PL/CT - A Terminal Version of PL/C
Release 2

User's Guide to the Cornell-CMS Version

C. G. Moore III, S. L. Worona & R. W. Conway

TR 75-259

September 1975

Department of Computer Science
Cornell University
Ithaca, NY 14853

Cornell University
Department of Computer Science -- Office of Computer Services

PL/CT - A Terminal Version of PL/C
Release 2

User's Guide to the Cornell-CMS Version

C. G. Moore III, S. L. Worona and R. W. Conway

PL/CT is a special version of PL/C designed to permit programs to be run interactively from a typewriter terminal. It is completely compatible with normal PL/C -- that is, the source languages accepted by PL/C and PL/CT are identical and the results of execution are exactly the same. Hence a program can be developed and tested under PL/CT and subsequently run under normal PL/C (or vice-versa).

PL/CT permits the user to interact with the program during its execution. Output will be printed on the terminal and input data may be requested from the terminal. The course and rate of execution can be controlled from the terminal. It is also possible to interrupt execution and display and alter the values of variables. However, the source program itself cannot be changed under PL/CT. PL/CT receives a complete program, compiles it, and then executes it in interactive mode. But to make any change in the program it is necessary to leave PL/CT, make the change under the CMS editor, and then present the modified program to PL/CT for complete recompilation.

This Guide provides only minimal information about CMS, perhaps sufficient for very straightforward programming tasks. For additional information see the following publications: IBM VM/370: Command Language Guide for General Users (GC20-1804); IBM VM/370: EDIT Guide (GC20-1805); IBM VM/370: Terminal User's Guide (GC20-1810).

Contents

<u>in-on Procedure and Terminal Usage</u> (2741 terminal)

To initiate a CMS session from a dial-up 2741 terminal with acoustic coupler, perform the following steps:

1. Turn on the terminal (switch at right of keyboard)
2. Place telephone handset in coupler.
3. Turn on the coupler.
4. Dial (9-) 257-1410
5. Wait for terminal keyboard to unlock, then enter <u>pcpc</u> in lower case (no shift) and press "ATTN" to transmit the entry to the system. The system will then respond with the line "VM/370 Online". Press "ATTN".
6. Enter <u>logon act mask</u> (again in lower case) where "act" is your three-character account number. Press "RETURN" to transmit the line.
7. The system will respond with "ENTER PASSWORD:". Enter your password and press "RETURN".
8. The terminal will then reply "CORNELL CMS" indicating that it is in <u>command mode</u> and ready to receive a CMS command.

After you enter a line on the terminal it is transmitted to e system by pressing "RETURN". You must then wait for the stem to process the line before entering another line. The rminal keyboard will be locked until this processing is done. many cases the terminal will indicate its readiness with some ompting message, but in some cases there will only be a faint ick as the keyboard unlocks ready for your next entry.

To correct a typing error on the terminal, two control aracters are available: the "at" sign (@), and the cent sign). Typing an @ deletes the previously typed character; two s delete the previous two characters, etc. For example:

GLUB@@@@QQSVV@ is equivalent to QQSV. The ¢ character deletes all characters previously typed on that line, so that GLUB¢QQSVV@ is equivalent to QQSV. Note that you cannot use @ or ¢ in your program -- there is no way to input these characters.

Normal practice is to enter letters in lower case, since this helps to distinguish your input from the terminal response on the listing. Letters are automatically stored internally in upper case, no matter how they are typed in.

Levels and Modes

The most complicated aspect of using CMS-PL/CT is understanding that you are communicating with the system at several different levels. Sometimes you are entering commands telling the command processor what to do, sometimes you are entering lines that are PL/I source statements, and sometimes you are entering data required for the execution of your PL/I program. It is essential that you understand the difference between these levels, and that you understand the means by which you indicate the proper level to the system.

CMS handles this problem of levels by establishing different "modes" of communication. The highest level is called command mode. When the system is in command mode it assumes that anything entered is a command (and not a program line or data). Two of these commands change the mode of the system: the EDIT command causes the system to enter EDIT mode; the PLC command causes it to enter PL/CT mode. When the system is in EDIT mode it assumes that anything entered is an EDIT sub-command. One of these sub-commands causes the system to shift to INPUT mode in which it assumes that everything entered is a line to be stored in a dataset. Similarly there are modes within PL/CT that determine whether input from the terminal is considered to be source program lines, execution data, or execution debugging sub-commands. It is difficult to describe, but it works fairly naturally in practice. The relationship between levels, modes, commands and sub-commands can be summarized in the following table:

CMS Command Mode

Input:

- EDIT to enter EDIT mode
- ERASE to delete a dataset
- LISTFILE to list the names of datasets
- LOGOFF to end the terminal session
- PLC to enter PL/CT mode

EDIT Mode

Input: (sub-commands)

- INPUT to enter INPUT mode
- DELETE to delete 1 or more lines
- LOCATE to locate a certain line

```
        CHANGE to change part of a line
        TYPE to display 1 or more lines
        TOP, BOTTOM, UP, DOWN to move the line pointer
        SAVE to save a copy of a dataset
        QUIT to return to command mode

    INPUT Mode
        Input:
            Lines to be inserted in dataset
            Null line to return to EDIT mode
PL/CT Mode
    Input:
        Source program lines
        Execution data
        Debug commands
    Returns to CMS command mode when program execution
            is completed, or STOPped, or two
            consecutive ATTNs occur.
```

Command Mode

The system indicates that it is in command mode with the prompting message 'R; T=runtime time-of-day'.

Commands (optional abbreviation below full form)

EDIT dataset-name PLC

"Dataset-name" is a string of not more than 8 characters, beginning with a letter.

If "dataset-name" is the name of an existing dataset, then EDIT will retrieve that dataset and EDIT mode is entered to allow you to make modifications to that dataset. If "dataset-name" is not the name of any existing dataset, then EDIT will create a new dataset with that name. In this case, when you enter EDIT mode, EDIT will type "NEW FILE", to indicate it has created a new, empty dataset. You should then use the INPUT sub-command to enter lines into the new dataset.

ERASE dataset-name
Delete the indicated dataset.

LISTFILE * * A

List the contents of your dataset catalog -- the names of all your datasets. (The names will appear in "full" form, rather than the simple form that is sufficient for the purposes described here.)

LOGOFF
End the terminal session. The system will reply with a

line giving the cost of the session. The system leaves command mode and will accept no further commands. Turn off the terminal and the coupler and replace the telephone handset.

PLC sp-list DATA(d-list) OPTIONS(op-list) PAUSE SAVE n
Cause the system to enter PL/CT mode to compile and execute a PL/C program. "Sp-list" specifies the source program; "d-list" specifies the input data for execution of the program; "op-list" specifies PL/C options; PAUSE causes a return to the terminal before beginning execution of the program; "SAVE n" saves n lines of the source listing for display during execution. All of the phrases after PLC are optional; their order is immaterial except that sp-list (if given) must come first.

Because of CMS restrictions a PLC command must not contain a sequence of more than 8 characters without a blank character appearing. This will not be a problem if you use the following rules in typing a PLC command:
(1) always type a blank after a comma
(2) always type a blank before or after a parenthesis (left or right)
(3) use dataset-names of 6 characters or less

The source-program dataset (specified by sp-list) contains lines equivalent to the source program cards submitted to batch-PL/C -- including PL/C control cards (*PL/C, *PROCESS, etc.) Sp-list should be given in one of the following forms:
1. a dataset name
2. an asterisk, indicating program will be entered from the terminal. This is the default assumption if no source-program-list is given.
3. a list of dataset names, separated by commas and enclosed in parentheses. The datasets listed are "concatenated" -- the first line of one dataset follows the last line of the predecessor dataset on the list -- and presented as a single dataset to PL/CT.

Data for the execution of the program is obtained in one of the following ways:
1. If the concatenated source-program dataset contains a *DATA line, input data will be drawn from that dataset. The DATA option on the PLC command should be omitted.
2. If there is no *DATA line in the source-program dataset and the DATA option is omitted, then DATA(*) is assumed and PL/CT will return to the terminal for input data.
3. If there is no *DATA line in the source-program dataset and the DATA option is given on the PLC command then input data will be drawn from the dataset concatenated from the items given in d-list. Items

may be datasets or asterisks, as in sp-list.

When a dataset is entered from the terminal, its end must be indicated by entering a null line consisting of simply a carriage return.

Op-list specifies PL/C options to be applied _after_ any options that may be given on a *PL/C card in the source-program dataset. (This may be used to override the *PL/C options. A *PL/C card need not be present.) Remember when typing in this list that CMS will not accept more than 8 characters without a blank.

PAUSE is given if you want your program to enter debug mode just before program execution begins. It is equivalent to pressing "ATTN" just as program execution begins (which is hard to do). This will give you the opportunity to set PAUSEs in the program before it begins to execute.

PL/CT will save a copy of the source listing for display during execution unless the NOSAVE option is given. SAVE n saves the _first n lines_ of the listing. The default is SAVE 200. This feature uses a lot of memory, so if you don't need it, specify NOSAVE; if you have a small program, specify an n < 200.

Examples:

PLC
: Both source program and input data are to come from the terminal. The source program lines must conclude with a *DATA line to initiate execution, just as in batch-PL/C.

PLC PROB1
: Source program is to come from dataset PROB1. If this includes a *DATA card it will also supply data, otherwise data will come from the terminal.

PLC * DATA (P1DATA) SAVE 50
: Source program is to come from the terminal; input data from P1DATA. Save only 50 lines of the source listing.

PLC PROX4 DATA (XDATA) OPTIONS (ATR, XREF)
: Source program is to come from PROX4, data from XDATA. The cross-reference and attribute listing is to be printed.

PLC (*, PRG1) PAUSE NOSAVE
: Source program is to come first from the terminal (perhaps to supply a *PL/C card) and then from PRG1; input data is to come from the terminal (unless PRG1 includes a *DATA line). Return to the terminal before beginning execution. Do not save the source listing.

PLC (CS104, LIBR) DATA (INIT, *)
: Source program is to come from CS104 followed by LIBR, data is initially to come from INIT and

from the terminal when that is exhausted.

Special note: There is a difference between the two commands PLC * and PLC * DATA (*). The first command expects the input from the terminal to consist of the source program, followed by a *DATA card, followed by the program data. The second form expects the input to consist of the source program, followed by an end-of-file (carriage return), followed by the program data.

EDIT Mode

The EDIT facility permits the creation and modification of datasets. The following describes a portion of the full EDIT, assuming you are working with a relatively small dataset and that you have a listing of that dataset available. The full EDIT is much more powerful and flexible than what is described here -- see the IBM VM/370 EDIT Guide (GC20-1805).

The editor keeps track of a "current-line-pointer", which always points to the "current line" in your dataset. Most EDIT sub-commands use the current-line-pointer to determine where editing is to be done, and most alter the current-line-pointer as part of their action. For example, the DELETE command deletes lines beginning with the line pointed to by the current-line-pointer, and then sets the current-line-pointer to point to the next line after those deleted.

__EDIT Sub-commands__ (optional abbreviation below full form)

INPUT
I

Enter the __input__ sub-mode of EDIT. Subsequent lines are inserted into the dataset after the current line. To terminate this sub-mode enter a "null line" (carriage retu-n only).

DELETE n
DEL

Delete n lines, beginning with the current line. If n is omitted, delete only the current line. The new current line is the first line __after__ those deleted.

LOCATE /string/ or LOCATE 'string'
L

Beginning with the current line, search the dataset for the first line containing the sequence of characters given by "string", and make that line the new current line. Use the first form (delimited by slashes) if the string you are searching for contains no /; use the second form (delimited by quotes) if it contains a / but no quotes. If the search is unsuccessful (the string is not found in the portion of the dataset searched) the current-line-pointer is set to point to the last line of the dataset. For example: LOCATE /QQSV/ searches the dataset from the current line to the end of the dataset for the first occurrence of the string of characters "QQSV".

HANGE /string1/string2/ or CHANGE 'string1'string2'

Replace the leftmost occurrence of "string1" in the current line with "string2". "String1" and "string2" do not need to be of the same length. For example, if the current line is: "THIS IS A LINE" the sub-command: CHANGE /IS/WAS/ would change the line to: "THWAS IS A LINE".

YPE n

Print n lines of the dataset, beginning with the current line. The last line printed becomes the current line.

OP

Set the current-line-pointer to an imaginary line before the first line of the dataset (so that INPUT can be used to insert lines at the beginning).

OTTOM

Set the current-line-pointer to the last line of the dataset.

P n

Move the current-line-pointer up n lines (1 line if n is omitted).

OWN n
O

Move the current-line-pointer down n lines (1 line if n is omitted).

AVE or SAVE dataset-name PLC

First form: copy the current version of the dataset into the file whose name was given in the EDIT command, replacing the old version. Second form: copy the current version of the dataset into the file whose name is given in the SAVE sub-command. EDIT makes a _temporary copy_ of the specified dataset when you give the EDIT command, and all _changes are performed on this copy_. Therefore, if you do not specify SAVE before QUITing, your changes will be lost.

UIT

Terminates the EDIT command. Normally this will be given just after a SAVE sub-command.

PL/CT Mode

The PL/CT source language is identical to PL/C, but the following default options are different (to reduce the amount of printing):

CMPRS	NOHDRPG
NODUMP, NODUMPE, NODUMPT	NOOPLIST
FLAGE	NOSOURCE

PL/CT Source Program Entry

Programs can be entered in one of two ways:

1. directly to PL/CT from the terminal, or

2. by preparing a dataset which is then presented to PL/CT.

For all but trivial programs the second method should be used, since it provides a means of saving the source program for subsequent reuse and/or modification. If a program is presented directly to PL/CT it is not saved in the system and must be reentered to be re-run.

Note that once a source line has been presented to PL/CT (either from the terminal or from a dataset) there is no way within PL/CT to change that line. You must leave PL/CT, change the program, and then re-invoke PL/CT.

Terminal Use During Execution

During execution of a program the PL/CT terminal has two distinct roles:

1. It serves as the normal (that is, SYSIN/SYSPRINT) input/output device for the program. The printed output from PUT statements will appear on the terminal; GET statements will request input data from the terminal (assuming the d-list of the PLC command specifies the terminal). There is <u>no automatic prompt</u> when the program is requesting input data -- hence it is generally good programming practice to place a PUT statement with a prompting message <u>immediately before</u> each terminal GET.

2. In "debug mode" the terminal is used to enter PL/CT "debug commands" -- statements for immediate execution. Debug mode can be entered in the following ways:

   a. Give the <u>PAUSE option</u> on the PLC command that invokes PL/CT. This simulates an "attention interrupt" during the first statement of the program, and enters debug mode before the second statement. This gives you an opportunity to set PAUSEs in the program before its execution begins.

b. Strike the "attention" ("ATTN") key at any time during program execution. The program will complete execution of whatever statement is being executed and enter debug mode before beginning the next statement. If the statement being executed is a PUT causing printing on the terminal, the statement will be completed, but actual printing of the final lines will be suppressed. Moreover, because of the buffering of printed output both in PL/C and in the operating system, the handling of printed output on an attention interrupt is sometimes rather difficult to understand. Be careful that you hit attention only once, and then give the system a chance to respond. Hitting two consecutive attentions (without any intervening action) will cause the system to leave PL/CT and return to command mode, losing all trace of the program's execution.

c. After each non-fatal execution error, PL/CT will automatically enter debug mode. PL/CT prints the usual error message, makes the usual PL/C error repair, and then enters debug mode before beginning the next statement.

d. When a PAUSE, or "breakpoint" is encountered in the source program, PL/CT enters debug mode. PAUSEs may be set and removed by the debug commands, described below.

e. When a specified number of statements of the original program have been executed, PL/CT will enter debug mode. This "STEP interval" can be set by debug commands, described below.

n each case, PL/CT will print a message indicating the reason or entering debug mode, and the statement number of the next atement to be executed. This message will end with the rompting symbol "DBC:", indicating that PL/CT is in debug mode, aiting for a debug command to be entered on the terminal. ter each debug command line the prompt "DBC:" will be repeated o indicate that the system is still in debug mode and is ready o receive another debug command.

The "debug mode" and the "input data mode" are completely istinct -- you cannot enter data when PL/CT expects a debug ommand, and you cannot enter a debug command when it expects ata.

PL/CT Debug Commands

When the system is in debug mode any of the following commands may be given. Each command is executed immediately; it is not saved, and does not become part of the source program. The format for commands is free-field -- essentially the same as for statements in PL/C, except:

1. Comments are not allowed.
2. Commands may begin in position 1 of the line.
3. Commands cannot be continued onto a second line.

PUT SKIP LIST(variable, ...);
PUT SKIP DATA(variable, ...);

A restricted form of the PL/C PUT statement. The variable specified can be a scalar, an array, a structure or a subscripted variable with a constant subscript. Variables must be accessible at the point of interrupt under normal PL/C scope rules. Neither expressions nor literals can be given.

SKIP is assumed and need not be given.

If neither LIST nor DATA is specified the default output format will be used. If either LIST or DATA is specified, either in a PUT or as a separate command (see below), this sets the default output format. Initially the default is LIST.

This command may be abbreviated as just "PUT variable;" or just as the variable name alone. That is, assuming that LIST is the default output format, "X;" and "PUT X;" are equivalent to PUT SKIP LIST(X);".

LIST;

Set the default output format (for debug commands only) to be LIST.

DATA;

Set the default output format (for debug commands only) to be DATA.

PUT m, n;

m is a statement number from the source listing, and n is an integer. Display n source lines beginning with the line on which statement m started. If n is omitted from the command, 1 is assumed.

m can also be given as a label or entry-name, accessible from the point of interrupt under the normal PL/C scope rules.

Variable = constant;

A restricted form of the PL/C assignment statement. The target variable must be a scalar or a subscripted variable with constant subscript(s). It cannot be a label variable, an array or a structure. Structure elements must be fully-qualified. Multiple left sides and BY NAME assignment are not allowed. The right side can only be an arithmetic or string constant -- neither a variable nor an expression is allowed.

STEP n;

n is an integer. Reset the STEP interval to n, so that PL/CT will re-enter debug mode after execution of n statements of the source program. If n is omitted, 1 is assumed. This STEP interval remains in effect until changed -- it does not just apply to the first RETURN. Note that statements are counted in a manner comparable to PL/C numbering -- that is, END, PROCEDURE, DO are also counted as statements.

NOSTEP;

Reset the STEP interval to the default value: STEP 2^{16}.

PAUSE AT s;

Establish a PAUSE before statement(s) s. s can be given in several forms:

- a statement number, as given on the PL/C source listing

- a label or entry-name, which is accessible at the point of interrupt under normal PL/C scope rules

- an accessible label or entry-name modified by an integer. For example:

```
PAUSE AT ERRORPROC+6;
PAUSE AT TERMLOOP-3;
```

- an inclusive range of statements: "s^1 TO s^2" where s^1 and s^2 are any of the forms listed above. s^2 can also be the word END, implying the last statement of the program. For example:

```
PAUSE AT 14 TO TERM_LOOP;
PAUSE AT EVALPROC+3 TO EVALPROC+14;
PAUSE AT PRINT+6 TO END;
```

- ALL, which means "1 TO END".

The PAUSE command may be abbreviated by giving s (or s^1 TO s^2) alone. That is, if a command consists of any of the valid forms for s, "PAUSE AT s;" is assumed. For example, "36;" is equivalent to "PAUSE AT 36;".

PAUSEs are maintained in a list of fixed length within PL/CT. When this list is full, further PAUSE commands will be rejected. You will have to remove some PAUSEs before new ones can be added.

NOPAUSE AT s;

Remove the PAUSE (if any) before statement(s) s. s is given in the same forms as for the PAUSE command. Note that NOPAUSE can have a range <u>but not a list</u> of arguments. That is, "NOPAUSE AT s^1, s^2;" is <u>not valid</u>. (s^2 will be considered a separate command -- an abbreviation of "PAUSE AT s^2;".) Also note that since removing the middle of a PAUSE range actually creates two ranges, it is possible for NOPAUSE to cause overflow of the PAUSE list.

IGNORE n;

n is an integer. During program execution ignore the first n PAUSEs encountered; re-enter debug mode on the n+1st PAUSE. If n is omitted, 2^{16} is assumed. This IGNORE count remains in effect until changed -- it does not just apply to the first RETURN. Initially, the IGNORE count is 0 -- that is, PL/CT will stop on every PAUSE unless you set the PAUSE count to some non-zero value.

NOCHECK;

Suppress the printing of CHECK output, exactly as in PL/C.

CHECK;

Resume the printing of CHECK output, as in PL/C except that no parameters are allowed on the command.

NOFLOW;

Suppress the printing of FLOW output, exactly as in PL/C.

FLOW;

Resume the printing of FLOW output, as in PL/C except that no parameters are allowed on the command.

PUT OFF;

Suppress printing of SYSPRINT output, exactly as in PL/C.

PUT ON;

Resume printing of SYSPRINT output, exactly as in PL/C.

PUT ALL;

Display the current values of all automatic, scalar variables in the blocks active at the point of interrupt, as well as the current values of all static and external scalar variables, exactly as in PL/C.

PUT ARRAY;

Same as PUT ALL but also includes arrays, exactly as in PL/C.

PUT FLOW;

Display recent FLOW history, exactly as in PL/C.

PUT SNAP;

Display recent calling history, exactly as in PL/C.

ETURN;
Leave debug mode and resume execution of the source program. RETURN can be indicated by a null line. That is, after the "DBC:" prompt a carriage return with an empty line is equivalent to a RETURN command.

O To label;
Leave debug mode and resume execution of the source program starting with the statement whose label is given. This label must be accessible from the point of interrupt under the normal PL/C scope rules.

TOP;
Terminate execution of the PL/CT program, exactly as in PL/C.

L/CT Errors

When errors are detected during compilation of a program the sual PL/C action is taken. That is, a message is printed, some epair is automatically effected, compilation continues and xecution will be attempted. (There are a few cases in which hese errors are "fatal" and execution is suppressed.) If the epair is not satisfactory you must leave PL/CT mode, alter the ource program, and then re-submit it to PL/CT.

Similarly, during execution of the program PL/CT gives the tandard PL/C response -- message and repair -- but then returns ontrol to the terminal (before executing the next statement of he program) and requests a debug sub-command.

An error in a debug command will cause a message to be rinted, followed by a prompt "DBC:" for re-entry of the ommand. The complete command must be re-entered -- not just rom the point of error. However, if several debug commands ere given on the line containing the error, commands to the eft of the erroneous command will have already been executed nd should not be re-entered.