SC33-0051-0
File No. S360/S370-29

Program Product

DOS PL/I Optimizing Compiler: CMS User's Guide

Program Numbers 5736-PL1
5736-LM4
5736-LM5
(These program products are available as composite package 5736-PL3)

IBM

First Edition (March 1976)

This Edition applies to Version 1, Release 4, Modification 1 of the DOS PL/I Optimizing Compiler under Release 3 of VM/370 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes will continually be made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 and System/370 Bibliography SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM United Kingdom Laboratories Ltd., Programming Publications, Hursley Park, Winchester, Hampshire, England. Comments become the property of IBM.

Preface

This manual explains, for the DOS users of the Conversational Monitor System (CMS) component of the IBM Virtual Machine Facility/370 (VM/370), how to invoke the DOS PL/I Optimizing Compiler under CMS and execute and debug programs compiled by it.

The reader is assumed to have a basic knowledge of PL/I, DOS and of CMS.

Chapter 1 is an introduction to PL/I under CMS. It aims to give enough information to allow the reader to enter, compile, execute, and debug a straightforward PL/I program under CMS. It also aims to act as a guide to further sources of information and to provide enough background material to allow the reader to make full use of the potentialities of the optimizing compiler under CMS.

Chapter 2 is the reference source for the special restrictions and conventions that apply to PL/I when it is compiled by the DOS optimizing compiler and executed under CMS.

Chapter 3 is the reference source for the DOSPLI command and the DOS PL/I optimizing compiler options.

Figure P.1 is a guide to using this book.

REFERENCE PUBLICATIONS

This book makes reference to the following publications for related information that is beyond its scope.

Virtual Machine Facility/370:

- CMS User's Guide, Order No. GC20-1819
- CP Command Reference for General Users, Order No. GC20-1820
- CMS Command and Macro Reference, Order No. GC20-1818
- Terminal User's Guide, Order No. GC20-1810
- Planning and System Generation Guide, Order No. GC20-1801

DOS PL/I Optimizing Compiler: Language Reference Manual, Order No. SC33-0005

DOS PL/I Optimizing Compiler: Programmer's Guide, Order No. SC33-0008

DOS PL/I Optimizing Compiler: Program Logic, Order No. SC33-0006

DOS PL/I Optimizing Compiler: Messages, Order No. SC33-0021

AVAILABILITY OF PUBLICATIONS

The availability of a publication is indicated by its use key, the first letter in the order number. The use keys are:

G - General: available to users of IBM systems, products, and services without charge, in quantities to meet their normal requirements; can also be purchased by anyone through IBM branch offices.

S - Sell: can be purchased by anyone through IBM branch offices.

L - Licensed materials, property of IBM: available only to licensees of the related program products under the terms of the license agreement.

Contents

Figures

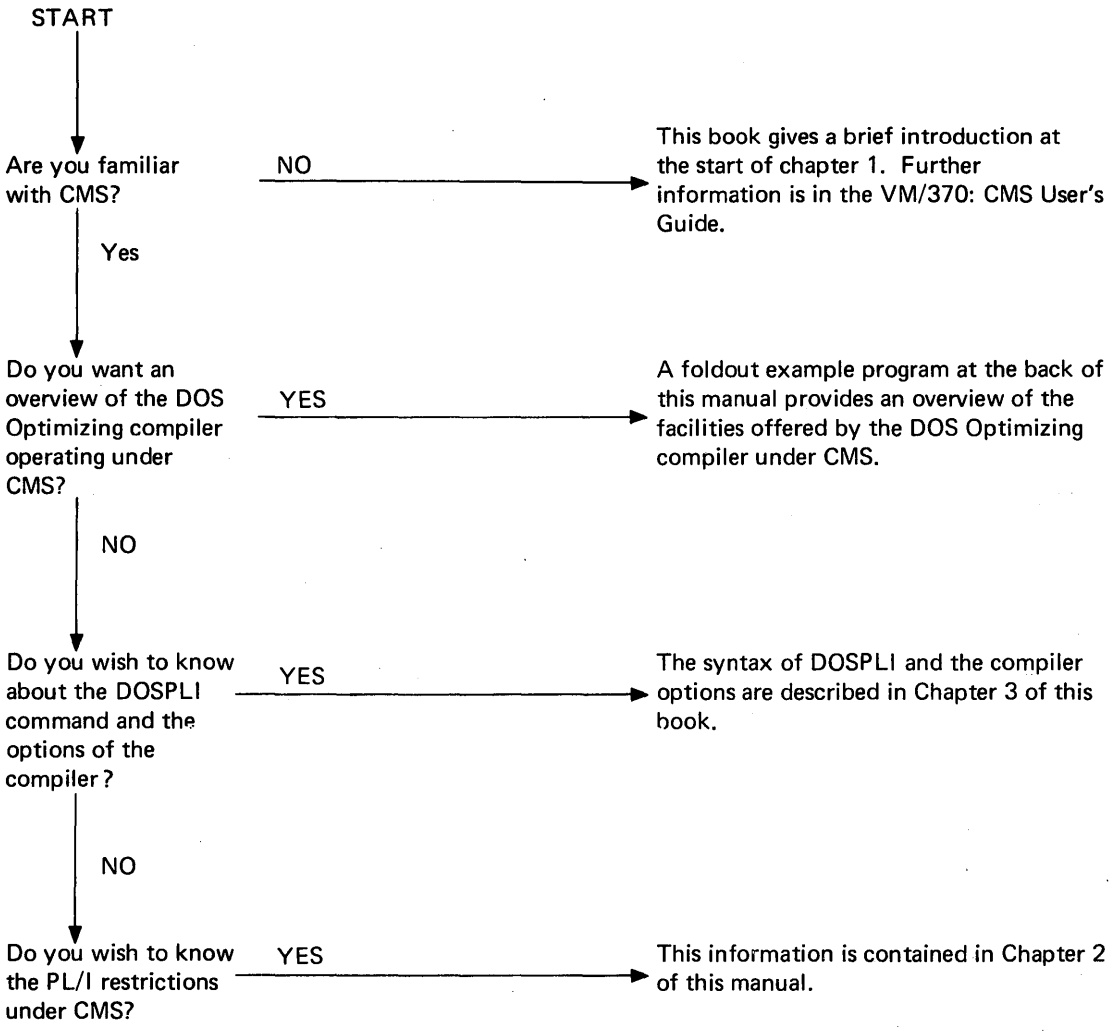

Figure P.1. How to use this book

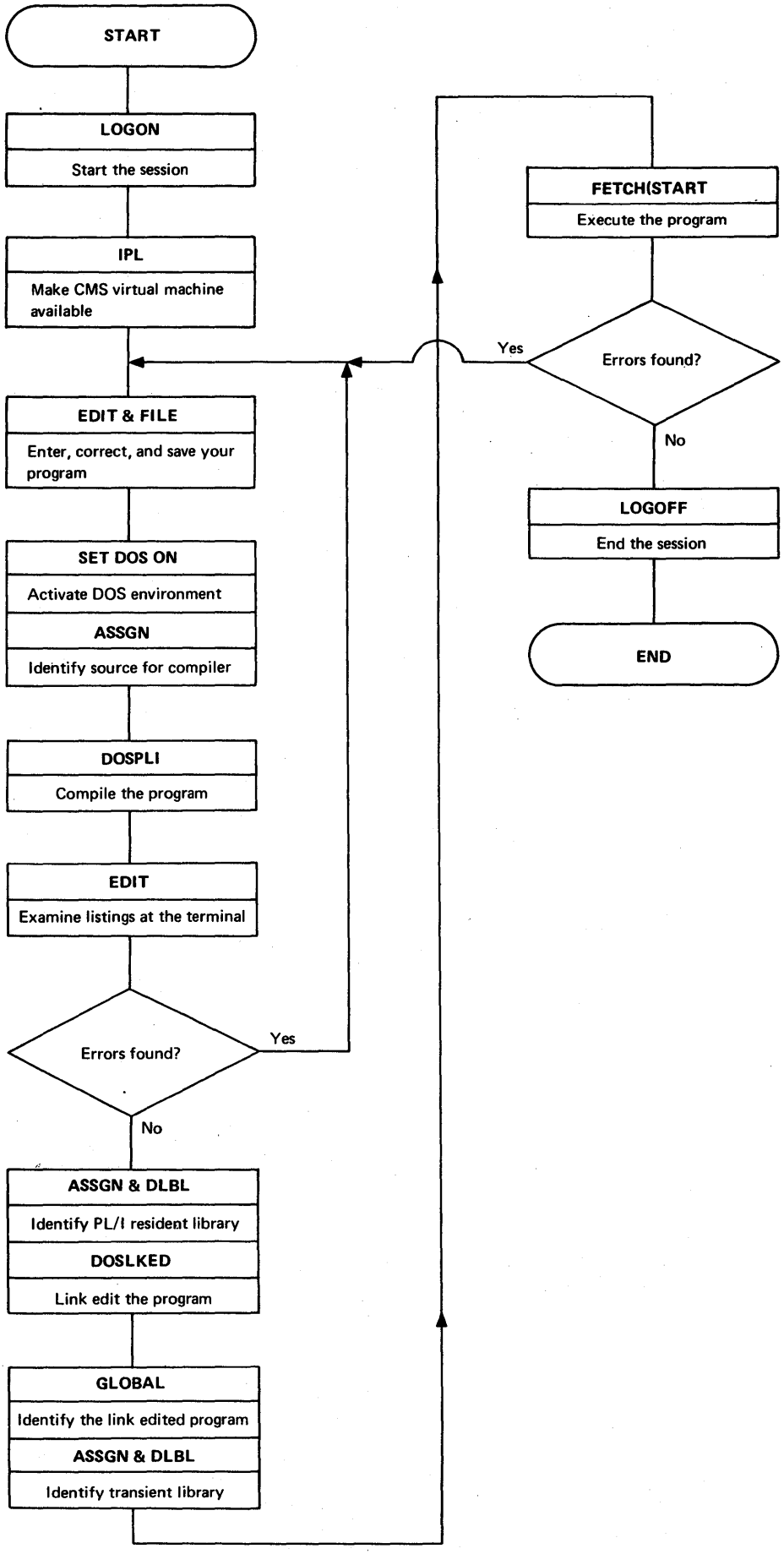

Figure 1.1. The steps involved in entering and executing a DOS PL/I program under CMS

S PL/I

Chapter 1: Writing and Running a PL/I Program Under CMS

Introduction

Running a DOS PL/I program under CMS is a very simple process. You will need to carry out the following steps using commands at a terminal:

1. LOGON at the terminal.
2. IPL CMS.
3. Write or alter a source program using the CMS editor.
4. Activate the DOS environment using the SET DOS command, assign the source file to SYSIPT using the ASSGN command, and compile the source program using the DOSPLI command.
5. Access the PL/I resident library using ASSGN and DLBL commands, then link edit the object program into an executable CMS/DOS phase library (DOSLIB) using the DOSLKED command.
6. Execute the compiled program using the DLBL and ASSGN commands to access the PL/I transient library and the GLOBAL command for the DOSLIB, followed by the FETCH and START commands.
7. End the session.

After a brief introduction to VM/370 and CMS the remainder of this chapter leads you through the steps listed above one by one. Because CMS/DOS is intended primarily for program development, a section on debugging is also included at the end of the chapter. A standard approach has been adopted for each step. The format is:

1. Summary and example. These give the essential information for a novice user to run straightforward programs and list any special cases that require additional action. These are the only sections you will need to look at during your first CMS sessions.
2. Background information. This amplifies the information in the summary and is intended to enable the user, as he gets more experienced, to get the best possible results from using DOS PL/I under CMS.
3. Special considerations. This explains what to do in the special cases listed in the summary. Special cases have been kept separate to prevent them making a simple process appear complex. This section is omitted if there are no special cases.
4. Sources of further information. This lists the manuals that you will require for any further information.

A sample terminal session can be folded out from the end of the book. This shows all steps involved on one page and can be used for quick reference.

Other chapters in this book are for reference. Chapter 2 lists the special restrictions and conventions that apply to PL/I when compiled by the DOS PL/I Optimizing Compiler and executed under CMS. Chapter 3 shows the syntax of the DOSPLI command and lists the compiler options.

System requirements: The PL/I Optimizing Compiler requires a minimum of 320K bytes of virtual storage for the CMS virtual machine. This figure is the same as the suggested minimum for CMS.

USING VM/370

DOS PL/I under CMS is a small part of the VM/370 system. VM/370 is a powerful system allowing you to develop and execute programs from the terminal, enter and access data from the terminal, and giving you the power of a full teleprocessing network. To make the most effective use of DOS PL/I it will pay you to know the underlying principles of VM/370. These are explained briefly below with particular reference to CMS/DOS.

VM stands for virtual machine. This is because as a VM/370 user you can behave as if you had a computer all to yourself. You have a "virtual machine". On this machine you have a set of disks on which your programs and your data are held. You have a reader to which you, and other VM/370 users, can pass data which can then be read onto your disks. You have a printer which can print copies of your files or of your program output, you have a CPU which you can use to execute programs or the commands that are available under the VM/370 and you have a console which is your terminal.

In fact, many of the resources will be shared with other users. The sharing of the resources is handled by the control program CP. When you sit at the terminal you are always working under the control of CP.

CP is the program that presents you with your virtual machine. To operate the virtual machine, you must have an operating system. You must ask CP to load a system using an IPL command. Many types of operating systems can be loaded. You will normally load CMS, a system specially designed for interactive programming. If you make a serious error under CMS or any other operating system it will ABEND and you will be returned to the direct control of CP. You can then re-IPL CMS and start again, having harmed no one but yourself.

The chief features of CMS are a file handling system and a facility for interactive programming. Both PL/I programs and the data they use, will be held as CMS files.

Although, when you are using CMS you will in fact have two operating systems, CMS and CP to which you can specify commands, this will not normally be apparent to you and you will appear to have one large set of commands at your disposal.

The commands that have most interest to the PL/I programmer can usefully be divided into four sets; those that control the faciltiies and configuration of your virtual machine, those that handle files, those that allow you to query the status of your virtual machine or your file system, and those that enable you to execute programs.

The most important are shown below:

Controlling your Virtual Machine Configuration

LINK and ACCESS	Connect disks to your virtual machine so that you can access the data, programs, or storage space you require.
SPOOL	Directs output so that it can be printed or sent to your reader for subsequent recovery.

Controlling your Files

EDIT	Allows you to enter and update files from your terminal.
ERASE	Erases files.
TYPE	Allows files to be displayed at the terminal.
PRINT	Allows files to be printed on the high-speed printer.

Querying your Status

QUERY	Allows you to determine the current status of your virtual machine.
LISTFILE	Gives a listing of your CMS files.

Plus various specific DOS oriented commands listed below:

Executing Programs

These commands give access to various assemblers and compilers including the OS and DOS PL/I compilers and also simulate DOS and OS job control language.

As a DOS PL/I programmer, your major concern will be with CMS/DOS, a fuller description of this is given below.

The DOS System under CMS

The DOS system under CMS, known as CMS/DOS, is a simulated system. It uses part of the actual DOS system for functions that include link-editing and control of input/output, and simulates other items.

All programs that are executed under CMS/DOS must be stored in the DOS core image library or in a DOSLIB file, which is a simulation of a core image library. A DOSLIB file is stored on a CMS disk.

Real DOS libraries are readable from the CMS virtual machine but no writing can be done on them. Items from the libraries can, however, be copied to CMS disks by means of special service commands. They then become CMS files and can be edited using the CMS editor.

Relevant Commands

The commands available to users of CMS/DOS fall into four groups:

1. Commands to initalize and support the CMS/DOS system.
2. Commands for simulating DOS job control language.
3. Commands for simulation of DOS functions such as compiling, link-editing, and executing programs.
4. Commands for simulating librarian services.

All commands except those for program product compilation are described in VM370: CMS Command and Macro Reference. The PL/I compilation command is described in this manual and the COBOL

compilation command in the VM/370: CMS User's Guide for COBOL, Order No. SC28-6469.

The commands that are liable to be of most use to the PL/I programmer are briefly described below.

Commands to initialize and support the DOS system are:

SET — Used to activate the CMS/DOS environment and set the UPSI byte, which is a DOS housekeeping field.

QUERY — Used to discover whether the CMS/DOS system is active and to test the settings of the UPSI byte.

LISTDS — Used to list the status of DOS files accessed in CMS.

Commands for simulation of DOS job control language are:

ASSGN — Used to associate a symbolic device name with an actual unit. The unit may be a CMS disk or a virtual unit such as a virtual punch.

DLBL — Used to identify a particular DOS data set or CMS file on a physical unit and associate it with the name used to reference the file in the PL/I program.

Commands for simulating DOS services are:

DOSPLI — Used to compile PL/I programs. (The full syntax of DOSPLI is given in Chapter 3 of this manual.)

FCOBOL — Used to compile COBOL programs.

DOSLKED — Used to link-edit programs ready for execution and place them in a DOSLIB.

FETCH — Used to bring DOSLIB files into storage for execution under CMS/DOS.

The commands for simulating DOS librarian and associated services are:

DSERV — Used to display directories of the various DOS libraries.

ESERV — Used to copy onto a CMS disk, display, punch or print an edited (compressed) element of a DOS/VS source statement library.

RSERV — used to copy DOS relocatable library modules to CMS disks, to a virtual device, or to display them at a terminal.

PSERV — used to copy procedures from the DOS procedure library to CMS disks, or to a virtual device, or to display them at a terminal.

SSERV — used to copy DOS source statement library books to CMS disks, display them at the terminal or spool them to the virtual punch or printer.

DOSLIB — used to handle DOSLIB libraries, for example, to delete, rename, or add members.

The remainder of this chapter leads you through the steps involved in keying in and executing a program under CMS/DOS. The next page shows you how to log on to your virtual machine.

Starting the Session - the LOGON Command

SUMMARY

To start a terminal session, you switch on the terminal and enter the LOGON command, specifying the identifier of your virtual machine. The terminal responds by requesting your password if one is required by your installation. After you have entered the password, the system responds with a log message. You are now in the control program environment of VM/370, and can invoke CMS.

Example of use of the LOGON Command

EXAMPLE OF LOGON

Terminal Printout	Notes and comments
(You switch on the terminal)	
VM/370 ONLINE (you may have to press attention key to unlock the terminal keyboard.)	Message shows VM/370 is available.
logon skylark	LOGON command followed by identifier for your virtual machine. (Normally known as userid.)
ENTER PASSWORD: (password entered here)	System requests password. You enter password. The printing of the password will normally be suppressed or overprinted for security.
LOGMSG - 09:12:09 04/02/76 RUNNING SYS010 - COLD START AT 09:00 LOGON AT 09:13:04 THURSDAY 04/03/76	Log message showing time and date of message, system identification and start time, time and date of signing on.

Conventions:

1. A carriage return (or equivalent) is assumed after all programmer input.

2. The character | in column two implies spacing has been added to accommodate notes.

3. System response is in upper case (capital) letters; programmer input in lower case.

CP and Your Virtual Machine

When you have keyed in your LOGON command and your password, you are in control of a virtual machine. Your terminal can be considered as the console of your virtual machine. You can thus carry out many of the operations of the operator of the real machine. This includes the ability to invoke a number of operating systems, among them CMS.

Your virtual machine is controlled in the real machine by a control program known as Control Program/370 or CP. When you have received the log message, you are in control of your virtual machine and are said to be in the "CP environment".

SOURCES OF FURTHER INFORMATION

Topic	Reference Source
LOGON command	VM/370: CP Command Reference For General User's
LOGMSG meaning	VM/370: Terminal User's Guide

Invoking CMS - the IPL Command

SUMMARY

To invoke CMS, you issue the IPL (Initial Program Load) command.

Example of use of the IPL command

EXAMPLE OF IPL

Terminal Printout	Notes and comments
ipl cms	The IPL CMS command.
|	
CMS 3.0 PLC 0 WEDNESDAY 04/03/76 09.13.50	
|	
|	Message confirms CMS is invoked and
|	that CMS commands may be entered.

Conventions:

1. A carriage return (or equivalent) is assumed after all programmer input.

2. The character | in column two implies spacing has been added to accommodate notes.

3. System response is in upper case (capital) letters, programmer input in lower case.

Entering Commands and Data Under CMS

Unless you are operating in a submode of CMS, such as input mode within the editor, everything you enter at the terminal is taken to be a CMS command. If the command is correct, it is carried out and a Ready message typed to confirm that the command is complete and that the system is ready for further commands. If the command is not correct, an error message is typed. Data is transmitted to the system when you press the carriage return key for typewriter type terminals or the ENTER key for display type terminals.

On a typewriter type terminal when a CMS command is being executed, the terminal keyboard is locked so that you cannot enter any further data until the system is ready to receive it.

Line editing symbols

VM/370 provides four symbols to alter, delete, or split up the line you key in at the terminal. These four symbols are known as logical line editing symbols and are @, ¢, #, and " by default. For some terminals, ¢ becomes [or (. The symbols are removed from your input and treated as editing symbols unless they are preceded by the escape symbol (see "Using line editing symbols as normal characters" below). The line editing symbols can be used to alter or delete lines before you press the carriage return key, or to enter a number of commands on one line to save time.

Deleting a line: If you wish to delete a line you are typing and to reenter it completely, you should use the logical line delete symbol and then press the carriage return key. By default the logical line delete symbol is ¢. Thus to delete a line you could enter:

```
this is an example of deleting a line ¢
```

(¢ becomes [or (on some terminals.)

Altering a line: If you wish to alter a line you have not completed typing, and then transmit it to the system, you must, on a typewriter-type terminal use the logical character delete symbol, (sometimes called the logical backspace symbol). On a display terminal you can simply backspace the cursor and reenter the input. By default, the logical character delete symbol is @. If the logical character delete symbol is entered once it deletes the previous character, if it is entered twice it deletes the previous two characters, and so on. Thus to alter the line you are typing you could enter:

```
this is an example of altering an amusing little wine@@@@line
```

Many programmers prefer to use the actual backspace key on the terminal as the character delete symbol. This saves the trouble of having to count back to the character you wish to change. Instead you can just backspace to the incorrect character and reenter the line from that point. To set the backspace as the character delete symbol you must use the TERMINAL command thus:

```
TERMINAL CHARDEL (you press the backspace key at this point)
```

Entering more than one command per line: If you want to save time at the terminal by entering more than one command per line, you must use the logical line end symbol. By default this is #. The characters following the # are treated as a new line. The line end character can be used to split any type of input although its chief use is for commands. For example if you wanted to split a line, you might enter:

this is an example of splitting#a line

The system would see "a line" as a separate input line.

Using line editing symbols as normal characters: If you wish to use any of the line editing symbols as a normal character, you must precede it with the escape symbol. By default this is ". For example to enter the line 'this is an example of using the escape symbol to enter @' you would enter:

this is an example of using the escape symbol to enter "@

The escape character can be used preceding itself.

Attention key: If you are a normal user, TERMINAL MODE VM will be in effect at your terminal and the use of the Attention key or its equivalent will have the following results. If you press it once while under the control of CMS, it causes an attention interrupt. If a CMS command is being executed, this allows you to key in further CMS commands that will normally be executed when the current command has been completed. However, there are a number of commands that are executed immediately. These are called Immediate commands. HT - halt typing or displaying, HX - halt execution and RT - resume typing or displaying can be useful when running PL/I programs. The Immediate commands are described in the VM/370: CMS Command and Macro Reference. Note HX (halt execution) clears all previously entered DLBL commands.

If a CMS command is not being executed, pressing the Attention key once deletes anything entered on the current line, but otherwise has no effect.

If you press the Attention key twice in quick succession while in the CMS environment, control is returned to CP. The system then types "CP" at your terminal. If you wish to return to CMS, you can press the attention key again or enter the BEGIN command and control will be returned to CMS.

Profile EXEC

When the first CMS command after IPL is executed, a CMS disk must be accessed. If the first command is an ACCESS command, the disk accessed will be the disk named in the ACCESS command. If any other command is used, the 191 disk will be accessed by default and set up as your A-disk.

When the first disk is accessed, the disk is searched for a CMS EXEC procedure with the name PROFILE. (An EXEC procedure is a set of CMS commands that, typically, carry out repetitive housekeeping tasks such as assigning files. These commands are executed by entering the name of the EXEC procedure as a command.) If an EXEC procedure with the name PROFILE is found on the first disk accessed, it is automatically executed. Many installations use this feature to handle repetitive housekeeping tasks that need to be done at the start of every session.

A PROFILE EXEC or other EXEC procedure is a suitable method of establishing the DOS environment and handling the assignments necessary for compiling and running a PL/I program. A discussion of this and an example of a suitable EXEC procedure are given in Appendix A.

SOURCES OF FURTHER INFORMATION

Topic	Reference source
CMS background	VM/370: CMS User's Guide
IPL command	VM/370: CMS User's Guide VM/370: CP Command Reference for General Users
PROFILE EXECs	VM/370: CMS User's Guide Appendix A of this Manual

Entering the Program - the EDIT and FILE Commands

SUMMARY

To enter or alter a PL/I source program under CMS, it is necessary to use the CMS Editor. You enter the EDIT command followed by the filename of your choice and the filetype PLIOPT or PLI. You then use the editing facilities either to enter new input or, if you are updating, to alter the existing program. The facilities available for manipulating and altering text using the Editor are not described in this manual. If you are not aware of them, you will find them in the VM/370: CMS User's Guide and the VM/370: CMS Command and Macro Reference. The facilities for correcting lines before you press the carriage return key are described in the previous section under the heading "Line Editing Symbols".

When you are satisfied with your input or alterations, you use the FILE subcommand to create a CMS file that can be compiled using the DOSPLI command. In addition to storing the file on a CMS disk, the FILE subcommand returns you from the edit submode to the CMS environment.

If you are entering a new PL/I program, you must choose a new filename which follows the CMS conventions. That is, the name can consist of up to eight characters, which may be any alphameric character plus the special characters $, @, and #. (Remember however that @ and # are default line editing symbols and special action may be required if you wish to use them.) If you are altering an existing program, you specify the existing filename. Your input must be typed in columns 1 through 71. The editor will insert one blank to the left of your input so that the actual margins will be 2,72. You can type your input in either capitals or lowercase letters or any combination of the two; it will be translated into capitals (uppercase) by the Editor.

If you intend to execute your program under CMS, you should be aware of the special conventions and restrictions that apply to PL/I when it is used under CMS. These are listed in chapter 3 of this manual. If you intend to compile your program under CMS but to execute it under the control of DOS, then there are no special restrictions on the language facilities you may use.

Special action will be required in the following circumstances

1. If you wish to use compiler options other than your installation defaults.

2. If your program uses lowercase character string constants.

3. If you wish to use any of the line editing symbols as normal characters in your program. The line editing symbols are @, #, ¢, and " by default.

4. If you wish to create a file of secondary input text for inclusion by use of the %INCLUDE statement.

The action is described under the heading "Special Considerations" later in this section.

Examples of Use of the EDIT and FILE Commands

EXAMPLE OF ENTERING A NEW PROGRAM

Terminal Printout	Notes and comments
edit rabbit pliopt	Key in EDIT command followed by filename and filetype.
NEW FILE:	Message shows that you have no PLIOPT file called "rabbit".
EDIT:	Message shows you are in edit mode.
input	Keying in INPUT subcommand causes the input mode to be entered.
INPUT:	Message shows you are in input mode.
rabbit:proc options (main);	PL/I statements must be keyed in
display	columns 1 through 71.
('the rabbit squeaks to the world');	
end;	
	Null line (carriage return only on a line) causes return from input to edit mode.
EDIT:	Message shows change of mode.
top	Places the Editor's line pointer at the top of the file.
TOF	Message shows the pointer is at top of file.
type *	Have the contents of the file displayed at the terminal.
RABBIT: PROC OPTIONS (MAIN);	Note program has been translated to
DISPLAY	uppercase and moved one column
	right.
('THE RABBIT SQUEAKS TO THE WORLD');	
END;	
EOF	Means end-of-file reached.
file	Keying in FILE command results in your input being stored with the filename and type you specified. It also ends edit mode.
R;	Ready message indicates further commands can be entered.

Conventions:

1. A carriage return (or equivalent) is assumed after all programmer input.

2. The character | in column two implies spacing has been added to accommodate notes in the right hand column.

3. System response is in uppercase (capital) letters, programmer input in lowercase.

EXAMPLE OF ALTERING AN EXISTING PROGRAM

Terminal Printout	Notes and comments
edit dig pliopt	Issue EDIT command specifying existing PLIOPT file "dig".
EDIT:	System confirms that it is in edit mode with a copy of the file available. (If there was no PLIOPT file "dig" it would respond "NEW FILE:".) The line pointer is placed at the top of the file.
locate/pht/	Locate the errors with the LOCATE subcommand.
PHT EDIT(X)(A);	If the VERIFY option is in effect, the first line containing the incorrect word is displayed. If it is not the default, VERIFY can be specified by entering VERIFY in edit mode.
change/ht/ut	Issue CHANGE subcommand.
PUT EDIT(X)(A);	The corrected line is displayed. For details of CHANGE and other edit subcommands, see VM/370 CMS User's Guide.
file	FILE subcommand requests that the altered copy be stored as the file "dig" and that the previous copy be discarded.
R;	Ready message indicates further CMS commands may be entered.

Conventions:

1. A carriage return (or equivalent) is assumed after all programmer input.

2. The character | in column two implies spacing has been added to accommodate notes in the right hand column.

3. System response is in uppercase (capital) letters, programmer input in lowercase.

BACKGROUND

The CMS Editor

The CMS Editor allows you to create and update sequential CMS files from your terminal. It is used to create PLIOPT or PLI files which can be compiled by the PL/I compilers. (PLI files were the filetype available for PL/I under CP/67 and can still be used under the VM/370 system. Their format is identical to PLIOPT files.) The CMS Editor has two modes, the edit mode and the input mode. The edit mode allows you to use various subcommands to change, rearrange, or add to the copy of the

file in main storage. The input mode assumes that all items keyed in at the terminal are to be included in the file you are creating. To enter the input mode, you issue the subcommand INPUT. To return from the input mode to the edit mode, you enter a null line; that is, a line that consists only of a carriage return (or pressing the Enter key for display terminals.) (If you want a blank line in your PLIOPT file you must, therefore, key in at least one blank character in the line.)

When you issue the EDIT command, you must specify a filename and a filetype. CMS searches your disks for the file and if you have such a file, brings a copy of it into main storage and displays the message "EDIT:" indicating that you are in EDIT mode. If you do not have such a file, it assumes you intend to create one and displays the message "NEW FILE" followed by "EDIT".

To return from the edit mode to CMS, you must issue an edit subcommand that specifies what is to be done to the copy of the file that you have been editing. This can be done by using either the FILE subcommand or the QUIT subcommand. The FILE subcommand stores the copy of the file you have been creating and discards the previous copy, if any. The QUIT subcommand discards the copy of the file that you have been editing. If you wish to retain both the original copy of the file and the copy of the file that you have been editing, you can specify a new name in the FILE subcommand thus

```
file rabbit2
```

the previous version of the file rabbit would then remain available.

If you wish to still remain in edit mode but store what you have edited so far, you can use the SAVE subcommand.

A full description of the EDIT command and EDIT subcommands is given in the VM/370: CMS command and Macro Reference. For examples of how to use the Editor see the VM/370: CMS User's Guide.

Correcting Typing Errors

If you wish to correct a line before pressing the carriage return key you can use the line editing characters described under the heading "Line Editing Characters" in the previous section of this chapter. If you wish to correct a line when it has been transmitted, you must use the editing facilities that are described in the VM/370: CMS User's Guide.

Using the Editor to Alter Non-CMS Files

The CMS Editor can only be used on CMS files, however, if you wish to alter DOS catalogued procedures or source statement books, these may be copied to CMS files by use of the PSERV, ESERV, and SSERV commands. The PSERV command can be used to copy a DOS catalogue procedure onto a CMS disk so that the job stream could be modified via the CMS Editor. The resultant job stream preceded by the CATALP control cards must then be spooled back to the DOS/VS virtual machine in a manner similar to that shown in the %INCLUDE example that follows.

Since DOSPLI compilations cannot access CMS MACLIB (macro) libraries, the only practical use of the SSERV and ESERV commands would be to copy DOS source books and edited source books, respectively, onto CMS disks

in order to manipulate them using the CMS Editor. They, too, must then be spooled back to the DOS virtual machine.

For more information on the PSERV, ESERV, and SSERV commands, refer to the VM/370: CMS User's Guide.

Format of PLIOPT files

PLIOPT and PLI files created by the editor have 80 byte fixed length records. Sequence numbers are in columns 73 through 80. PLI files are an alternative type of file of the same format. The standard tab settings for PLIOPT files are 2, 4, 7, 10, 13, 16, 19, 22, 25, 31, 37, 43, 49, 55, 79, and 80. The zones are columns 2 and 72; input is truncated at column 72.

SPECIAL CONSIDERATIONS

Use of Non Default Compiler Options

Non-default compiler options are entered in a *PROCESS statement that precedes the PL/I source statements. Special action is required to enter them because the * must appear in column 1 and, by default, the editor moves all input to PLIOPT and PLI files one column to the right. The method used depends on the type of terminal you are using. If you are using a typewriter terminal with a backspace key such as an IBM 2741 the backspace key must be used before the *. The *PROCESS statement takes the form:

```
(you press the backspace key)*process attributes xref;
```

If you are using the backspace character as a character delete symbol it must be preceded by the escape symbol. (See "Line Editing Symbols" under "Invoking CMS - the IPL Command" earlier in this chapter.) If you are using backspace for character deletion and the escape symbol is the default " you must enter:

```
"(you press the backspace key) *PROCESS attributes xref;
```

If you are using a display type terminal without an explicit backspace key, such as an IBM 3277, the simplest method is to reset the tabs to 1 and then enter the *PROCESS statement starting in column 1. The subcommands used could be as follows:

```
preserve                    save standard PLIOPT tab settings
tabset 1                    set tabs allowing column 1 to be used
top                         set pointer to start of file
input                       enter input mode
*PROCESS FLOW (10,20);      key in *PROCESS statement starting in
                            column one
                            null line returns to edit mode
restore                     restore previous tab settings
```

Lowercase Character String Constants

When you are editing a PLIOPT file, the CMS editor automatically translates any lowercase characters you enter to uppercase. If you wish to enter lowercase character string constants in your program it is necessary to take special action. Enter:

```
CASE M
```

This must be done when you are in Edit mode. Your input will then be transmitted as entered. As the PL/I optimizing compiler accepts both upper and lowercase input, you can still enter your program in either uppercase or lowercase. During compilation the compiler will translate all PL/I into uppercase. Items appearing between quotes or comment delimiters will not be translated. The listing will show your program with everything in uppercase except comments and data between quotes.

To return to automatic translation to uppercase during your edit session, issue a CASE U subcommand. First enter a null line (carriage return only on a line) to return to the edit mode, then enter:

```
CASE U
```

Use of the Line Editing Symbols in your Program

If you wish to use any of the line editing symbols as normal input to your program you must precede them by the escape symbol. By default, the line editing symbols are @, #, ¢, ", but all or any of them may be changed with the TERMINAL command, and ¢ becomes [or (on certain terminals. If the defaults are in effect, and you wish to refer to a variable called DOCUMENT#2, it is necessary to enter the #, which is the default line-end symbol, preceded by " which is the default escape symbol, thus:

```
DOCUMENT"#2
```

Details of the line editing symbols are given in the previous section of this chapter under the heading "Line Editing Symbols".

Files for Inclusion by %INCLUDE Statement

Any text included in your PL/I source program by means of the %INCLUDE statement must be included from a DOS source statement library. If you create such text using the Editor, you can create it as a file of the MACRO or COPY filetype, spool your virtual punch to the userid of a DOS virtual machine, and punch out the text preceded by the DOS JCL to catalog the data in the DOS source statement library. The following steps indicate one method of accomplishing this assuming the file is called MACFILE:

1. Invoke the CMS Editor as follows:

   ```
   EDIT MACFILE MACRO
   ```

2. Key in

   ```
   INPUT
   ```

To get into input mode and key in the following:

```
// JOB CATALS
// EXEC MAINT
  CATALS sublib,bookname
control statement
(%INCLUDE statements go here)
   .
   .
/*
/&
```

Then enter a null line to get into edit mode and issue the FILE subcommand

```
FILE
```

3. To spool the newly created card images to your DOS virtual machine, key in the following:

```
SPOOL PUNCH TO dosuserid
PUNCH MACFILE MACRO (NOHEADER)
```

The card images are then sent to the virtual card reader of the DOS virtual machine specified in "dosuserid" where they will be processed by the MAINT program. Full explanations and alternatives are given in the VM/370: CMS User's Guide. Examples of the use of the PSERV command are given in the VM/370: CMS User's Guide.

SOURCES OF FURTHER INFORMATION

Topic	Reference source
Format of PLIOPT and PLI files	VM/370: CMS User's Guide
Using the CMS editor	VM/370: CMS User's Guide
Spooling data to a DOS virtual machine	VM/370: CMS User's Guide

Compiling the Program - the DOSPLI Command

SUMMARY

To compile a DOS PL/I program under CMS, you issue the DOSPLI command specifying the filename of the PLIOPT or PLI file you wish to be compiled. For example:

```
DOSPLI RABBIT
```

where RABBIT is the name of the file.

Before the DOSPLI command is issued you must:

1. Activate the DOS environment by use of the SET command. For example:

   ```
   SET DOS ON G
   ```

   (This assumes that DOS SYSRES has already been accessed as your G-disk. See "Accessing the Compiler and its Libraries" later in this section.)

2. Assign the input file SYSIPT to the CMS disk that contains the file to be compiled. For example:

   ```
   ASSGN SYSIPT A
   ```

Two potential causes of unsuccessful compilation should be guarded against:

1. Assignment of logical units prior to entering the DOSPLI command in such a way that the command cannot be executed.

2. The NODECK compiler option being in effect.

Assignment of Logical Units before Executing DOSPLI

If you have made no previous assignments, the DOSPLI command will assign the logical units it requires for successful compilation and you have no cause to worry about logical units. However, if you have already assigned the logical units that the compiler uses, the DOSPLI command will not alter these assignments, assuming that you wish to use the compiler in some non-default manner. The logical units that the compiler uses are SYSIPT, which you must assign yourself, and SYSLST, SYSPCH, SYS001, and SYS002 which may be assigned in the DOSPLI command. SYSLST defines the destination of your listing file, SYSPCH defines the destination of your object program, and SYS001 and SYS002 are used for compiler work files. In the default situation, all these files will be on CMS read/write disks, however, you may vary SYSLST and SYSPCH to get the listing file or the object program transmitted elsewhere. Before issuing the DOSPLI command you should therefore ensure that these units are either correctly assigned or are unassigned. To unassign a logical unit, you use the ASSGN command thus:

```
ASSGN SYSxxx UA
```

where xxx is the identifier of the logical unit.

Compiling with the NODECK Option

On CMS/DOS the optimizing compiler places the object program on SYSPCH. The NODECK option specifies that SYSPCH will not be created consequently if NODECK is in effect no object program is produced. If the copy of the compiler you use is for CMS only DECK will almost certainly be the default option, and you will have no cause to worry about the NODECK option. However, if your copy of the compiler is shared between CMS/DOS virtual machine and DOS/VS virtual machines NODECK may have been chosen as the default compiler option to suit the DOS/VS machines. If this is the case, it will be necessary to enter the DECK compiler option on a *PROCESS record at the start of your program. The previous section explains how to do this. To discover the defaults you should ask your systems programmer or use the OPTIONS compiler option on its own in a *PROCESS statement. This last will result in a listing of the compiler options used being generated in the listings file.

During compilation, provided you have followed the instructions above, two new disk files will be produced. They will have the filetypes TEXT and LISTING and the same filename as the file specified in the DOSPLI command. The TEXT file contains the compiled code. The LISTING file contains the listings produced during compilation. When compilation is complete, CMS transmits a Ready message.

Unless the compiler has terminated abnormally, no return code will be produced with the Ready message. You must inspect the listing file under the EDIT command to examine the messages to discover the results of compilation. On a display-type screen (such as an IBM 3270), you can scroll through the listing on the screen. On a typewriter type terminal (such as an IBM 2741), you must locate the messages using the LOCATE subcommand of EDIT and type them at the terminal.

If the listing file reveals source program errors, you can alter the PLIOPT file that contains the source by use of the Editor. You can then reissue the DOSPLI command. This results in the creation of new TEXT and LISTING files corresponding to the newly edited source program. If previous versions existed they will be overwritten. When you have a satisfactory compilation, you are ready to execute the program, which is now in the form of a TEXT file. The next section of the chapter tells you how to do this.

Special action will be required if your source program is not on a CMS disk or if you are including text from a DOS source statement library.

The action required is described later under the heading "Special Considerations."

Example of Use of the DOSPLI Command

EXAMPLE OF USE OF THE DOSPLI COMMAND	
Terminal Printout	Notes and comments
access 390 g	Access the DOS SYSRES volume as your G-disk (example assumes that SYSRES is in your directory as 390 and that the compiler resides in the system core image library. You must discover how to access SYSRES in your installation).
R;	
set dos on g	Set up the DOS environment so that the DOS compiler can operate, specifying the CMS disk that has been accessed for SYSRES. (Note that this may be done automatically in your profile exec.)
R;	
assgn sysipt a	Assign source file.
R;	
dospli rabbit	Issue the DOSPLI command to compile the PLIOPT file rabbit.
R;	Ready message shows compilation is complete. The LISTING file produced by the compiler must now be inspected to see if the compilation was successful. A return code 'R(00xx)'; with the Ready message would indicate an abnormal termination of the compiler.
edit rabbit listing	Enter edit mode to inspect listing file.
EDIT:	Response shows you are in edit mode.
locate/compiler diagnostic	Search for the words "compiler diagnostic" in the listing file. On a display type terminal you could scroll through the file. Note that the character in column 1 will be a carriage control character. It should be ignored.
NOT FOUND	
EOF:	
	Shows that no diagnostic messages have been produced for this compilation, and that the compilation is therefore correct. Had the line read "COMPILER DIAGNOSTIC MESSAGES...." it would have been necessary to type the messages at the terminal by means of the subcommmand TYPE *. When enough had been typed, typing could be stopped by pressing the Attention key and using the HT (halt typing) command.
quit	Leave edit mode.
R;	Ready message, showing you are back in the CMS environment.

Conventions:

1. A carriage return is assumed after all programmer input.
2. The character | in column two implies spacing has been added to accommodate notes in the right hand column.
3. System response is in upper case (capital) letters, programmer input in lower case.

Compiler Output and its Destination

When you issue the DOSPLI command, CMS calls the DOS PL/I Optimizing Compiler to compile your source program. The compiler creates two new files during its execution. One file contains the compiled code that can be link-edited into an executable phase so that you can execute your program. The other file contains diagnostic messages about the compilation, and, optionally, listings of your source program and the compiled code. (The various options controlling the listing produced by the compiler are described in chapter 3 of this manual.)

Unless SYSLST or SYSPCH are assigned elswhere the two newly created files will be placed on CMS disks. They will have the same filename as the file that contains the source program but a different filetype. The compiled code will have the filetype TEXT and the listing will have the filetype LISTING. Thus, if you compiled a PLIOPT file called ROBIN you would, by default, create two further files called ROBIN; a TEXT file containing the compiled code and a LISTING file containing the listing information. These files would be placed on your CMS disks according to the rules shown in figure 1.2. (The relationship between CMS disks is explained in the VM/370: CMS User's Guide.)

SOURCE DISK	OUTPUT DISK
source disk read/write	source disk
source disk read/only with parent disk read/write	parent disk
source disk read/only with parent disk read/only and A-disk read/write	A-disk
source disk read only with no parent and A-disk read/write	A-disk
source disk read/only with no parent disk or parent disk read/only and A disk read/only	program terminates unless you have directed output to a non DASD device. (See VM/370 CMS User's Guide for information on how to do this).

Figure 1.2. The disks on which the compiler output is stored

Handling Listing Information

The DOS optimizing compiler places all diagnostic information on a listing file. This file always contains diagnostic error messages or a message saying that no errors have been found. In addition, compiler options can specify that a number of listings are to be generated. The possibilities include a list of source program statements, a list showing the attributes of all variables and other identifiers, a list showing all statements in which variables or identifiers are found (the cross-reference listing), and a list showing the size of the aggregates used in a program. (A list of all the possible listings can be found in

Figure 3.1 in chapter 3.) The production of some or all of these listings may be the default in the system you use. To discover which are the defaults, the OPTIONS compiler option can be used. This option results in a list of the options used being generated. If no options other than the OPTIONS option are specified in a *PROCESS statement, the default options of your installation will be shown.

The listing file is generated as a print file and is placed on one of your CMS disks in accordance with the rules shown in Figure 1.2. Because the listing is a print file, its formatting is controlled by a carriage control character in column 1. Therefore when the file is inspected at the terminal using the EDIT command, any character in the first column should be ignored.

To make the best use of the listing file the programmer needs a source listing, and needs to ensure that either an offset listing is generated or that the GOSTMT option is in effect. The GOSTMT option or an OFFSET listing is necessary to ensure that statements mentioned in compile-time and execution-time error messages, can be easily identified. The attribute and cross-reference listing can be extremely useful during program debugging, enabling the programmer to check that variables have been correctly declared or defaulted, and enabling a history of a particular variable to be followed through a program.

It is recommended therefore that either the GOSTMT or the OFFSET option is specified and that the SOURCE option is always used during program debugging. If these are not your installation defaults, a suitable *PROCESS statement should be included at the start of the PLIOPT file containing your source program.

At the end of a session, the listing file may be printed on a high-speed printer to retain a printed record of the source program and its compilation. This is done by use of the PRINT command. For example PRINT RABBIT LISTING would transmit the listing file to the printer. The listing file could then be erased from the CMS disk to save space by issuing the command:

```
erase rabbit listing
```

Files used by the compiler

During compilation the compiler uses a number of files. These files are allocated by the DOSPLI command's EXEC procedure that invokes the compiler. The files used are shown in figure 1.3. All files except SYSIPT will be assigned by the EXEC procedure unless already assigned. SYSIPT must be assigned by the programmer to the device where the source program is located. If SYS001 and SYS002 are assigned before the DOSPLI command is issued they must be assigned to CMS disks, because they are used by the compiler as work files.

FILES USED BY COMPILER	
Logical Unit	Filenames
SYSIPT/SYSIN (SYSIPT on DLBL |SYSIPT/SYSIN on ASSGN)	XINPUT
SYSLST	XPRINT
SYSPCH	XPUNCH
SYSSLB	IJSYSSL
SYS001	IJSYS01
SYS002	IJSYS02

Figure 1.3. Files that may be used by the compiler

Accessing Disks Under CMS

In addition to disks defined in your VM/370 directory, you can temporarily obtain disks via use of the CP LINK and DEFINE commands. CMS must also know about these disks, and you must use the CMS ACCESS command to establish a filemode letter for them:

```
ACCESS 197 f
```

CMS uses the filemode letter to manage your files during your terminal session. By using the ACCESS command, you can:

- Control whether the disk is to be read-only (that is, you cannot write on it), or is read/write.
- Control the minidisk search sequence used by CMS.
- Control which disks are to contain new files that you create.

For the most part, you will use your primary 191 minidisk, that is, your A-disk, your DOS and/or OS read-only disks, and, if needed, your VSAM disks. For more detailed information on the CMS ACCESS command and the CP LINK and DEFINE commands, refer to the VM/370: CMS User's Guide.

Accessing the Compiler and its Libraries

The compiler may be held either on DOS SYSRES or in a private DOS core image library.

If the compiler is on DOS SYSRES, you will probably access it before you set DOS on. In the example, DOS SYSRES is accessed on disk 198 with the CMS mode letter of C thus:

```
ACCESS 198 C
SET DOS ON C
```

Here the letter C in the SET command tells CMS that the DOS SYSRES volume is at 198. If the compiler is on DOS SYSRES, access to the compiler will be faster if no GLOBAL commands for DOSLIB libraries are in force when the DOSPLI command is issued, see the note below.

If the DOS PL/I compiler and the PL/I libraries are in private libraries, you must use ASSGN and DLBL commands to make them accessible. Asssuming that the compiler and PL/I transient library are in a core

image library whose dataset name is DOSPLI.CLIB, that the PL/I resident library is in a private relocatable library whose dataset name is DOSPLI.RLIB, that the disk is at 196, and that you wish to access it as your D-disk, the commands might be as follows:

```
ACCESS 196 D
ASSGN SYSCLB D
DLBL IJSYSCL D DSN DOSPLI CLIB (SYSCLB
ASSGN SYSRLB D
DLBL IJSYSRL D DSN DOSPLI RLIB (SYSRLB
```

DOSPLI.CLIB and DOSPLI.RLIB are examples of the DOS file-ids for the private core image and relocatable libraries, respectively. Because DSN is specified in the DLBL command line, CMS automatically inserts the period.

Note: If the DOS PL/I compiler resides in the DOS system core image library and not in the private core image library, the time required to compile your PL/I program will be significantly increased if any DOSLIB libraries were previously activated via a GLOBAL DOSLIB libename1... libenameN command. This is because CMS's fetch routine uses the following search order to find the PL/I compiler:

1. In a DOS/VS private core image library, if one had been previously assigned via a DLBL command such as the following:

   ```
   DLBL IJSYSCL DSN DOSPLI CLIB (SYSCLB)
   ```

2. If a GLOBAL DOSLIB command specifying DOSLIB libraries had been previously issued, all such libraries are searched in the order specified in the GLOBAL command.

3. In the DOS/VS system core image library, if the CMS disk mode letter (fm) was specified in the SET DOS ON fm command.

Therefore, if the PL/I compiler is in the DOS system core image library, you should issue the GLOBAL DOSLIB command with no additional operands to eliminate the searching of the DOSLIB libraries.

SPECIAL CONSIDERATIONS

Secondary Input Text - %INCLUDE Statements

If your program uses %INCLUDE statements to include previously written PL/I statements or procedures, the DOS/VS source statement library on which they are held must be made available to CMS before issuing the DOSPLI command. If the text to be included is on a private DOS/VS source statement library, you must use ASSGN and DLBL commands to access the source statement library on which the secondary input text is held, and an ACCESS command to make the DOS disk on which it is held available to the system. Assuming the source statement library is on 193 the command might take the form:

```
ACCESS 193 B
ASSGN SYSSLB B
DLBL  IJSYSSL B DSN PLI INCLUDE (SYSSLB
```

The DOSPLI command used to compile the program must specify either the INCLUDE or the MACRO option.

Source Program not on a CMS Disk

If your source program is on a DOS disk, it can still be compiled using the DOSPLI command under CMS. However the disk itself will only be accessible from CMS in read-only form and so it will not be possible to make corrections using the CMS Editor if an error is discovered.

To compile a program on a DOS disk, ACCESS, ASSGN, and DLBL commands must be used. Assuming the program was on disk located at 192, the commands might take the form:

```
ACCESS 192 B
ASSGN SYSIPT B
DLBL XINPUT B DSN MY PL1 PROG (SYSIPT
DOSPLI program name
```

An alternative would be to move the program to a CMS disk using a SSERV or ESERV command. Examples of the use of the commands are given in the VM/370: CMS User's Guide.

A detailed description of the ASSGN and DLBL commands is given in the CMS Command and Macro Reference.

SOURCES OF FURTHER INFORMATION

Topic	Reference source
Error message explanations	
CMS (numbered DMSxxxx)	VM/370: System Messages Manual
PL/I (numbered IELxxxx)	DOS PL/I Optimizing Compiler Messages
DLBL command	VM/370: CMS Command and Macro Reference
ASSGN command	VM/370: CMS Command and Macro Reference
PL/I language	PL/I Optimizing Compiler Language Reference Manual
DOSPLI command	Chapter 3 of this manual

Executing a DOS PL/I Program

SUMMARY

To execute a program under CMS/DOS you must have an executable program phase on a DOSLIB file or in a DOS/VS core image library. Such a phase is produced on a DOSLIB file by issuing the DOSLKED command which produces an executable program phase from either a CMS TEXT file or an object module in a DOS library.

Before the DOSLKED command is issued, the SET DOS ON command must have been issued and the PL/I resident libary must be made available so that the necessary library modules may be link-edited in the executable program phase. When an executable program phase has been link edited and placed on a DOSLIB file, the PL/I transient library must be made available so that transiently loaded modules are available for loading during execution. The GLOBAL command must then be issued to identify the DOSLIB library that holds the program. Because PL/I programs issue error messages on SYSLST, SYSLST must be assigned to a suitable device before executing a program. When a file has been link-edited and placed on a DOSLIB library, a FETCH command with the START option will load the program into main storage and execute it. When a program has been placed as an executable program phase in a DOSLIB it remains there until deleted and can always be executed by making the transient library available, specifying the DOSLIB in a GLOBAL command and issuing a FETCH command with the START option.

Executing a program held as a TEXT file therefore, involves the following steps:

1. Issuing an ASSGN command to make the DOS/VS PL/I resident library available for link-editing.

2. Issuing a DLBL command to identify the PL/I resident library.

3. Issuing a DOSLKED command to link-edit the text file and place it on a DOSLIB library. The program is now link edited and may be retained in this form.

4. Issuing a GLOBAL command for the DOSLIB on which the executable program phase has been placed.

5. Issuing an ASSGN command for the PL/I transient library.

6. Issuing a DLBL command for the PL/I transient library.

7. Assigning SYSLST so that any error messages will be accessible.

8. Issuing a FETCH command with the START option specifying the name of the executable phase.

These steps are shown in the example that follows.

Special action will be required if error messages are generated. See under the heading "Special Considerations" later in this section.

EXAMPLE OF EXECUTING A PL/I PROGRAM

PREPARATORY WORK - CREATING AN EXECUTABLE PROGRAM PHASE AND PLACING IT ON A DOSLIB FILE

```
set dos on
access 193 b
assgn sysrlb b
R;                                                  Make PL/I resident
dlbl ijsysrl b dsn privat reloc lib (sysrlb         library (on disk 193)
 |                                                  available as B-disk.
R;
 |
doslked rabbit mylib                        Link edit the object program
 |                                          with PL/I resident library
R;                                          READY message shows link-edit-
 |                                          ing successful.
```

EXECUTING THE DOSLIB PHASE

```
assgn sysclb b                              Assign PL/I transient library
R;
dlbl ijsyscl b dsn privat corim lib (sysclb     Make PL/I transient
 |                                              library available.
R;
global doslib mylib                         Indicate the DOSLIB to be sear-
R;                                          ched by FETCH command.
assgn syslst printer                        Ensure error messages are sent
R;                                          to the virtual printer.
spool print to *                            Have error message file spooled
R;                                          to your virtual reader so that
 |                                          it will be accessible at
 |                                          terminal if messages are
 |                                          generated.
 |                                          If error messages or other
 |                                          output to SYSLST is generated
 |                                          a message reading "PRINT FILE
 |                                          TO your userid etc" will be
 |                                          be displayed.
fetch rabbit(start                          FETCH command fetches the link-
EXECUTION BEGINS...                         edited module into storage.
 |                                          START option begins execution.
 |                                          The two commands can be issued
 |                                          separately or as shown.
THE RABBIT SQUEAKS TO THE WORLD             The message in the DISPLAY
 |                                          statement is transmitted to the
 |                                          terminal which acts as the
 |                                          console of the virtual machine.
R;                                          Ready message indicates that
 |                                          execution is complete.
```

Conventions

1. A carriage return is assumed after all programmer input.

2. The character | in column two implies spacing has been added to accommodate notes in the right hand column.

3. System response is in upper case (capital) letters, programmer input in lower case.

The CMS/DOS Linkage Editor

The CMS/DOS linkage editor prepares a previously compiled object program for execution and places it in a DOSLIB file in core image format. All compiled PL/I programs need link-editing before execution to resolve addresses and to include resident library modules that are used to handle program initialization and other routine tasks.

For most purposes, the link-editing command DOSLKED can be used by simply specifying the TEXT file produced by compilation, and the name of the DOSLIB file on which the link-edited module is to be placed. For example:

```
DOSLKED RABBIT MYLIB
```

causes the link-edited version of the TEXT file RABBIT to be placed on the DOSLIB library called MYLIB. A DOSLIB file is a type of library that can hold a number of link-edited programs. If the DOSLIB name is omitted, a new DOSLIB library with the name of the TEXT file will be generated.

For more complex link-editing jobs, input to the linkage editor must be provided in a CMS file with the filetype DOSLNK which will normally be created using the CMS Editor. The name of the DOSLNK file and the name of the DOSLIB library on which the link-edited module is going to be placed are then specified in the DOSLKED command. The example that follows shows the creation of a DOSLNK file and the execution of the DOSLKED command that specifies it. If you wish to examine the results of a DOSLKED command, these are, by default, placed on a CMS file with the filetype MAP and the filename of the file specified in the DOSLKED command. Like the LISTING file generated by the DOSPLI command, it can be examined using the Editor. Options of the DOSLKED command allow you to have your linkage editor output sent to your terminal (TERM option), or to the logical device associated with SYSLST (PRINT option). (SYSLST is normally associated with the system printer.)

In this example the file LINKER is used to create an overlay program in which the root phase is called RABBIT and the two overlay phases are called LAPIN and HARE.

```
edit linker doslnk
NEW FILE:
EDIT
input
INPUT
 action rel                  Link editor statements to be executed
 phase rabbit,root
 include rabbit
 phase lapin,*
 include lapin
 phase hare,lapin
 include hare
                             Null line to return to edit mode

EDIT
file                         Store disk with filename LINKER and
R;                           filetype DOSLNK
doslked linker mylib         DOSLKED command specifying file of
                             linkage editor statements
```

Note: Each DOSLKED function executed extends the size of the DOSLIB.

Thus Fetch time is increased. If possible, a separate DOSLIB should be created for each program. This DOSLIB should then be erased before another DOSLKED for the same program is executed. If several programs must reside in the same DOSLIB library the DOSLIB should be condensed periodically using the "DOSLIB COMP libename" command. A fuller explanation of how to use the DOSLKED command and DOSLIB libraries is given in the VM/370: CMS User's Guide.

Using CMS Files and DOS Data Sets

CMS files and DOS and OS data sets can, with varying levels of restrictions, be written and read by programs executed under CMS/DOS.

CMS files are completely accessible to CMS programs to read, write, and update. They are also available to create and update via the CMS Editor, and to manipulate with CMS file-handling commands. Such files can be made available to a number of virtual machines, but are not accessible from outside the CMS system except by copying and recreation. (It should be noted that only sequential and VSAM files are supported under CMS.)

VSAM data sets are available both to CMS virtual machines and to the DOS system proper. OS VSAM data sets are also available because DOS and OS VSAM data sets are, with minor exceptions, compatible. VSAM data sets provide a method of sharing data between CMS and outside systems. VSAM data sets cannot be manipulated by the CMS Editor or by CMS file-handling commands.

DOS data sets are available on a read-only basis to CMS programs provided that they are consecutive, that is, sequential, files.

Three elements are used under CMS/DOS to associate PL/I files with external data. Within a program, the file is identified by the declared name or the title option. (The title option allows a file name to be associated with different external data throughout the program.) Outside the program, the DLBL command associates the filename with a particular data set on a symbolic device, and the ASSGN command associates the symbolic device name with an actual physical device.

Users of DOS job control language will be familiar with DLBL and ASSGN as JCL statements. The CMS DLBL and ASSGN commands are very similar to their DOS/VS equivalents, however, it should be noted that there is no EXTENT command under CMS. Extent information is not necessary for CMS files because the space allocation is handled by CMS. For VSAM data sets however, you will be prompted for the extent information when you issue the DLBL command if it is required. VSAM data sets differ from other types in that they have their housekeeping handled by a set of programs known as Access Method Services, these programs are available to the CMS user by use of the AMSERV command which uses a file containing Access Method Services statements to specify the functions required.

Three examples follow showing the PL/I statements and the CMS commands necessary to access CMS files, VSAM data sets, and non-VSAM DOS data sets respectively. The full syntax for DLBL, ASSGN, AMSERV and other commands is given in VM/370: CMS Command and Macro Reference, and further examples of their use are given in the VM/370: CMS User's Guide.

Accessing CMS Files

To access a CMS file, you issue an ASSGN command associating a logical

unit with a particular device, and a DLBL command associating the PL/I file identifier with a particular file on the symbolic device.

In the example that follows, the PL/I program reads the file known in the program as OLDRAB. This refers to the file RABBIT1 DATA on the CMS B-disk. The program writes the file known in the program as NEWRAB, creating a CMS file on the disk that will be known as RABBIT2 DATA. A further file, PL/I file RABPRINT is assigned to the virtual printer.

PL/I Program Statements

```
DCL OLDRAB FILE RECORD INPUT ENV (MEDIUM(SYS009) F RECSIZE(40)),
    NEWRAB FILE RECORD OUTPUT ENV (MEDIUM(SYS008) F RECSIZE(40)),
    RABPRINT FILE STREAM PRINT ENV (MEDIUM(SYSLST));
```

CMS Commands and Responses

`assgn sys009 b` `R;`	Assign SYS009 to CMS B-disk.
`dlbl oldrab b CMS rabbit1 data (sys009` `R;`	Associate OLDRAB with RABBIT1 DATA on B disk. The keyword CMS indicates that it is a CMS file. "Data" is the filetype.
`assgn sys008 a` `R;`	Assign SYS008 to CMS A-disk.
`dlbl newrab a CMS rabbit2 data (sys008` `R;`	Associate NEWRAB with the file RABBIT2 to be placed on the A-disk.
`assgn syslst printer` `R;`	Assign SYSLST to printer. There is no need for a DLBL command for the file RAB-PRINT, because DLBL is only used for disks.

Accessing VSAM Data Sets

VSAM data sets are available to read, write, and update, both from within and from outside the CMS system. Before VSAM data sets can be accessed, or the AMSERV command used, the command

```
SET DOS ON (VSAM
```

must be issued. This results in VSAM initialization being carried out. SET DOS ON (VSAM can be specified outside the DOS environment or when DOS is already active. It should not be used unless VSAM processing is to take place, because it requires a sizeable virtual storage overhead compared with SET DOS ON.

VSAM data sets differ from other data sets in that they are always accessed through a catalog and that they have their routine housekeeping carried out by Access Method Services. The CMS/DOS user uses the AMSERV command for VSAM housekeeping functions and the ASSGN and DLBL commands to establish the location of a catalog and to associate an actual VSAM data set with a symbolic device and the file identifier in a PL/I program.

To use the AMSERV command, a file of the filetype AMSERV must be created using the CMS editor. The file should contain the necessary Access Method Services statements. An AMSERV command specifying the name of this file is then issued and the appropriate functions are carried out. The AMSERV command must always be used, for cataloguing and formatting purposes, before creating a VSAM data set. It is also

used for deleting, renaming, making portable copies, and other routine tasks. Before use, the minidisk or disk must be initialized using the IBCDASDI program supplied with VM/370, or, for disks other than minidisks, any other DOS or OS initialization program.

For VSAM data sets, information normally supplied in the ENVIRONMENT option of the PL/I file is placed in the VSAM catalog. Catalog entries are created by the DEFINE statement of Access Method Services, they contain such information as the space used or reserved for the data set, the record size, and the position of a key within the record. The catalog entry also contains the address of the data set.

To use a VSAM data set, the CMS user has to identify the catalog to be searched, to assign a unit to the symbolic device that contains the data set, and to associate the PL/I file with the VSAM data set. The DLBL command is used to specify the catalog and to associate the PL/I file with the data set, and the ASSGN command is used to associate the symbolic device with an actual unit. Note that VSAM data sets cannot be written to or read from your A-disk, the filemode must be other than A. Where the data set is being newly created, the AMSERV command must be specified to catalog and define the data set.

The relevant PL/I statements and CMS commands to access an existing VSAM data set and to create a new VSAM data set are shown below.

The PL/I program reads the file OLDRAB from the VSAM data set called RABBIT1 on the CMS C-disk. It writes the file NEWRAB onto the data set RABBIT2 also on the CMS C-disk. RABBIT2 is defined using an AMSERV command. It is assumed that the master catalog is defined to be on the C-disk and that VSAM space is also defined before executing the example.

PL/I File Declarations

```
DCL OLDRAB FILE RECORD SEQUENTIAL KEYED INPUT ENV(VSAM);
DCL NEWRAB FILE RECORD SEQUENTIAL KEYED OUTPUT ENV(VSAM);
```

CMS Commands

```
access 195 c                                  Access the VSAM master
assgn syscat c                                catalog on 195 as your C-disk.
dlbl ijsysct c dsn mastcat (syscat perm       Issue a DLBL for the
R;                                            master catalog.
                                              Note that this normally needs
                                              to be done only once per
                                              terminal session.
edit amsin amserv                             Create an AMSERV file.
NEW FILE:
EDIT
input
INPUT
 define cluster(name(rabbit2) vol(cmsdev) -
    cyl(1,1) recsz(130,130) nonindexed) -
    catalog (mastcat)

EDIT
file
R;
amserv amsin                                  Execute Access Method Services
R;                                            statements in the file to
                                              catalog and format data set.
assgn sys002 c                                Assign symbolic devices
R;                                            for VSAM data sets.
assgn sys001 c
```

```
R;
dlbl oldrab c dsn rabbit1(sys001 vsam
R;
dlbl newrab c dsn rabbit2(sys002 vsam

R;
```

Issued DLBL commands to associate PL/I files with the VSAM data sets. The fact that it is a VSAM data set must be specified in the DLBL command.

Accessing DOS Data Sets

To access a DOS data set it must first be made available to your virtual machine using the LINK command. Then, using the ACCESS command, it can be given a CMS filemode letter. Once this has been done, ASSGN and DLBL commands can be used to access the data set. Any attempt made to write onto the data set will be detected and an error message generated.

In the example that follows the PL/I file OLDRAB is used to access the DOS data set CONEY1.
It is assumed that the disk has been mounted and is held as VM disk number 196.

PL/I Program Statement

```
DCL OLDRAB FILE RECORD ENV (MEDIUM(SYS009)F RECSIZE(40));
```

CMS Commands

```
access 196 g

Y 196 G R/O-DOS
R;
assgn sys009 g
R;
dlbl oldrab g dsn coney1(sys009
R;
```

Connect DOS disk to your virtual machine.

Message to confirm DOS disk is accessed in read/only mode.

Assign SYS009 to CMS G-disk.

Associate PL/I file OLDRAB with DOS data set CONEY1 and symbolic unit SYS009.

SPECIAL CONSIDERATIONS

Accessing Error Messages

If an error or some type of exceptional condition occurs during the execution of a PL/I program, a message will be generated and sent to the file SYSPRINT which is sometimes truncated to SYSPRIN and which is normally assigned to SYSLST. You should make assignments to this file so that error messages will be accessible at the terminal if an error occurs. The method shown in the example is to assign SYSLST to the printer and spool the printer output to your reader. If a message is generated, CP will then inform you that a file has been spooled to your reader. You then issue the READCARD command which reads it onto a CMS disk and giving it the filename specified in the READCARD command. It will then be accesible for you to type or edit at the terminal.

If a print file is spooled to your reader CP will transmit a message to the terminal taking the form "PRT FILE file-number TO userid COPY 01

NOHOLD". The exact form will depend on your virtual machine configuration. If the message was caused by an error and you have no PL/I ERROR on-unit containing a GOTO statement, the message DMSDOS160S. "JOB 'jobname' CANCELLED DUE TO PROGRAM REQUEST" will be displayed indicating that the PL/I error handler has terminated your program. It should be noted that the output from the COUNT and FLOW compiler options and program output to SYSPRINT will also be spooled to your reader and result in the message "PRT FILE file-number TO userid ..."being displayed", so the message does not always indicate an error.

An example of accessing error messages is shown below. There are many other methods of making error messages available to the terminal. You can, for example, issue ASSGN and DLBL statements to transmit SYSLST output to your CMS disks. The advantage of the suggested method is that you will be informed when an error message has been generated.

```
┌──────────────────────────────────────────────────────────────────┐
│            EXAMPLE OF ACCESSING ERROR MESSAGES                   │
├──────────────────────────────────────────────────────────────────┤
│assgn syslst print                          Assign error message  │
│  |                                         to printer.           │
│R;                                                                │
│spool prt to *                              Spool printer to      │
│R;                                          reader.               │
│fetch vrab (start                           Start execution.      │
│EXECUTION BEGINS...                                               │
│PRT FILE 4809  TO  SKYLARK  COPY 01 NOHOLD  Message indicating    │
│  |                                         file spooled to       │
│DMSDOS160S JOB 'VRAB' CANCELLED DUE TO PROGRAM REQUEST.           │
│  |                                         reader.               │
│R;                                          Job cancellation      │
│readcard vrab output                        message.              │
│  |                                         READCARD command      │
│  |                                         stores file on CMS    │
│RECORD LENGTH IS '132' BYTES.               A-disk and names it   │
│  |                                         VRAB OUTPUT.          │
│R;                                          Message indicating    │
│type vrab output                            record length of      │
│  |                                         VRAB OUTPUT.          │
│                                            Type command to       │
│IBM209I  'ONCODE'=0082 'UNDEFINEDFILE' CONDITION RAISED           │
│   CONFLICTING ATTRIBUTES AND FILE ORGANISATION ('ONFILE'=NEWRAB)  │
│   IN STATEMENT 5 AT OFFSET =000124 IN PROCEDURE WITH ENTRY VRAB  │
│                                            display the file      │
│                                            at the terminal.      │
│                                            PL/I error message    │
│                                            displayed at your     │
│                                            terminal.             │
└──────────────────────────────────────────────────────────────────┘
```

Link-Editing an Object Program Stored on a DOS Relocatable Library

If the program is stored on the same relocatable library as the PL/I resident library it is possible to link it by issuing ASSGN and DLBL commands to identify the library, as in the example at the start of this section, and then specify the name of program in the DOSLKED command. Provided you have no CMS TEXT file of the same name, the program will be link edited. The following steps are only necessary if the PL/I resident library is not a part of the system relocated library. If the program is on a library different from the PL/I resident library, you should create it as a CMS TEXT file by use of the RSERV command and then link edit it as in the example at the start of this section. The command might take the form:

```
assgn sysrlb e
dlbl  ijsysrlb e dsn obj mod (sysrlb
rserv pliprog
```

This will create a CMS TEXT file called PLIPROG which you will be able to link edit. To link edit the PLIPROG, you can use the CMS Editor to create a DOSLNK file (that is, a file with a filetype of DOSLNK) that contains the following link-edit cards:

```
ACTION REL,MAP
PHASE PLIPROG,S
INCLUDE PLIPROG
```

Note: A blank character must precede each of the above records.

Then issue the DOSLKED command thus:

```
DOSLKED fn libename (options...
```

To link edit the PLIPROG program using the DOSLNK link-edit control statements.

SOURCES OF FURTHER INFORMATION

Topic	Reference Source
DOSLKED command	VM/370: CMS Command and Macro Reference
ASSGN command	VM/370: CMS Command and Macro Reference
AMSERV command	VM/370: CMS Command and Macro Reference
GLOBAL command	VM/370: CMS Command and Macro Reference
DLBL command	VM/370: CMS Command and Macro Reference
FETCH command	VM/370: CMS Command and Macro Reference

A description of how to use all these commands is given in the VM/370: CMS User's Guide.

Ending the Terminal Session - the LOGOFF Command

SUMMARY

To end a CMS session you enter the CP LOGOFF command from the CMS or the CP environment. LOGOUT can be used as an alias for LOGOFF.

Before finishing the session you may wish to erase some of the files. This is done by using the ERASE command for CMS files, or the DOSLIB command for deleting members of a DOSLIB library.

Special action will be required if you are using a switched line connection and you do not wish to be disconnected. See "Special Considerations" later in this section.

Example of ending the session

EXAMPLE OF LOGOFF

Terminal Printout	Notes and comments
logoff	You enter the LOGOFF command.
CONNECT=hh:mm:ss VIRTCPU=mm.ss.ss TOTCPU=mm:ss.ss	Message tells you the connect time, the actual length of the session, and virtual and the real CPU time in minutes, seconds, and hundredths of seconds.
LOGOFF AT hh:mm:ss (zone) day-of-week mm/dd/yy	Message shows time and date of logging off.
(you switch off terminal)	

Conventions:

1. A carriage return (or equivalent) is assumed after all programmer input.
2. The character | in column two implies spacing has been added.
3. System response is in upper case (capital) letters, programmer input in lower case.

BACKGROUND

Deleting Files

If you wish to delete CMS files you use the ERASE command. The command must specify the filename, the filetype, and the filemode if it is not

on the A-disk. For example, if you wished to delete the PLIOPT file "rabbit", that previous examples have created on your A-disk, you would enter:

```
erase rabbit pliopt
```

If you wished to delete all the files called "rabbit" on your A-disk you would enter:

```
erase rabbit *
```

To delete an executable program phase you will have to delete the appropriate member of a DOSLIB library using the DOSLIB command. For example:

```
DOSLIB DEL MYLIB RABBIT
```

deletes the executable phase RABBIT from the DOSLIB MYLIB. To delete a VSAM file, use the Access Method Services DELETE statement.

SPECIAL CONSIDERATIONS

Retaining a Switched Line Connection

If you are using a switched line to a computer, the use of the LOGOFF command as shown results in the connection to the computer being broken. If you wish to retain the connection, you must enter "logoff hold". The action is the same as for logoff except that the switched line is not disconnected.

SOURCE OF FURTHER INFORMATION

Topic	Reference source
ERASE command	VM/370: CMS Command and Macro Reference
LOGOFF command	VM/370: CP Command Reference for General Users

Debugging a Program

To debug a PL/I program under CMS you compile the program, look for errors in the listing file, correct them using the Editor on your source program, and repeat the process until all compilation errors are corrected. You then continue into execution, checking the results or the error messages to see if there are any logic errors. If logic errors are found, the errors are again corrected in the source program, and the program recompiled and executed. This process is continued until it is established that the program is correct.

The example that follows shows the process. The program is a simple program to calculate the average of numbers entered at the terminal. It contains two errors; a missing parenthesis, discovered during compilation, and failure to divide the total by the correct number, discovered during execution.

The suggested debugging procedure involves switching between the source file and the listing file. Care should be taken not to alter the listing file in an attempt to correct the source program. Such erroneous alterations will, of course, have no effect on the source program.

Use of CP Debug

Most PL/I programs can be debugged at the source level, and it should be noted that VM/370 users are in a position to use the OS PL/I program products including the Checkout Compiler. The Checkout compiler provides an array of symbolic debugging aids such as the setting of breakpoints, and the interactive correction and saving of the source program. The OS PL/I compilers can be used to check all OS compatible language.

Should it prove necessary to debug at machine language level, the CP debug facilities can be used. These are described in the VM/370: CMS User's Guide. To obtain a machine language listing of your compiled PL/I source program, you must specify the LIST option in the *PROCESS statement. The statement with offset X'00' in this listing will be loaded at an offset X'10' beyond the point specified in the message that is generated when you specify the FETCH command without the START option. Information on the execution time logic of PL/I programs is given in the DOS PL/I Optimizing Compiler: Execution Logic, Order No. SC33-0019.

```
------------------------------------------------------------------------
EXAMPLE OF DEBUGGING A PROGRAM
------------------------------------------------------------------------
access 193 d                            Access DOS SYSRES as D-disk.
R;
set dos on d                            Enter the simulated DOS environ-
                                        ment.
------------------------------------------------------------------------
Stage 1 - Compilation and Correction of Compiler Detected Errors
------------------------------------------------------------------------
assgn sysipt c
R;                                      Assign the source file as C-disk.
dospli nmav                             Call the DOS optimizing compiler
                                        located in the DOS system core
                                        image library to compile the
                                        program nmav.
R;                                      Ready message means that compi-
                                        lation is completed. Next inspect
                                        the listing file for errors.
                                        Examine listing file.
edit nmav listing
EDIT:
l/compiler diagnostic                   Find message in file.
-COMPILER DIAGNOSTIC MESSAGES           LOCATE subcommand can be
                                        abbreviated to l.
type *                                  Print messages at terminal.
-COMPILER DIAGNOSTIC MESSAGES
-ERROR ID L  STMT  MESSAGE DESCRIPTION Ignore first character which
0                                       is a carriage control for printer.
0SERVERE AND ERROR DIAGNOSTIC MESSAGES
0
0IEL0400I E  5  RIGHT PARENTHESIS ASSUMED AFTER 'RS, END WITH 99999''.
                                        An additional warning message
                                        may appear if the LINK compiler
                                        option is your default.
0
0END OF COMPILER DIAGNOSTIC MESSAGES
-COMPILE TIME  0.02M>                   Press Attention key and enter HT
ht                                      command to halt typing.
quit                                    Leave the listing file.
R;
edit nmav pliopt                        Edit source program.
EDIT:
1/99999/                                Locate erroneous statement.
     DISPLAY('ENTER NUMBERS, END WITH 99999' REPLY(STRING);
c/9'/9')                                Insert change using CHANGE sub-
                                        command abbreviated to "c".
     DISPLAY('ENTER NUMBERS, END WITH 99999') REPLY(STRING);
file                                    File the correct program.
R;
dospli nmav                             Recompile.
R;
edit nmav listing                       Examine listing file for errors.
EDIT:
locate/compiler diagnostic              Look for messages.
NOT FOUND:
EOF
top                                     Return to top of file.
TOF:
locate/source                           Locate "source".
```

```
------------------------------------------------------------------------
EXAMPLE OF DEBUGGING A PROGRAMMING PART 2
------------------------------------------------------------------------
type *                                  Print compiled source for reference.
-                SOURCE LISTING
-  STMT LEV NT
0
      1      0  AVERAGE: PROC OPTIONS(MAIN);                              NMA00010
      2   1  0  DCL (TOTAL,NUMBER) FIXED DECIMAL,                        NMA00020
                   STRING CHAR (10);                                      NMA00030
      3   1  0    TOTAL=0;                                                NMA00040
      4   1  0     DO I=1 BY 1;                                           NMA00050
      5   1  1      DISPLAY('ENTER NUMBER, END WITH 99999') REPLY(STRING); NMA00060
      6   1  1      NUMBER=STRING;                                        NMA00070
      7   1  1      IF NUMBER=99999 THEN GOTO LAB;                        NMA00080
      8   1  1      TOTAL=TOTAL+NUMBER;                                   NMA00090
      9   1  1     END;                                                   NMA00100
     10   1  0   LAB:                                                    NMA00110
                  DISPLAY('AVERAGE='||TOTAL/I);                          NMA00120
     11   1  0  END;                                                     NMA00130
1PL/I OPTIMIZING COMPILER       >       Press attention and enter HT
                                        command to stop typing.
ht
quit                                    Return from Edit mode to CMS.
R;
------------------------------------------------------------------------
Stage 2:  Execution and Correction of Execution-time Errors
------------------------------------------------------------------------
access 196 b                            Access the DOS private re-
                                        locatable and core image
                                        libraries as your B-disk.
assgn sysrlb b
R;
dlbl ijsysrl b dsn privat relocat lib(sysrlb     Make PL/I resident
R;                                               library available.
assgn sysclb b
dlbl ijsyscl b dsn privat corim lib(sysclb       Make PL/I transient
R;                                               library available.
doslked nmav mylib                      Link edit the program nmav.
R;
global doslib mylib                     Make DOSLIB available to CMS.
R;
fetch nmav(start                        Fetch program and start to execute.
EXECUTION BEGINS...
ENTER NUMBER, END WITH 99999            Message from PL/I DISPLAY statement
IBM007A WAITING FOR REPLY TO 'DISPLAY'  MESSAGE Prompt for data
10                                      Key in 1st number.
ENTER NUMBER, END WITH 99999
IBM007A WAITING FOR REPLY TO 'DISPLAY'  MESSAGE Prompt for data
11                                      Key in 2nd number.
ENTER NUMBER, END WITH 99999
IBM007A WAITING FOR REPLY TO 'DISPLAY'  MESSAGE Prompt for data
12                                      Key in 3rd number.
ENTER NUMBER, END WITH 99999
IBM007A WAITING FOR REPLY TO 'DISPLAY'  MESSAGE System Message
99999                                   Key in 99999 to mark end.
AVERAGE=         8.2500                 Program displays the result but
R;                                      you see answer is wrong.
                                        Your examination of the source
                                        listing shows that TOTAL is
                                        divided by 1 more than it should
                                        be, because I is incremented by
                                        1 for the end-of-data marker 99999.
```

EXAMPLE OF DEBUGGING A PROGRAM PART 3

```
edit nmav pliopt                          Edit the source file to correct
                                          error.
EDIT:
locate/display                            Locate "display".
     DISPLAY('ENTER NUMBERS, END WITH 99999') REPLY(STRING);
locate/display                            Wrong one, look for next occurance.
     DISPLAY('AVERAGE='||TOTAL/I);
change/i/(i-1)                            Make the change.
     DISPLAY('AVERAGE='||TOTAL/(I-1));    Corrected statement is displayed.
file                                      Store the corrected program.
R;
dospli nmav                               Recompile the program. The old
                                          LISTING and TEXT files are
R;                                        overwritten.
edit nmav listing                         Examine listing file for
EDIT:                                     errors.
locate/compiler diagnostic
NOT FOUND:                                Error free compilation.
EOF
quit                                      Return to CMS.
R;
doslked nmav mylib                        Link edit corrected program.
R;                                        Note that libraries etc are
                                          already assigned.
fetch nmav (start                         Re-execute the program.
EXECUTION BEGINS...
ENTER NUMBER, END WITH 99999
IBM007 WAITING FOR REPLY TO 'DISPLAY' MESSAGE Prompt for data
10
ENTER NUMBER, END WITH 99999
IBM007 WAITING FOR REPLY TO 'DISPLAY' MESSAGE Prompt for data
11
ENTER NUMBER, END WITH 99999
IBM007 WAITING FOR REPLY TO 'DISPLAY' MESSAGE Prompt for data
12
ENTER NUMBER, END WITH 99999
IBM007 WAITING FOR REPLY TO 'DISPLAY' MESSAGE System Message
99999
AVERAGE=         11.0000                  Correct result
R;
print nmav listing                        Print the listing file
                                          cn the system printer for
                                          reference.
logoff
CONNECT= 00:45:04 VIRTCPU= 000:26.70 TOTCPU= 001:19.44
LOGOFF AT 17:03:29 GMT THURSDAY 05/08/76
```

Chapter 2: PL/I Conventions and Restrictions Under CMS

Restrictions

The PL/I features that may not be used under CMS and restrictions on other features are shown in figure 2.1.

DO NOT USE UNDER CMS
ASCII data sets
ASSOCFLE option of the ENVIRONMENT attribute
BACKWARDS attribute with magnetic tapes
Files with INDEXED environment option that access ISAM datasets
FUNC option of the ENVIRONMENT attribute
PL/I checkpoint restart facilities (PLICKPT)
PL/I sort facilities (PLISRT)
REGIONAL files
Assembler language subroutines using multitasking, multipartition, or teleprocessing operations. CMS does not support any of these functions.

Figure 2.1. Restrictions on the PL/I functions that can be executed under CMS.

The results of using PL/I features that are not available under CMS are summarized below.

SORT	Results undefined
CHECKPOINT/RESTART	Results undefined
ISAM DATASETS (Files with the INDEXED ENVIRONMENT attribute)	CMS error message DMSBOP088E "UNSUPPORTED DTF TYPE 'dtftype'" will be displayed.
Use of TCAM, or spanned records on BDAM, or the BACKWARDS attribute.	CMS error message number DMSBOP063E "OPEN ERROR CODE x ON ddname" will be displayed.

ASCII data sets
BACKWARDS attribute
ASSOCFLE environment attribute
FUNC environment attribute

CMS error message DMSBOP089E "OPEN ERROR CODE CODE 'nn' ON fn/SYSxxx/TAPEn" will be displayed.
where:

- n=4 Attempting to open DTFCD or DTFPR with ASSOCFLE/FUNC parameters specified on the DTF macro.
- n=9 The parameter 'READ=BACK' has been specified on the DTFMT macro. CMS/DOS will only support tape processing in a forward direction.
- n=11 Attempting to open DTFMT (tape data files) and 'ASCII=YES' was specified.

Conventions

Certain conventions apply to PL/I under CMS because the terminal is treated as the console of the virtual machine. Thus the DISPLAY statement and REPLY option, normally used to communicate with the operator, can be used for communication between the program and the terminal.

No prompting or other facilities are provided for I/O at the terminal. DISPLAY and REPLY should therefore be used to communicate between the program and the terminal.

DISPLAY AND REPLY UNDER CMS

Because the terminal is considered to be the console of the virtual machine, the DISPLAY statement and the REPLY option can be used to create conversational programs. The DISPLAY statement transmits the message to your terminal, and the REPLY option allows you to respond. For example, the PL/I statement:

```
DISPLAY ('ENTER NAME') REPLY (NAME);
```

would result in the message "ENTER NAME" being printed at your terminal. The program would then issue a message wait for your response. When you had responded, your data would be placed in the variable NAME after you pressed the carriage return key. The terminal printout would look like this:

```
ENTER NAME
IBM007A WAITING FOR REPLY TO DISPLAY MESSAGE
LESLEY RIERA
```

Chapter 3: The DOSPLI Command and Compiler Options

How to Use This Chapter

This chapter shows the syntax of the DOSPLI command, the options that can be used with the DOS optimizing compiler, and the standard defaults that will apply if you do not specify values for certain options.

There are five sections:

1. A summary of the syntax notation used.
2. A description of the DOSPLI command.
3. A table of the options available with the optimizing compiler arranged in alphabetical order.
4. A table of options arranged by function.
5. An alphabetical list of options with detailed descriptions and syntax notation.

If you wish to accept the default compiler options, you will only need to look at the section on the DOSPLI command and possibly the section on syntax notation. It should be noted that the default values for the options may have been altered by your installation and may not correspond to those shown. If you wish to look up a particular option, you should look for it in the alphabetical section. If you want a summary of the options that are available, you should look at the alphabetical list of options. If you are looking for an option to serve a specific purpose, you should look in the table of options listed by function. Before using an option found in either of the tables you should check in the alphabetical section to discover the full syntax.

A general discussion of the DOSPLI command is given in chapter 1 under the heading "Compiling the Program - the DOSPLI Command".

Syntax Notation

The syntax notation used to illustrate the command in this part of the manual is the same as that used in the VM/370: CMS Command and Macro Reference. Briefly, the conventions are as follows:

Items in brackets [ ] are optional.

Items in braces { } are alternatives; choose only one.

An item underlined applies unless an alternative is specified.

Note: Defaults shown are suggested defaults and may have been changed for your system.

Items written in uppercase (capital) letters are keywords and must be spelled as shown.

Items written in lowercase letters must be replaced by appropriate names or values.

Separate the command name from the operands, options, and suboptions by one or more blanks.

The four special characters ' ( ) * (single quote, left parenthesis, right parenthesis and asterisk) must be included where shown.

DOSPLI Command

The DOSPLI command invokes the DOS PL/I Optimizing Compiler to compile a program written in PL/I source language. Provided the DECK compiler option is in effect, the compiler produces a TEXT file containing machine code and a LISTING file containing listings and diagnostics. Other files may also be produced depending on compiler options. Compiler options are either defaulted from the defaults specified by your installation, or specified in the *PROCESS statement at the start of the PL/I source program.

FORMAT:

where:

filename

is the name of the file that contains the DOS PL/I source program. The filetype must be PLIOPT or PLI.

USAGE

The DOSPLI command compiles a PL/I program into machine language object code. If the source program is stored as a CMS file on disk, it must have the filetype PLIOPT or PLI and an ASSGN command must be issued associating SYSIPT (or SYSIN) with the disk containing the source program. If it is not on a CMS disk, it must be defined to the system with ASSGN and DLBL commands.

The CMS/DOS environment must previously have been activated by a SET DOS ON command. If it is not active, an error message is generated. See chapter 1 or the foldout at the end of this manual for an example of DOSPLI including the necessary file assignments.

Example: To compile the PLIOPT file called CONEY

```
DOSPLI CONEY
```

Compiler Options

Compiler options are used to tailor compilations to your needs. They control items such as the listings produced, the space used by the compiler, the form of code produced, and where the object module is placed. They are also used to indicate the format in which the input is presented.

Compiler options are specified in the *PROCESS statement that precedes the source program. The method used is described in Chapter 1, in the section on the EDIT command.

The majority of compiler options offer alternative methods of processing (for example, OPTIMIZE or NOOPTIMIZE), one of which must be taken. A system of defaults specifies which options will be taken when neither one from a set is specified. The defaults are determined by your installation when the compiler is installed.. During installation certain options may be "deleted". These options are not then available unless he knows the password of the CONTROL option. This manual shows the defaults suggested by IBM, however, different defaults may have been chosen by your installation.

To simplify using the many options available, the options are presented in two summary forms, first alphabetically (Figure 3.1), then by function (Figure 3.2). These summaries are followed by the full description. The alphabetical summary is intended for quick reference to check spelling and allowable abbreviations. The functional summary is intended to summarize the possibilities that are open to you. The full description, in which options are arranged alphabetically, gives all necessary details about the options and their use.

COMPILER OPTIONS ALPHABETICAL SUMMARY

Compiler Option IBM suggested default for CMS underlined	Abbreviated Name	Use
AGGREGATE|NOAGGREGATE	AG|NAG	Lists aggregates.
ATTRIBUTES|NOATTRIBUTES	A|NA	Lists attributes of variables.
CATALOG('name')	-	Generates CATALR record.
CHARSET([48|60][EBCDIC|BCD])	CS([48|60][EB|B])	Specifies character set of input.
COMPILE|NOCOMPILE[(W|E|S)]	C|NC[(W|E|S)]	Specifies if compiler continues after syntax check.
CONTROL[('password')]	-	Allows use of deleted options.
COUNT|NOCOUNT	CT|NCT	Generates code for statement count.
DECK|NODECK[1]	D|ND	Generates a TEXT file.
DUMP|NODUMP	DU|NDU	Causes dump after compiler error.
DYNBUF|NODYNBUF	-	Specifies method of buffer allocation.
ESD|NOESD	-	Lists external symbol dictionary.
FLAG[(I|W|E|S)]	F[(I|W|E|S)]	Suppresses certain diagnostic messages.
FLOW[(n,m)]|NOFLOW	-	Generates code for flow trace.
GOSTMT|NOGOSTMT	GS|NGS	Produce statement numbers in execution time messages.
INCLUDE|NOINCLUDE	INC|NINC	Allows inclusion of text with minimum overhead.
INSOURCE|NOINSOURCE	IS|NIS	Lists preprocessor input.
LIMSCONV|NOLIMSCONV	LSC|NLSC	Limits conversions and reduces space overhead.
LINECOUNT(n) LINECOUNT(55)	LC(n)	Specifies lines on listing page
LINK|NOLINK[(W|E|S)]	-	No effect under CMS.
LIST[(m[,n])]|NOLIST	-	Lists compiled code.
MACRO|NOMACRO	M|NM	Handles all preprocessor statements.
MAP|NOMAP	-	Lists static code.
MARGINI('c')|NOMARGINI	MI('c')|NMI	Marks margins of source.
MARGINS(m,n[,c]) MARGINS(2,72)	MAR(m,n[,c])	Specifies margins used.
MDECK|NOMDECK	MD|NMD	Generating preprocessed card deck.
NEST|NONEST	-	Shows block arrangement on listing.
OFFSET|NOOFFSET	OF|NOF	Lists offsets of each statement
OPTIMIZE(TIME| 0|2|NOOPTIMIZE)	OPT(TIME| 0|2|NOPT)	Specifies optimization.
OPTIONS|NOOPTIONS	OP|NOP	Lists the options used.
SIZE(yyyyyy|nnnnK|MAX)	SZ(yyyyyy|nnnnK|MAX)	Specifies storage to be used by compiler.
SOURCE|NOSOURCE	S|NS	Lists source program.
STORAGE|NOSTORAGE	STG|NSTG	Lists storage requirements.
SYNTAX|NOSYNTAX[(W|E|S)]	SYN|NSYN[(W|E|S)]	Specifies conditions under which compiler proceeds after preprocessing.
WORKFILE(device type) WORKFILE(2311)	-	Specifies type of workfile to be used.
XREF|NOXREF	X|NX	Lists variable use by statement.

[1]DECK is default for non-CMS DOS/VS.

Figure 3.1. Compiler Options and IBM Recommended defaults arranged alphabetically

COMPILER OPTIONS: FUNCTIONAL SUMMARY PART 1		
LISTING OPTIONS		
Control listings produced	AGGREGATE	List of aggregates and their sizes.
	ATTRIBUTES	List of attributes of all identifiers.
	ESD	List of external symbol dictionary.
	FLAG(I|W|E|S)	Suppress diagnostic messages below a certain severity.
	INSOURCE	List of preprocessor input.
	LIST	List of compiled code produced by compiler.
	MAP	List contents of static control section produced by compiler.
	OPTIONS	List of options used.
	SOURCE	List of source program or preprocessor output.
	STORAGE	List of storage used.
	XREF	List of statements in which each identifier is used.
Improve readibility of source listing		
	NEST	Indicates do-group and block level by numbering in margin.
	MARGINI	Highlights any source outside margins.
Control lines per page of listing		
	LINECOUNT	Specify number of lines per page on listing.
INPUT OPTIONS		
To define character set and margins of input		
	CHARSET	Identify the character set used in source.
	MARGINS	Identify the columns used for source program, and identify position of a carriage control character

Figure 3.2. (Part 1 of 3) Compiler options arranged by function

COMPILER OPTIONS: FUNCTIONAL SUMMARY PART 2		
OPTIONS TO PREVENT UNNECESSARY PROCESSING		
Control whether compilation should end if errors above a certain level are found		
	NOSYNTAX(W|E|S)	Stop processing after errors are found in preprocessing.
	NOCOMPILE(W|E|S)	Stop processing after errors are found in syntax checking.
OPTIONS FOR PREPROCESSING		
	MACRO	Allow full use of the preprocessor facility.
	INCLUDE	Allow inclusion of text without overheads incurred by MACRO.
	MDECK	Produce a source deck from preprocessor output.
	INSOURCE	List the input to the preprocessor.
OPTIONS TO USE WHEN PRODUCING AN OBJECT MODULE		
	DECK	Produce an object module.
	CATALOG	Produce an object deck on virtual card punch with a CATALR card.
OPTIONS TO CONTROL STORAGE USED DURING COMPILATION		
	SIZE	Control the amount of storage used by the compiler.
OPTIONS TO REDUCE EXECUTION TIME STORAGE		
	DYNBUF	Allocate buffer space during execution.
	LIMSCONV	Specify that certain conversions will not be used in stream I/O, consequently reducing number of library modules link-edited.

Figure 3.2. (Part 2 of 3) Compiler options arranged by function

COMPILER OPTIONS FUNCTIONAL SUMMARY PART 3		
STATEMENT NUMBERING OPTIONS		
	GOSTMT	Retain a statement number table into execution so that execution time messages can specify statement number.
	OFFSET	Specify that a listing associating statement numbers with offsets will be generated. Enables you to identify statements from offsets given in execution time error messages.
OPTIONS FOR USE WHEN DEBUGGING		
	FLOW	Generate code that will result in a trace of executed statements being retained.
	COUNT	Generate code that will result in a count of the number of times each statement is executed being printed at the end of the program.
OPTION TO IMPROVE COMPILATION/EXECUTION SPEED		
	OPTIMIZE(TIME)	Reduce execution time at the expense of compilation.
	NOOPTIMIZE	Reduce compilation time at the expense of execution.
OPTION FOR USE WHEN DEBUGGING THE COMPILER		
	DUMP	Produce a dump if the compiler terminates abnormally.
OPTIONS TO SPECIFY DEVICES USED BY COMPILER		
	WORKFILE(device type)	Specify device type used for compiler workfiles.
OPTIONS FOR SYSTEMS PROGRAMMING		
	CONTROL('password')	Allows access to deleted options for those who know password.

Figure 3.2. (Part 3 of 3) Compiler options arranged by function

ALPHABETICAL LIST OF OPTIONS

IBM suggested defaults are underlined

AGGREGATE|NOAGGREGATE
AG|NAG

The AGGREGATE option specifies that the compiler is to produce an aggregate length table, giving the lengths of all arrays and major structures in the source program.

Example: To get a listing of the size of aggregates in a program:

```
*PROCESS AGGREGATE;
```

ATTRIBUTES|NOATTRIBUTES
A|NA

The ATTRIBUTES option specifies that the compiler is to include in the compiler listing a table of all source-program identifiers and their attributes. If both ATTRIBUTES and XREF apply, the attribute table is combined with the cross reference table.

Example: To get a list of program identifiers and their attributes:

```
*PROCESS ATTRIBUTES;
```

CATALOG ('name')

The CATALOG option specifies that the compiler will write an object deck to the virtual card punch and will precede it with a CATALR statement specifying the name under which the object module is to be cataloged. The module cannot be cataloged onto the system relocatable library under CMS. The name can contain up to eight chracters, the first of which may not be an asterisk. It must be enclosed in quotes.

Example: To specify that an object module will be written to the virtual card punch specifying the name BIGWIG on the CATALR card:

```
*PROCESS CATALOG ('BIGWIG');
```

CHARSET([48|60]|[EBCDIC|BCD])
CS([48|60]|[EB|B])

The CHARSET option specifies the character set and data code used in the source program. The compiler will accept source programs written in the 60-character set or the 48-character set, and in the Extended Binary Coded Decimal Interchange Code (EBCDIC) or Binary Coded Decimal (BCD).

60- or 48-character set: If the source program is written in the 60-character set, specify CHARSET (60); if it is written in the 48-character set, specify CHARSET (48). The language reference manual for this compiler lists both of these character sets. (The compiler will accept source programs written in either character set if CHARSET(48) is specified.

However, if 48-character set reserved keywords, for example CAT or LE are used as identifiers in a program using the 60 character set, errors may occur if it is compiled with the CHARSET(48) option).

BCD or EBCDIC: If the source program is written in BCD, specify CHARSET (BCD); if it is written in EBCDIC, specify CHARSET (EBCDIC). The language reference manual for this compiler lists the EBCDIC representation of both the 48-character set and the 60-character set.

If two arguments (48 and BCD or 60 and EBCDIC) are specified, either argument may appear first. One or more blanks must separate the arguments.

Example: To specify that the source program is in the 48 character set:

```
*PROCESS CHARSET (48);
```

COMPILE|NOCOMPILE[(W|E|S)]
C|NC|[(W|E|S)]

The COMPILE options control whether the final stage of processing is carried out by the optimizing compiler.

The PL/I optimizing compiler compiles in three stages; preprocessing, syntax checking, and compilation. Preprocessing is the expansion of macro statements, syntax checking is checking that the procedure is syntactically valid PL/I, compilation is the actual production of compiled code. Processing can be stopped after either of the first two stages if errors are found. The COMPILE|NOCOMPILE options can be used to prevent unnecessary processing if errors are found during preprocessing or syntax checking, or to force compilation regardless of errors. Errors are divided into four classes:

W Warning. An error may have occurred.

E Error. An error has been detected but execution may be successful.

S Severe Error. An error has been detected which will prevent successful execution.

U Unrecoverable Error. An error has been detected that prevents further processing by the compiler.

The various COMPILE|NOCOMPILE options have the following meanings:

NOCOMPILE No compilation in any circumstances.

NOCOMPILE(W) No compilation if a warning, error, severe error, or unrecoverable error is detected.

NOCOMPILE(E) No compilation, if error, severe error, or unrecoverable error is detected.

NOCOMPILE(S) No compilation if a severe error or unrecoverable error is detected.

COMPILE Compilation will proceed regardless of errors found except unrecoverable errors.

Example: To prevent compilation if an error of severity E is found:

```
*PROCESS NOCOMPILE (E);
```

CONTROL('password')

The CONTROL option specifies that any compiler options deleted when the compiler was installed are to be available for this compilation. You must still specify the appropriate keywords to use the options. The CONTROL option must be specified with a password that is established for each installation; use of an incorrect password will cause processing to be terminated.

'password' is a character string, not exceeding eight characters in length.

Example: To specify that the deleted option DUMP applies to this compilation where the password is SESAME:

```
*PROCESS CONTROL ('SESAME') DUMP;
```

COUNT|NOCOUNT
CT|NCT

The COUNT option specifies that code will be generated that causes a count to be kept of the number of times each statement is executed in a particular run of a program. The results are written as a table to SYSLST when the program terminates.

Programs compiled with the COUNT option require SYSLST to be assigned to a suitable device when they are executed. For example:

```
assgn syslst printer
```

The COUNT option requires that the GOSTMT option is also specified. If it is not, a message is generated during execution and a statement frequency count table is not produced.

Example: To compile a program that will generate a statement frequency count table:

```
*PROCESS COUNT GOSTMT;
```

DECK|NODECK
D|ND

The DECK option on CMS specifies that the compiler is to produce TEXT file and write it to the file or device assigned to SYSPCH. Columns 73-76 of each card contain a code to identify the object module; this code comprises the first four characters of the first label in the external procedure represented by the object module. Columns 77-80 contain a 4-digit decimal number: the first card is numbered 0001, the second 0002, and so on.

Note that under CMS, the object program is written to SYSPCH and DECK must be specified if an object program is required. This differs from DOS/VS practice where the object program is written to SYSLNK and the DECK option results in an additional copy of the object program being generated. Under DOS/VS, NODECK is the recommended default.

Example: To specify that an object module be produced:

```
*PROCESS DECK;
```

DUMP|NODUMP
DU|NDU

The DUMP option specifies that the compiler is to produce a formatted dump of main storage if the compilation terminates abnormally (usually due to an I/O error or compiler error). This dump is written on the file associated with ddname SYSPRINT. Dump has a number of suboptions useful to those dealing with the internals of the compiler. Details of the suboptions of DUMP are given in the DOS PL/I Optimizing Compiler: Program Logic.

Example: To produce a dump after abnormal termination of the compiler:

```
*PROCESS DUMP;
```

DYNBUF|NODYNBUF

The DYNBUF option controls how the storage for file buffers is acquired. If DYNBUF is specified, the storage is acquired dynamically during execution. If NODYNBUF is specified, the storage is acquired during compilation and becomes part of the object module generated by the compiler.

The advantage of specifying DYNBUF is that the storage for buffers is only acquired when needed. Thus DYNBUF can reduce the object module storage requirements and, provided that all files are not open at one time, the execution time storage requirements.

The advantage of specifying NODYNBUF is that it saves the time overhead involved in acquiring buffer storage during execution.

When DYNBUF is specified, it pays to open and close files as they are needed so that space is available for some other use when it is not required for file buffers. If NODYNBUF is specified, the buffer space forms a permanent part of the object module and there is no advantage in opening and closing files as they are required. The advantage lies in opening and closing all files at once, because this is quicker than opening and closing files individually.

Example: To specify that buffers for files will be acquired dynamically during execution of the compiled program:

```
*PROCESS DYNBUF;
```

ESD|NOESD

The ESD option specifies that the external symbol dictionary

(ESD) is to be listed in the compiler listing.

Example: To produce a listing of the external symbol dictionary:

```
*PROCESS ESD;
```

FLAG(I|W|E|S)
F(I|W|E|S)

The FLAG option specifies the minimum severity of error for which a message is to be listed in the compiler listing. The format of the FLAG option is:

FLAG(I)	List all messages.
FLAG(W)	List all except informatory messages. (If you specify FLAG, with no argument FLAG(W) is assumed.)
FLAG(E)	List all except warning and informatory messages.
FLAG(S)	List only severe error and unrecoverable error messages.

Example: To specify that all messages including informatory messages be listed:

```
*PROCESS FLAG(I);
```

FLOW(n, m)|NOFLOW

The FLOW compiler option specifies that code will be produced enabling the transfers of control most recently executed in a program to be listed when an ON statement with the SNAP option, or when a CALL PLIDUMP statement is executed. This enables you to follow the path through the most recently executed statements when an error occurs during execution. The format of the FLOW option is:

FLOW(n, m)

n	is the number of transfers of control that will be listed with associated statement numbers.
m	is the number of transfers of control between procedures that will be listed with associated procedure names.

n and m must be decimal integers and may not exceed 32767. If either value is zero, the associated listing will not be produced.

The list will start with the earliest available information and continue to the point where the CALL PLIDUMP statement or the ON statement with the SNAP option was executed.

Programs compiled with the FLOW option require SYSLST to be assigned when they are executed.

For example: assgn syslst printer

Example: To specify that a flow trace will be kept containing 50 branches between statements and 20 branches between procedures:

```
*PROCESS FLOW (50, 20);
```

GOSTMT|NOGOSTMT
GS|NGS

The GOSTMT option specifies that statement numbers from the source program will be included in execution-time error messages. Alternatively, these statement numbers can be derived by using the offset address, and the table produced by the OFFSET option. The offset address is always included in execution-time messages.

The GOSTMT option results in the compiler generating a statement number table and thus has a space overhead in the object module.

Example: To specify that statement numbers will be included in execution time messages:

```
*PROCESS GOSTMT;
```

INCLUDE|NOINCLUDE
INC|NINC

The INCLUDE option specifies that %INCLUDE statements are to be handled without the overhead of using the full preprocessor facilities. If preprocessor statements other than %INCLUDE are used in the program, the MACRO option must be used.

The INCLUDE option will be overridden if the MACRO option is also specified.

Example: To specify that text is to be included but that no other preprocessor facilities are required:

```
*PROCESS INCLUDE;
```

INSOURCE|NOINSOURCE
IS|NIS

The INSOURCE option specifies that the compiler is to include a listing of the source program (including preprocessor statements) in the compiler listing. This option is applicable only when the preprocessor is used, therefore the MACRO option must also apply.

Example: To specify that a listing showing the source program before preprocessing is to be generated:

```
*PROCESS INSOURCE MACRO;
```

LIMSCONV|NOLIMSCONV
LSC|NLSC

The LIMSCONV option specifies that the compiled program will not have to handle certain types of conversion in data- or

list-directed input. This reduces the size of the object module produced because the modules to handle such input need not be included in the executable program phase.

If the LIMSCONV option is in effect, only the following types of input are allowed for the variable types shown:

Bit (or character containing bit strings) to bit variable.

Character to character or picture character variable.

Fixed- or floating-point decimal constants (or character strings that represent such constants) to arithmetic variable.

Thus all the usual conversions are allowed, and only the more unusual forms which are allowed by PL/I but seldom used are prohibited. If one of the prohibited types of input is found by a program compiled with the LIMSCONV option, the CONVERSION condition is raised and an on-code generated. The on-codes are listed in the language reference manual for this compiler. The LIMSCONV option only affects programs that contain list- or data-directed input.

Example: To limit the types of input for list- or data-directed stream I/O to those listed above, and thus reduce the size of the executable program by preventing the linking of unnecessary library modules:

```
*PROCESS LIMSCONV;
```

LINECOUNT(n)|LINECOUNT(55)
LC(n)

The LINECOUNT option specifies the number of lines to be included in each page of the compiler listing, including heading lines and blank lines. The format of the LINECOUNT option is:

```
LINECOUNT(n)
```

where "n" is the number of lines. It must be in the range 1 through 32767, but if you specify less than 7, only the heading of the listing will be printed.

Example: To specify that compiler listings will be written 30 lines to a page:

```
*PROCESS LINECOUNT (30);
```

LINK|NOLINK(E|W|S)

The LINK|NOLINK options are intended for use on non-terminal systems and have no effect on CMS/DOS.

LIST[(m, n)]|NOLIST

The LIST option specifies that the compiler is to include a listing of the compiled code (in a form similar to IBM System/360 assembler language instructions) in the compiler listing.

The values m and n allow you to specify the range of statements for which the list will be produced. If m and n are omitted the complete program is included in the listing.

Example: To specify that a listing of compiled code from statement 10 through statement 20 is to be generated:

```
*PROCESS LIST(10, 20);
```

MACRO|NOMACRO
M|NM

The MACRO option specifies that the source program is to be processed by the preprocessor. This option should only be used when preprocessor facilities other than inclusion are required. For inclusion, the INCLUDE option provides better performance.

Example: To specify that the program is to be processed by the preprocessor:

```
*PROCESS MACRO;
```

MAP|NOMAP

The MAP option specifies that the compiler is to produce tables showing the organization of the static storage for the object module. These tables consist of a static internal storage map and the static external control sections. The MAP option is normally used with the LIST option.

Use of the MAP option also results in the generation of a variables offset map which lists static internal and automatic variables with the offsets from their defining bases. This simplifies finding variables in a dump.

Example: To specify that a listing of static storage and a variable offset map be produced:

```
*PROCESS MAP;
```

MARGINI('c')|NOMARGINI
MI('c')|NMI

The MARGINI option specifies that the compiler is to indicate the position of the margins by including in the listings of the PL/I program a specified character in the column preceding the left-hand margin, and in the column following the right-hand margin. Any text in the source input which precedes the left-hand margin will be shifted left one column, and any text that follows the right-hand margin will be shifted right one column. Thus the text outside the source margins can be easily detected. The MARGINI option applies to both the SOURCE and INSOURCE listings.

The MARGINI option has the format:

```
MARGINI('c')
```

where "c" is the character to be printed as the margin indicator.

Example: To specify that the margins of the source program are

to be marked with an @ in the compiler listings:

```
*PROCESS MARGINI('@');
```

MAR(m, n [,c])
MARGINS(m,n[,c])
IBM Default: MARGINS(2,72)

The MARGINS option specifies which part of each compiler input record will be scanned by the compiler for PL/I statements, and the position of any ANS control character used to format the listing.

The format of the MARGINS options is:

MARGINS(m,n,c)

where:

m is the column number of the leftmost column that will be scanned by the compiler. m must not exceed 100.

n is the column number of the rightmost column that will be scanned by the compiler. n must not be less than m, nor greater than 100.

c is the column of the ANS printer control character. It must not exceed 100 and it must be outside the values specified for m and n. A value of 0 for c indicates that no ANS control character is present. The control character applies only to listings on a line printer. Only the following control characters can be used:

(blank) Skip one line before printing.

0 Skip two lines before printing.

- Skip three lines before printing.

+ Skip no lines before printing.

1 Start new page.

Any other character is taken to be blank. If the value c is greater than the maximum length of a source statement record, the compiler will not be able to recognize it; consequently the listing will not have the required format.

Example: To specify that the source program is contained in columns 1 to 70 of the input file and a printer control character appears in column 80:

```
*PROCESS MARGINS(1,70,80);
```

MDECK|NOMDECK
MD|NMD

The MDECK option specifies that the preprocessor is to produce a copy of its output (see MACRO option) and write it to the file defined by SYSPCH. The MACRO option produces 84 byte records; however, the last four bytes, which contain sequence numbers, are ignored for the output from MDECK option. Thus

MDECK allows you to retain the output from the preprocessor as a deck of 80-column punched cards.

Example: To specify that a copy of preprocessor output is to be written onto the virtual card punch:

```
*PROCESS MDECK;
```

NAME('object-module-name')
N('object-module-name')

The NAME option should not be used on CMS/DOS.

NEST|NONEST

The NEST option specifies that the listing resulting from the SOURCE option will indicate, for each statement, the begin-block level and the do-group level; thus displaying the program structure. The levels are shown by numbers in the left hand margin.

Example: To specify that the source listing will contain indications of begin-block and do-group level:

```
*PROCESS NEST SOURCE;
```

OFFSET|NOOFFSET
OF|NOF

The OFFSET option specifies that the compiler is to print a table of statement numbers for each procedure with their offset addresses relative to the primary entry point of the procedure. This table can be used to identify a statement from an execution-time error message if the GOSTMT option is not in effect.
Example: To specify that a table associating statement numbers and offsets in compiled code is to be generated in the listing file:

```
* PROCESS OFFSET;
```

OPTIMIZE(TIME|0|2)|NOOPTIMIZE
OPT(TIME|0|2)|NOPT

The OPTIMIZE option specifies the type of optimization required:

NOOPTIMIZE specifies fast compilation speed, but inhibits optimization for faster execution and reduced main storage requirements.

OPTIMIZE (TIME) specifies that the compiler is to optimize the machine instructions generated to produce a very efficient object program. A secondary effect of this type of optimization can be a reduction in the amount of main storage required for the object module. The use of OPTIMIZE(TIME) can result in a substantial increase in compile time over NOOPTIMIZE.

OPTIMIZE(0) is the equivalent of NOOPTIMIZE.

OPTIMIZE(2) is the equivalent of OPTIMIZE(TIME).

The language reference manual for this compiler includes a full discussion of optimization. OPTIMIZE will be accepted if spelled in the English manner: OPTIMISE.

Example: To specify that the compiled code will be optimized for the best performance:

```
*PROCESS OPTIMISE;
```

OPTIONS|NOOPTIONS
OP|NOP

The OPTIONS option specifies that the compiler is to include in the compiler listing a list showing the compiler options used during this compilation. This list includes those options applied by default, and those specified in a *PROCESS statement.

Example: To specify that a listing will be produced showing all options used for a compilation:

```
*PROCESS OPTIONS;
```

SIZE(yyyyyy|yyyK|MAX)
SZ(yyyyyy|yyyK|MAX)

The SIZE option can be used to limit the amount of main storage used by the compiler. The SIZE option can be expressed in three forms:

SIZE(yyyyyy)	Specify that the compiler should attempt to obtain YYYYYY bytes of main storage for compilation. Leading zeros are not required.
SIZE(yyyK)	Specify that the compiler should attempt to obtain YYYK bytes of main storage for compilation (1K=1024). Leading zeros are not required.
SIZE(MAX)	Specify that the compiler should attempt to obtain as much main storage as it can.

The IBM default, and the most usual value to be used, is SIZE(MAX). This permits the compiler to use as much main storage in the partition or region as it can.

When a limit is specified, the amount of main storage used by the compiler depends on how the operating system has been generated, and the method used for storage allocation. The compiler assumes that buffers, data management routines, and processing phases take up a fixed amount of main storage, but this amount can vary unknown to the compiler.

Example: To specify that the compiler will operate in approximately 100K:

```
*PROCESS SIZE (100K);
```

SOURCE|NOSOURCE
S|NS

The SOURCE option specifies that the compiler is to include in the compiler listing a listing of the source program. The source program listed is either the original source input or, if the MACRO or INCLUDE option applies, the output from the preprocessor.

Example: To produce a listing of the source program:

```
*PROCESS SOURCE;
```

STORAGE|NOSTORAGE
STG|NSTG

The STORAGE option specifies that the compiler is to include in the compiler listing a table giving the main storage requirements for the object module.

Example: To specify that a table giving main storage requirements for the program will be generated:

```
*PROCESS STORAGE;
```

SYNTAX|NOSYNTAX[(W|E|S)]
SYN|NSYN[(W|E|S)]

The SYNTAX options control whether the compiler is to continue into syntax checking after initialization (or after preprocessing if the MACRO option applies).

The PL/I optimizing compiler compiles in three stages; preprocessing, syntax checking, and compilation. Preprocessing is the expansion of PL/I macro statements, syntax checking is checking that the program is syntactically valid PL/I, compilation is the actual production of compiled code. Processing can be stopped after preprocessing or initialization, either unconditionally, or if a certain level of error is found. Alternatively, syntax checking can be forced regardless of any errors (except unrecoverable errors) found in preprocessing, or compiler initialization.

Errors are divided into four classes:

W Warning. An error may have occurred.

E Error. An error has been detected but execution may be successful.

S Severe Error. An error has been detected which will prevent successful execution.

U Unrecoverable Error. An error has been detected that prevents further compilation.

The various SYNTAX options have the following effects:

NOSYNTAX The compiler will not continue into the syntax checking phase.

NOSYNTAX(W) No syntax checking if a warning, error, severe

error, or unrecoverable error is detected.

NOSYNTAX(E) — No syntax checking if an error, severe error, or unrecoverable error is detected.

NOSYNTAX(S) — No syntax checking if a severe error or unrecoverable error is detected.

SYNTAX — The compiler will carry out syntax checking phase regardless of any errors (apart from unrecoverable errors) found in preprocessing.

If the SOURCE option applies, the compiler will generate a source listing even if syntax checking is not performed.

The use of this option can prevent wasted runs when debugging a PL/I program that uses the preprocessor.

Example: To prevent syntax checking if a severe error is found in preprocessing:

```
*PROCESS NOSYNTAX(S);
```

WORKFILE(direct-access-storage-device-type)
IBM default: WORKFILE(2311)

The WORKFILE option specifies the type of direct access storage device that will be used by the compiler for work files during compilation. The WORKFILE option is intended to be used where a non-standard device type is required for a particular compilation.

When the WORKFILE option is used, the symbolic device names SYS001 and SYS002 must be assigned the channel and devices used, and an ASSGN and a DLBL command must be issued to define the data sets for each workfile. The file names used in the commands must be IJSYS01 and IJSYS02. The amount of space required for the data sets is described in DOS: PL/I Optimizing Compiler: System Information.

Optimum compilation speed is achieved if SYS001 and SYS002 are on different volumes with full cylinders allocated to each data set. If only one volume is available, SYS001 and SYS002 should use a split-cylinder extent allocation with the cylinders divided equally between the data sets.

The size and total number of records written by the compiler onto these data sets is listed at the end of the compilation; it varies widely according to the size and nature of the source program and the amount of main storage available. However, 250K bytes of storage for each data set should be sufficient for compiling programs containing up to 500 source statements.

Example: To specify that the compiler is to use a 2314 direct access storage device for its workfiles:

```
*PROCESS WORKFILE(2314);
```

Note that a number of ASSGN and DLBL commands would also be required to achieve the result. See above.

XREF|NOXREF

The XREF option specifies that the compiler is to include in the compiler listing, a list of all identifiers used in the PL/I program, together with the numbers of the statements in which they are declared or referenced. This is known as a cross-reference listing.

(Label reference on END statements are not included. For example, assume that statement number 20 in the procedure PROC1 is END PROC1;. In this situation statement number 20 will not appear in the cross reference listing for PROC1.)

If both ATTRIBUTES and XREF apply, the two listings are combined into one table.

Example: To specify that an attribute and cross-reference listing will be produced:

```
*PROCESS ATTRIBUTES XREF;
```

Appendix A: An EXEC Procedure for the PL/I User

EXEC procedures are sets of CMS commands that are held in an EXEC file and executed by specifying the name of the file. They simplify carrying out repetitive tasks and are well suited to the job of setting up the assignments and DLBL commands necessary to compile and execute a DOS PL/I program.

If you use CMS only for developing and executing DOS PL/I programs, the EXEC procedure shown below could be adopted as all or part of, your PROFILE EXEC. If you only occassionally use DOS PL/I, the EXEC procedure could be given a suitable name, stored, and executed when required.

Creating an EXEC Procedure

An EXEC procedure is created using the Editor, specifying the filetype EXEC. You use the input mode of the editor to key in the input, just as you do with PLIOPT files. For example, to create an EXEC procedure called PLDOS you would enter:

```
edit pldos exec
NEW FILE:
EDIT:
input
INPUT:
(You would enter the EXEC procedure here)
```

When you had completed the procedure you would leave the input mode by entering a null line and, when you had corrected any errors, file it with a FILE subcommand. It could then be executed by specifying the name PLDOS.

The name you choose for your EXEC file should not be the name of a CMS command. If it is, it will be impossible to execute the command until the EXEC procedure is either renamed or deleted.

The EXEC procedure shown below sets DOS on, issues the ASSGN and DLBL commands for the PL/I libraries, and the GLOBAL command for a DOSLIB. It also spools SYSLST to your virtual reader so that any execution time error messages will be available for reading at the terminal. When this EXEC procedure has been executed, it will be possible to compile simply by issuing the DOSPLI command, and to execute by issuing the FETCH and START commands. Thus by use of this or a similar EXEC procedure, you can free yourself to concentrate on the PL/I program itself, rather than being concerned with the mechanics of running it.

The example below is fully commented so that you can choose which parts of the procedure are suitable for you. Comments are the lines starting with an asterisk and can be omitted from the procedure you use. EXEC procedures are a powerful tool. They can be passed arguments, and execution of commands can be made conditional upon the success of previous operations or on other factors. Full details are given in the VM/370: CMS User's Guide. It should be noted that the DOSPLI command is itself an EXEC procedure. Consequently, if you wish to specify DOSPLI in an EXEC procedure of your own, you must specify:

```
EXEC DOSPLI
```

Example of a suitable EXEC

```
* DEVICE 350 IS ASSUMED TO CONTAIN DOS/VS SYSRES
CP LINK SYSTEM 350 350 RR ALL
ACCESS 350 F
* DEVICE 353 IS ASSUMED TO CONTAIN A PRIVATE RELOCAT LIBRARY
* CONTAINING PL/I RESIDENT LIBRARY MODULES NECESSARY DURING LINK EDITING
CP LINK SYSTEM 353 353 RR ALL
* DEVICE 352 IS ASSUMED TO BE A PRIVATE CORE IMAGE LIBRARY
* CONTAINING THE PL/I TRANSIENT LIBRARY, NECESSARY DURING EXECUTION
CP LINK SYSTEM 352 352 RR ALL
* SET DOS ON WITH MODE OF DISK CONTAINING DOS SYSRES
SET DOS ON F
* ASSIGN SYSIPT TO CMS DISK THAT WILL CONTAIN SOURCE PROGRAM
ASSGN SYSIPT A
* ISSUE ACCESS; ASSGN, AND DLBL COMMANDS FOR PL/I LIBRARIES
ACCESS 352 E
ACCESS 353 G
ASSGN SYSCLB E
ASSGN SYSRLB G
DLBL IJSYSCL E DSN PRIVAT CORE IMAGE LIB ( SYSCLB PERM
DLBL IJSYSRL G DSN PRIVAT RELOCAT LIB ( SYSRLB PERM
* ISSUE GLOBAL FOR DOSLIB USED. NOTE THAT THIS DOSLIB SHOULD BE
* SPECIFIED IN DOSLKED COMMAND
GLOBAL DOSLIB YOURLIB
* SPOOL PRINTED OUTPUT TO READER SO THAT EXECUTION TIME ERROR MESSAGES
* WILL BE ACCESSIBLE FROM TERMINAL
ASSGN SYSLST PRINT
CP SPOOL PRINTER TO *
```

Index

```
Reference    Terminal Printout                    Comments
Line
-------------------------------------------------------------------------------
35   *The source program is now created, and is ready to compile, link edit
36   *and execute.
37   access 390 g                          Access the DOS SYSRES volume as one
38   G (350) R/O - DOS                     of your CMS disks.
39   R;
40   set dos on g                          Activate CMS/DOS environment.
41   R;
42   assgn sysipt a                        Assign SYSIPT to CMS disk containing
43   R;                                    source program.
44   dospli bunny                          Enter compilation command.
45   R;
46   edit bunny listing                    Inspect listing file for results.
47   EDIT
48   locate/compiler diagnostic messages   Look for messages.
49   NOT FOUND                             Not found so compilation OK.
50   EOF
51   quit                                  Leave the edit mode.
52   R;
*******************************************************************************
     * The program has now been compiled and a TEXT file created which
53   * can be link-edited with PL/I resident library modules so that an
54   * executable phase can be created on a DOSLIB library.
55   assgn sysrlb b                        Assign PL/I resident library.
56   R;
57   dlbl ijsysrl b dsn privat relocat lib(sysrlb perm
58   R;                                    DLBL for resident library.
59   doslked bunny mylib                   Link edit the TEXT file onto a DOSLIB
60   R;                                    library called MYLIB.
*******************************************************************************
     * The source program is now link edited and is ready for execution.
61   * it can be retained in this form.
62   assgn sysclb b                        Assign the PL/I transient library.
63   R;
64   dlbl ijsyscl b dsn privat corim lib (sysclb perm
65   R;                                    DLBL for transient library.
66   global doslib mylib                   Specify doslib to be searched for
67   R;                                    program phase specified in FETCH.
68   assgn syslst printer                  Assign SYSLST so that error
69   R;                                    messages will be available.
70   spool printer to *                    Spool error messages to reader, so that
71   R;                                    they are available at terminal.
72   fetch bunny (start                    Start the execution.
73   EXECUTION BEGINS...                   Message from CMS.
74   WHAT'S UP DOC?                        Display message from program because
75   R;                                    terminal is virtual console.
76   print bunny listing                   Print the listing file to retain record.
77   R;
78   erase bunny *                         Erase files created by program
79   R;                                    to save space.
80   logoff                                End the terminal session.
81   CONNECT= 00:40:24 VIRTCPU=000:03:.95 TOTCPU= 000:22:09
82   LOGOFF AT 09:30:21 GMT 07/04/75       Message when you log off.
```

Figure F.1. Sample Terminal Session

SAMPLE TERMINAL SESSION

Reference Line	Terminal Printout	Comments
	Programmer input in lower case. System response in upper case.	
1	VM/370 ONLINE	Message when you switch on.
2	logon patti	Logon with name of virtual machine.
3	ENTER PASSWORD:	
4	(password entered here)	Printing normally suppressed.
5	LOGMSG 09/12/00 07/04/75	Message when you logon.
6	(other data depending on your installation)	
7	ipl cms	You request CMS to be loaded.
8	CMS 3.0 PLC 0 WEDNESDAY 04/07/75	Shows version of CMS loaded.
9	* CMS has been entered and the file containing the PL/I program	
10	*can now be created. Note that lines starting with * are comments in CMS	
11	edit bunny pliopt	EDIT command naming PLIOPT file
12	NEW FILE	to be created.
13	EDIT	
14	preserve	Alter tabs to enter *PROCESS
15	tabs 1	statement, which must start in column
16	input	one. By default all input to PLIOPT
17	INPUT	files is moved on column right.
18	*PROCESS OPTIMIZE(TIME);	Enter *PROCESS statement.
19		Null line(carriage return only)
20	EDIT	to return to edit mode.
21	restore	Restore standard PLIOPT tabs.
22	input	Return to input to enter rest of program.
	INPUT	Message shows you are in input mode.
23	bunny:proc options(main);	Start of program. Column one can be used.
24	display('what''s down@@@@up doc?');	"@@@@" are line editing symbols. See notes.
25	end;	
26		Null line to return to edit mode.
27	type	Edit subcommand to get file contents
28	*PROCESS OPTIMIZE (TIME);	displayed.
29	BUNNY:PROC OPTIONS(MAIN);	
30	DISPLAY('WHAT''S UP DOC?');	Note that input has been moved one column
31	END;	right and changed to upper case (see notes).
32	EOF	End of file reached.
33	file	File the program on your CMS A-disk.
34	R;	Ready message. See notes.

Explanation of Programming Example

The diagram shows a terminal session that covers the complete process of keying-in, compiling, link-editing, and executing a PL/I program under CMS/DOS. In the figure, the numbers on the left are used as references to the notes below, which either provide fuller explanations than there is room for on the figure, or point to the section of the book where further information is available. The Contents List at the start of the book will enable you to find these sections.

It is intended that this figure should be used as an overview by those who are familiar with terminal systems, and, as such, it aims to highlight the important features of using PL/I under CMS/DOS. The figure can also be used as an aide-memoire to remind you of the steps involved in running a DOS PL/I program. The PL/I program has been deliberately kept trivial so that it will not obscure the points being made about CMS/DOS.

Notes

Line Number	Comments
1-4	For information on logging-on see "Starting the Session - the LOGON command" in chapter 1.
1	On typewriter-type terminals, such as the IBM 2741, it may be necessary to press the attention key or its equivalent to unlock the keyboard.
6-8	For information on loading CMS see "Invoking CMS - the IPL Command" in chapter 1.
11-34	For information on keying-in and filing your PL/I program see "Entering the Program - the EDIT and FILE Commands" in chapter 1.
14-19	For information on using the *PROCESS statement, see "Use of non-default Compiler Options" under "Entering the Program - the EDIT and FILE commands" in chapter 1.
18	For information on OPTIMIZE and other DOS PL/I Optimizing Compiler options see chapter 3.
24	"@@@@" deletes the previous four characters. For information on this and other logical line editing symbols see "Line Editing Symbols" under "Invoking CMS - the IPL Command". For information on the effects of the PL/I DISPLAY statement under CMS see "Conventions" in chapter 2.
30,31	PLIOPT files have their input moved one column to the right because PL/I default margins are 2 through 72. Thus the movement obviates the need to key in the leading blank before PL/I statements. By default, all input is translated to upper case. To override this see "Lowercase Character String Constants" under "Entering the Program - The EDIT and FILE Commands" in chapter 1.
34	The Ready message may take the form shown, or a longer form giving information on CPU usage.
37-52	For information on entering the CMS/DOS environmnet and compiling the program see "Compiling the Program - The DOSPLI Command" in chapter 1.
54-75	For information on link-editing and execution see "Executing a DOS PL/I Program" in chapter 1.
68-71	For information on accessing execution-time error messages see "Accessing Error Messages" under "Executing a DOS PL/I Program" in chapter 1.
78-82	For information on logging off and erasing files see "Ending the Terminal Session - The LOGOFF Command" in chapter 1.

DOS
PL/I Optimizing Compiler:
CMS User's Guide

READER'S COMMENT FORM

Order No. SC33-0051-0

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? _____

Number of latest Technical Newsletter (if any) concerning this publication: _____

Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)