Indicates that the first byte in each record is a carriage control character. This option is assumed if the filetype is LISTING. It should be specified for display-format files written with BASIC's DEVICE PRINTER option.

TRC Indicates that the first byte in each record after the carriage control character is a table reference (or font control) character. This character determines which translate table an IBM 3800 printing subsystem is to use to print that record. This option must be specified for display-format files written with the BASIC DEVICE 3800 option.

The PRINT command can be used to print display-format files created by BASIC programs (filetype BASDATA), log files (filetype BASLOG), or BASIC programs (filetype BASIC).

PUNCH Command

The PUNCH command punches a CMS disk file to your virtual card punch.

Format

COMMAND	OPERANDS
PUNCH	fn ft fm (options...)

where:

fn Specifies the filename of the file to be punched.

ft Specifies the filetype of the file to be punched.

fm Specifies the filemode of the file to be punched.

options The following options may be specified:

HEADER Inserts a control card in front of the punched output. This is the default.

NOHEADER Does not punch a header control card. This option is necessary for jobs submitted to CMS batch.

The PUNCH command is used to send jobs to the CMS batch virtual machine by spooling your virtual card punch to the virtual reader of the batch virtual machine.

RENAME Command

The RENAME command changes the file identification of one or more CMS files on a read/write disk.

Note: This command is a CMS command that differs from BASIC's RENAME command.

Format

COMMAND	OPERANDS
Rename	fileid1 fileid2

where:

fileid1 Specifies the file identifier of the original CMS file to be renamed. All components of the fileid must be coded with either a name or an asterisk (*).

fileid2 Specifies the new file identifier. All components of the fileid must be coded with either a name or an equal sign (=). The equal sign indicates that the corresponding component of the original fileid is to be left unchanged.

The RENAME command is useful to keep from overwriting successive LISTING and TEXT files.

START Command

The START command begins execution of a previously loaded program.

Format

COMMAND	OPERANDS
START	entry

where:

entry Passes control to the control section name or entry point name at execution time. Entry may be a filename only if the filename is identical to a control section name or an entry point name.

If entry is not specified, control is passed to a default entry point. See *IBM Virtual Machine/System Product: CMS Command and Macro Reference* for details.

You can use the LOAD and INCLUDE commands to load a BASIC object program and then use the START command to execute it.

XEDIT Command

The XEDIT command invokes the VM/System Product editor to create, modify, and manipulate CMS disk files.

Format

COMMAND	OPERANDS
XEDIT	fn ft fm (options...)

Additional information on the use of this command and the XEDIT subcommands can be found in *IBM VM/SP: System Product Editor User's Guide* and *IBM VM/SP: System Product Editor Command and Macro Reference.*

where:

fn Specifies the filename of the file to be created or edited. If the file specified is a new file, issue the INPUT subcommand, and all data lines entered become input to the new file. If the file specified exists, you issue the XEDIT subcommands to change the file.

ft Specifies the filetype of the file to be created or edited. Filetypes useful with BASIC are:

```
BASIC   - for BASIC source files
AMSERV  - VSAM control commands
EXEC    - for EXEC procedures
BASDATA - for BASIC data files
LISTING - for BASIC listing files
```

fm Specifies the filemode of the file to be edited, indicating the disk on which the file resides. The editor determines the filemode of the edited file as follows:

If a specific *fm* is specified, that filemode is used.

If *fm* is specified as an asterisk (*), all accessed disks are searched for the specified file.

options The options that may be specified are fully described in *IBM Virtual Machine/System Product: CMS Command and Macro Reference.*

Instead of the BASIC editing facilities, you can use the XEDIT command and its subcommands; in that case, however, there is no syntax checking of BASIC statements, and BASIC editing commands are not available.

XEDIT (or another non-BASIC editor) can be used to create files not containing BASIC programs, such as AMSERV files or CMS batch control files.

Control is returned to the CMS environment by issuing the XEDIT subcommand FILE, QUIT, or QQUIT.

The XEDIT command places the terminal session in XEDIT mode and makes XEDIT subcommands available for file creation and alteration.

These subcommands and additional options for the XEDIT command are fully described in *IBM Virtual Machine/System Product: CMS Command and Macro Reference*.

Appendix A. Minimum List of IBM BASIC/VM Library Modules

The following library modules are needed in a minimum library in order to execute a minimum program consisting of an END statement.

BLIDIRA	BLIOCOP	BLIOC27	BLIOC63
BLIDIRC	BLIOCRT	BLIOC28	BLIOC65
BLIDIRI	BLIOCOO	BLIOC29	BLIOC72
BLIDIRL			
BLIDIRM	BLIOCO1	BLIOC36	BLIOC74
BLIDIRN	BLIOCO3	BLIOC37	BLIOC86
BLIDIRO	BLIOCO4	BLIOC51	BLISCTL
BLIDIRS	BLIOC10	BLIOC52	BLISEMC
BLIDIRX	BLIOC11	BLIOC53	BLISEMT
BLIDIRZ	BLIOC12	BLIOC54	BLISERT
BLIDIR2	BLIOC13	BLIOC55	BLISINT
BLIICL	BLIOC14	BLIOC56	BLISSYS
BLIICNV	BLIOC17	BLIOC57	BLISTRM
BLIIEND	BLIOC20	BLIOC58	
BLIIIOL	BLIOC21	BLIOC59	
BLIIPR	BLIOC23	BLIOC60	
BLIIRD	BLIOC24	BLIOC61	
BLIIVAL	BLIOC26	BLIOC62	

These modules are also those needed to execute a simple PRINT statement:

```
PRINT "ABC",A,B%
```

Because you may need additional modules for your program, the entire list of modules called by the IBM BASIC Library appears in Appendix B, "IBM BASIC/VM Library Module Cross-Reference" on page 105.

Appendix B. IBM BASIC/VM Library Module Cross-Reference

The following list shows the modules called by each IBM BASIC/VM Library module.

In most cases, the called modules will not automatically be included in a load module with the calling module.

However, some called modules are automatically included in the load module when the calling module is included; these modules are identified with an exclamation point (!) following the module name. These modules will be automatically included only if the AUTO option is specified on the CMS INCLUDE command or the AUTO option is the default.

Modules identified with a less-than symbol (<) are entry points in GDDM or in other languages. See "Calling Programs Written in Other Languages" on page 35.

For a list of the minimum library modules you would need to run a program consisting solely of an END statement, refer to Appendix A, "Minimum List of IBM BASIC/VM Library Modules" on page 103.

If you have compiled your program using Release 1 of IBM BASIC, see Appendix C, "IBM BASIC/VM Release 1 Library" on page 111.

CALLING MODULE	CALLED MODULE				
BLIAASN	BLIMCDI	BLIMCID	BLISERT	BLISTMM	
	BLIMCIR	BLIMCRD	BLIMCRI	BLIMCDR	
BLIACAT	BLISERT	BLISTMM			
BLIACIX	BLISERT	BLISSTR	BLISTMM		
BLIACSC	BLISERT	BLISTMM			
BLIACSS	BLISERT	BLISSTR	BLISTMM		
BLIADAA	BLIAASN	BLIMDAS	BLISERT	BLISTMM	
BLIADCN	BLIARDM	BLISERT			
BLIADDO	BLIAASN	BLIMDAS	BLIMDM	BLISERT	
BLIADID	BLISERT	BLIADRM			
BLIADIN	BLIAASN	BLIMDAS	BLIMDC	BLIMDD	BLIMDM
	BLIMDST	BLISERT	BLISTMM	BLIMCDR	BLIMCRD
BLIADIX	BLIMDC	BLISERT	BLISTMM		
BLIADML	BLIAASN	BLIMDAS	BLIMDM	BLISERT	BLISTMM
BLIADPS	BLIMDAS	BLIMDM	BLISERT		
BLIADSS	BLIMDC	BLISERT	BLISTMM		
BLIADZR	BLIARDM	BLISERT			
BLIAICN	BLIARDM	BLISERT			
BLIAIID	BLISERT	BLIADRM			
BLIAIIX	BLISERT	BLISTMM			
BLIAIPS	BLISERT				
BLIAISS	BLISERT	BLISTMM			
BLIAIZR	BLIARDM	BLISERT			
BLIAMFN	BLISERT	BLISTMM	BLIMCDI	BLIMCDR	
	BLIMCID	BLIMCIR	BLIMCRD	BLIMCRI	
BLIANUL	BLIARDM	BLISERT			
BLIARCN	BLIARDM	BLISERT			
BLIARDM	BLIAASN	BLISERT			
BLIARID	BLIARDM	BLISERT			
BLIARIX	BLISERT	BLISTMM			
BLIARPS	BLISERT	BLISTMM			
BLIARSS	BLISERT	BLISTMM			
BLIARZR	BLIARDM	BLISERT			
BLIASAM	BLIAASN	BLIMCID	BLIMDM	BLISERT	BLIMCIR
	BLIMCRI				
BLIASRC	BLISERT				
BLIATRN	BLISERT	BLISTMM			
BLIAUSZ	BLISERT				
BLICCAS	BLISERT	BLISTMM			
BLICCHR	BLISERT	BLISTMM			
BLICNST	BLISERT	BLISTMM	BLIMCID	BLIMCRD	
BLICPAD	BLISERT	BLISTMM			
BLICPRM	BLISERT	BLISTMM			
BLICRPT	BLISERT	BLISTMM			
BLICSRE	BLISERT	BLISTMM			
BLICTRM	BLISERT	BLISTMM			
BLIDIRA					
BLIDIRC					
BLIDIRI					
BLIDIRL					
BLIDIRM					
BLIDIRN					
BLIDIRO					
BLIDIRS					
BLIDIRX					
BLIDIRZ	BLIDIRA!	BLIDIRC!	BLIDIRI!	BLIDIRL!	BLIDIRM!
	BLIDIRN!	BLIDIRO!	BLIDIRS!	BLIDIRX!	
BLIDIR2					
BLIICL	BLIIVAL	BLIOC17	BLIOC24	BLISERT	
BLIICNV	BLIIVAL	BLISERT	BLISTRM		

BLIIDEL	BLIIVAL	BLIOC15	BLISERT		
BLIIEND	BLIICNV	BLIIIOL	BLIIVAL	BLIOC12	BLIOC13
	BLIOC36	BLIOC37	BLISERT	BLISTRM	BLIOC23
	BLIOC24				
BLIIFNC	BLIIVAL	BLIOC21	BLISERT		
BLIIGPT	BLIIVAL	BLISERT			
BLIIIN	BLIIRD	BLIIVAL	BLISERT	BLIIEND	BLIOC24
BLIIINT	BLIIIOL	BLISERT	BLISTRM	BLIOC24	
BLIIIOL	BLIICNV	BLIIRD	BLIIVAL	BLIOC12	BLIOC20
	BLIOC23	BLIOC24	BLIOC29	BLISERT	BLISTRM
	BLIIEND	BLIMCDR	BLIMCIR	BLIMCRD	BLIMCRI
BLIIMAR	BLIIVAL	BLISERT			
BLIIOPN	BLIICL	BLIIVAL	BLIOC10	BLIOC23	BLIOC24
	BLISERT				
BLIIPOS	BLIIVAL	BLIOC16	BLISTRM		
BLIIPR	BLIIVAL	BLIOC29	BLISERT	BLISTRM	BLIIEND
BLIIRD	BLIIVAL	BLIOC11	BLISERT		
BLIIRER	BLIIVAL	BLISERT			
BLIIREW	BLIIRD	BLIIVAL	BLISERT	BLIIEND	
BLIIRIF	BLIIVAL	BLISERT			
BLIIRS	BLIIVAL	BLIOC16	BLISERT		
BLIISCR	BLIIVAL	BLIOC18	BLISERT		
BLIITAB	BLIIVAL	BLISERT			
BLIIVAL	BLIOC14	BLIOC21	BLISERT	BLIOC12	
	BLIOC23	BLIOC24			
BLIIWRT	BLIIVAL	BLISERT			
BLIMCDI	BLISERT				
BLIMCDR	BLISERT				
BLIMCID					
BLIMCIR					
BLIMCRD	BLIMDD	BLIMDM			
BLIMCRI	BLISERT				
BLIMDAS	BLISERT				
BLIMDC	BLISERT				
BLIMDD	BLISERT				
BLIMDDP	BLIMDM	BLIMDIP	BLINEXP	BLINLNE	BLISERT
BLIMDIP	BLIMDD	BLISERT			
BLIMDM	BLISERT				
BLIMDST	BLISERT				
BLIMRPI	BLISERT				
BLIMRPR	BLINEXP	BLINLNE	BLISERT	BLIMCRI	
BLINDAA	BLIMCDR	BLIMCRD	BLINSQR	BLISERT	
BLINDAB					
BLINDAT	BLIMDD	BLIMCDR	BLIMCID	BLIMCRD	BLISERT
BLINDBL	BLIMCDR	BLIMCIR	BLISERT		
BLINDCS					
BLINDDG	BLIMDAS	BLIMDM	BLISERT		
BLINDEC	BLIMCDR	BLIMCRD			
BLINDFI					
BLINDHH	BLIMCDR	BLIMCRD	BLINEXP	BLISERT	
BLINDII					
BLINDRR	BLIMDM				
BLINDRT	BLIMCDR	BLIMCRD			
BLINDSG					
BLINDSS	BLIMCDR	BLIMCRD	BLISERT		
BLINDTC	BLIMCDR	BLIMCRD	BLISERT		
BLINEXP	BLIMCDR	BLIMCRD	BLISERT	BLIMDM	
BLINFIX	BLIMCDI	BLIMCRI	BLISERT		
BLINIAB	BLISERT				
BLINIFI					

BLINIII					
BLINISG					
BLINLEN					
BLINLG2	BLIMDM	BLINLNE	BLIMDAS	BLIMCDR	BLIMCID
	BLIMCRD				
BLINLNE	BLIMCDR	BLIMCRD	BLISERT	BLIMDAS	BLIMDML
	BLIMCID				
BLINLOG	BLINLNE	BLIMDAS	BLIMCDR	BLIMCID	BLIMCRD
BLINMRM	BLIMDD	BLISERT	BLIMDM	BLIMDAS	BLIMCID
BLINMXN	BLINDFI	BLINDII	BLINRFI	BLINRII	
BLINORD	BLISERT				
BLINPOS					
BLINRAB					
BLINRFI					
BLINRII					
BLINRSG					
BLINSNG	BLIMCDR	BLIMCIR	BLISERT		
BLINSQR	BLIMCDR	BLIMCRD	BLISERT		
BLINTNH	BLIMCDR	BLIMCRD	BLINEXP		
BLINVAL	BLIMCDR	BLISERT			
BLIOCOP					
BLIOCRT	BLIOC00	BLIOC03	BLIDIRZ	BLIFLGC	BLIFLGI
	BLISERT				
BLIOC00	BLIOC01	BLIOC20	BLIOC23	BLIOC26	BLIOC27
	BLIOC28	BLIOCOP	BLIOC10	BLIOC11	BLIOC17
	BLIOC03	BLIOC12	BLIOC21	BLISERT	
BLIOC01	BLIOC24	BLIOC21			
BLIOC02					
BLIOC03	BLIOC23	BLIOC24	BLIOC10	BLIOC12	BLIOC17
	BLIOC19				
BLIOC04	BLIOC24				
BLIOC05	BLIOC03	BLIOC04			
BLIOC06	BLIOC93!				
BLIOC07					
BLIOC10	BLIOC23	BLIOC24	BLIOC51	BLIOC52	BLIOC53
	BLIOC72	BLIOC74			
BLIOC11	BLIOC57	BLIOC58	BLIOC59		
BLIOC12	BLIOC60	BLIOC61	BLIOC62		
BLIOC13	BLIOC63	BLIOC65			
BLIOC14	BLIOC60	BLIOC61	BLIOC62		
BLIOC15	BLIOC69	BLIOC71			
BLIOC16	BLIOC66	BLIOC67	BLIOC68		
BLIOC17	BLIOC24	BLIOC54	BLIOC55	BLIOC56	
BLIOC18	BLIOC79	BLIOC81			
BLIOC19	BLIOC74	BLIOC76			
BLIOC20	BLIOC12	BLIOC21	BLIOC86		
BLIOC21	BLIOC12	BLIOC86	BLIOC23		
BLIOC22	BLIOC24				
BLIOC23					
BLIOC24					
BLIOC25	BLIOC23				
BLIOC26					
BLIOC27					
BLIOC28	BLIOC01	BLIOC20	BLIOC21	BLIOC23	BLIOC86
BLIOC29	BLIOC21	BLIOC86			
BLIOC30	BLIOC21				
BLIOC33	BLIOC23				
BLIOC34	BLIOC10	BLIOC14	BLIOC17	BLIOC21	BLIOC23
	BLIOC24	BLIOC74			
BLIOC35					
BLIOC36	BLIOC21	BLIOC23	BLIOC86		
BLIOC37	BLIOC21	BLIOC86			

BLIOC51	BLIOC23	BLIOC24			
BLIOC52	BLIOC23	BLIOC24			
BLIOC53	BLIOC23	BLIOC24			
BLIOC54					
BLIOC55					
BLIOC56	BLIOC24				
BLIOC57					
BLIOC58					
BLIOC59					
BLIOC60					
BLIOC61					
BLIOC62					
BLIOC63					
BLIOC65					
BLIOC66					
BLIOC67					
BLIOC68					
BLIOC69					
BLIOC71					
BLIOC72					
BLIOC74					
BLIOC76					
BLIOC79					
BLIOC81					
BLIOC86	BLIOC23	BLIOC24			
BLIOC93					
BLISBAS	BLIOC29				
BLISBRK					
BLISCLK	BLIMCDI	BLIOC26	BLIOC27	BLISERT	BLISTMM
BLISCRC	BLISERT				
BLISCSY	BLIOC30	BLISERT			
BLISCTL	BLIOC23	BLIOC24	BLISERT	BLISTRM	
BLISEMC					
BLISEMT					
BLISERT	BLIIEND	BLIIPR	BLIOC21	BLIOC24	BLIOC29
	BLISEMC	BLISEMT	BLISTRM		
BLISGDD	ADMASP <	BLIMCDI	BLIOC03		
	BLIOC10	BLIOC12	BLIOC17	BLIOC19	BLIOC23
	BLIOC24	BLISERT			
BLISILC	BLIMCDR	BLIOC03	BLIOC04	BLIOC23	BLIOC24
	BLIOC28	BLISCOT<	BLISERT	BLISFOT<	BLISPLT<
	ILBOSTBO<	PLICALLA<	PLIMAIN<	VSCOM#<	
BLISINT	BLIOC01	BLIOC23	BLIOC24	BLISCTL	BLISEMC
	BLISEMT	BLISERT	BLISSYS	BLISTRM	
BLISPAU	BLIOC21	BLISERT			
BLISRND	BLIMCDI	BLIMCID	BLIMDD	BLIMDM	BLIOC27
BLISSCR	BLISERT				
BLISSQF					
BLISSQL	BLISSQF!	BLISSQM!	BLISSQO!	BLISSQP!	BLISSQS!
	BLISSQT!	BLISSQU!	BLISSQV!		
BLISSQM	BLISERT				
BLISSQO					
BLISSQP					
BLISSQS					
BLISSQT					
BLISSQU					
BLISSQV	BLIMCRD				
BLISSTR	BLISERT	BLISTMM			
BLISSYS					
BLISTMM	BLIOC23	BLIOC24	BLISTRM		
BLISTRC	BLIIEND	BLIIIOL	BLIIPR	BLIOC23	BLIOC24
	BLISERT	BLISTRM			
BLISTRM	BLIICL	BLIIEND	BLIIPR	BLIOC01	BLIOC03
	BLIOC04	BLIOC10	BLIOC17	BLIOC21	BLIOC23
	BLIOC24	BLISERT			

Appendix C. IBM BASIC/VM Release 1 Library

If you have compiled your program under Release 1 of IBM BASIC and are linking it with the Release 2 Library, you should be aware of the following changes.

If the list of library modules generated by the MAP option using Release 1 of IBM BASIC refers to any of the following modules, you should use the specified Release 2 library modules instead.

Release 1	Release 2
BLIADTR	BLIATRN
BLIAITR	BLIATRN
BLIMDPD	BLIMDDP
BLIMDPI	BLIMDIP
BLINDHT	BLINTNH
BLINDLT	BLINLOG
BLINDL2	BLINLG2
BLINDVL	BLINVAL
BLISCOB	BLISILC
BLISFOR	BLISILC
BLISPL	BLISILC

Appendix D. Shared Segment Starter

The following program can be used to start a compiled BASIC program which has been placed in a discontiguous shared segment.

```
BEGDCSS  TITLE 'BASIC SHARED SEGMENT STARTER'
*****************************************************************
*
* This program will find the DCSS containing a linked BASIC
* program.  To use this program do the following:
*   EDIT this file changing 'BASPGM' to the name of the DCSS
*     (see label 'DCSSNAME' in this file)
*   GLOBAL MACLIB DMSSP CMSLIB
*   ASSEMBLE BEGDCSS
*   LOAD BEGDCSS (CLEAR
*   GENMOD NAME
*      (where 'NAME' is the desired name of starter program)
*
* To build the DCSS (you must have the appropriate privileges; see
* your System Administrator to setup DCSS; you must have storage
* higher than DCSS Base + Size of module):
*   LOAD BLIOCRT (ORIGIN DCSSBASE CLEAR
*   INCLUDE PROG
*   (include any desired compiled BASIC subprograms)
*   (include Library parts as needed; omit if Library is in
*     separate DCSS; if Library parts are included you must
*     include BLIDIRZ)
*   SETKEY 13 BASPGM
*   CP SAVESYS BASPGM
*      (where 'BASPGM' is the desired name of DCSS as given in
*      starter program; 'DCSSBASE' is address of DCSS; 'PROG' is the
*      name of the compiled BASIC program; there may only be one
*      main program in DCSS; chaining is not allowed)
*
* Note that BLIOCRT must be first in DCSS.
* After doing above, make sure you re-DEFINE your storage below
* DCSSBASE.
*
* To execute, just enter:
*   NAME parms
*       (where optional 'parms' may be retrieved by the program
*       using the intrinsic function PARM$)
*
*****************************************************************
```

```
         EJECT
BEGDCSS  CSECT
         STM   14,12,12(13)       SAVE CALLER'S REGISTERS
         LR    1,13               SAVE CALLER'S R13
         LR    10,15              USE R10 AS BASE
         USING BEGDCSS,10         ESTABLISH ADDRESSABILITY
         LA    13,SAVEAREA        POINT TO NEW SAVE AREA
         ST    1,4(13)            BACKCHAIN CALLER'S SAVE AREA
         LA    1,DCSSNAME         POINT TO NAME OF DCSS
         BAL   14,FINDDCSS        GET ADDRESS OF DCSS
* R15 = ADDRESS OF DCSS
         L     13,4(13)           RESTORE REGISTERS AND
         L     14,12(13)            AND START PROGRAM BY
         LM    0,12,20(13)          BRANCHING TO BLIOCRT
         BR    15                   (WHICH MUST BE FIRST IN DCSS)
*
         DS    0D
DCSSNAME DC    CL8'BASPGM'        NAME OF DCSS
         EJECT
*
* LOOK FOR DCSS AND LOAD IT AS NEEDED
*   ON ENTRY:  R1 = ADDRESS OF NAME OF DCSS
*   RETURNS:   R15 = ADDRESS OF DCSS
*   ERRORS: DISPLAY MESSAGE AND ABORT
FINDDCSS EQU   *
* GET VIRTUAL STORAGE SIZE IN REGISTER 2 USING DIAG X'60'
         DC    OH'0',X'83',AL.4(2,0),Y(X'0060')
* TRY TO FIND DCSS USING DIAG X'64'
         LR    3,1                GET DCSS NAME
         LA    4,12               FIND CODE
         DC    OH'0',X'83',AL.4(3,4),Y(X'0064')
         BC    8,INCOR            ALREADY IN STORAGE
         BC    4,LOADDCSS         FOUND, BUT NOT IN STORAGE
         C     4,PAGECODE         PAGING ERRORS?
         BNE   NODCSS             NO, MEANS NO DCSS
         B     PAGERR             OTHERWISE MUST BE PAGING ERRORS
*
LOADDCSS CR    3,2                WOULD SEGMENT OVERLAP DEFINED
         BL    OVERLAP              STORAGE? YES, THEN ABORT
* LOAD DCSS USING DIAG X'64'
         LR    3,1                GET DCSS NAME
         LA    4,0                LOAD CODE
         DC    OH'0',X'83',AL.4(3,4),Y(X'0064')
         BC    12,GOTIT
         B     NODCSS
INCOR    CR    3,2                WOULD SEGMENT OVERLAP DEFINED
         BL    OVERLAP              STORAGE? YES, THEN ABORT
GOTIT    EQU   *
         LR    15,3
         BR    14
         EJECT
PAGERR   EQU   *
         LA    2,PAGMSG
         LA    3,L'PAGMSG
         LA    4,108              PAGING ERROR RETURN CODE
         B     ABORT
```

```
OVERLAP  EQU   *
         LA    2,OLAPMSG
         LA    3,L'OLAPMSG
         LA    4,104               OVERLAP RETURN CODE
         B     ABORT

* NO PROGRAM DCSS (OR ITS BAD)
NODCSS   EQU   *
         LA    2,NDCSSMSG
         LA    3,L'NDCSSMSG
         LA    4,100               NO DCSS RETURN CODE
ABORT    EQU   *
         WRTERM  (2),(3)           DISPLAY MESSAGE
* R4 = RETURN CODE
         LR    15,4                MOVE RETURN CODE TO R15
         L     13,4(13)            RESTORE REGISTERS
         L     14,12(13)
         LM    0,12,20(13)
         BR    14                  RETURN TO OPERATING SYSTEM
         EJECT
*
* LOCAL DATA AREA
*
         DS    0D
SAVEAREA DS    18F
PAGECODE DC    F'177'
*
* MESSAGES
*
NDCSSMSG DC    C'##### DCSS CANNOT BE FOUND'
OLAPMSG  DC    C'##### USER STORAGE OVERLAPS DCSS'
PAGMSG   DC    C'##### PAGING ERRORS IN DCSS'
         END    BEGDCSS
```

Index

A

B

C

D

E

O

P

Q

R

Numerics

Reader's Comment Form

IBM BASIC/VM System Services
SC26-4028-2

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM BASIC/VM System Services (File No. S370-40) Printed in U.S.A. SC26-4028-2

Reader's Comment Form

IBM BASIC/VM System Services
SC26-4028-2

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

SC26-4028-2

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM

IBM BASIC/VM System Services (File No. S370-40) Printed in U.S.A. SC26-4028-2

IBM BASIC/VM
System Services

File Number S370-40

SC26-4028-02

Printed in U.S.A.