Program Product

# OS
# PL/I Optimizing Compiler:
# CMS User's Guide

Program Numbers 5734-PL1
5734-LM4
5734-LM5

(These program products are available as composite package 5734-PL3)

Fourth Edition (October 1976)

This is a major revision of and obsoletes SC33-0037-2. This edition applies to Version 1, Release 3, Modification 0 of OS PL/I Optimizing Compiler, Program Products 5734-PL1, 5734-LM4, and 5734-LM5, and to any subsequent version, release, and modification.

Changes made for this edition

Information has been included on the facilities that have been introduced with release 3 of the compiler, notably the INTERRUPT option and attention handling. The SHORT and FULL suboptions of XREF and ATTRIBUTES are introduced. In addition, a number of editorial changes and technical corrections have been made throughout the manual.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest IBM System/370 Bibliography, GC20-0001, and the technical newsletters that amend the bibliography, to learn which edition and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California. All comments and suggestions become the property of IBM.

# Preface

This manual explains, for the users of the Conversational Monitor System (CMS) component of the IBM Virtual Machine Facility/370, how to invoke the PL/I Optimizing Compiler and use its conversational I/O capabilities.

The reader is assumed to have a basic knowledge of PL/I 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, and execute 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 optimizing compiler and executed under CMS.

Chapter 3 is the reference source for the PLIOPT command and its options.

Chapter 4 is the reference source for the execution time options that are available when executing programs compiled by the PL/I Optimizing Compiler.

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.

IBM Virtual Machine Facility/370:

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

OS PL/I Checkout and Optimizing Compilers: Language Reference Manual, Order No. SC33-0009-1

OS PL/I Optimizing Compiler: Programmer's Guide, Order No. SC33-0006-1

OS PL/I Optimizing Compiler: Program Logic, Order No. SC33-0006-0

## 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

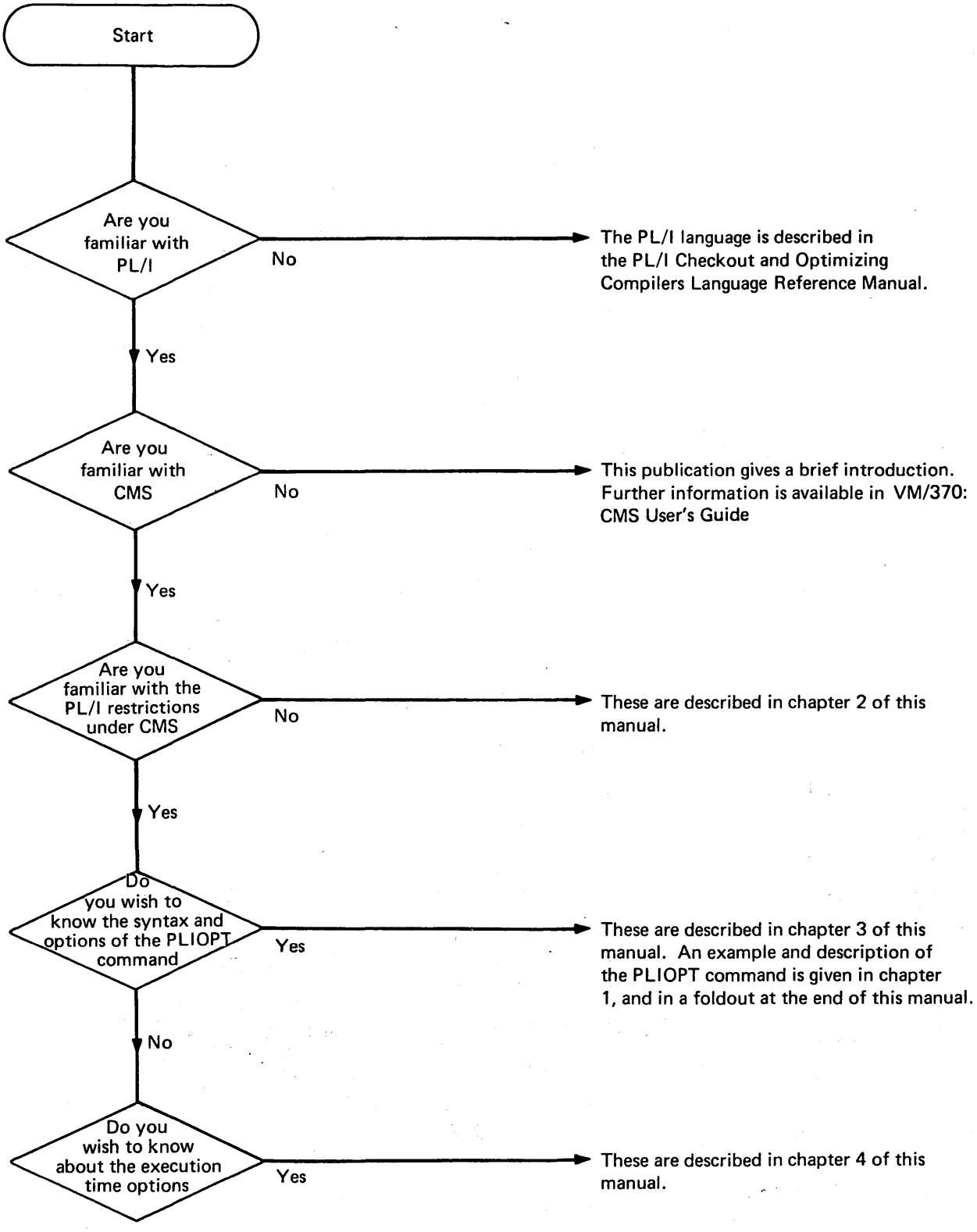



Figure P.1. How to use this book

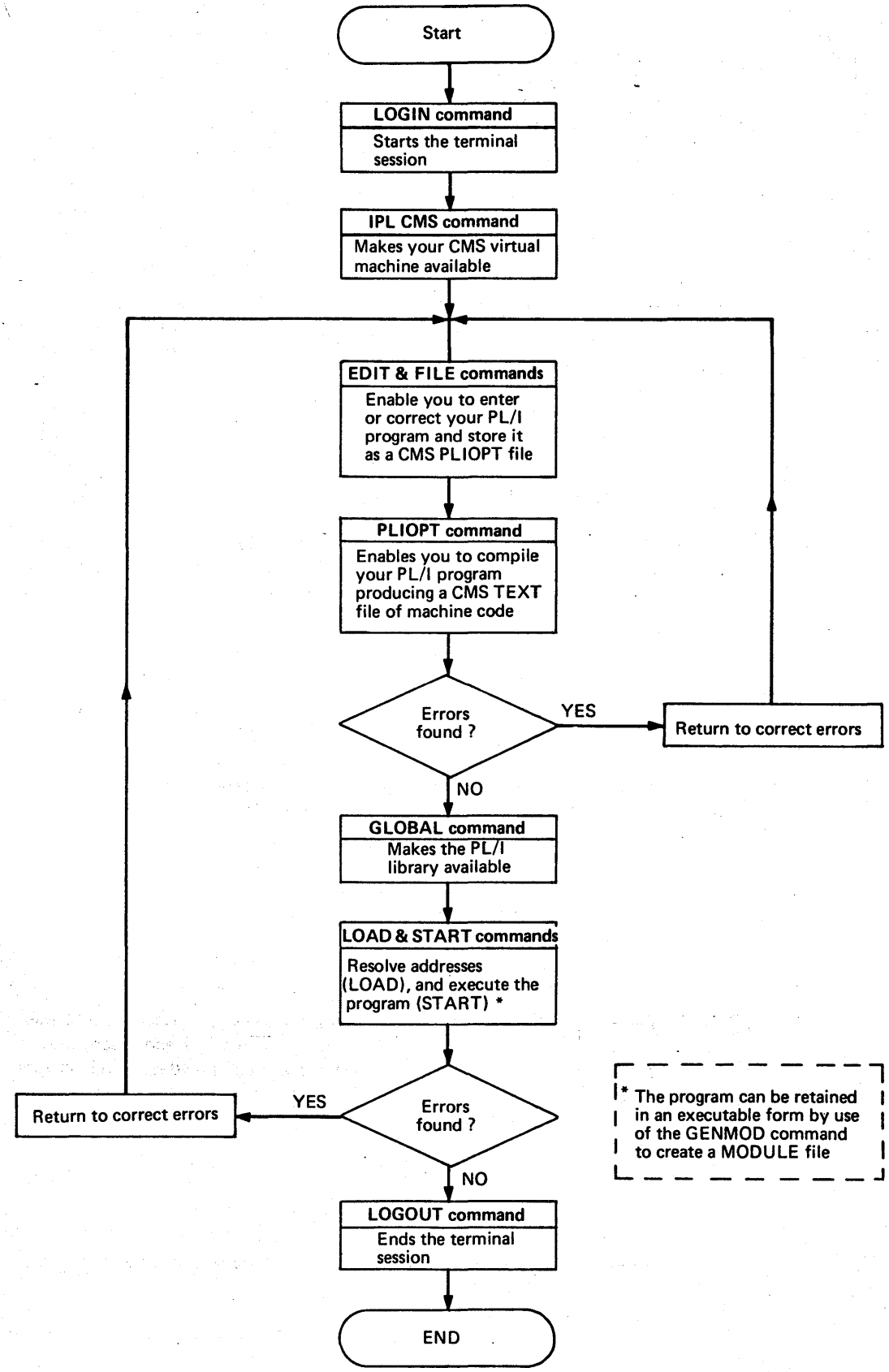



Figure 1.1. The steps involved in entering and executing a PL/I program under CMS

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

## Introduction

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

1. LOGIN at the terminal.
2. IPL CMS.
3. Write or alter a source program using the CMS editor.
4. Compile the source program using the PLIOPT command.
5. Execute the compiled program using the GLOBAL command to access the PL/I libraries followed by the LOAD and START commands. Or create a MODULE file using the GLOBAL, LOAD, and GENMOD commands for subsequent execution without further use of the LOAD command.
6. End the session.

The remainder of this chapter leads you through the steps listed above one by one. A standard approach has been adopted for each step. The format is:

1. Summary and example. These give you the essential information 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 to get the best possible results from using 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 where there are no special cases.
4. Sources of further information. This lists the manuals that you will require for any further information you may need.

A sample terminal session can be folded out from the end of the book. This shows all seven 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 that is compiled by the optimizing compiler and executed under CMS. Chapter 3 lists the options and syntax of the PLIOPT command. Chapter 4 lists the execution time options that are available for programs compiled by the optimizing compiler. The manual was written with users of a 2741 terminal in mind. Certain information has been included for the 3277. For other terminals you may need the terminal manual.

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.

The next page shows you how to start a CMS session.

# 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. (LOGIN is an alias for LOGON and can be used with identical effects.)

Example of use of the LOGON Command

| EXAMPLE OF LOGON | |
|---|---|
| Terminal Printout | Notes and comments |
| (you switch on the terminal) | |
| VM/370 ONLINE<br>(you may have to press attention key to unlock the terminal keyboard.) | Message shows a virtual machine is available. |
| logon skylark | LOGON command followed by identifier for your virtual machine. |
| ENTER PASSWORD:<br>(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 04/02/72<br>RUNNING SYS010 - COLD START AT 09:00<br>LOGON AT 09:13:04 THURSDAY 04/02/72 | 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. | |

## BACKGROUND

### 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 (CP/370). When you have received the log message, you are in control of your virtual machine and said to be in the "CP environment".

## SOURCES OF FURTHER INFORMATION

| Topic | Reference Source |
|---|---|
| LOGON command | VM/370: CP Command Reference for General Users |
| 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 1.0 PLC 5 WEDNESDAY 04/07/72 09.13.50 | Message confirms CMS is invoked and that CMS commands may be entered. |
| Conventions: | |
| 1. A carriage return is assumed after all programmer input. | |
| 2. System response is in upper case (capital) letters, programmer input in lower case. | |

BACKGROUND

Entering 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.

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 characters

VM/370 provides four characters to alter, delete, or split up the line you key in at the terminal. These four characters are known as line editing characters and are @, ¢, #, and " by default. For some terminals ¢ becomes [ or (. They are removed from your input and treated as editing characters unless they are preceded by the escape character (see "Using line editing characters as normal characters" below). The line editing characters 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 character and then press the carriage return key. By default the logical line delete character is ¢. Thus to delete a line you might enter:

this is an example of deleting a line ¢

(¢ becomes [ or ( on some terminals.)

Altering a line: If you wish to alter a line and then transmit it to the system, you must use the logical character delete character, (sometimes called the logical backspace character). By default the logical character delete character is @. If the logical character delete character 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 might enter:

this is an example of altering a wine@@@@line

Many programmers prefer to use the actual backspace key on the terminal as the character delete character. 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 character you must use the terminal command thus:

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

(Note: This cannot be done in EDIT mode.)

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 character. 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

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

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

The escape character can be used preceding itself.

Attention Interrupts. Under CMS, attention interrupts can be used either to halt processing of the current CMS command or to return control to CP. To halt processing of the current CMS command you must reflect an interrupt to your virtual machine. Unless the current command has its own attention processing, this will allow you to key in further commands. Most such commands will be executed when the current command is completed. However, there are a number of immediate commands that will be executed immediately. The most useful are: HT - halt typing, HX - halt execution, and RT - resume typing. If a CMS command is not being executed, an attention interrupt deletes anything on the current line but otherwise has no effect. If you cause an attention interrupt during execution of a PL/I program, the result depends on whether it was compiled with the INTERRUPT option (see the section headed "Compiling the Program - the PLIOPT Command", later in this chapter).

To return to CP, you must reflect an interrupt to CP. When this has been done, you can return to the point where you were executing in CMS

by use of the BEGIN command.

The method of causing attention interrupts varies according to the type of terminal you are using. On a 2741, an interrupt is reflected to your virtual machine by pressing the ATTN key once, and to CP by pressing it twice. On a 3277 you reflect an interrupt to your virtual machine by pressing the ENTER key, first moving the cursor one space to the left if the screen status message reads VMREAD. You reflect an attention interrupt to CP by pressing the PA1 key.

## 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 defining files. These commands are executed by calling the EXEC procedure.) 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.

Information on issuing and writing a PROFILE EXEC is given in the VM/370: CMS User's Guide.

## SOURCES OF FURTHER INFORMATION

| Topic | Reference source |
|---|---|
| CMS background | VM/370: CMS User's Guide |
| IPL command | VM/370: CP Command Reference for General Users |
| PROFILE EXECs | VM/370: CMS User's Guide |
| ACCESS command | VM/370: CMS User's Guide |

# 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 file name of your choice and the file type 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. The facilities for correcting lines before you press the carriage return key are described in the previous section under the heading "Line Editing Characters".

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 PLIOPT command. In addition to creating a file, the FILE subcommand also ends the edit submode.

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 characters and special action may be required if you wish to use them. Also care should be taken not to choose a CMS or CP command as a name, because this can cause problems if you wish to create a module file.) 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.

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 OS then there are no special restrictions on the language you may use.

Special action will be required in the following circumstances

1. If your program uses lower case character string constants.

2. If you wish to use a *PROCESS statement.

3. If you wish to use any of the line editing characters as normal characters in your program. The line editing characters 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" below.

## Examples of Use of the EDIT and FILE Commands

**EXAMPLE OF ENTERING A NEW PROGRAM**

| Terminal Printout | Notes and comments |
|---|---|
| edit rabbit pliopt | The EDIT command followed by file name and file type. |
| NEW FILE: | Message shows that you have no PLIOPT file called "rabbit". |
| EDIT: | Message shows you are in EDIT mode. |
| input | INPUT subcommand causes the INPUT mode to be entered. |
| INPUT: | Message shows you are in INPUT mode. |
| rabbit:proc options (main); | PL/I must appear in columns 1 through 71. |
| display ('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 line pointer at the top of the file. |
| TOF | Message shows pointer is at top of file. |
| type * | Have the contents of the file displayed at the terminal. |
| RABBIT: PROC OPTIONS (MAIN); | |
| DISPLAY ('THE RABBIT SQUEAKS TO THE WORLD'); | |
| END; | |
| FOF | Means end of file reached |
| file | 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 |
|---|---|
| (This example assumes that you are correcting an error on line 10) | |
| 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. |
| next 10 | Position line pointer to incorrect line. |
| PHT EDIT(X)(A); | The tenth line beyond the original line pointer position is displayed. |
| c/ht/ut | CHANGE subcommand. |
| PUT EDIT(X)(A); | Corrected line displayed. For details of CHANGE and other edit subcommands see the VM/370: CMS Command and Macro Reference. |
| file | FILE command 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 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 lowercase.

BACKGROUND

## The EDIT Facility of CMS

The EDIT facility of CMS allows you to create and update sequential files from your terminal. It is used to create PLIOPT or PLI files which can be compiled by the PL/I Optimizing Compiler. (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 EDIT facility has two modes, the EDIT mode and the INPUT mode. The EDIT mode allows you to use various EDIT 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. (If you want a blank line in your PLIOPT file you must, therefore, key in at

least one blank in the line.)

When you issue the EDIT command, you must specify a file name and a file type. CMS searches your disks for the file and if you have such a file brings a copy of it into main storage and types 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 types the message "NEW FILE" and "EDIT". To enter the INPUT mode you must enter the INPUT subcommand.

To return from the EDIT mode to CMS, you must issue a command 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 command or the QUIT command. The FILE command stores the copy of the file you have been creating and discards the previous copy, if any. The QUIT command 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 use the FNAME subcommand to rename the copy of the file on which you are working. You could enter:

```
fname rabbit2
```

Then, when you issued the FILE command, the altered file would be stored with the name rabbit2 and the original file rabbit would still be available.

If you wish to save your input and still remain in EDIT mode you can use the SAVE command.

A full description of the EDIT command and EDIT subcommands is given in the VM/370: CMS Command and Macro Reference.

### 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 which are described in 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. Further information can be found in the VM/370: CMS User's Guide. PLI files are an alternative type of file that can be handled by the optimizing compiler.

## SPECIAL CONSIDERATIONS

### Lowercase Character String Constants

When you are editing a PLIOPT file, the CMS editor automatically translates any lower case characters you enter to upper case. If you wish to enter lower case 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 except comments and data between quotes in upper case.

To return to automatic translation to upper case 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 *PROCESS Statements

Special action is required if you use the *PROCESS statement. This is because the * must appear in column 1 and, by default, the editor moves all input to PLIOPT files one column to the right. Accordingly 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 as a character delete character it must be preceded by the escape character. (See "Line Editing Characters" under "Invoking CMS - the IPL Command" earlier in this chapter.)

## Use of the line editing characters in your program

If you wish to use any of the line editing characters as normal input to your program you must precede them by the escape character. By default, the line editing characters 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 character, preceded by " which is the default escape character, thus:

```
DOCUMENT"#2
```

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

## Creating a file for inclusion by %INCLUDE statement

If you wish to create a file of secondary input text that you will subsequently be able to include in your program by use of the %INCLUDE statement, you will need to create a COPY file and to store it on a macro library by use of the MACLIB command.

Creating a copy file is similar to creating a PLIOPT file, however, data must be typed in columns 2 through 72 if you intend to use the standard PL/I margins. This is necessary because the text is not shifted one column to the right as it is for PLIOPT files. When you have created your copy file and used the FILE command to store it, you will need to issue a MACLIB command to place it on a macro library. The MACLIB command takes the form:

```
MACLIB  ADD file-name macro-library-name
        GEN
```

If you are adding a new file to an existing library you use "ADD" as the second operand. If you are creating the macro library you use "GEN" as the second operand. An example of creating a file of inclusion by the

use of %INCLUDE statements is shown below.

```
        EXAMPLE OF CREATING SECONDARY INPUT TEXT FOR INCLUSION
        BY %INCLUDE STATEMENTS
=============================================================================
edit cuckoo copy            Filetype COPY must be used
NEW FILE:
EDIT:                       Enter EDIT mode
input                       Enter INPUT submode
INPUT:
 DISPLAY('TEST DATA FOR %INCLUDE');   Column 1 must be left blank to
   |                        allow for standard PL/I margins
   |
                            Null line causes return to EDIT mode.
EDIT                        Return from INPUT to EDIT mode.
file                        Store the file.
R;
maclib add mylib cuckoo     Store the file on the macro library mylib.
   |                        If the macro library did not exist, you
   |                        would use "GEN" instead of "ADD" this
   |                        would generate a macro library called
   |                        "mylib".
R;
-----------------------------------------------------------------------------
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 upper case (capital) letters, programmer
    input in lower case.
```

SOURCES OF FURTHER INFORMATION

| Topic | Reference source |
|---|---|
| Format of PLIOPT and PLI files | VM/370: CMS User's Guide |
| Using the VM/370 editor | VM/370: CMS User's Guide |
| Using your terminal | VM/370: Terminal User's Guide |

# Compiling the Program — The PLIOPT Command

SUMMARY

To compile a program under CMS, you use the PLIOPT command followed by the name of the file that contains the source program. If you wish to specify any compiler or PLIOPT options, these must follow the file name and be preceded by a left parenthesis. Options are separated from each other by blanks, the abbreviated forms should always be used.

If the file containing the source program is not a PLIOPT or PLI file, it will be necessary to specify the filetype after the filename. If the file is not on the A disk it will also be necessary to specify the filemode for the file naming the disk where the file is held. Chapter 3 shows the syntax for the PLIOPT command.

During compilation, two new disk files will be produced. They will have the file types TEXT and LISTING and the same file name as the file specified in the PLIOPT command. The TEXT file contains the compiled code. The LISTING file contains the listings produced during compilation. Any error messages produced will be transmitted to your terminal.

If compilation reveals source program errors, you can alter the PLIOPT file that contains the source by use of the CMS editor. You can then reissue the PLIOPT command. This results in the creation of new TEXT and LISTING files corresponding to the newly edited source program. If previous versions were available they will be overwritten. When you have a satisfactory compilation, you can 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 in the following circumstances:

1. If your source uses the %INCLUDE statement to incorporate secondary input text.
2. If your source program is not on a CMS disk.
3. If you intend to execute your program under the control of OS.
4. If you wish to place the compiled program on a text library as you may if you want to use separately compiled subroutines.
5. If you wish to use attention interrupts as an integral part of your program.

The action required is described in the sections below under the heading "Special Considerations."

## Example of Use of the PLIOPT Command

**EXAMPLE OF USE OF THE PLIOPT COMMAND**

| Terminal Printout | Notes and comments |
|---|---|
| pliopt rabbit (xref | The PLIOPT command |
| | 1. Options must appear after a left parenthesis and be separated by blanks. If any exceed 8 characters see "Special Considerations" below. |
| | 2. The right parenthesis is not necessary. |
| | 3. During compilation the system will issue an in-operation signal for every 2 seconds of virtual CPU time used, this is known as the BLIP signal. |
| NO MESSAGES PRODUCED FOR THIS COMPILATION<br>COMPILE TIME 0.01 MINS SPILL FILE 0 RECORDS SIZE 4051 | |
| R; | READY message. If the compiler failed or found errors of severity W or higher, CMS responds R(return code); |

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 upper case (capital) letters, programmer input in lower case.

# BACKGROUND INFORMATION

## Compiler Output and its Destination

When you issue the PLIOPT command, CMS calls the 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 will be executed when you wish to 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.)

By default, the two newly created files will be placed on CMS disks. They will have the same file name as the file that contains the source program but a different file type. The compiled code will have the file type TEXT and the listing will have the file type 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.)

It is possible to specify a name for the TEXT file other than that of the file compiled in the PLIOPT command by specifying a filename with the OBJECT option.

The creation of the LISTING file can be suppressed by use of the NOPRINT option of the PLIOPT command. (See below under "Listing Options".) The creation of the TEXT file can be suppressed by use of the NOOBJECT option of the PLIOPT command.

| 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 by using a CMS FILEDEF command. (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

## Choosing the Information to be sent to your Terminal - Listing Options

Options of the PLIOPT command and other CMS facilities offer you a wide choice in the amount of listing information that can be made available to you at the terminal.

Three factors are relevant:

1. The compiler option TERMINAL which allows you to have sections of the listing printed at the terminal as well as being included in the normal listing file. TERMINAL can be followed by a parenthesized options lists specifying those parts of the listing that you wish to be transmitted to your terminal. Chapter 3 of this manual gives details. By default the TERMINAL option is specified without an options list and compiler diagnostic messages are transmitted to the terminal.

2. The CMS option PRINT|DISK|TYPE|NOPRINT, which allows you to direct the listing file to a printer (PRINT), to a CMS file (DISK...this is the default) to the terminal (TYPE), or to be discarded (NOPRINT).

The TERMINAL and PRINT options are described in chapter 3 of this manual. The FILEDEF command is described in the VM/370: CMS User's Guide.

The CMS defaults are TERMINAL with no options list and DISK. When you have received the messages passed to your file as specified in the

TERMINAL option, you can decide whether to examine the LISTING file using the EDIT mode, to pass it to a printer, or to discard it.

Only one copy of the listing is transmitted to the terminal if you use both the TERMINAL option and assign the listing file to the terminal.

## Compiler Options

The PLIOPT command expects all options to be not more than eight characters long. It is therefore, necessary to use the abbreviated form of certain compiler options such as ATTRIBUTES, and advisable always to use the abbreviated form. All options and sub-options must be separated by blanks. Parentheses need not be separated from options or suboptions even if the option has a length of more than 8 characters. Thus TERMINAL(XREF A) is acceptable, although the total length is greater than 8 characters.

## Files used by the compiler

During compilation the compiler uses a number of files. These files are allocated by the interface module that invokes the compiler. The files used are shown in figure 1.3.

| Name | Function | Device Type | When Required |
|---|---|---|---|
| PLIOPT | System input | DASD, magnetic tape, card reader | Always |
| LISTING | System print | DASD, magnetic tape, printer | Always |
| TEXT | Object module output | DASD, magnetic tape | When object module is to be created |
| SYSPUNCH | System punch | DASD, magnetic tape, card punch | When object module required in card image format |
| SYSUT1 | Spill | DASD | When insufficient main storage available |
| MACLIB | Preprocessor %INCLUDE | DASD | When %INCLUDE is used from CMS disks |
| SYSLIB | Preprocessor %INCLUDE | DASD | When %INCLUDE is used from OS Library |

Figure 1.3. Files that may be used by the compiler

SPECIAL CONSIDERATIONS

## Secondary Input Text - %INCLUDE Statements

If your program uses %INCLUDE statements to include previously written PL/I statements or procedures, the libraries on which they are held must be made available to CMS before issuing the PLIOPT command. To do this you must insert the statements into a CMS MACLIB using the MACLIB command. You then issue a GLOBAL command taking the form "GLOBAL MACLIB filename." For example, if your secondary input text was held in MACLIB called "mylib" you would enter:

```
global maclib mylib
```

before issuing the PLIOPT command. The PLIOPT command must specify either the INCLUDE or the MACRO option.

If your %INCLUDE statement takes the form %INCLUDE MYLIB CUCKOO, as opposed to %INCLUDE CUCKOO, you will also need to specify a FILEDEF command for MYLIB. This should take the form:

```
filedef mylib disk mylib maclib
```

If in the maclib the LRECL is not 80 and the BLOCKSIZE not 400, format information must be included in the filedef command.

## Source Program not on a CMS Disk

If your source program is not held on a CMS disk you can either read or move it to a CMS disk from a card reader, OS disk, or tape using the READCARD or MOVEFILE commands of CMS, or issue a FILEDEF command to define the PL/I source as coming from the reader, OS disk, or tape device and then compile it.

Moving the file onto a CMS disk offers the advantage that the source can subsequently be altered from the terminal. This may be necessary if compilation reveals errors in the source program. The method is given in the VM/370: CMS User's Guide.

To compile a program held on card or tape it is necessary to issue a FILEDEF command before the PLIOPT command. Thus to compile a program held on card you might use the following sequence:

```
FILEDEF PLIOPT READER (LRECL 80 RECFM F

PLIOPT fname (option 1 .... option n)
```

If the source program is on an OS disk, issue the following commands:

```
ACCESS 290 D
FILEDEF PLIOPT DISK D DSN?
    [then key in the data set name]
PLIOPT fname
```

The keywords 'PLIOPT DISK D' in the FILEDEF command tells CMS that the PL/I source statements are to be read in from the user's D-disk.

Any filename can be used for "fname". The name specified will be given to the LISTING, TEXT, and UTILITY files produced by the compiler.

A description of the FILEDEF command is given in the VM/370: CMS Command and Macro Reference.

## Compiling a Program to be Executed Under OS

If you intend to execute your program under OS, you should specify the OSDECK option thus:

```
PLIOPT RABBIT (OSDECK
```

This prevents the compiler from issuing a CMS loader ENTRY command, specifying the CMS execution time interface module. An attempt to execute a program compiled without the OSDECK option under OS, results in an OS linkage editor error of severity level 8.

It is possible to execute a program compiled with the OSDECK option under CMS, but special action is required. See "Executing a File not Compiled Under CMS or Compiled with the OSDECK option" in the following section, "Executing a PL/I Program."

## Compiling a program to be placed on a TEXT library

If you intend to include the compiled TEXT file as a member of a text library it is necessary to use the NAME option when you specify the PLIOPT command. This is because members of a TXTLIB file are given the name of their primary entry point if they have no external name. The primary entry point of every TEXT file produced by the optimizing compiler is the same, consequently only one compiled program can be included in a TXTLIB if the NAME option is not used. (The NAME option gives the TEXT file an external name.)

Commands required to create a TEXT file suitable for including in a TXTLIB is shown below. This code gives the file the external name used in the PLIOPT command. However any other name can be used, provided that it does not exceed six characters. It should ne noted that, if the name exceeds six characters the NAME option will be ignored.

The advantage of using a TXTLIB is that files on the TXTLIB will automatically be linked when a LOAD command is issued if a GLOBAL for the TXTLIB has been issued. Thus separately compiled subroutines are most easily managed if they are held on a TXTLIB.

The commands below compile a PLIOPT file RABBIT with the external name RABBIT and adds it to an existing text library called BIOLIB.

```
pliopt rabbit (name('rabbit'

(compiler messages etc)

txtlib add biolib rabbit
```

If the TXTLIB did not yet exist GEN would be used instead of add.

## The INTERRUPT Option and Attention Interrupts

If you have written a program that relies on the raising of the ATTENTION condition, it must be compiled with the INTERRUPT option, for example:

```
     PLIOPT BLUETIT (INT
```

Full details of how to use the INTERRUPT option and attention interrupts are given in the next section under the heading "Attention Interrupts".

SOURCES OF FURTHER INFORMATION

| Topic | Reference source |
|---|---|
| Error message explanations<br>CMS (numbered DMSxxxx) | VM/370: System Messages |
| PL/I (numbered IELxxxx) | PL/I Optimizing Compiler Messages |
| FILEDEF command | VM/370: CMS Command and Macro Reference |
| GLOBAL command | VM/370: CMS Command and Macro Reference |
| MOVEFILE command | VM/370: CMS Command and Macro Reference |
| PL/I language | PL/I Checkout and Optimizing Compilers Language Reference Manual |
| PLIOPT command | Chapter 3 of this manual |
| READCARD command | VM/370: CMS Command and Macro Reference |
| TXTLIB command | VM/370: CMS Command and Macro Reference |

# Executing a PL/I Program

SUMMARY

To execute a PL/I program under CMS, you must have either a CMS TEXT file or a CMS MODULE file. If your program is not in either of these forms, see the earlier sections of this chapter. (A MODULE file is created by using the LOAD command to resolve addresses in a TEXT file. Details are given below.)

If you have a TEXT file execution requires three steps:

1. Issuing a GLOBAL command for the PL/I libraries.
2. Issuing the LOAD command with the START option if you wish execution to begin.
3. If the START option was not issued with the LOAD command, issuing the START command.

These steps are shown in example 1 below.

If you have a MODULE file execution requires 2 steps:

1. Issuing a GLOBAL command for the PL/I libraries.
2. Issuing the filename as a CMS command.

These steps are shown in example 3 below.

To create a MODULE file, you issue the GENMOD command after issuing the GLOBAL and LOAD commands. Because of the structure of the the PL/I object program, you must include the operand (FROM PLISTART and if you do not include a filename, the resulting module file will be called DMSIBM. To produce a MODULE file you must therefore issue a GENMOD command of the form:

GENMOD filename (FROM PLISTART

where filename is the file name of your choice.

The PL/I standard files, SYSIN, SYSPRINT, and PLIDUMP are automatically assigned before the PL/I program begins execution. SYSIN and SYSPRINT are assigned to the terminal, and PLIDUMP is assigned to a printer. If you wish to override these assignments you must issue FILEDEF commands before the start of execution. See "PL/I Files and CMS Defaults" below.

Special action will be required in the following circumstances:

1. If you wish to pass parameters to your program.
2. If your program uses any PL/I files that do not match the CMS default definitions.
3. If you wish to execute a program that was compiled under OS, or was compiled under CMS with the OSDECK option.

The action required is described in the sections below under the heading "Special Considerations."

## Examples of Executing a PL/I Program

**EXAMPLE 1. EXECUTING A TEXT FILE**

| Terminal printout | Notes and comments |
|---|---|
| global txtlib plilib | GLOBAL command makes the PL/I libraries available. |
| R; | READY message. |
| load rabbit | LOAD command generates an executable program in main storage from the TEXT file. (An alternative is LOAD RABBIT (START, if you want immediate execution). |
| R; | READY message. |
| start | START command starts execution<br>1. If you wish to pass parameters, follow "start" with a blank, an asterisk, another blank, and then the parameters; thus: start * / 123. See "Special Considerations" below. |
| EXECUTION BEGINS... | Message at start of execution. For every 2 seconds of CPU time used an in-operation signal is given. |
| THE RABBIT SQUEAKS TO THE WORLD | The message in the sample program is passed to the terminal. |
| R; | The READY message indicates that further CMS commands may be entered. |

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.

**EXAMPLE 2. CREATING A MODULE FILE**

| Terminal Printout | Notes and Comments |
|---|---|
| global txtlib plilib | Make PL/I libraries available |
| R; | |
| load rabbit | LOAD command creates an executable program from TEXT file and library modules. |
| genmod rabbit(from plistart | Creates a copy of the loaded program as a CMS MODULE file called rabbit. The program can now be executed by use of the file name as a command. Because of the structure of compiler output, the operand (FROM PLISTART must be included, if it is not, the file will not be executable. If the name of the file is omitted it will be called DMSIBM. |
| R; | |

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.

**EXAMPLE 3. EXECUTING A MODULE FILE**

| Terminal Printout | Notes and comments |
|---|---|
| global txtlib plilib | GLOBAL command makes the PL/I libraries available. This is necessary as some library modules are loaded dynamically. |
| R; | READY message |
| rabbit | For a MODULE file the filename can be used as a CMS command.<br><br>1. If you wish to pass parameters, they must appear after the filename and be preceded by a blank thus: rabbit / 1234. See "Special Considerations" below. |
| THE RABBIT SQUEAKS TO THE WORLD | The message in the sample program is passed to the terminal. |
| R; | The READY message indicates that further CMS commands may be entered. |

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

**EXAMPLE 4. COMPILING AND EXECUTING A PROGRAM WITH AN EXTERNAL SUBROUTINE**

| | |
|---|---|
| pliopt rabbit | Compile main PL/I program. |
| (compiler messages omitted) | |
| R; | |
| pliopt subrab(name('subrab' | Compile subroutine. Use of NAME |
| (compiler messages omitted) | option necessary or all members will |
| R; | be called DMSIBM. |
| txtlib add biolib subrab | Include the subroutine in the existing TXTLIB BIOLIB. This simplifies linking. If the TXTLIB did not exist, GEN would be used in place of ADD. |
| R; | |
| global txtlib plilib biolib | Issue GLOBAL command for TXTLIBs that hold PL/I library (PLILIB) and subroutine (BIOLIB). |
| R; | |
| load rabbit(start | The load command can now be issued and linkages will be automatic because the TXTLIB is globalled. If it was not, it would be necessary to specify the TEXT file SUBRAB in the LOAD command thus:<br>load rabbit subrab(start |
| R; | |

## MODULE and TEXT Files

During compilation the PL/I optimizing compiler produces code that requires further processing before it can be executed. Addresses within the code must be resolved and external modules referenced within the code must be included. These references will always include modules from the PL/I library.

The resolution of addresses is initiated by the LOAD command. The processed data can then be retained with the addresses resolved by the use of a GENMOD command specifying the filename. This command produces a CMS MODULE file that can be executed without going through the process of issuing the LOAD command on each occasion. Because of the structure of compiler output it is necessary to specify a name for the file explicitly and to include the operand (FROM PLISTART.

## Using CMS Files and OS Data Sets

CMS files, and other OS data sets can be written and read by programs executed under CMS with varying restrictions.

CMS files are completely accessible to programs running under CMS to read, write, and update. 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 ISAM files are not supported under CMS, but VSAM is supported and that there are restrictions on the use of REGIONAL files. The restrictions are described in chapter 2.)

VSAM data sets are available both to CMS virtual machines and to the OS system proper. DOS VSAM data sets are also available because DOS and OS VSAM data sets are compatible. VSAM data sets provide a method of sharing data betwen CMS and outside systems. However, under CMS they are handled by the DOS data management routines which must be incorporated under CMS during CMS system generation. The user's program will not be affected by the use of DOS routines, but certain OS Access Method Services functions will not be available for data set handling. Full details of this and other aspects of CMS VSAM are given in the CMS Users Guide.

OS data sets are available on a read-only basis to CMS programs, provided they are sequential.

Two elements are used under CMS 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 sets of external data throughout the program.) Outside the program, the FILEDEF command, or the DLBL command for VSAM, associates the filename with a particular data set and provides the function of the DD statement under OS.

VSAM data sets differ from other types in that they have their housekeeping handled by a set of programs known as Access Method Services, the DOS/VS versions of these programs are available to the CMS user by use of the AMSERV command which uses a previously created file containing Access Method Services statements to specify the services required.

Three examples follow, showing the PL/I statements and the CMS commands necessary to access CMS files, VSAM data sets, and non-VSAM OS data sets respectively.

## Accessing CMS Files

To access a CMS file you issue a FILEDEF command associating the PL/I file identifier with a particular CMS file.

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 on the A-disk a CMS file that will be known as RABBIT2 DATA. A third file, PL/I file RABPRNT is assigned to the virtual printer.

PL/I Program Statements

```
DCL OLDRAB FILE RECORD INPUT ENV (F RECSIZE(40)),
    NEWRAB FILE RECORD OUTPUT ENV (F RECSIZE(40)),
    RABPRNT FILE STREAM PRINT;
```

CMS Commands and Responses

| | |
|---|---|
| filedef oldrab disk rabbit1 data b R; | Associate OLDRAB with RABBIT1 DATA on B-disk |
| filedef newrab disk rabbit2 data a R; | Associate NEWRAB with the file RABBIT2 to be placed on the A-disk |
| filedef rabprnt printer R; | Associate the file RABPRNT with the virtual printer |

If no FILEDEF command is explicitly issued, one is issued by CMS, this assumes that the CMS file has the filename FILE, has a filetype the same as the name of the PL/I file, and the location is the A-disk.

The full syntax of the FILEDEF and other commands is given in VM/370: CMS Command and Macro Reference.

## Accessing VSAM Data Sets

VSAM data sets can be read, written, and updated, both from within, and from outside the CMS system. 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 user uses the AMSERV command to access Access Method Services functions and DLBL command to associate an actual VSAM data set with 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 Access Method Services are carried out. Such services must always be used for cataloging and formatting purposes before creating a VSAM data set. They are also used for deleting, renaming, making portable copies, and other routine tasks.

For VSAM data sets, information normally supplied in the ENVIRONMENT option of the PL/I file or the JCL DD statement is placed in the VSAM catalog. Catalogue 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, and to associate the PL/I file with the VSAM data set. The DLBL command is used for both these purposes. Where the data set is being newly created, the AMSERV command must be specified to catalog and define the data set before the PL/I program is executed. Details of how

to use VSAM under CMS are given in the VM/370: CMS User's Guide.

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 in the example that follows.

The PL/I program reads the file OLDRAB from the VSAM data set called RABBIT1 on the CMS B-disk. It writes the file NEWRAB onto the data set RABBIT2 also on the CMS B-disk. RABBIT2 is defined using an AMSERV command. In the example it is assumed that this master catalog is already assigned and that VSAM space is also already assigned.

PL/I File Declarations

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

CMS Commands

```
dlbl ijsyscat b dsn mastcat (perm        Issue a DLBL for the
R;                                       master catalog.
                                         Note that this need only
                                         be done once for terminal
                                         session if PERM is specified.
edit amsin amserv                        Create an AMSERV file to
NEWFILE                                  contain Access Method
EDIT                                     Services statements.
input
INPUT
define cluster(name(rabbit2) vol(mamal2)) -
       data (cyl(4,1) keys(20,0) recsz(23,23) -
       freespace(20,30)) -
       index(cyl(1,1))

EDIT
file
R;
amserv amsin                             Execute statements in the
R;                                       AMSERV file to catalog
                                         and format data set.
dlbl oldrab b dsn rabbit1 (vsam)         Issue DLBL commands to
R;                                       associate PL/I files with
dlbl newrab b dsn rabbit2 (vsam)         the VSAM data sets.
R;
```

Notes:
1. The closing parenthesis is optional in CMS commands but required in Access Method Services commands.

2. OS PL/I programs with files declared with ENV(INDEXED) may in certain instances execute VSAM I/O requests if the library routines detect that the data set being accessed is a VSAM data set. Refer to the OS PL/I Programmer's Guide under the heading 'The Compatibility Interface' for information on obtaining 'native VSAM' support.

## Accessing OS Data Sets

Before you access an OS data set it must be made available to your virtual machine. Then using the ACCESS command, it can be accessed as one of your CMS disks. Once this has been done, a FILEDEF command can be used to access the disk in the usual manner.

In the example that follows, the PL/I file OLDRAB is used to access the OS data set RABBIT.OS.DATA. It is assumed that the disk containing the data set has been mounted and is known to the user as virtual disk number 196.

PL/I Statement

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

CMS Commands

```
access 196 g                          Connect disk containing data
196 G R/O                             set to your virtual machine.
R;                                    Message confirms that it is
  |                                   accessed in read/only mode.
filedef oldrab g dsn rabbit os data   Associate PL/I file OLDRAB
                                      with OS data set RABBIT.OS.DATA
```

Assigning Files to the Terminal

To assign the terminal to a file it is necessary only to use TERM in your FILEDEF command. For example if you wished to assign a file called OUTPUT1 to the terminal you would do it as follows:

```
FILEDEF OUTPUT1 TERM
```

Because synchonization is only automatically handled for STREAM files, RECORD files should not normally be assigned to the terminal.

A number of FILEDEF commands are issued by the interface module DMSIBM. They assign SYSIN and SYSPRINT to the terminal for conversational I/O and PLIDUMP to a printer. If you wish to override these default assignments, you must issue suitable FILEDEF commands before starting the execution of the PL/I program.

An Alternative Method of Executing a MODULE File

A module file can be executed by a LOADMOD command followed by a START command.

## Attention Interrupts

The INTERRUPT option allows attention interrupts to become an integral part of programming with the optimizing compiler, and this gives the programmer considerable interactive control of the program.

If the INTERRUPT option is in effect during compilation, the compiled program will respond to one attention interrupt by searching for an established ATTENTION on-unit, executing it if it finds one, and continuing with the processing if it does not. When the execution of an ATTENTION on-unit is complete, control will return to the point of interrupt unless directed elsewhere by means of a GOTO statement. Two attention interrupts return you to CP.

If NOINTERRUPT was in effect during compilation, the compiled program will be halted if one attention interrupt occurs.

## How to Cause an Attention Interrupt

An attention interrupt to your virtual machine is caused on a 2741 by pressing the attention button once. An attention interrupt to CP is caused by pressing the attention button twice in quick succession. On a 3277, an attention to your virtual machine is caused by pressing the ENTER key (first moving the cursor one column left if the screen status message reads "VMREAD"), and an attention to CP by pressing the PA1 key. For other terminals see the appropriate manuals.

## How to use Attention Interrupts

The ability given by the INTERRUPT option to respond or not respond to attention interrupts allows for two possible uses:

1. Attention interrupts can be used purely as a debugging feature with ATTENTION on units used to supply debugging data. The program can finally be compiled with NOINTERRUPT for production purposes.

2. Attention interrupts can be used to produce an interactive system in which attention interrupts are used to control execution of the program.

## Attention on-units used for Debugging

When debugging under the optimizing compiler, ATTENTION on-units can be used to transmit values to the terminal when an attention interrupt is caused. For example, an ATTENTION on-unit might read:

```
ON ATTENTION PUT DATA(A,B,C,ICOUNT);
```

These values would then be transmitted to the terminal when an attention interrupt was caused. When the program had been debugged, the unit could be retained and the program compiled with the NOINTERRUPT option. This would prevent code to poll for attention interrupts being included in the load module and so there would be no time overhead, there would, however, be a small space overhead because the on-unit itself would be compiled.

The use of NOINTERRUPT also allows programs compiled on the checkout compiler with debugging ATTENTION on-units to be compiled on the optimizing compiler without producing an execution time overhead.

## Attention on-units used for Interactive Systems

The availability of attention interrupt also makes it possible to write fully interactive programs in PL/I and execute them when compiled by the optimizing compiler.

Typically the ATTENTION on-unit will prompt the user for input and carry out some action determined by that input. For example:

```
   ON ATTENTION BEGIN;
     ERRCOUNT=0;
 FIRST:
     PUT EDIT
     ('ENTER 1 FOR NEXT TABLE, 2 FOR REPETITION OF CURRENT TABLE 3 TO END OUTPUT:')
```

```
      GET LIST(NUM);
          SELECT(NUM);
            WHEN(1) GOTO NEXT;
            WHEN(2) GOTO START;
            WHEN(3) GOTO FINAL;
          OTHERWISE DO;
            ERRCOUNT=ERRCOUNT+1;
            IF ERRCOUNT<3 THEN  DO;
               PUT EDIT('INCORRECT INPUT.  TRY AGAIN')(A);
               GOTO FIRST;
               END;
            ELSE SIGNAL ERROR;
          END;  /*OTHERWISE CLAUSE*/
        END;  /*SELECT*/
      END;  /*ON-UNIT*/
```

The terminal interaction resulting from causing an attention interrupt could be as follows:

```
call bicent
THIS PROGRAM LISTS TABLES OF DATA RELATING
TO AMERICAN BICENTENNIAL CELEBRATIONS
USE ATTENTION INTERRUPT TO CHANGE TABLE
LIST OF STATE BIRDS

AMERICAN ROBIN                CONNECTICUT, MICHIGAN, WISCONSIN
BALTIMORE ORIOLE              MARYLAND
BLACK CAPPED CHICKADEE        MAINE,MASSACHUSETS
(Attention interrupt here)

ENTER 1 FOR NEXT TABLE, 2 FOR REPEAT OF CURRENT TABLE, 3 TO END OUTPUT:1

LIST OF STATE FLOWERS
AMERICAN BEAUTY ROSE          DISTRICT OF COLUMBIA
APPLE BLOSSOM                 ARKANSAS,MICHIGAN
ARBUTUS                       MASSACHUSETS
etc
```

## Background to Attention Handling

If you are going to make extensive use of attention interrupts, it is important to understand something of how they are implemented by the optimizing compiler.

Essentially, causing an attention interrupt sets a switch immediately and this switch is tested by means of polling at suitable points in the compiled program.

In procedures compiled with the INTERRUPT option, polling takes place between PL/I statements at branch-in points. Polling also takes place in all stream I/O statements to and from the terminal if <u>any</u> procedure in the load module was compiled with the INTERRUPT option. This arrangement allows maximum control of terminal input and output with minimum performance overheads. It also ensures that the ATTENTION condition is raised between PL/I statements.

## Pitfalls when using Attention Interrupts

The synchronization of terminal printout and processing by the CPU and the method used of implementing the ATTENTION condition cause various pitfalls for the user of attention interrupts. These are described below.

### Synchronization

When output is being transmitted to the terminal, the statement being executed in the CPU may be well beyond the point where the output is transmitted. (The number of buffers allocated determines how far.) Consequently an attention interrupt will often cause loss of output that is held in buffers. In addition an attempt to end excessive output to the terminal by use of an attention interrupt may have unexpected results if the program is not actually executing the output statement when the attention interrupt is caused.

Consider the on-unit

```
     ON ATTN BEGIN;
     /*Unit illustrates a potential pitfall*/
         ON ATTN GOTO ENDUNIT; /*Second ON statement kills the
                                 output if too long, by allowing
                                 attention interrupt during output*/
         PUT DATA;
 ENDUNIT:END;
```

An attention interrupt entered when you have seen enough output may in fact occur when the unit has completed executing. Thus the attention far from ending the output will just cause another set of output to begin.

Synchronization is only carried out when a GET statement to the terminal is executed. Therefore a GET statement at the end of the unit would solve the problem. A corrected on-unit could read:

```
     ON ATTN BEGIN;
      ON ATTN GOTO ENDUNIT;
      PUT EDIT
('TO END OUTPUT CAUSE ATTENTION. THEN ENTER GO TO CONTINUE OR STOP TO STOP')(A);
      DCL ANS CHAR(4);
      PUT DATA;
 ENDUNIT:
      /*Execution will wait here to synchronise the GET statement.*/
      /*Therefore attention interrupts entered during output of data*/
      /*will occur within the scope of the on-unit, so output will be*/
      /*ended by second ON ATTN statement*/
      GET EDIT (ANS)(A(4));
      UNSPEC(ANS)=UNSPEC(ANS)|(4) '01000000'B;
         /*Fold to upper case because input may be in upper or lower*/
      IF ANS='STOP' THEN STOP;
     END;
```

Note that the prompt for the GET statement must appear before the PUT DATA or it will be lost when an attention interrupt occurs.

## Programs Partly Compiled with the INTERRUPT Option

If any procedures within a load module have been compiled with the INTERRUPT option, a STAX macro instruction is issued at the start of execution. Consequently, an attention interrupt will be noted whenever it is caused and will raise the ATTENTION condition when polling occurs. This will be during stream I/O to or from the terminal in all procedures and at branch-in points in procedures compiled witht the INTERRUPT option. If you wish to use attention interrupts principally to control stream I/O, this can be very useful, as it achieves your requirements with the minimum performance overhead. However, if you wish to use attention interrupts for debugging purposes the results may be unexpected because any attention on units will be executed regardless of the option with which the procedure that contains them was compiled.

SPECIAL CONSIDERATIONS

## Passing Parameters and Execution Time Options

When passing parameters two sets of restrictions have to be born in mind, those that are imposed by CMS, and those imposed by the PL/I optimizing compiler.

Under CMS, parameters must be passed to the program in tokens containing no more than eight characters. These tokens must be separated by blanks.

The PL/I Optimizing Compiler allows you to pass two types of parameters to a PL/I program. The first is a set of execution time options, sometimes called program management parameters (these are listed in Chapter 4 of this manual). The second is a single parameter that is passed to the PL/I main procedure. The two types of parameter are separated by a / symbol which must itself have a blank on either side. Anything preceding this symbol is taken to be an execution time option. If no execution time option is passed, the main procedure parameter must be preceded by the three characters blank, oblique stroke, and blank( / ).

Under the PL/I Optimizing Compiler, the main procedure parameter must be a character string, and, because blanks are used as delimiters in CMS blanks cannot be passed in the string. Blanks are removed from the string and the two separated items concatenated.

Suppose you wished to pass to a program the execution time options NOSPIE AND REPORT and a character string consisting of a name of more than eight characters and three sets of figures, this could be passed in the form:

```
start * NOSPIE REPORT / CARPENTE R,38,24, 38
```

this would be passed to the program in the form of

```
CARPENTER,38,24,38
```

## Executing a File Not Compiled Under CMS or Compiled with the OSDECK Option

If you wish to execute under CMS a program that was compiled under OS or was compiled under CMS with the OSDECK option, it is necessary to explicitly load the execution time interface module. (An entry statement for this module is automatically included in the TEXT file for any PL/I program compiled under CMS without the OSDECK option.) Assuming the program that you wish to execute is on a CMS TEXT file and is called SEAGULL, the following commands are required.

```
global txtlib plilib
load seagull dmsibm
start dmsibm
```

The GLOBAL and LOAD commands make the PL/I library available and load the program and the interface module. The START command passes control to the interface module, which, in turn, passes control to the program.

If you wish to create a MODULE file from the load module you have created, you must issue a GENMOD command after the LOAD command. This will produce a MODULE file with the name of the file used in the LOAD command (SEAGULL in the example). The MODULE file can then be executed in the normal manner.

SOURCES OF FURTHER INFORMATION

| Topic | Reference Source |
|---|---|
| FILEDEF command | VM/370: CMS Command and Macro Reference and VM/370: CMS User's Guide |
| filename as a command | VM/370: CMS User's Guide |
| GENMOD command | VM/370: CMS Command and Macro Reference and VM/370: CMS User's Guide |
| GLOBAL command | VM/370: CMS Command and Macro Reference and VM/370: CMS User's Guide |
| LOADMOD command | VM/370: CMS Command and Macro Reference and VM/370: CMS User's Guide |
| START command | VM/370: CMS Command and Macro Reference and 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 is an alias that can be used instead of LOGOFF.)

Before finishing the session you may wish to erase some of the files. This is done by using the ERASE command.

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

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 and virtual and the real CPU time in minutes, seconds, and hundredths of seconds. |
| LOGOFF AT hh:mm:ss (time-zone) day-of-week mm/dd/yy | Message shows time and date of logging off. |
| (you switch off terminal) | |

Conventions:

1. A carriage return 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 files you use the ERASE command. The command must specify the file name and the file type. For example if you wished to delete the PLIOPT file "rabbit", you would enter:

```
erase rabbit pliopt
```

If you wished to delete all the files called "rabbit" you would enter:

```
erase rabbit *
```

## 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 logout 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 (or LOGOUT) command | VM/370: CP Command Reference for General User's |

# 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.

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

MULTITASKING — PL/I error message number IBM576I will be generated. This reads "ATTEMPT TO CALL A TASK IN NON-TASKING ENVIRONMENT" The associated ONCODE is 3915.

SORT — The ERROR condition will be raised and PL/I error message 881I will be generated. This reads "SORT/MERGE NOT SUPPORTED IN CMS". The associated ONCODE is 9201.

FETCH/RELEASE — The ERROR condition will be raised and PL/I error message 592I will be generated. This reads "FETCH/RELEASE NOT SUPPORTED IN CMS". The associated ONCODE is 9252.

CHECKPOINT/RESTART — PL/I error message 926I will be generated and execution will continue without a checkpoint being taken. The message reads "CHECKPOINT RESTART NOT SUPPORTED IN CMS". There is no ONCODE as the ERROR condition is not raised.

ISAM FILES — The UNDEFINEDFILE condition is raised.

Use of TCAM, or spanned records on BDAM, or the BACKWARDS attribute.

CMS error message number DMSOP063E will be generated. This reads "OPEN ERROR CODE x ON ddname."

DELAY statement — The statement is executed without raising an error message but does not cause a delay.

| DO NOT USE UNDER CMS |
|---|
| ASCII data sets |
| BACKWARDS attribute with magnetic tapes |
| DELAY statement |
| FETCH and RELEASE statements |
| INDEXED files (Except for use with VSAM) |
| PL/I checkpoint restart facilities (PLICKPT) |
| PL/I sort facilities (PLISORT) |
| Tasking |
| Teleprocessing files |
| VS or VBS record formats |

| OTHER RESTRICTIONS UNDER CMS | |
|---|---|
| READ....EVENT | can only be used if the NCP parameter is included in the ENVIRONMENT option of the PL/I file. |
| Blanks | cannot be passed in the parameter string to the main procedure. The blanks are removed from the string and the items separated by them are concatenated. |
| TIME | The TIME builtin function returns values calculated to the nearest second. |

Restrictions on REGIONAL files and VSAM

| | |
|---|---|
| REGIONAL files | More than one regional file with keys may not be open at the same time.<br>The minimum logical record length for REGIONAL 2 and 3 files is 8 bytes.<br>Files must not be written with a dependancy on the physical track length of a direct access device.<br><br>REGIONAL 3 files with varying length records BLKSIZE specified in the FILEDEF command must be 8 bytes longer than logical record length.<br><br>KEY (TRACKID/REGION NUMBER) must not be incremented unless 255 records are written on the first logical track, and 256 records on each subsequent logical track.<br><br>When a file is created, the XTENT option of the FILEDEP command must be specified and it must be equal to the number of records in the file to be created. |
| VSAM | VSAM data sets can be used only if DOS/VS VSAM was incorporated into CMS during CMS system generation procedure. DOS VSAM is used and any features not available to DOS VSAM cannot be used. CMS/DOS must also be generated into CMS. For details of how to do this, see VM/370: Planning and System Generation Guide. Environment options: SIS cannot be used, SKIP cannot be used on ESDS. |
| Interlanguage Communication | See section headed "Interlanguage Communication" later in this chapter. |

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

## Using Record I/O at the Terminal

No provision is made for automatic prompting or synchronization of output for RECORD files at the terminal. It is envisaged that stream I/O will normally be used. If you do use record I/O at the terminal the following points should be born in mind.

Output: Output files should be declared with BUFFERS(1) if you are using them for prompting. Otherwise messages may be held in the buffer when you expect a prompt. V-format records should be used otherwise trailing blanks will be printed.

Input: V-format records should be used, as otherwise the RECORD condition will be raised unless the record is filled out with trailing blanks. Note that when V-format records are used, and the data is read into a fixed length string, the string is not padded with blanks. By default, RECORD files assigned to the terminal are given F-format records with the record length the same as the linesize for the terminal.

## Interlanguage Communication

Interlanguage facilities are restricted under CMS, and special action is required to execute programs that contain them.

The restriction is that all calls must be in one direction only. (This is caused because a loader restriction on CMS prevents the full implementation of the PL/I scheme, which requires the resolution of one control section that appears in a number of library modules.) The restriction means that:

1. A PL/I program can call either FORTRAN or COBOL subroutines, but not both.

2. A COBOL or FORTRAN program can call PL/I subroutines.

3. In neither case can the subroutines make any further interlanguage calls.

Non-standard action is required when PL/I calls a FORTRAN subroutine, and when PL/I is called from FORTRAN or COBOL. It ensures that the correct entry point is used for the program. The action required is described below.

## PL/I Program with FORTRAN Subroutines

To run a PL/I program with one or more FORTRAN subroutines, you must specify DMSIBM (the PL/I-CMS interface) as the starting point. Otherwise entry statements in the FORTRAN can result in entry to a FORTRAN routine. DMSIBM can be specified either in the start command or in the RESET operand of the LOAD command. The latter should be used if you wish to create a module file. Commands to execute a PL/I program (PPROG) with FORTRAN subroutine (FSUB) might take the form shown below:

```
pliopt pprog
fortgi fsub
global txtlib plilib ppfortg1
load pprog fsub     or   load pprog fortsub reset(dmsibm
start                    genmod intprog (from plistart
                         intprog
```

As can be seen in the example on the right LOAD...RESET... allows a module file to be created that will have the correct entry point. The module file is called intprog in the example.

FORTRAN or COBOL with PL/I subroutines

When FORTRAN or COBOL routines are to be executed with PL/I subroutines, special action must be taken to avoid entry being made through the PL/I-CMS interface, DMSIBM. The PL/I program should be compiled with the OSDECK option to prevent entry via DMSIBM. The fact that DMSIBM is not used, means that FILEDEF commands are not issued for SYSIN, SYSPRINT, and PLIDUMP, and that formatting of stream I/O will not always be correct. You can issue your own FILEDEF commands but there is no satisfactory solution to the stream I/O problem.

Commands to execute a FORTRAN program (FPROG) with a PL/I subroutine (PSUB) are shown below. FILEDEF commands normally executed in DMSIBM are included.

```
fortgi fprog
pliopt plisub(osdeck
filedef sysin term
filedef sysprint term
filedef plidump printer
global txtlib plilib ppfortg1
load fprog plisub
start
```

## Conventions

Two types of convention apply to PL/I when used under CMS. The first are those adopted to make input/output simpler and more efficient at the terminal. The second type are those that result from the terminal being considered as the console of a virtual machine. These affect the DISPLAY statement and the REPLY option.

No prompting or other facilities are provided for record I/O at the terminal. You are therefore strongly advised to use stream I/O for any transmission to or from a terminal.

### STREAM I/O CONVENTIONS AT THE TERMINAL

To simplify input/output at the terminal various conventions have been adopted for stream files that are assigned to the terminal. Three areas are affected.

1. The formatting of PRINT files
2. The automatic prompting feature
3. The spacing and punctuation rules for input

#### Formatting Conventions for PRINT Files

When a PRINT file is assigned to the terminal, it is assumed that it will be read as it is being printed. Spacing is therefore reduced to a minimum to reduce printing time. The following rules apply to the PAGE, SKIP, and ENDPAGE keywords.

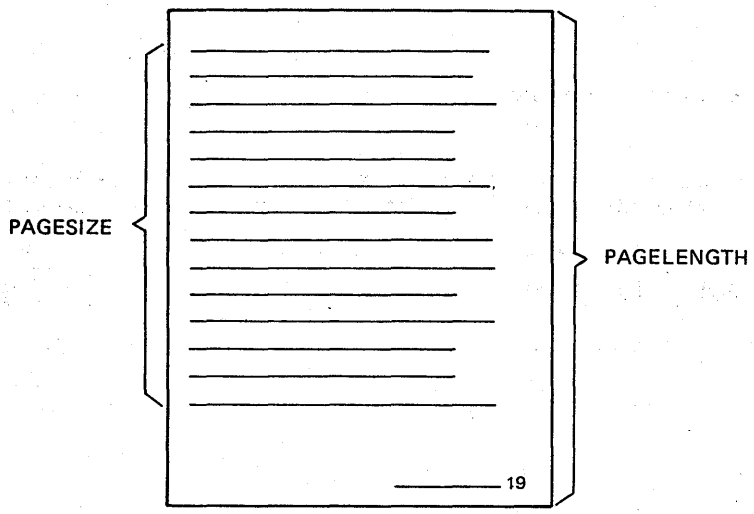
- PAGE options or format items result in three lines being skipped.
- SKIP options or format items larger than SKIP (2) result in three lines being skipped. SKIP (2) or less is treated in the usual manner.
- The ENDPAGE condition is never raised.

Overriding the formatting conventions for PRINT files: If you wish normal spacing to apply to output from a PRINT file at the terminal, it is necessary to supply your own tab table for PL/I. This is done by declaring an external structure called PLITABS in the program and initializing the element PAGELENGTH to the number of lines that can fit on your page. This value differs from PAGESIZE which defines the number of lines you want to be printed on the page before ENDPAGE is raised, see figure 2.2. If you required a pagelength of 64 lines you would declare PLITABS thus:

```
DCL 1 PLITABS STATIC EXTERNAL,
  ( 2   OFFSET INIT (14),
    2    PAGESIZE INIT (60),
    2    LINESIZE INIT (120),
    2    PAGELENGTH INIT (64),
    2    FILL1 INIT (0),
    2    FILL2 INIT (0),
    2    FILL3 INIT (0),
    2    NUMBER_OF_TABS INIT (5),
    2    TAB1 INIT (25),
    2    TAB2 INIT (49),
```

```
2    TAB3 INIT (73),
2    TAB4 INIT (97),
2    TAB5 INIT (121)) FIXED BIN (15,0);
```

This declaration gives the standard page size, line size and tabulating positions.




PAGELENGTH: is the number of lines that could be printed on a page.

PAGESIZE: is the number of lines that will be printed on a page before the ENDPAGE condition is raised.

Figure 2.2. PAGELENGTH defines the size of your paper, PAGESIZE the number of lines printed in the main printing area.

## Automatic Prompting

When the program requires input from a file that is associated with a terminal, it issues a prompt. This takes the form of printing a colon on the next line and then skipping to column 1 on the line following the colon. This gives you a full line to enter your input thus:

```
:
(space for entry of your data)
```

This type of prompt is referred to as a primary prompt.

If the data you transmit from the terminal does not complete the requirements of the GET statement, a further prompt is issued. This takes the form of printing a plus sign followed by a colon thus:

```
+:(space for entry of your data)
```

This type of prompt is referred to as the secondary prompt.

Overriding Automatic Prompting: It is possible to override the primary prompt by making a colon the last item in the request for the data. The secondary prompt cannot be overridden. Take the two PL/I statements

```
PUT SKIP EDIT ('ENTER TIME OF PERIHELION');
GET EDIT (PERITIME) (A(10));
```

As they stand they would result in the terminal printing

```
ENTER TIME OF PERIHELION
:            (automatic prompt)
 (space for entry of data)
```

However, if the first statement had a colon at the end of the output thus:

```
PUT EDIT ('ENTER TIME OF PERIHELION:') (A);
```

the sequence would be:

```
ENTER TIME OF PERIHELION:(space for entry of data)
```

Note: The override remains in force for only one prompt. You will be automatically prompted for the next item unless the automatic prompt is again overridden.

## Spacing and Punctuation Conventions for Input

Line continuation character. If you wish to transmit as one data item data that requires 2 or more lines of space at the terminal a hyphen must be typed as the last character in each line except the last line. For example, if you wanted to transmit the sentence enter:

```
this data must be transmitted -
as one unit.
```

Transmission would not occur until the carriage return after the word "unit". The hyphen would be removed. The item transmitted is referred to as a "logical line".

Note: This convention means that a line whose last character is a hyphen or a PL/I minus sign can only be transmitted by entering two hyphens at the end of the line and following them by a carriage return only on the next line thus:

```
xyz--
(carriage return only on this line.)
```

## Simplified Punctuation for GET LIST and GET DATA Statements

For GET LIST and GET DATA statements, a comma is added to the end of each logical line transmitted from the terminal if it is omitted by the programmer. Thus there is no need to enter blanks or commas to delimit items if they are entered on separate logical lines. For the PL/I statement GET LIST(A,B,C); you could enter at the terminal.

```
1
+:2
+:3
```

However this rule also applies when entering character string data. A character string must therefore be transmitted as one logical line, otherwise commas are placed at the break points. For example. if you entered:

```
'COMMAS SHOULD NOT BREAK
+: UP A CLAUSE'
```

The resulting string would read 'COMMAS SHOULD NOT BREAK, UP A CLAUSE'

Automatic Padding for GET EDIT: For a GET EDIT statement there is no need to enter blanks at the end of the line. The item will be padded to the specified length. Thus for the PL/I statement GET EDIT (NAME) (A(15)); you could enter SMITH followed immediately by a carriage return. The item would automatically be padded with ten blanks so that the program received the fifteen characters "SMITH ".

Note: This means that a single item must be transmitted as a logical line, otherwise the first line transmitted will be padded with the necessary blanks and considered to be the complete item.

Use of SKIP for terminal input: SKIP in a GET statement has little meaning if the file involved is allocated to a terminal. The program is apparently being asked to skip data that has not yet been keyed in. For this reason, all uses of SKIP for input are taken to be SKIP(1) when the file is allocated to the terminal. SKIP(1) is treated as an instruction to ignore all unused data on the currently available logical line.

## Endfile

The end of file can be entered at the terminal by keying in a logical line that contains the characters "/*" followed by a carriage return. Any further attempts to use the file without closing it result in the ENDFILE condition being raised.

## 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 wait for your response and 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
|John Karpinski
```

# Chapter 3: The PLIOPT Command and its Options

How to Use This Chapter

This chapter shows the syntax of the PLIOPT command, the options that can be used with the command, 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 PLIOPT command and its options showing the default option values suggested by IBM.
3. A discussion of two general points. First the differences between options of the PLIOPT command and options of the PL/I Optimizing Compiler, and, second, the relationship between the various statement numbering options.
4. A table of options listed by function.
5. An alphabetical list of options with detailed descriptions and syntax notation.

If you wish to accept the default options, you will only need to look at the section on the PLIOPT command and possibly the section on syntax notation if you are not already familiar with this. It should be noted that the default values may have been altered by your installation and may not correspond to those shown in the table. 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, or 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 this table you should check in the alphabetical section to discover the syntax.

If you intend to use options in a *PROCESS statement, you should read the discussion headed "PLIOPT Options and Compiler Options". It should not be necessary to read the section headed "Relationship of Statement Numbering Options" unless you need amplification of the information supplied in the descriptions of the statement numbering options in the alphabetical section.

A general discussion of the PLIOPT command is given in chapter 2 under the heading "Compiling the Program the PLIOPT 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.

# PLIOPT Command

The PLIOPT command invokes the PL/I Optimizing Compiler to compile a program written in PL/I source language. The compiler produces a TEXT file containing machine code and a LISTING file containing listings and diagnostics. Other files may be produced depending on compiler options.

FORMAT:

```
PLIOPT  filename [filetype[filemode]] (option1 option 2...)

        Options:    AG|NAG
                    A[(FULL|SHORT)]|NA
                    CHARSET ([48|60][EBCDIC|BCD])
                    COMPILE|NC[(W|E|S)]
                    CONTROL('password')
                    COUNT|NOCOUNT
                    DECK|NODECK
                    DUMP|NODUMP
                    ESD|NOESD
                    FLAG[(I|W|E|S)]
                    FLOW[(n m)]|NOFLOW
                    GONUMBER|NGN
                    GOSTMT|NOGOSTMT
                    INCLUDE|NINC
                    IMP|NIMP
                    INT|NINT
                    INSOURCE|NIS
                    LC(n)|LC(55)
                    LIST[(m n)]|NOLIST
                    LMESSAGE|SMESSAGE
                    MACRO|NOMACRO
                    MAP|NOMAP
                    MARGINI('c')|NMI
                    MARGINS(m n[c])|MARGINS(2 72)
                    MDECK|NOMDECK
                    NAME('name')
                    NEST|NONEST
                    NUMBER|NONUMBER
                    OBJECT[(filename)]|NOOBJECT
                    OFFSET|NOOFFSET
                    OPTIMIZE(TIME|0|2)|OPTIMISE(TIME|0|2)|NOPT
                    OPTIONS|NOP
                    OSDECK [1]
                    PRINT|DISK|TYPE|NOPRINT[1]
                    SEQUENCE(m n)|NOSEQUENCE
                    SIZE (yyyyyyyy|yyyyyK|MAX)
                    SOURCE|NOSOURCE
                    STMT|NOSTMT
                    STORAGE|NSTG
                    SYNTAX|NSYN[(W|E|S)]
                    TERMINAL[(opt-list)]|NTERM
                    XREF[(FULL|SHORT)]|NOXREF

        [1]Note: These are options of the PLIOPT command and not
         compiler options, see discussion below.
```

filename[filetype[filemode]]

is the file identification of the file that contains the PL/I source program. If filetype is omitted, a search will be made first for PLIOPT files of the specified filename and then for PLI files of the specified filename. If filemode is omitted, A will be assumed.

option1 option2

are a series of compiler or PLIOPT options. They must be separated from each other by at least one blank. The right hand parenthesis is optional. If contradicting options are specified, the rightmost option applies.

## USAGE

The PLIOPT command compiles a PL/I program or a series of PL/I programs into machine language object code. If the program is held as a CMS file on disk it must have the file type PLIOPT or PLI. If it is not on disk, it must be defined to the system with a FILEDEF command.

The options governing compiler operation and output are specified in any order. Any combination of options is accepted. When conflicting options are specified, the last specified option is used. The majority of options have positive and negative forms one of which is used by default if neither form is specified. Figure 3.1 summarizes the compiler options by function and enable you to quickly grasp the possibilities available with the PL/I Optimizing Compiler.

## PLIOPT OPTIONS AND COMPILER OPTIONS

The majority of options of the PLIOPT command are options of the optimizing compiler. This means that they can be specified in the *PROCESS statement as well as in the PLIOPT command. All options except DISK, NOPRINT, OSDECK, and PRINT can be specified in the * PROCESS statement. DISK, NOPRINT, OSDECK, and PRINT cannot be specified because they are PLIOPT options and not a compiler options. DUMP cannot be specified in the * PROCESS statement unless it is also specified in the PLIOPT command. This is because extra space must be acquired for the DUMP option before the * PROCESS statement is processed.

Where options of the PLIOPT command contradict those of the *PROCESS statement, the options in the *PROCESS statement override those in the PLIOPT command. For options whose length is greater than eight characters, the abbreviation for that option must be used in the PLIOPT command.

### Relationship of Statement Numbering Options

The optimizing compiler provides two methods of numbering statements. Statements can have their numbers taken from the sequence field of the record; this is the method used when NUMBER or GONUMBER is specified and is the default for CMS. Alternatively, they can be numbered sequentially starting from 1; this is the method used when STMT or GOSTMT is specified.

The numbers of the statements are used in compiler diagnostic messages and listings. If the GONUMBER or GOSTMT option is specified, the numbers are retained in a table generated by the compiler and are used in execution time diagnostic messages. When numbers are required during execution, the same numbering system as that which applied during compilation will be used. This means that specifying certain options implies that certain other options will be used. Three rules apply:

1. Because one or other statement numbering system must be used during compilation, NOSTMT is taken as equivalent to NUMBER, and similarly, NONUMBER is taken as equivalent to STMT.

2. Because the same numbering system must be used during compilation and execution, either of the GO options is taken to imply that the corresponding numbering system is to apply during compilation. Thus GONUMBER implies NUMBER and GOSTMT implies STMT.

3. It is not possible to use both numbering systems in one compilation therefore GOSTMT implies NOGONUMBER, and GONUMBER implies NOGOSTMT.

If contradictory options are specified, the last option found is used and any implications are taken from that option.

The use of GONUMBER or GOSTMT involves a space overhead because the numbers are retained in a table generated by the compiler. If statement numbers are not retained into execution, execution-time diagnostic messages identify the location of the error by an offset from a procedure entry point. The use of the OFFSET option results in the generation of a listing at compile time that associates statement numbers with offsets and consequently enables you to identify the PL/I statement mentioned in an execution time error message.

The OFFSET option is separate from the numbering options and must be specified if required.

Examples:

To compile a PLIOPT or PLI file called rabbit on the A disk with the options TYPE and SOURCE.

```
plic rabbit (type source
```

To compile a file with the name rabbit and the type FORMAT on the B disk with the options PRINT, XREF and ATTRIBUTES. Note that the abbreviations are used for these options.

```
plic rabbit format b (x a pri
```

| OPTIONS LISTED BY FUNCTION PART 1 | | |
|---|---|---|
| LISTING OPTIONS | | |
| Control destination of listing file | | |
| | PRINT\|DISK\|TYPE \|NOPRINT* | Determine whether listing goes to printer (PRINT), CMS disk (DISK), the terminal (TYPE), or is discarded. |
| Control listings produced | AGGREGATE | list of aggregates and their sizes. |
| | ATTRIBUTES | list of attributes of all identifiers. |
| | ESD | list of external symbol dictionary. |
| | INSOURCE | list of preprocessor input. |
| | FLAG(I\|W\|E\|S) | suppress diagnostic messages below a certain severity. |
| | LIST | list of compiled code produced by compiler. |
| | MAP | list offsets of variables in static control section and DSAs. |
| | 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 | | |
| | LC | specifies number of lines per page on listing. |
| * Options marked thus are ignored if used in the *PROCESS statement | | |

Figure 3.1. (Part 1 of 3) Compiler options arranged by function

| OPTIONS LISTED BY FUNCTION PART 2 | | |
|---|---|---|
| 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 |
| | SEQUENCE | identify the columns used for sequence numbers. |
| 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. |
| | NC(W\|E\|S) | stop processing after errors are found in syntax checking. |
| OPTIONS FOR PREPROCESSING | | |
| | MACRO | allows full use of the preprocessor facility. |
| | INCLUDE | allows inclusion of text without overheads incurred MACRO. |
| | MDECK | produces a source deck from preprocessor output. |
| OPTIONS TO USE WHEN PRODUCING AN OBJECT MODULE | | |
| | OBJECT | produce an object module from translated output. |
| | NAME | specify the name of the object module produced. |
| | DECK | produce an object module on punched cards. |
| OPTIONS TO CONTROL STORAGE USED | | |
| | SIZE | controls the amount of storage used by the compiler. |
| OPTIONS TO IMPROVE USABILITY AT A TERMINAL | | |
| | TERMINAL | specifies how much of listing is transmitted to terminal. |
| | SMESSAGE/LMESSAGE | enables you to specify concise or full message format. |

Figure 3.1. (Part 2 of 3) Compiler options arranged by function

| OPTIONS LISTED BY FUNCTION PART 3 | | |
|---|---|---|
| OPTIONS TO SPECIFY STATEMENT NUMBERING SYSTEM USED | | |
| | NUMBER & GONUMBER | numbers statements according to line on which they start. |
| | STMT & GOSTMT | numbers statements sequentially. |
| | OFFSET | specifies that a listing associating statement numbers with offsets will be generated Thus enabling you to identify statements from offsets given in execution time 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. |
| OPTION TO IMPROVE COMPILATION/EXECUTION SPEED | | |
| | OPTIMIZE(TIME) | reduce execution time at the expense of compilation. |
| | NOPT | reduce compilation time at the expense of execution. |
| OPTION TO CONTROL EFFECT OF ATTENTION INTERRUPTS | | |
| | INT | specifies that the ATTENTION condition will be raised when the attention interrupt occurs. For NINT, attention halts execution. |
| OPTION TO ALLOW EXECUTION UNDER OS | | |
| | OSDECK* | specifies that compiler will produce OS compatible code. |
| OPTION FOR USE WHEN DEBUGGING THE COMPILER | | |
| | DUMP | produces a dump if the compiler terminates abnormally. |
| OPTION FOR USE ON IMPRECISE INTERRUPT MACHINES | | |
| | IMP | allows imprecise interrupts to be correctly handled. |
| OPTIONS FOR SYSTEMS PROGRAMMING | | |
| | CONTROL('password') | gives access to deleted options. |
| * Options marked thus are ignored if used in the *PROCESS statement | | |

Figure 3.1. (Part 3 of 3) Compiler options arranged by function

AGGREGATE|NOAGGREGATE
AG|NAG

The abbreviated form must be used in PLIOPT command.

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.

|ATTRIBUTES[(FULL|SHORT)]|NOATTRIBUTES
|A[(F|S)]|NA

The ATTRIBUTES option specifies that the compiler is to include in the compiler listing a table of source-program identifiers and their attributes. If both ATTRIBUTES and XREF apply, the two tables are combined.

If SHORT is specified, unreferenced identifiers are omitted, making the listing more manageable.

If both ATTRIBUTES and XREF apply, and there is a conflict between SHORT and FULL, the usage is determined by the last option found. For example, ATTRIBUTES(SHORT) XREF (FULL) results in FULL applying to the combined listing.

The default FULL means that FULL applies if the option is specified with no sub-option.

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

The CHARSET option specifies the character set and data code that you have used to create 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 the 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.

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

The abbreviated form must be used in PLIOPT command for NOCOMPILE.

The COMPILE option specifies that the compiler is to compile the source program unless an unrecoverable error was detected during preprocessing or syntax checking. The NOCOMPILE option without an argument causes processing to stop unconditionally after syntax checking. With an argument, continuation depends on the severity of errors detected after the syntax checking phase as follows:

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.

CONTROL('password')

The CONTROL option specifies that any compiler options deleted for your installation are to be available for this compilation. You must still be 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 six characters in length.

If the option is specified in the PLIOPT command, the password may not contain a left parenthesis "(" or a right parenthesis ")".

COUNT|NOCOUNT
CT|NCT

The COUNT option specifies that code will be generated to allow a count to be kept of the number of times each statement is executed in a particular run of a program to be generated at the end of the run.

Unless overridden at execution time by the NOCOUNT option, it will result in a count of the number of times each statement in a program has been executed being printed on the PLIDUMP file or, if there is none, on the SYSPRINT file, after the execution of the compiled program.

The CODE generated for the COUNT option also allows a trace of the most recently executed statements to be retained if the FLOW option is specified at execution time.

The COUNT option implies the GONUMBER option if the NUMBER option is in effect and the GOSTMT option if the STMT option is in effect.

DECK|NODECK
D|ND

The DECK option specifies that the compiler is to produce an object module in the form of 80-column card images and store it in the data set defined by the DD statement with the name SYSPUNCH. 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.

DUMP|NODUMP
DU|NDU

Do not use in *PROCESS statement unless also used in the PLIOPT command

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. Details of the suboptions of DUMP are given in the OS PL/I Optimizing Compiler Program Logic.

ESD|NOESD

The ESD option specifies that the external symbol dictionary (ESD) is to be listed in the compiler listing.

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

The FLAG option specifies the minimum severity of error that requires a message 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, FLAG(W) is assumed. |
| FLAG(E) | List all except warning and informatory messages. |
| FLAG(S) | List only severe error and unrecoverable error messages. |

FLOW(n m)|NOFLOW

The FLOW COMPILER OPTION SPECIFIES THAT CODE WILL BE PRODUCED ENABLing the transfers of control most recently executred 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. The format of the FLOW option is:

FLOW(n m)

| | |
|---|---|
| where "n" | is the number of transfers of control that will be listed with associated statement numbers. |

where "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 32768. 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.

The code generated for the FLOW compiler option allows the COUNT execution time option to be used if it is specified at execution time.

GONUMBER|NOGONUMBER
GN|NGN

The abbreviated form must be used in the PLIOPT command for NOGONUMBER.

The GONUMBER options specifies that the compiler is to produce additional information that will allow line numbers from the source program to be included in execution-time messages. Alternatively, these line numbers can be derived by using the offset address, which is always included in execution-time messages, and the table produced by the OFFSET option.

Use of the GONUMBER option implies that the NUMBER option will apply. See "Relationship of Statement Numbering Options" at the start of this chapter.

GOSTMT|NOGOSTMT
GS|NGS

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

Use of the GOSTMT option implies that the STMT option will also apply. See "Relationship of Statement Numbering Options" at the start of this chapter.

IMPRECISE|NOIMPRECISE
IMP|NIMP

The abbreviated form must be used in the PLIOPT command.

The IMPRECISE option specifies that the compiler is to include extra text in the object module to localize imprecise interrupts when executing the program with an IBM System/360 Model 91 or 195, or System 370 model 195. This extra text ensures that if interrupts occur, the correct on-units will be entered.

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.

INSOURCE|NOINSOURCE
IS|NIS

The abbreviated form must be used in the PLIOPT command for NOINSOURCE.

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.

|INTERRUPT|NOINTERRUPT
|INT|NINT

The abbreviated form must be used in the PLIOPT command. This option determines the effect of attention interrupts when the compiled PL/I program is being executed.

If INTERRUPT was in effect during compilation, an established on ATTENTION on-unit will be executed when one attention interrupt is caused. If there is no such on-unit, processing will continue. Two attention interrupts will cause control to return to CP.

If NOINTERRUPT was in effect during compilation, one attention interrupt will end the execution of the program and cause control to return to CMS.

It should be noted that if any procedure within a load module was compiled with the INTERRUPT option, an attention interrupt at any time will lead to the ATTENTION condition being raised if polling is carried out, and execution continuing with no apparent effect if polling is not carried out. Polling is carried out during the execution of stream I/O for all modules, and, additionally, at branching points for modules compiled with the INTERRUPT option. Because the ATTENTION condition is raised when polling is done, an attention interrupt in a program partly compiled with the INTERRUPT option can lead to unexpected results.

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

The abbreviated form must be used in the PLIOPT command.

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.

LIST[(m n)]|NOLIST

The LIST option specifies that the compiler is to include a listing of the object module (in a form similar to IBM System/360 assembler language instructions) in the compiler listing. When used in conjunction with MAP it increases the information generated by MAP (see MAP later in this section).

m and n are statement numbers which 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. If only one statement number is specified, that statement only will be listed.

LMESSAGE|SMESSAGE
LMSG|SMSG

The LMESSAGE and SMESSAGE options specify that the compiler is to produce messages in a long form (specify LMESSAGE) or in a short form (specify SMESSAGE). Short messages save printing time at the terminal.

MACRO|NOMACRO
M|NM

The MACRO option specifies that the source program is to be processed by the preprocessor.

MAP|NOMAP

The MAP option specifies that the compiler is to produce tables showing the organization of storage for the object module. These tables show how variables are mapped in the static internal control section and in DSAs, thus enabling STATIC INTERNAL and AUTOMATIC variables to be found in a PLIDUMP.

If LIST is also specified, the MAP option produces tables showing constants, control blocks, and INITIAL variable values. LIST generates a listing of the compiled code in pseudo assembler language format. If you want a complete MAP but not a complete LIST, you can specify a single statement number as an argument for LIST. For example:

```
* PROCESS MAP LIST(1);
```

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

The abbreviated form must be used in the PLIOPT command for NOMARGINI.

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. If the option is specified in the PLIOPT command, the charater may not be left parenthesis "(" or right parenthesis ")".

MARGINS(2,72,1) (F-format records)
MARGINS (10,100,0) (V-format records)

MAR(m n [c])

The MARGINS option specifies which part of each compiler input record contains PL/I statements, and the position of the ANS control character that formats the listing. The MARGINS option is used to override the default margin positions that are set up during compiler installation by the FMARGINS and VMARGINS options.

The FMARGINS default applies to F-format records and the VMARGINS default applies to V-format or U-format records. Only one of these defaults is overridden by the MARGINS option. If the first input record to the compiler is F-format, the FMARGINS default is overridden. If the first input record to the compiler is a V- or U-format record the VMARGINS default is overridden by the MARGINS option. Default values are assumed if a record with a different type of format is encountered by the compiler.

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; it is ignored in conversational-mode listings at the terminal. Only the following control 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.

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 the ddname SYSPUNCH. 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.

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

No default applies. NAME must be specified if required.

The NAME option specifies that the TEXT file created by the compiler will be given the specified external name. This allows you to create more than one text file when doing batch compilation and also allows you to produce TEXT files suitable for inclusion in a text library (see section headed "Compiling the Program - the PLIOPT Command".)

The name option has the format:

NAME('object-module-name')

where "object-module-name" has from one through six alphabetic or numeric characters, and begins with an alphabetic character.

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.

NUMBER|NONUMBER
NUM|NNUM

The NUMBER option specifies that the numbers specified in the sequence fields in the source input records are to be used to derive the statement numbers used in the compiler listings.

The position of the sequence field can be specified in the SEQUENCE option. Alternatively, the following default positions are assumed:

- Last 8 columns for fixed-length source input records.
- First 8 columns for undefined-length or variable-length source input records. In this case, 8 is added to the values used in the MARGINS option.

These defaults are the positions used for line numbers generated by CMS; thus it is not necessary to specify the SEQUENCE option, or change the MARGINS defaults when using the line numbers generated by CMS. Note that the preprocessor output has fixed-length records irrespective of the format of the primary input. Any sequence numbers in the primary input

are repositioned in columns 73-80.

The line number is calculated from the five right-hand characters of the sequence number (or the number specified, if less than five). These characters are converted to decimal digits if necessary. Each time a line number is found which is not greater than the preceding one, 100000 is added to this and all following line numbers.

If there is more than one statement on a line, a suffix is used to identify the actual statement in the messages. For example, the second statement beginning on the line 40 is numbered 40.2. The maximum value for this suffix is 31. Thus the thirty-first and subsequent statements on a line have the same number.

The use of NONUMBER is equivalent to the use of STMT, and GONUMBER implies NUMBER see "Relationship of Statement Numbering Options" at the start of this chapter.

OBJECT[(filename)]|NOOBJECT
OBJ[(filename)]|NOBJ

The OBJECT option specifies that the compiler is to create an object module and store it on a TEXT file.

filename is the name that will be given to the text file. If it is omitted, the text file will be given the same name as the file specified in the PLIOPT command. The TEXT file will be placed on one of your disks in accordance with the rules shown in figure 1.2.

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 GONUMBER or GOSTMT option is not in effect.

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

The abbreviated form must be used in the PLIOPT command for NOOPTIMIZE.

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) could 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 OPTIMISE.

OPTIONS|NOOPTIONS
OP|NOP

The abbreviated form must be used in the PLIOPT command for NOOPTIONS.

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

OSDECK
OSD

This is a PLIOPT option and is ignored if used in the *PROCESS statement.

The OSDECK option specifies that the compiler will produce output that can be executed under the control of OS. If the OSDECK option is not used, the first record in the TEXT and SYSPUNCH files is a CMS loader control card specifying the execution time interface module as the entry point. This record results in an error of severity level 8 if it is passed to the OS linkage editor.

There is no negative form, and OSDECK must be specified if it is required.

PRINT|DISK|TYPE|NOPRINT
PRI|DI|TY|NOPRI or NPRI

This is a PLIOPT option and is ignored if used in the *PROCESS statement.

Directs the compiler listing file to the printer (PRINT), a CMS disk (DISK -- this is the default), or to the terminal (TYPE). If NOPRINT is specified, the file is not written.

See figure 1.2 in chapter 1 to determine on which disk the listing file will be placed when you use the DISK option.

SEQUENCE(m n)|NOSEQUENCE
SEQ(m n)|NSEQ

IBM-default: F-format records SEQUENCE(73 80)
V- or U-format records SEQUENCE(1 8)

The SEQUENCE option defines the section of the input record from which the compiler will take the sequence number. (Sequence numbers are used to calculate statement numbers if the NUMBER option is in effect.)

During compiler installation, two default values are set up. One value is for F-format records, the other is for V- or U-format records. The SEQUENCE option overrides only one of

these values. The value overridden is the value that applies to the first record read by the compiler. If a second type of record is found the default sequence values will apply to this type of record.

SEQUENCE(n m)

where:

m specifies the column number of the leftmost digit of the sequence number.

n specifies the column number of the rightmost digit of the sequence number

SIZE(yyyyyyyy|yyyyyK|MAX)
SZ(yyyyyyyy|yyyyyK|MAX)

This option can be used to limit the amount of main storage used by the compiler. This is of value, for example, when dynamically invoking the compiler, to ensure that space is left for other purposes. The SIZE option can be expressed in three forms:

| | |
|---|---|
| SIZE(yyyyyyyy) | specifies that the compiler should attempt to obtain YYYYYYYY bytes of main storage for compilation. Leading zeros are not required. |
| SIZE(yyyyyK) | specifies that the compiler should attempt to obtain YYYYYK bytes of main storage for compilation (1K=1024). Leading zeros are not required. |

SIZE(MAX) obtain as much main storage as it can.

The IBM default, and the most usual value to be used, is SIZE(MAX), which 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.

Note: Under CMS, SIZE(MAX) should always be used unless it is essential to limit the space used. If a limit is set in the SIZE option, the value used will exceed that which is specified. This is because storage is handled by a CMS/compiler interface routine and not directly by the compiler.

The value specified in the SIZE option cannot exceed the main storage available for the job step and cannot be changed after processing has begun. This means that in a batched compilation the value established when the compiler is invoked cannot be changed for later programs in the batch. Thus it is ignored if specified in a *PROCESS statement.

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 option applies, the output from the preprocessor.

STMT|NOSTMT -kh

The STMT option specifies that statements in the source program are to be counted, and that the resulting number is to be used to identify statements in the compiler listings. If NOSTMT is specified, NUMBER is implied. STMT is implied by NONUMBER or GOSTMT. (For further information see "Relationship of Statement Numbering Options" earlier in this chapter.)

STORAGE|NOSTORAGE
STG|NSTG

The abbreviated form must be used in the PLIOPT command for NOSTORAGE.

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.

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

The SYNTAX option specifies that the compiler is to continue into syntax checking after initialization (or after preprocessing if the MACRO option applies) unless an unrecoverable error is detected. The NOSYNTAX option without an argument causes processing to stop unconditionally after initialization (or preprocessing). With an argument, continuation depends on the severity of errors detected during preprocessing, as follows:

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.

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.

TERMINAL[(opt-list)]|NOTERMINAL
TERM[(opt-list)]|NTERM

The abbreviation must be used in the PLIOPT command for NOTERMINAL.

The TERMINAL option is applicable only in a conversational environment. It specifies that some or all of the compiler listing is to be printed at the terminal. If TERMINAL is specified without an options list, diagnostic and informatory messages are printed at the terminal. You can add an argument, which takes the form of an option list, to specify other parts of the compiler listing that are to be printed at the terminal.

The listing at the terminal is independent of that written on the LISTING file. However, if the ddname LISTING is associated

with the terminal, only one copy of each listing requested will be printed, even if it is requested in the TERMINAL option and also as an independent option. The following option keywords, their negative forms, or their abbreviated forms, can be specified in the option list:

AGGREGATE, ATTRIBUTES, ESD, INSOURCE, LIST, MAP, OPTIONS, SOURCE, STORAGE, and XREF.

If the option does not apply to the compiler listing, specifying it in the TERMINAL option has no effect. In the PLIOPT command, abbreviations must be used for any option that exceeds eight characters in length. Values for the other options and suboptions that relate to the listing (that is, FLAG, NUMBER, STMT, LINECOUNT, LMESSAGE/SMESSAGE, MARGINI, NEST, NUMBER and the SHORT and FULL suboptions of ATTRIBUTES and XREF) will be the same as for the LISTING file.

XREF[(FULL|SHORT)]|NOXREF
X[(F|S)]|NX

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. (The only exception is that label references 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 SHORT is specified, unreferenced identifiers are omitted, making the listing more manageable.

If both ATTRIBUTES and XREF apply, and there is a conflict between SHORT and FULL, the usage is determined by the last option found. For example, ATTRIBUTES(SHORT) XREF(FULL) results in FULL applying to the combined listing.

The default suboption FULL means that FULL applies if the option is specified with no sub-option.

# Chapter 4: Execution Time Options

The PL/I Optimizing Compiler produces compiled code to which various execution time options may be passed. These options enable you to control the amount of storage used during execution, and to override the PL/I error handler's attempts to intercept program check interrupts and ABENDs. and, provided that either FLOW or COUNT HAS BEEN SPECIFIED as a compiler option, to specify that a count of the number of times each statement has been executed be generated or that a trace of the most recently executed statements be retained, or both. Execution time options are sometimes called program management parameters.

A set of default execution time options are established during system generation. These can be overridden by options specified in a PL/I variable PLIXOPT, and these in turn can be overridden by options specified with the START command or with the filename when it is used as a command.

To specify execution time options within a PL/I program, you must use the following declaration:

```
DCL PLIXOPT CHAR(len) VAR INIT ('strg') STATIC EXTERNAL;
```

where "strg" is a list of options separated by blanks or commas, and "len" is a constant equal to or greater than the length of "strg". Similarly PLIXHD allows you to specify headings for REPORT and COUNT output. For example:

```
DCL PLIXHD CHAR(50) VAR STATIC EXTERNAL INIT('RUN WITH DUMMY DATA');
```

If more than one external procedure in a job declares PLIXOPT as STATIC EXTERNAL, the PLIXOPT value in the first program passed to the loader will be taken as the list of options and the second and subsequent PLIXOPT valuess ignored. The same is true of PLIXHD.

The execution time options can be specified with the START command or with the filename of a MODULE file when it is used as a command. If a parameter is also being passed to the main procedure, it must follow any execution time options and be preceded by the characters blank, oblique stroke, blank( / ). Program management parameters must be separated from each other by blanks.

A typical START command specifying execution time options (NOSPIE and REPORT) and a main procedure parameter (734) might be:

```
start * nospie report / 734
```

## List of Execution Time Options

The following is a list of execution time options:

COUNT

> specifies that a COUNT of the number of times each statement in the program was executed will be produced if either the COUNT or FLOW option was specified at as a compiler option. (If neither was specified as a compiler option, an error message is issued and the request for COUNT is ignored.)
>
> The count is transmitted to the PLIDUMP file when the program has completed execution. To highlight statements that have not

been executed, a separate list of such statements is produced. Output is headed by the name of the main procedure, the date and time of the start of execution, and the value of PLIXHD if PLIXHD is used.

NOCOUNT

specifies that a count of the number of times each statement has been executed will not be produced. NOCOUNT is used to prevent a program compiled with the COUNT option from producing count information. Even when NOCOUNT is specified, a considerable time and space overhead is incurred by a program compiled with the COUNT option. To get the best performance a debugged program must be recompiled without the COUNT option.

FLOW

specifies that a trace of the most recently executed statements will be retained and that this will be printed when an on-unit with the SNAP option is entered or when a call to PLIDUMP with the trace option is made. The option is only effective if either FLOW or COUNT was specified as a compiler option. (If neither was specified an error message is issued and the option is ignored.)

The format of the FLOW option is FLOW [(n m)] where n specifies the number of branch-out/branch-in statement number pairs to be retained, and m specifies the number of changes of procedure or on-unit that are retained. n and m can have different values from those specified in the compiler FLOW option. If n and m are omitted both at compiler time and at execution time, default values of 25 for n and 10 for m are assumed.

The trace is transmitted to SYSPRINT and takes the form:

```
3 TO 8 IN TESTER
12 TO 17
22 TO 3 IN DRIVER
```

Meaning that a branch was made from statement three to statement 8 which is in the procedure named TESTER, than ran sequentially to statement 12 when a branch to 17 was made then ran sequentially to 22 where a branch to statement 3 which is in DRIVER was made.

NOFLOW

Specifies that a trace of the most recently executed statements will not be retained. It is used to override the FLOW compiler option.

Even when NOFLOW is specified, considerable time and space overheads are incurred by programs compiled with the FLOW option. When a program has been debugged it should be recompiled without the FLOW COMPIler option to achieve maximum efficiency.

ISASIZE ([-]yyyyy|[-]yyyyyK)

specifies the amount of main storage initially acquired for automatic, controlled, and based variables, and compiled code workspace. The area obtained is known as the ISA (initial storage area).

If a negative value is specified, all available storage minus the amount specified is used. "Available storage" is what is left in the region when the load module has been allocated.

Allocation of PL/I dynamic storage when PROCEDURE or BEGIN blocks are entered or BASED or CONTROLLED variables are allocated storage is allocated as far as possible within the ISA. When there is insufficient room, storage is acquired from the system and a time overhead is involved. However if a large value is specified in ISASIZE, storage may be wasted, and there may be insufficient storage for I/O buffers and transient library routines.

If ISASIZE is not specified, a default value is applied. This value is half of the storage remaining in the region after storage for the load module has been allocated rounded up to the nearest 2K bytes.

The REPORT execution time option can be used to help work out the optimum ISASIZE.

**REPORT** specifies that a table showing the use of storage by the program will be transmitted to the PLIDUMP file at the end of the execution of the program. Under CMS the PLIDUMP file is assigned to the printer by default. Output is headed by the name of the main procedure, the date and time of the start of execution and the value of PLIXHD if PLIXHD is used.

The REPORT option should be used to help calculate the best value to specify in ISASIZE. The value given in the REPORT table "Amount of PL/I Storage Used" would give the fastest execution with the minimum total waste of storage if specified as the ISASIZE. However, if a number of PL/I blocks or controlled or based variables are little used during the program, the programmer may prefer to have storage for some of these allocated by the system. In this situation, specifying a smaller ISASIZE value may enable the program to run in a smaller region, although execution time may increase. For a fuller discussion see the Programmer's Guide for the compiler.

Note: The use of the REPORT option considerably slows execution. It is intended as an aid for program development, not for regular use.

**NOREPORT** specifies that a report table will not be generated. It is the default.

**STAE** specifies that when an ABEND occurs, an attempt will be made to call the PL/I error handler and raise the PL/I ERROR condition. It is the default.

**NOSTAE** specifies that on program initialization, a STAE macro instruction is not to be issued, and consequently the PL/I error handler will not be called to attempt to raise the ERROR condition when an ABEND occurs.

**SPIE** specifies that when a program interrupt occurs, an attempt will be made to call the PL/I error handler to raise the ERROR condition. It is the default.

**NOSPIE** specifies that on program initialization, a SPIE macro instruction is not to be issued, and consequently the PL/I error handler will not be called to raise the ERROR condition when a program check interrupt occurs.

# Index

Explanation of sample terminal session

The terminal session has been planned to show various features of CMS. The program is a simple conversational program that responds with one of two well known quotations when the correct author is specified. It has been written to show the conversational I/O and parameter conventions of PL/I under CMS.

The first column in the figure shows whether the terminal print out is entered by the user or is transmitted by the system. The second column shows the terminal printout. Where an action from the user would not result in words appearing on the terminal printout, the action to be taken is placed in parentheses. For example "(you switch on terminal)" in line 1. The third column contains notes and comments. The fourth column gives the page of the book where a fuller explanation of the point being illustrated can be found. Throughout the example certain blank lines have been omitted to allow the complete session to appear on one page.

| Action by | Terminal Printout | Notes and comments | Page for more data |
|---|---|---|---|
| | | | |
| user | (you switch on terminal) | | Page 2 |
| system | d'x38z irvy; vm370 online | Message when you switch on | |
| user | (you press attention key to unlock terminal) | | |
| | login robin | Enter 'login' followed by name of virtual machine | |
| system | ENTER PASSWORD: | | |
| user | (you enter password) | Printing of password normally suppressed | |
| system | LOGMSG 08:09:08 GMT MONDAY 05/13/73 | | |
| | LOGON AT 08:25:34 GMT MONDAY 05/13/73 | Log message from system | |
| user | ipl cms | Invoke CMS | Page 4 |
| system | CMS 1.0 PLC 5 | Message shows CMS version in use | |
| user | edit skylark pliopt | Edit mode to enter program as a CMS PLIOPT file | Page 8 |
| system | NEW FILE: | Shows that you have no PLIOPT file called skylark | |
| | EDIT: | Shows that you are in edit mode | |
| user | input | Tell system further input will be part of file | |
| system | INPUT: | Shows that you are in input submode | |
| user | skylark:proc (charparm) options (main); | PL/I program entered in either capitals or lower | |
| | dcl (charparm,quotation,string) char (100) var; | case letters. Use column 1 through 71 | Page 8 |
| | string=translate(charparm,' ',','); | CMS interface removes blanks from main procedure | |
| | on endfile (sysin) goto finis; | parameter. Program uses commas and translates. | |
| | start: | | |
| | if string='percy bysshe shelley' then quotation= | | |
| | 'hail to thee blythe spirit'; | | |
| | esl@@lse if string='william blake' then quotation= | @@ deletes two previous incorrect characters | Page 4 |
| | 'a skylark wounded on the wing/a cherubim doth cease to sing'. | | |
| | else quotation='no known quotation'; | | |
| | put skip edit (quotation,'enter new name or endfile') | | |
| | (a,skip(5)); | Skip(5) is interpreted as skip(3) at terminal | Page 39 |
| | get edit(string)(a(80)); goto start; | | |
| | finis: display('thank you for your company'); | Sent to terminal (console of virtual machine) | Page 42 |
| | end skylark; | | |
| user | (you press carriage return key) | Ends input submode | Page 9 |
| system | EDIT: | Message confirms you are back in edit mode | |
| user | file | Stores input as PLIOPT file skylark | |
| system | R; T=0.35/0.91 08.26.32 | Ready message, CMS ready for further commands | |
| user | pliopt skylark (xref a | Compile command, options preceded by ( | Page 14 |
| system | PL/I OPTIMIZING COMPILER V1 R1.2 TIME 08.34.51 DATE 13 MAY 1973 | | |
| | OPTIONS SPECIFIED | | |
| | XREF,A,TERM | TERM specified by CMS Interface module | |
| | | | |
| | NO MESSAGES PRODUCED FOR THIS COMPILATION | | |
| | COMPILE TIME 0.02 MINS SPILL FILE 0 RECORDS SIZE 4051 | | |
| system | R; T=2.74/4.41 08.38.37 | | |
| user | global txtlib plilib | Make the PL/I library available | Page 21 |
| system | R; T=0.03/0.04 08.41.49 | | |
| user | load skylark | resolve addresses in PL/I program | |
| system | R;T=1;11/1.85 08.50.06 | | |
| user | start * / percy, bysshe, shelley | Note parameter must be divided into 8 character | |
| | | tokens. Blanks are removed. Note also blanks | |
| | | after * and / | |
| system | EXECUTION BEGINS... | Message from CMS | |
| | HAIL TO THEE BLYTHE SPIRIT | Output from program | |
| | | | |
| | | | |
| | ENTER NEW NAME OR ENDFILE | | |
| | : | Prompt shows input required from terminal | Page 40 |
| user | /* | Endfile marker | |
| system | THANK YOU FOR YOUR COMPANY | Message from DISPLAY statement | Page 42 |
| | R;T=1.52/2.46 08.45.06 | | |
| user | logout | Command ends terminal session | Page 33 |
| system | CONNECT=00.33.59 VIRTCPU=000:09.11 TOTCPU=000:16.55 | | |
| | LOGOFF AT 08:59:33 GMT MONDAY MAY 13 1973 | Logoff message | |
| user | (you switch off terminal) | | |

Figure F.1 A sample terminal session

OS
PL/I Optimizing Compiler:
CMS User's Guide
Order No. SC33-0037-3

**Reader's Comment Form**

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

**Fold on two lines, staple, and mail.** No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your

SC33-0037-3

Fold and Staple

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

**IBM Corporation**
**P.O. Box 50020**
**Programming Publishing**
**San Jose, California 95150**

Fold and Staple

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of

OS PL/I Optimizing Compiler: CMS User's Guide (File No. S360/S370-29) Printed in U.S.A. SC33-0037-3