File No. S360-29
GC28-2049-1

TSS

IBM Systems Reference Library

Version 8.1

IBM System/360 Time Sharing System PL/I Programmer's Guide

This publication is a companion volume to IBM System/360 Time Sharing System: PL/I Language Reference Manual, GC28-2045. Together, the two books form a guide to the writing and execution of PL/I programs under the control of an IBM System/360 Time Sharing System that includes a PL/I compiler. This publication is concerned with the relationship between a PL/I program and the Time Sharing System. It explains how to compile and execute a PL/I program, and introduces the command system, data management, and other essential features of TSS/360.

PREFACE

This publication is a guide to the facilities of the IBM System/360 Time Sharing System (TSS/360) for the PL/I user. It explains how to compile and execute a PL/I program, and introduces the command system, data management, and other essential features of TSS/360.

Part I explains how to use PL/I on TSS/360 without previous knowledge of TSS/360; Parts II and III describe the more advanced facilities of the TSS/360 PL/I compiler and provide a brief survey of TSS/360 features available to the PL/I user.

PREREQUISITE KNOWLEDGE

Readers should be familiar with the PL/I language, since this book does not describe the language, but rather the use of the system.

The PL/I user will find the language specified in these publications:

IBM System/360 Time Sharing System: PL/I Language Reference Manual, GC28-2045.

IBM System/360 Time Sharing System: PL/I Library Computational Subroutines, GC28-2046.

If additional knowledge of the time-sharing system is needed, the following publications should be referred to:

IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003, provides a broader system survey than does Part I of this manual.

IBM System/360 Time Sharing System: Command System User's Guide, GC28-2001, describes the entire command language.

IBM System/360 Time Sharing System: Data Management Facilities, GC28-2056, describes, in detail, the system's facilities for data management.

IBM System/360 Time Sharing System: System Messages, GC28-2037, lists all of the messages produced by the system.

IBM System/360 Time Sharing System: Terminal User's Guide, GC28-2017, gives details of the facilities and operations of the various terminals supported by TSS/360.

If you have access to a remote job entry (RJE) device and you want to use the RJE feature, see IBM System/360 Time Sharing System: Remote Job Entry, GC28-2057.

Second Edition (September, 1971)

This is a major revision of, and replaces, the previous edition, GC28-2049-0, and Technical Newsletter GN28-3160.

This edition documents changes to the PLI command and the TSS/360 command system. Use of the linkage editor and the program control system are explained in greater detail. Sharing of PL/I modules is explained. Two new sections, "Terminal I/O," and "Interface between PL/I and Assembler Programs," have been added to Part II.

This edition is current with Version 8, Modification 1, of IBM System/360 Time Sharing System (TSS/360), and remains in effect for all subsequent versions or modifications of TSS/360 unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, GC28-2043, which may contain information pertinent to the topics covered in this edition. The Addendum also lists the editions of all TSS/360 publications that are applicable and current.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System/360 Programming Publications, Department 641, Neighborhood Road, Kingston, N.Y. 12401

CONTENTS

ILLUSTRATIONS

PART I: BASIC PROGRAMMING WITH THE PL/I COMPILER

Time Sharing System/360 is a comprehensive programming system used in conjunction with IBM System/360 computers that have time-sharing features. The primary purpose of TSS/360 is to provide many users with simultaneous conversational (online) access to a computing system that may have a single processor, or multiple processors. The combination of machine and program features gives you the impression that you have sole possession of the system. You use the system as if it had a main-storage capacity equal to the largest address that can be written, rather than its actual main-storage capacity.

In TSS/360 you can run a program conversationally: you and the system exchange information during the execution of your program. You can also run a program nonconversationally, without access to the system during program execution.

You can run in mixed mode -- that is, start a program conversationally and switch to nonconversational processing. Once a program is running nonconversationally you cannot switch back to conversational processing, however.

THE SYSTEM

TSS/360 is a set of programs that makes use of a computer easier:

- A supervisor program controls the overall operation of the system, and provides the time sharing environment that lets a number of users employ the system concurrently.
- A group of service routines perform program control and data management functions for each user, as well as for the system.
- A third set of programs is provided to allow you to compile and develop your problem programs.

This publication explains how to use these programs, without involving you in their structure or their internal operations.

VIRTUAL STORAGE

Virtual storage is the name given to the address space referenced directly by the processing unit of a System/360 that is equipped with time-sharing machine features. This address space is as large as the addressing capability of the system; that is, if you can write an address, the location addressed can be included in your virtual storage. The number of addressable positions is not limited by the size of main storage; in TSS/360, you are not directly concerned with the installation's physical limitations on main storage.

Although the addressing range of virtual storage is equal to the addressing capability of the system, you are constrained to a virtual storage capacity that is somewhat less than that limit. The constraints are determined by the installation, on the basis of such considerations as main storage capacity, secondary storage capacity, and number of permissible users.

Your virtual storage capacity is extremely large; however, efficient programming is still important. Performance can be degraded by excessive demands on the available storage at an installation.

When you initiate communication with the system (conversationally or nonconversationally), the system routines essential to your task[1] are loaded into your virtual storage. These routines are a permanent part of your virtual storage, that is, they remain there through your task.

You obtain other system routines by issuing commands and executing programs. These routines are loaded into, and unloaded from, your virtual storage on a demand basis.

An important aspect of TSS/360 virtual storage management is the protection it provides. Another user cannot interfere with your executing programs.

SHARING TIME

Others may be using the system at the same time you are; your terminal is one of many that are connected to the same computer center. The system appears to be serving each of you exclusively because it is repetitively giving each of you an interval during which all the facilities required by

---

[1]A task is the work done between the time you begin conversational or nonconversational communication with the system and the time you end that communication.

your task, including the central processing unit and the supervisor program, are in fact exclusively yours. Unless the system is overloaded, its speed allows it to do your work, as well as that of other users, without apparent intervals.

TERMINAL SESSION ACTIVITY

If you have access to a terminal, you can use TSS/360 conversationally. You control the system step-by-step as you type commands, data, and the source statements for your programs. The system, in turn, responds to your requests, delivering its output at the terminal in the form of typed responses.

Commands are your principal means of communication with the system; they tell the system what you want it to do. They initiate and terminate tasks, compile programs, create, modify, and copy data sets, and obtain bulk output.

Entering Commands

To enter a command, you simply type the characters required and press RETURN. Each command has an operation part specifying what is to be done (as PRINT), and each may have one or more operands that qualifies the operation (as DSNAME=LIST: this qualifies the operation to mean "print my data set named LIST").

If you enter an incorrect command, the system issues a message that informs you of the error. The system also issues messages helpful in assessing the system's activity relative to your task. System messages are issued automatically as the conditions causing them arise.

DATA MANAGEMENT FACILITIES

You can use your terminal as an I/O device, typing data for your programs and having your programs type their results. You can also have data (including programs) stored by the system on its direct access devices, for use at a later time. A third alternative is to use your own private storage devices.

A PL/I Program that uses only terminal input and output can be compiled and executed using only the commands, PL/I statements, and data below. (The program name SIMPLE is used for this example.)

The procedure for readying your terminal for use is described in "Section 4: Communicating with the System." If you do not know how to operate the terminal, read that section or ask someone.

Note: The system translates the lowercase letters into capital letters.

```
logon  smith,passwd,24
pli  name=simple
0000100 simproc:  procedure options(main);
0000200           get data;
0000300           d=a**2+b*c;                    PL/I
0000400           e=sqrt(d);                     source
0000500           put data (d,e);                statements
0000600           end;
0000700_end
simple
:a=2,b=3,c=4                                     input data
D= 1.60000E+01          E= 4.00000E+00;          output data
logoff
```

COMMANDS AND PL/I STATEMENTS

logon smith,passwd,24

The LOGON command identifies you to the system. You must supply all LOGON parameters assigned by the system manager or administrator when you were joined to the system. (As a PL/I programmer, you want to make sure that a 24 appears after the second comma, unless your installation defaults that parameter to 24. See Part III, Example 1 for an explanation.) After entering the LOGON command, you can proceed with your work.

pli name=simple

The system types an underscore and a backspace, to indicate that it expects a command. You type a PLI command, to invoke the PL/I compiler, specifying that the name of your executable program is to be SIMPLE. (This name must be different from the procedure name.)

The "name=" could be omitted, and the command could be typed as:

```
pli simple
```

Since you do not specify a name for your collection of source statements, the system looks for a prestored collection of source statements named SOURCE.SIMPLE. Not finding SOURCE.SIMPLE, the system then assumes that you are going to enter your source statements from the terminal. The system prompts you to enter your source statements by typing line numbers at the terminal.

PL/I source statements

simproc: procedure options(main); -- The label of your main procedure must not be the same as the name assigned to your executable program - SIMPLE.

get data; -- Since you have not told the system anything about the data to be used by your program, it assumes that you will enter the data from the terminal while your program is running conversationally.

```
d = a ** 2 + b * c;
e = sqrt(d);
```

-- These statements perform calculations.

put data (d,e); -- Since you have not told the system where the output data is to be placed, it assumes that you want the data printed at your terminal while your program is running conversationally.

end; -- This statement is the last of your PL/I source statements; it marks the end of SIMPROC.

_end

The END command, which is different from the PL/I END statement, indicates to the system that you have finished entering PL/I statements. The system now compiles your program (that is, converts your PL/I statements to machine-executable instructions) and stores the compiled program in your user library so that it is available for use.

You type an underscore before the END command, to inform the system that a command, not a PL/I statement, is being entered.

simple

This command invokes procedure SIMPROC. Since SIMPROC is invoked from the command mode, it must be invoked by its program (that is, object-module) name, SIMPLE. Take care that you do not invoke a PL/I program by a name other than its object-module name, unless you invoke it from another PL/I procedure. A PL/I program invoked from another PL/I procedure is invoked by its procedure name (not by its module name). When your program is ready to accept your input data, a colon is issued to your terminal; you then enter your input data as follows:

```
a = 2 b = 3 c = 4
```

Your program then performs the calculation and prints the values of the program variables at the terminal as follows:

```
D= 1.60000E+01    E= 4.00000E+00;
```

logoff

The LOGOFF command indicates the end of your communication with the system. The next time you want to communicate with the system, you must log on again.

Note: If you want to compile another program or do any other work, you need not log off until you are finished.

CORRECTING ERRORS WHEN COMPILING

If you see an error in a line that is still being typed, it can be corrected by backspacing to the error and retyping from that point on. If you don't notice the error until after pressing RETURN, you can make the correction by issuing an UPDATE command and then, after the system unlocks the keyboard, typing the line number and the correct statement. The line number and the statement must be separated by a blank.

Example 1:

```
logon (your LOGON parameters)

pli  simple
```

As you are typing the first PL/I statement, you notice an error. You backspace to the error, move the paper up one line to avoid overtyping, and retype from that point on:

```
0000100 simproc!  pr
                :  procedure;
0000200           d=a**2+b*c;
0000300           e=sqrt(d);
```

You notice the omission of OPTIONS (MAIN) in line 100, so you type an UPDATE command, preceding the command by a break character because the system is expecting data.

```
0000400 _update
```

After the system unlocks the keyboard, you make the correction.

```
100 simproc:  procedure options(main);
```

The system again unlocks the keyboard. You type an INSERT command, preceding it with a break character because the system is expecting data, giving the number of the last-entered line as an operand; this causes the system to resume prompting you for the unentered source statements.

```
_insert 300
0000400        put data (d,e);
0000500        end;
0000600 _end
```

You notice that a source statement is missing and that another statement is without a semicolon. To halt compilation and unlock the keyboard, you press the attention key; the system prompts you with an exclamation mark (!). Since you already typed an END command, you must type an EDIT command to regain access to the source statements. You then type an UPDATE command and enter corrections.

```
(you press attention key)
!
edit  source.simple
update
200    d=a**2+b*c;
150    get data;
_end
```

Now all the source statements are correct. (The line numbers differ from the line numbers in the preceding example, but this has no bearing on the operation of the program.) Since you changed some of the source statements, you must start compilation over from the beginning by typing "pli simple".

Example 2:

This example is the same as example 1, except that you discover the errors later.

```
logon (your LOGON parameters)
pli  simple
0000100 simproc!  procedure;
0000200           d=a**2+b*c
0000300           e=sqrt(d);
0000400           put data(d,e);
```

```
0000500           end;
0000600 _end
```

At this point the PL/I compiler informs you of the syntactical errors. You correct the errors:

```
edit  source.simple
update
100 simproc:  procedure options(main);
150           get data;
200           d=a**2+b*c;
_end
```

Now you can recompile SIMPLE, and it will be ready for execution.

Note: There may be times when you want to stop entering source statements and not compile them; for example, you may decide to start over from the beginning or you may have misspelled the name in the PLI command. Simply press the attention key. Do not type _END; this causes compilation. Even if you typed _END immediately after the prompt for line 100, the compiler would attempt to compile the empty data set.

DATA ASSOCIATED WITH COMPILATION

The system stores the collection of source statements under the name SOURCE.name, where "name" is the object-module name you gave in the PLI command. You can specify a different name than SOURCE.name, by using the PLI command's SOURCEDS operand as explained under "Invoking the Compiler," in Section 5. The object module itself, which consists of the executable machine instructions, is stored as USERLIB(name), unless you specify otherwise. (See Section 6.) The compiler also produces a listing of information about the source statements and object module; this listing is named LIST.name. You can erase any of this data at any time by typing ERASE SOURCE.name, ERASE USERLIB(name), or ERASE LIST.name.

EXECUTING A PREVIOUSLY COMPILED PROGRAM

To execute a program that you have already compiled, you do not have to reenter your source statements; you can call for execution of the program from the library in which it is stored. The following terminal session shows execution of program SIMPLE.

```
logon (your LOGON parameters)
simple
:a = 1, b = 5, c = 4      input data to your
                          program

D= 2.10000E+01            E= 4.58257E+00;
                          output data from
                          your program
logoff
```

FURTHER INFORMATION

Section 3 describes data manipulation techniques. Part II describes more sophisticated methods of using the PL/I compiler and available data processing facilities. Examples of how to use PL/I in TSS/360 are in Part III.

Note: You can enter commands, PL/I keywords, and names in either lowercase or capital letters; they will hereinafter be shown in capital letters, in order to distinguish them from the explanatory text.

SECTION 3: BASIC DATA MANIPULATION

This section explains how to create and access simple data sets typed at your terminal (using STREAM files) and data sets stored on magnetic tape or on a direct access device (using STREAM or RECORD files). It is intended to introduce the subject of data management, and to meet the needs of users who do not require the full I/O facilities of PL/I and TSS/360. Sections 7, 8, 9, and 10 give a full explanation of the relationship between the data management facilities provided by PL/I and those provided by the system.

A data set is any collection of data that can be created or accessed by a program. The data can be entered from, or printed at, your terminal. It can also be punched onto cards, or recorded on magnetic tape or on a direct access device.

A PL/I program can access data sets using either stream-oriented transmission or record-oriented transmission. When accessed by stream-oriented transmission, a data set can be thought of as a continuous stream of characters that are converted from character form to internal form on input and from internal form to character form on output. It can be processed without regard to its actual origin.

A data set accessed by record-oriented transmission is considered to be a collection of discrete data items (that is, records). No data conversion occurs during record transmission; on input the data is transmitted exactly as it is recorded in the data set, and on output it is transmitted exactly as it is recorded internally. To be accessed by record-oriented transmission, a data set must have either CONSECUTIVE or INDEXED organization. The records in INDEXED data sets are arranged according to 'keys' that you supply when you create the data sets. CONSECUTIVE data sets do not use keys; when you create such a data set, records are recorded consecutively in the order in which you present them. You can read the records from a CONSECUTIVE data set only in the order in which they were presented, except that in the case of a data set on magnetic tape, you can read them either in the order in which they were presented or in the reverse order.

Two types of information are required to create or retrieve a data set.

1. Appropriate input and output (I/O) statements in your PL/I program.
2. Information required by TSS/360 describing the data set and how it is to be handled. You don't have to specify this information explicitly if you are using the terminal as your I/O device

The I/O statements that you may need in your PL/I program are described by PL/I Language Reference Manual. Essentially, you must declare a file (explicitly or contextually) and open it (explicitly or implicitly) before you can begin to transmit data. A file is the means provided in PL/I for accessing a data set, and is related to a particular data set only while the file is open; when you close the file, the data set is no longer available to your program. This arrangement allows you to use the same file to access different data sets at different times, and to use different files to access the same data set.

TERMINAL I/O

TSS/360 allows you to enter data for your PL/I program and receive output while the program is executing. PL/I files for terminal I/O must be STREAM files. You do not have to specify that the files are STREAM; this is implied by the GET and PUT statements used to read and write data.

If your program includes a GET statement without the FILE option, the compiler assumes 'SYSIN'. If your program includes a PUT statement without the FILE option, the compiler assumes 'SYSPRINT'. In conversational mode, SYSIN and SYSPRINT are directed to your terminal. You do not have to supply a DDEF command when doing terminal I/O; when you omit a DDEF command for a STREAM file, the system assumes that you are using the terminal and supplies all required data management parameters.

The simplest way to enter data for a PL/I program is with the statement:

```
GET DATA;
```

The simplest way to transmit data to an external medium is with the statement:

```
PUT DATA (data-list);
```

This is all that is required to perform terminal I/O.

When your program is ready to accept input data, a colon (:) is typed at the terminal and the keyboard is unlocked.

EXAMPLE OF TERMINAL I/O: This example is a modification of the program in Section 2; it allows you to read and write more than one line during execution. The statement following the PUT DATA statement returns control to the GET DATA statement.

```
LOGON   (your LOGON parameters)

PLI     NAME=SIMPLE
```

The system types a line number (not shown) before each line you enter.

```
        TERMIO:   PROCEDURE OPTIONS (MAIN);

                  ON ENDFILE (SYSIN) GO TO
                   STOP;

        READIN:   GET DATA;

                  D = A**2 + B*C;

                  E = SQRT(D);

                  PUT DATA (D,E);

                  GO TO READIN;

        STOP:     END;

        _END

SIMPLE
```

When the system prompts you with a colon, enter values for A, B, and C, ending the list with a semicolon.

```
A = 2,B = 3,C = 4
```

The values of the variables in your program are then printed at the terminal.

```
D= 1.60000E+01   E= 4.00000E+00;
```

You can now enter new values for A, B, and C to find the new value of D.

```
A = 1,B = 17,C = 0
```

The system then prints:

```
D= 1.00000E+00   E= 1.00000E+00;
```

You do not want to calculate further values of D, so when the system prompts you to enter data, press RETURN. The null line entered indicates the end of your input data. Your program then terminates, and you can log off.

```
LOGOFF
```

DATA SETS ON SYSTEM STORAGE

It is not always convenient to have data records entered from and printed at a terminal. If a data set is very large, the time required to print it at a terminal might be quite long. If a data set is modified frequently, it is more convenient to keep a copy of the data set stored within the system and enter changes as necessary. Thus, only changes to the data set need be entered, and unchanged portions of the data set do not have to be reentered each time the data set is processed.

DATA SET NAMES

To retain a data set in system storage, you must assign a name to it. When you subsequently go to retrieve the data set, you can inform the system of the name assigned, and the system will locate the data set for you.

The name of a data set can be qualified to distinguish it from other data sets or to relate it to other data sets. For example, if payroll records for two departments, D561 and D58, were maintained and each had the simple name PAYROLL, the department name could be prefixed to each simple name to produce two qualified data set names -- D561.PAYROLL and D58.PAYROLL. Note that a period must separate the components of a name.

Figure 1 shows the names of data sets belonging to an imaginary user. The only unqualified name is RESEARCH; all other names, such as RECORDS.INVENTRY and RECORDS.PERSONEL.DEPT561, are qualified.

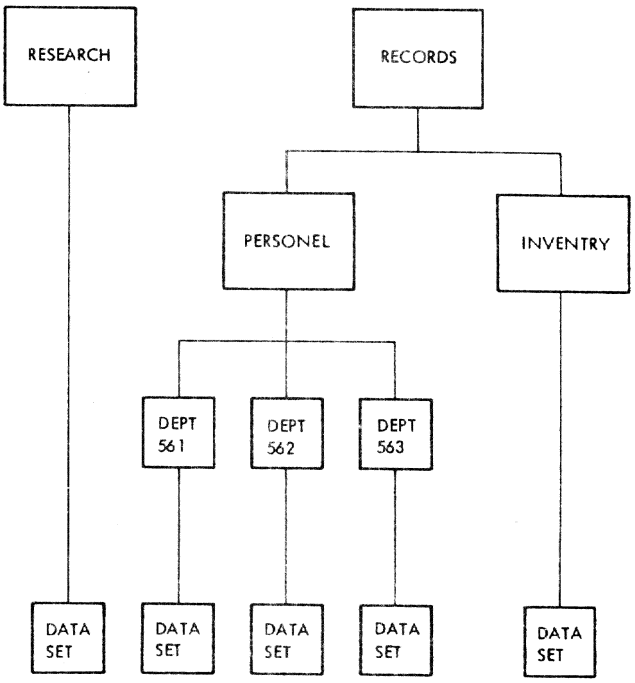


Figure 1. Qualified and Unqualified Names

A fully qualified data set name identifies an individual data set and includes all components of that data set's name. RECORDS.INVENTRY in Figure 1 is a fully qualified name. Fully qualified names are used in the DDEF command to identify the input and output data sets of a problem program. They are also used in connection with a variety of commands, such as those that manipulate data sets as an entity (for example, CDS and PRINT), and those that affect system functions relative to the data set (for example, CATALOG, DELETE, ERASE).

A partially qualified data set name identifies a group of data sets, and omits one or more of the rightmost components of a data set name. The group of data sets referred to includes all that have qualifiers identical to those present in the partially qualified name. In Figure 1, all records can be referred to by the partially qualified name RECORDS; records for all departments can be referred to as RECORDS.PERSONEL (no period after PERSONEL). Partially qualified names are used in several commands when it is convenient to refer to the specified data sets as a group; for example, in erasing the group, in removing it from the data set name structure, or in specifying that it can be shared by other users.

These rules must be observed in naming data sets:

1. Each component, or simple name, can consist of from one to eight alphameric characters (this is why "personel", in Figure 1, has only one N); the first character must always be alphabetic.

2. A period must be used to separate components.

3. The maximum number of characters (including periods) in the data set name is 44. For data sets used exclusively within TSS/360, you are limited to 35 characters, because the system automatically prefixes each name with your eight-character user identification[1] followed by a period. For data sets to be interchanged with the IBM System/360 Operating System, you can employ 44-character data set names. These data sets, however, cannot be referred to by name alone unless they are renamed; for data sets with 44-character names, the expanded DDEF command must be used (see Appendix D).

4. The maximum number of single-character qualification levels to a single-character basic name is 18, for data sets used in TSS/360. Normally, fewer qualification levels will be used.

5. The fully qualified names in your data set name structure must be unique; no fully qualified data set name can be used as a partial qualifier in another data set name.

[1]The user identification, that is, userid, is the first operand of the LOGON command. See Example 1 in "Part III: Examples." If less than eight characters, it is padded with blanks

THE DDEF COMMAND

The DDEF command is used to define the data sets used during execution of a program. You can also use it to define the data sets used by certain commands, and to define job libraries.

The DDEF command names a data set and supplies the system with information with which to retrieve it. The DDEF command has four basic operands:

- Data definition name (DDNAME) is the name of the DDEF command describing the data set. This information is required by the system to relate the data set to the file named in your I/O statements or to the TITLE option of an OPEN statement.

- Data set organization (DSORG) specifies the organization of the data set so that the appropriate system data management routines are made available to your program.

- Data set name (DSNAME) is the name of the data set being described. This information is required to locate the data set among those stored in the system.

- Disposition (DISP) specifies whether the data set is being created (NEW) or already exists (OLD).

Reserved Names

DDNAME=PCSOUT and all DDNAMEs beginning with the characters SYS are reserved for use by the system. DSNAME=USERLIB is reserved by the system; in addition, you should avoid specifying any DSNAME that begins with the characters SYS, since several data sets that the system uses for you have names beginning with SYS.

STREAM-ORIENTED TRANSMISSION

A data set that you process using stream-oriented transmission could have

been created using either stream- or record-oriented transmission. (See example under "Creating a Simple Consecutive Data Set", in this section.) If it was created using record-oriented transmission, it must have CONSECUTIVE organization, and all the data in it must be in character form. You can open the associated file for input and read the records that the data set contains, or you can open the file for output and extend the data set by adding records at the end.

EXAMPLE: To gain access to the data set in this example, you must supply a DDEF command so that the system can locate the data set. You must specify:

- The name of this DDEF command - WORK. (This name is the file name used within the PL/I program in such statements as OPEN, CLOSE and DECLARE.)
- The organization of the data set - VS. (VS stands for virtual sequential, a TSS/360 organization for sequential data sets on direct access devices.)
- The name under which the data set was previously stored-PEOPLE.
- The disposition of the data set - OLD (since the data set already exists).

The PL/I program that you write to update data set PEOPLE is called UPDAT. After you compile UPDAT, you enter update records for data set PEOPLE from your terminal.

```
LOGON              (your LOGON parameters)

DDEF     DSNAME=PEOPLE, DSORG=VS, DDNAME=WORK, DISP=OLD

PLI      NAME=UPDAT

The text editor types a line number (not
shown) before each line that you enter.

         UPDAT:   PROCEDURE OPTIONS (MAIN);

                  DCL WORK FILE STREAM OUTPUT PRINT,

                      1  REC,
                      2  FREC,
                       3  NAME CHAR (20),
                       3  NUM CHAR (1),
                       3  PAD CHAR (24),
                      2  VREC CHAR (35),

                  IN CHAR (80) DEF REC;

                  ON ENDFILE (SYSIN) GO TO FINISH;
                  OPEN FILE (WORK) LINESIZE (130);

         MORE:    GET FILE (SYSIN) EDIT (IN) ( A (80));

                  PUT FILE (WORK)SKIP EDIT (IN) (A(45+7*NUM));

                  GO TO MORE;

         FINISH:  CLOSE FILE (WORK);
                  END UPDATE;

_END

         You now call for execution of UPDAT.

UPDAT
```

As the system prompts you to enter your input data, you enter it; NUM must always be entered as the twenty-first character. A null line indicates the end of data.

```
:R.C.ANDERSON   0  202848  DOCTOR
:R.R.BENNETT    2  771239  PLUMBER    VICTOR  HAZEL
:R.E.COLE       4  698635  COOK       ELLEN   VICTOR  JOAN    ANN
:L.R.COOPER     4  418915  LAWYER     ROGER   THERESA LAURA   KATHY
:A.J.CORNELL    3  237837  BARBER     DONALD  NANCY   JOSEPH
:B.P.FERRIS     4  158636  CARPENTER  GERALD  ANNA    MARY    FRED
:(null line)
```

The data set is now updated in storage. The next time that you use the data set, the records that you have just entered will be a part of it.

RECORD-ORIENTED TRANSMISSION

In record-oriented transmission, data is transmitted to and from auxiliary storage exactly as it appears in the program variables; no data conversion takes place. A record in a data set corresponds to a variable in the program.

You can use record-oriented transmission to process data sets with CONSECUTIVE or INDEXED organization. This section describes processing of a CONSECUTIVE data set; for a description of processing INDEXED data sets and further description of processing CONSECUTIVE data sets, see Section 10.

CREATING A SIMPLE CONSECUTIVE DATA SET

The program in the next example creates a data set named ROOTS. The PL/I I/O statements (DECLARE, OPEN, and CLOSE) refer to the file DISK; therefore, the name of the DDEF command that defines the data set must be DISK (DDNAME operand). You specify that the organization of the data set is to be virtual sequential (the same as the STREAM file in the previous example). The name of the data set is ROOTS (DSNAME operand), and its disposition is NEW (you are creating the data set).

Since this data set is created, you must give the system some information about the format of the records in the data set. You can give record-format information in either your PL/I program (ENVIRONMENT attribute or LINESIZE option) or a DDEF command. This discussion refers only to the DDEF command and does not apply if you decide to give the information in your program. Refer to PL/I Language Reference Manual for a description of the ENVIRONMENT attribute and the LINESIZE option.

The records in a data set must have one of three formats: F (fixed length), V (variable length), or U (undefined length). If you do not specify a record format, format V is assumed. Since you specified that the organization of the data set is virtual sequential, you do not have to consider record blocking; the system handles this for you. If you had specified physical sequential organization, you could control the blocking of records. Record blocking is discussed in Section 10, under "CONSECUTIVE Files."

If you are using a PRINT file to produce printed output, you do not have to specify record size in the DDEF command or PL/I program; in the absence of other information, the compiler supplies default line size of 120 characters.

To give record-format information in a DDEF command, use the DCB (data control block) operand. The DCB operand passes information to the system for inclusion in the data control block, a collection of information maintained by TSS/360 data management routines for each data set in a task. The data control block contains a description of the data set and how it will be used. Suboperands of the DCB operand allow you to specify such information as record format (RECFM suboperand) and logical record length (LRECL suboperand). If the DCB operand includes more than one suboperand, enclose the list in parentheses. For example:

```
DCB = (RECFM = F, LRECL = 40)
```

EXAMPLE: Record-oriented transmission is used to create the data set ROOTS. Before each record is written in ROOTS, that record is entered from SYSIN (your terminal) and processed using stream-oriented transmission. Note how stream- and record-oriented transmission can be easily combined for a single data processing application.

```
DDEF  DDNAME=DISK, DSORG=VS, DSNAME=ROOTS,
      DISP=NEW, DCB=(RECFM=F,LRECL=40)

PLI   NAME = CREATE
```

The text editor types a line number (not shown) before each line that you enter.

```
   CREAT:  PROCEDURE OPTIONS (MAIN);

               DCL DISK FILE RECORD OUTPUT
                SEQUENTIAL,
                1 RECORD, 2(A, B, C, X1,X2)
                FLOAT DEC(6) COMPLEX;
```

The length of RECORD is 40 bytes; RECORD contains five items, each declared as FLOAT DECIMAL (6) COMPLEX; since the declared precision ≤ 6, short floating-point (fullword length) is used; the COMPLEX attribute doubles this to a length of one double word per item.

Note: If you specify the ATR or the XREF listing option of the PL/I compiler options, the lengths of all structures are shown on the listing, in a table called the aggregate length table. See Appendix G.

```
               ON ENDFILE (SYSIN) GO TO
                FINISH;

               OPEN FILE (DISK);

   NEXT:  GET FILE (SYSIN) LIST (A, B, C);
          X1=(-B+SQRT(B**2-4*A*C))/(2*A);
          X2=(-B-SQRT(B**2-4*A*C))/(2*A);
          WRITE FILE (DISK) FROM (RECORD);
          GO TO NEXT;

   FINISH:  CLOSE FILE (DISK);
            END CREAT;

   _END
CREATE
 :5   12   4
 :4  -10   4
 :5   16   2
 :4  -12  10
 :5   12   9
:29  -20   4
:  (null line)
```

Data set ROOTS now exists on system storage. The records in ROOTS consist of the values for A, B, C, X1, and X2.

RETRIEVING A CONSECUTIVE DATA SET

EXAMPLE: At a later time, you want to read data set ROOTS. The data set name is still ROOTS (DSNAME operand), and its organization is still virtual sequential (DSORG operand). The PL/I I/O statements in the program that reads ROOTS refer to file RESULTS, so you must specify DDNAME=RESULTS. Since ROOTS already exists, you specify its disposition as OLD and omit the DCB operand; the system can fill in the DCB from information in control blocks associated with the data set.

```
DDEF DDNAME=RESULTS, DSORG=VS, DSNAME=ROOTS, DISP=OLD

PLI  NAME=ACCESS

     ACCES:   PROCEDURE OPTIONS (MAIN);

              DCL RESULTS FILE RECORD INPUT SEQUENTIAL,

                  1 RECORD,  2(A, B, C, X1, X2)
                   FLOAT DEC(6) COMPLEX;

              ON ENDFILE (RESULTS) GO TO FINISH;

              PUT FILE (SYSOUT) EDIT
               ('A', 'B', 'C', 'X1', 'X2')

                  (X(7), 3(A,X(23)), A, X(22), A);

              OPEN FILE (RESULTS);

     NEXT:    READ FILE (RESULTS) INTO (RECORD);

              PUT FILE (SYSOUT) SKIP EDIT (RECORD)
               (C(F(12,2)));

              GO TO NEXT;

     FINISH:  CLOSE FILE (RESULTS);

              END ACCES;

     _END

     ACCESS
```

The data set ROOTS is now printed at your terminal.

PART II: USING ALL THE FACILITIES OF THE PL/I COMPILER

In TSS/360 you can run a program conversationally; you and the system exchange information during the entering and execution of your program.

You can also run a program nonconversationally; for instance, when a program is checked out and you know it will run satisfactorily, or when you do not want to stay at the terminal.

You can run in mixed mode: that is, start a program conversationally and switch to nonconversational processing. Once a program is running nonconversationally, you cannot switch back to conversational processing.

CONVERSATIONAL USE OF THE SYSTEM

In conversational processing, you communicate with the system by means of a terminal. The terminal is a typewriter-like device. One type of terminal, the IBM 2741 Communications Terminal, is an IBM SELECTRIC® typewriter specially equipped for terminal use; another type, the IBM 1050 Data Communications System, can include both a typewriter and a card reader. With the 1050 you can enter input into the system via the keyboard or the card reader. All types of terminals can be located either at the computer installation or at a remote location. In any event, all terminal operation is much the same: you enter a command directing the system to do certain work; the system responds; you enter another command, etc. You don't have to be an expert typist; correcting typing errors is simple, as shown in Part III, Example 1.

You will find that you do not require extensive computer training to use TSS/360. You must know three things:

- How to set up your terminal for operation. This is a matter of setting a few switches. The use of each terminal is discussed in this section; see the description for your terminal, or ask someone to show you how to set it up.
- The PL/I language, the language in which you express your problem-solving procedure. This language is used for illustration throughout this publication; it is explained in detail in PL/I Language Reference Manual.
- The TSS/360 commands you will use to converse with the system. Typical uses of many commands are shown in the examples section of this manual. The format of every command is shown in Appendix F. This section includes summaries of how to type commands and how to use commands to control nonconversational tasks. Part III, Example 17 shows how to execute a command from within a PL/I program. Should you need more information than is in this book, consult Command System User's Guide, which describes the commands in detail.

In conversational mode, you engage in dialog with the system. The system responds to your requests, confirms actions, and informs you of any errors. Complete details on command-response messages are presented in the System Messages publication. Messages produced by the PL/I compiler are explained in Appendix H of this publication. Some messages are issued during compilation by system routines other than the PL/I compiler; these messages are documented in System Messages.

The work done between logging on and logging off is called a task. You may run one or many programs as part of a single task. The work you do on a task at a terminal is called a session. Since a task may begin conversationally but end nonconversationally, task is not necessarily synonymous with session.

CONVERSATIONAL TASK INITIATION

The way in which you initiate a conversational task varies slightly with the type of terminal you are using. The available types are:

- IBM 1050 Data Communications System
- IBM 2741 Communications Terminal
- Teletype[1] Model 33 or 35 KSR

The terminal operation procedures are explained below.

After you initiate the LOGON procedure and the system is ready to receive input, enter the values assigned to you when you were joined to the system. (Note: To compile or run PL/I programs, you must log on with 24-bit addressing. This must be specified in the third operand of the LOGON command, unless your installation's default value for that operand is 24. See Example

---

[1]A trademark of the Teletype Corporation

1 in "Part III: Examples.") The system then completes initiation of your conversational task. If you cannot log on, notify your system manager or administrator.

IBM 1050 Data Communications System

The IBM 1050 Data Communications System as used with TSS/360 includes an IBM 1051 Control Unit, a 1052 Printer-Keyboard, and, optionally, a telephone-like modulator-demodulator, or MODEM. The MODEM is used to dial up TSS/360.

INITIATION PROCEDURE: To ready the IBM 1050 for use with TSS/360, proceed as follows:

1. Set the panel switches on the IBM 1052 Printer-Keyboard as directed in Table 1. If the 1052 has additional switches, set them to the OFF or HOME position. Do not change the switch positions while using the terminal.

2. Turn on the main-line switch. The POWER light should come on. If the DATA CHECK light is on, turn it off by pressing the DATA CHECK key.

3a. If the terminal is directly connected to the computer, initiate the LOGON procedure by pressing the ATTENTION/RESET LINE key.

3b. If the terminal has a telephone-like MODEM, press the MODEM's TALK button, dial the TSS/360 number, and when a continuous high-pitched tone is heard, press DATA. The terminal is now connected to the time-sharing system. The receiver of the MODEM can be replaced in its cradle.

KEYBOARD OPERATION: The numeric and special-character keys, the space bar and the SHIFT, LOCK, and TAB keys operate like their counterparts on standard typewriters.

Table 1. 1052 Switch Settings

Switch	Setting	Toggle Position
SYSTEM	ATTEND	up
PRINTER 1	SEND REC	middle
KEYBOARD	SEND	up
READER 1	ON	up
STOP CODE	OFF	down
SYSTEM	PROGRAM	up
SYSTEM		up
TEST	OFF	down
SINGLE CY	OFF	middle
RDR STOP	OFF	middle

Note: Set all other panel switches to OFF or HOME position.

PROCEED LIGHT: When the green PROCEED light is on, the keyboard is unlocked and data or commands can be entered. As soon as a line has been entered, the keyboard is locked; the PROCEED light turns off shortly afterwards. All keys except the ATTENTION/RESET LINE key are locked out while the PROCEED light is out.

ATTENTION KEY: The ATTENTION/RESET LINE key in the lower left-hand corner of the keyboard, hereinafter referred to as the ATTENTION key, cannot be locked out. It generates an attention interruption. (The ATTENTION key can also be used, as explained above, in initiating the LOGON procedure from a directly connected terminal.)

RETURN KEY: Pressing RETURN causes a line feed and print-head return at the terminal printer and transmits an end-of-block character to system. After RETURN is pressed, the keyboard is locked out (except for the ATTENTION key) and control passes to the system.

CONTINUATION LINES: When the hyphen is entered as the last character in a line, the system recognizes the next line as a continuation. The hyphen is not entered as part of the line.

CANCELING LINES: When a pound sign (#) is entered as the last character before the RETURN key is pressed, the entire line is canceled. The system will then expect the corrected line to be entered without additional prompting. The pound sign is defined as the line-kill character.

A line can also be canceled with the ALTN CODING key, at the upper left-hand side of the keyboard. To do this, hold down ALTN CODING and press the zero key.

CORRECTING LINES: A line that you have started to enter incorrectly can be corrected by backspacing to the first incorrect character with the BACKSPACE key and reentering the line from that point on.

DATA CHECK AND RESEND: The DATA CHECK light may come on after the terminal is first turned on; this light can be turned off with the DATA CHECK key. The RESEND light will come on briefly after the RETURN key is pressed; it should turn off when the line has been accepted by the system. If the DATA CHECK and RESEND lights are on together, an error is indicated. While the RESEND light is on, the system does not accept input from the terminal keyboard. Press the DATA CHECK and RESEND keys to turn off the lights and reenter the line.

IBM 2741 Communications Terminal

The IBM 2741 consists of an IBM SELECTRIC® typewriter mounted on a stand that includes the electronic controls needed for communication with TSS/360. If the terminal is directly connected to the system, merely turning on the terminal results in connection with the system. If not, it can be connected to the system through a modulator-demodulator, or MODEM, that resembles a telephone.

INITIATION PROCEDURE: To ready the 2741 for use with TSS/360, proceed as follows:

1. Check that the terminal mode switch on the left side of the stand is set to COM.

2. Press on the ON side of the power switch.

3a. If the terminal is directly connected to the computer, initiate the LOGON procedure by pressing the ATTN key at the upper right-hand corner of the keyboard.

3b. If the terminal has a telephone-like MODEM, press the TALK button, lift the receiver, dial the TSS/360 number, and, when you hear a continuous high-pitched tone, press the DATA button. The terminal is now connected to the system. The receiver of the MODEM can now be placed in its cradle.

KEYBOARD OPERATION: The terminal keyboard works like an IBM SELECTRIC® typewriter except for the ATTN key, which is used to generate attention interruptions. (It can also be used in initiating the LOGON procedure from directly connected terminals, as explained above.) The system unlocks the keyboard when it is expecting input; at other times, the keyboard is locked. The ATTN key is the only key that cannot be locked out.

Note that unless you issue a KA command or an equivalent K command, the system recognizes no distinction between capital and lowercase letters; they are all interpreted as capital letters. This saves you from having to use the SHIFT key in entering commands, which consist only of capital letters.

RETURN KEY: Pressing the RETURN key causes a line feed and carrier return at the terminal and transmits an end-of-transmission character to the system. RETURN must be pressed to end every line of input from the keyboard. After RETURN has been pressed, the keyboard is locked out (except for the ATTN key) and control passes to the system.

CONTINUATION LINES: When the hyphen is entered as the last character in a line, the system recognizes the next line as a continuation. The hyphen is not entered as part of the line.

CANCELING LINES: When a pound sign (#) is entered as the last character before RETURN is pressed, the entire line is canceled. The system then expects the corrected line to be entered without additional prompting. The pound sign is defined as the line-kill character.

CORRECTING LINES: A line that you have started to enter incorrectly can be corrected before RETURN is pressed by backspacing to the first incorrect character with the BACKSPACE key and reentering the line from that point on.

Teletype Model 33/35 KSR

The Teletype[1] Model 33 or 35 KSR (Keyboard Send-Receive) consists of a printer, a four-row keyboard, and a control unit, all mounted in a special cabinet.

INITIATION PROCEDURE: To ready the teletypewriter for use with TSS/360, proceed as follows:

1. Press the ORIG button. The lamp should light under the button, and the teletypewriter will be on.

2. Dial the TSS/360 number with the telephone dial. A continuous tone is heard momentarily as the connection is made. The LOGON procedure then begins.

KEYBOARD OPERATION: The alphameric and special-character keys, the space bar, and the SHIFT key all work like their counterparts on conventional typewriters (except that the SHIFT key does not lock in the down position). Only capital letters are provided; lowercase letters are not. The BREAK key is used to generate an attention interruption. After using the BREAK key, you must press the BRK-RLS key above the telephone dial to unlock the keyboard.

Do not use the keyboard when the system is not expecting input. The keyboard is not locked when the system is not expecting input, and pressing a key at such a time will cause the equivalent of an attention interruption.

Since the teletypewriter lacks keys for the backspace, underscore, and logical-NOT sign, you must use substitutes. The usual TSS/360 prompt of underscore and backspace, for instance, is represented on the tele-

---

[1]A trademark of the Teletype Corporation

typewriter as a right bracket and a left arrow:]⇐

Since the right bracket is the equivalent of the underscore, it is used as the command-prefix character for the text editor. The right bracket is obtained by holding down SHIFT and pressing the "M" key.

In addition, the backwards slash (\) is the teletypewriter equivalent for the logical-NOT sign (¬) . The backwards slash is obtained by holding down SHIFT and pressing the "L" key. The left bracket is obtained by holding down SHIFT and pressing the "K" key.

END-OF-LINE SEQUENCE: To signal the end of an input line, press the RETURN key, the LINE FEED key, and then hold down the CTRL (control) key while pressing the key marked C OFF. This end-of-line sequence must be executed to signal the end of each input line. After the end-of-line signal, control passes to the system. Do not use the keyboard again until prompted for further input, except to generate an attention interruption.

CONTINUATION LINES: When the hyphen is entered as the last character before the end-of-line sequence, the system recognizes the next line as a continuation. The hyphen is not entered as part of the line.

CANCELING LINES: When a pound sign (#) is entered as the last character before the end-of-line sequence, the entire line is canceled. The system then expects the corrected line to be entered without further prompting. The pound sign is defined as the line-kill character.

CORRECTING LINES: To correct a line that you have started to enter incorrectly, enter the left arrow (which is obtained by holding down SHIFT and pressing the "O" key) once for each character entered since the first erroneous character, and then reenter the line from that point on. In other words, use the left arrow as if it were the BACKSPACE key on a typewriter. For instance, if you have typed ERESE and want to change it to ERASE, your correction would look like this:

ERESE⇐⇐⇐ASE

The left arrows are not entered as part of the line; TSS/360 treats the teletypewriter left arrow as the equivalent of the backspace, as mentioned above.

SYSIN and SYSOUT

Now that you have initiated a task, you can converse with the system as if you alone were using it. You have unique communication paths in the system permitting it to read from and write to your terminal, independently of all other tasks. You can thus define work for the system by issuing commands, and the required programs and data will be loaded into main storage and processed, as you specify, regardless of the work other users may be simultaneously specifying.

Your task's input to the system contains the sequence of commands you issue; this sequence is called SYSIN. Your system input stream can also include data to be prestored in the system, or actual input records to an executing program. When you are in the conversational mode, your terminal is your task's SYSIN device. Your task's system output stream, called SYSOUT, is directed to the terminal. It consists basically of system messages; it may also contain output from your object programs if you so choose. Because the terminal is thus a combined SYSIN/SYSOUT device, it writes a mixture of the two system streams.

You and every other user have your own unique SYSIN/SYSOUT. You also have the following:

- Your own virtual storage space.
- A scheduled time interval in which your task is executed.
- Your own catalog.

CONVERSATIONAL TASK EXECUTION

After the initialization process is complete, the system asks you to enter your next command statement; that is, a command or series of commands and engages in a conversation with you. Your part of this dialog consists of any command and source language statements that you enter during execution of your task, and your replies to the messages issued by the system. The system's part of this dialog consists of responses to command statements, requests for next command statements, and messages. The system issues general information messages and messages informing you of error conditions.

INFORMATION MESSAGES: These messages prompt you to supply certain information when a mandatory operand has been omitted, or inform you of the actions the system has taken in executing a command statement.

DIAGNOSTIC MESSAGES: These messages warn you of errors made in entering a command name or operands; some messages request you to correct errors.

REQUEST FOR NEXT COMMAND STATEMENT: The system informs you that it is ready to accept the next command statement by printing an underscore character(_) in the first character position of a new line and then backspacing one space. (The same indication is given when you are entering command statements through the terminal card reader.)

ENTERING COMMAND STATEMENTS: Command statements can be entered into the system from the terminal keyboard, the terminal card reader (if any), the system card reader, or a magnetic input device in which the information is stored in card-image format. Command statements can be entered in either upper- or lowercase form, unless you specify otherwise by issuing a KA or CA command.

If a command statement contains more than one command, all the commands in it must be separated by semicolons.

The end of a command statement entered from the terminal keyboard is indicated by pressing RETURN. If a command statement requires more than one line, one hyphen must be typed at the end of the line before RETURN is pressed; the hyphen signals that the statement is not complete, and will be continued on the next line.

Note: The LOGON command must begin and end on the same line.

Command statements that are entered through the terminal card reader can utilize free-form format (that is, input is not restricted to particular card fields). The hyphen, following the command operands, is used to signify that the following line is a continuation line. For statements longer than 80 characters, with the terminal EOB switch on, the continuation character may appear in any available column. If the EOB switch is off, the continuation character is not needed unless the statement exceeds 260 characters.

Note: Nonconversational input through a computer center's high-speed card reader does not require the 11-5-9 punch to signify new line; its inclusion has no effect. An EOB is automatically inserted by the card reader at the end of every card. A continuation character must appear (in any column) for command statements that require more than one card.

Caution: In most cases, tab characters are treated as spaces and are valid characters in the command system. However, because of physical limitations in terminal devices, displaying tabs of more than 65 consecutive spaces at the terminal might cause the next character to be printed in the wrong place.

CONVERSATIONAL TASK OUTPUT

The messages produced by the system during execution of conversational tasks and the responses to command statement execution are printed at your terminal. The results of processing during task execution can be held in data sets within the system. When you want to examine these data sets, you can have them printed at your terminal or have them printed or punched in nonconversational mode. You can also use the dynamic I/O facilities of PL/I to obtain these results directly from execution of your program.

CONVERSATIONAL TASK TERMINATION

When you want to terminate your conversational task, issue a LOGOFF command. The system then updates its internal accounting tables to reflect your use of the system during the task. After issuing LOGOFF, turn the power switch to "off" if your terminal is an IBM 1050 Data Communications System or an IBM 2741 Communications Terminal, or press the CLEAR button on the control unit if your terminal is a Teletype[1] Model 33 or 35 KSR.

If you later want to communicate with the system again conversationally, you must again log on as described under "Conversational Task Initiation."

NONCONVERSATIONAL USE OF THE SYSTEM

While the system is operating conversationally, for many simultaneous users, it can also operate nonconversationally, with batch-type processing jobs, in the background. With minor exceptions, the commands available in conversational tasks, including commands for data manipulation, program compilation, and program execution, are also available in nonconversational tasks. However, in a nonconversational task, there is no communication between you and the system. You might want to execute a task nonconversationally if it is checked out and you know it will run satisfactorily, or if you cannot stay at the terminal to converse with the system.

NONCONVERSATIONAL TASK INITIATION

Figure 2 illustrates the various ways in which you can use the system for nonconversational processing.

The BACK command is used to continue a conversational task nonconversationally.

---

[1]A trademark of the Teletype Corporation

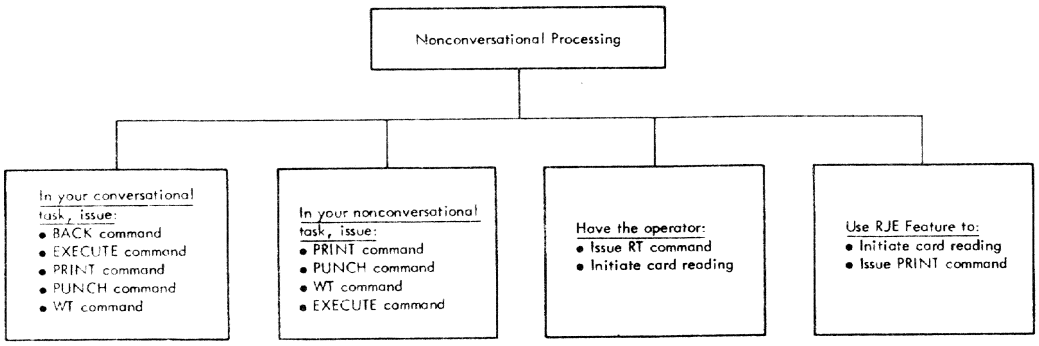


Figure 2. Nonconversational Task Initiation

See "Mixed Mode Use of the System," later in this section.

You can issue the EXECUTE command in a conversational or nonconversational task to initiate a nonconversational task. The EXECUTE command names a cataloged command procedure that is to be executed. The command procedure functions as the SYSIN data set for the nonconversational task. It must begin with a LOGON command; end with a LOGOFF command; and, you must prestore it in the system so that it can be retrieved merely by its name. If private devices are required in the task, a SECURE command must immediately follow the LOGON command.

You can issue PRINT, PUNCH, and WT commands in either a conversational or nonconversational task. These commands are, in effect, one-command-procedures. They initiate nonconversational tasks that transfer data from a direct access device to a printer, card punch, or tape unit.

You can also have the operator initiate nonconversational tasks for you. You supply him with a card deck or magnetic tape; the contents of the deck or tape depend on what you want done:

- If you want to enter data into the system for later use, that is, prestore it, you prepare a card deck (or magnetic tape) with a command procedure of the following form:

```
[data descriptor card]
Data cards              } card images
[%ENDDS card]
```

If you do this, the task set up by the operator transfers data from the input medium to a direct access device and catalogs it so that it is later available to you by its name. For the format of the data descriptor card, see Command System User's Guide.

- If you want to enter a command procedure, you prepare a card deck as follows:

```
[LOGON card]
Other commands & data cards
[LOGOFF card]
```

If you do this, the task that is set up by the operator executes the commands in the command procedure you have defined.

You can use the remote job entry (RJE) feature, if you have access to an RJE device. (See IBM System/360 Time Sharing System: Remote Job Entry.)

In all of the ways of initiating a nonconversational task, the system action is basically the same:

1. The request to set up the nonconversational task is enqueued and assigned a batch sequence number.

2. The individual requesting the task (you or the operator) is sent the batch sequence number (to later permit that individual to CANCEL that task if he wants).

3. The requested task is then executed when the required resources become available in the system.

When you use EXECUTE to initiate a nonconversational task, the commands are taken, one at a time beginning with LOGON, from the cataloged command procedure (SYSIN data set) you specified. The system specifies the task's SYSOUT. You can read SYSIN input in your programs, in a manner similar to conversational mode, if the data is properly positioned in the SYSIN data set. Similarly, you can write to SYSOUT from your program. Because there is no prompting in nonconversational processing, you must specify every command completely, take care to have the commands in proper sequence, and include a SECURE command to

obtain any devices needed for private volumes.

Nonconversational SYSIN Data Set

A nonconversational SYSIN data set is a series of command statements and associated data that are to be acted upon in the sequence in which they are presented to the system; they inform the system of the actions you want performed during execution of your nonconversational task. You create a nonconversational SYSIN data set in the same way you create any other type of data set. You can construct it at your terminal by using the text editing commands (or DATA or MODIFY), or you submit it on punched cards to the system operator for entry into the system via the installation's high-speed card reader. The data set must be VSAM or VISAM line.

Each nonconversational SYSIN data set begins with a LOGON command and ends with a LOGOFF command, unless the mode of the task is originally conversational. (See "Mixed Mode Use of the System," below). If any private I/O devices are to be used by the task, the SECURE command must immediately follow the LOGON command, preceding all requests for those devices.

Data that is to be read by your program during execution can be included in the SYSIN data set; this data must immediately follow the invocation of your program. The end-of-data record (0-6-9 punch in column 1), if required, must follow the last data record.

NONCONVERSATIONAL TASK EXECUTION

The system analyzes, in the order presented, each command of the nonconversational SYSIN data set and executes every valid command. If a command is invalid, the system ignores it and continues reading SYSIN until either a valid command is read or the task is abnormally terminated. After reading and executing a valid command, the system proceeds to process the next command, continuing until it processes LOGOFF, which completes the task.

While executing a nonconversational task, you can also execute a conversational task, but you cannot communicate with the nonconversational task or affect its operation except to cancel it by issuing a CANCEL command. To inquire about the status of each of your uncompleted nonconversational tasks, you can issue an EXHIBIT command.

NONCONVERSATIONAL TASK TERMINATION

The execution of a nonconversational task (except PRINT or PUNCH) is terminated when its LOGOFF command is executed. When this occurs, the system automatically prints the task's SYSOUT data set. This data set contains the output from commands that were executed, any data that your program writes to SYSOUT, and the compiler-issued diagnostic messages (if no listings were requested).

Tasks created by the PRINT and PUNCH commands terminate when the data transfer is complete.

You can also terminate a nonconversational task by issuing a CANCEL command identifying the task by its batch sequence number.

MIXED MODE USE OF THE SYSTEM

You can begin a task at your terminal and then issue a BACK command to have the task's execution completed nonconversationally. Before issuing the BACK command, you must store a SYSIN data set that is to function as the command procedure and, if desired, input data for the nonconversational portion of your task; in addition, you must issue DDEF commands for any private volumes you may need. The SYSIN data set must not contain a LOGON command, because you have already logged on, but it should end with a LOGOFF command.

When you issue a BACK command, the system checks that it can provide sufficient resources to continue your task nonconversationally. If it cannot, the system rejects your request (and you can try later).

Once your BACK request is accepted, the terminal is inactive. If you want to continue using the terminal, you must log on again to initiate a new conversational task.

SECTION 5: COMPILING A PL/I PROGRAM

Each external procedure, including any procedures nested within it, must be compiled separately. If appropriate control statements are inserted among the PL/I source statements, the compiler can process two or more external procedures in a single run by means of batch compilation.

The PL/I compiler translates the procedure's source statements into machine instructions; the set of machine instructions produced in one compilation (that is, for one external procedure) is an object module. The compiler does not generate all the machine instructions required to represent the source program; for frequently needed services -- computation, error-handling, data transmission, and storage management -- it inserts calls to standard subroutines that are stored in the PL/I library. These calls will be performed during execution of the object module, when the services are actually needed.

While it is processing a PL/I source program, the compiler produces a listing that contains information about the source program and the object module derived from it, together with diagnostic messages relating to errors or other conditions detected during compilation. Much of this information is optional, and is supplied only in response to the inclusion of appropriate "options" in the PLIOPT and PLCOPT operands of the PLI command that invokes the compiler.

The compiler also includes a facility, the preprocessor or compile-time processor, which can modify the source statements or insert additional source statements before compilation begins.

After a module (that is, object module) is compiled, it is processed by a routine called the Object Data Set Converter (ODC). ODC resolves certain constants and changes the module from Operating System format to Time Sharing System format. Input to ODC consists of compiled modules in card-image format; output consists of executable modules.

RELATIONSHIP WITH TSS/360

Figure 3 shows how your TSS/360 PL/I object module interfaces with the system programs.

When the TSS/360 command system encounters the PLI command, control is transferred to Program Language Control (PLC), the main interface of the PL/I Compiler with TSS/360. See Figure 4. PLC supervises:

- Creation of the source data set, which consists of the PL/I source statements.
- Invocation of the PL/I compiler.
- Printing, or not printing, of compiler-generated listings.
- Invocation of ODC to convert the compiled object module to TSS/360 code.

In addition, PLC serves as a communication area that the compiler references to con-

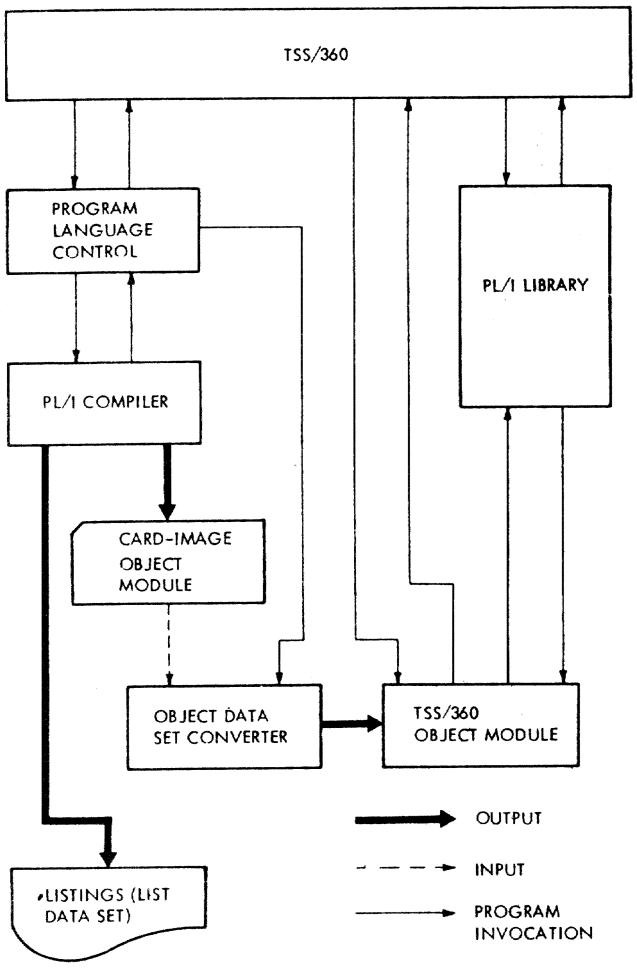


Figure 3. Relationship of a TSS/360 PL/I Object Module with the System Programs

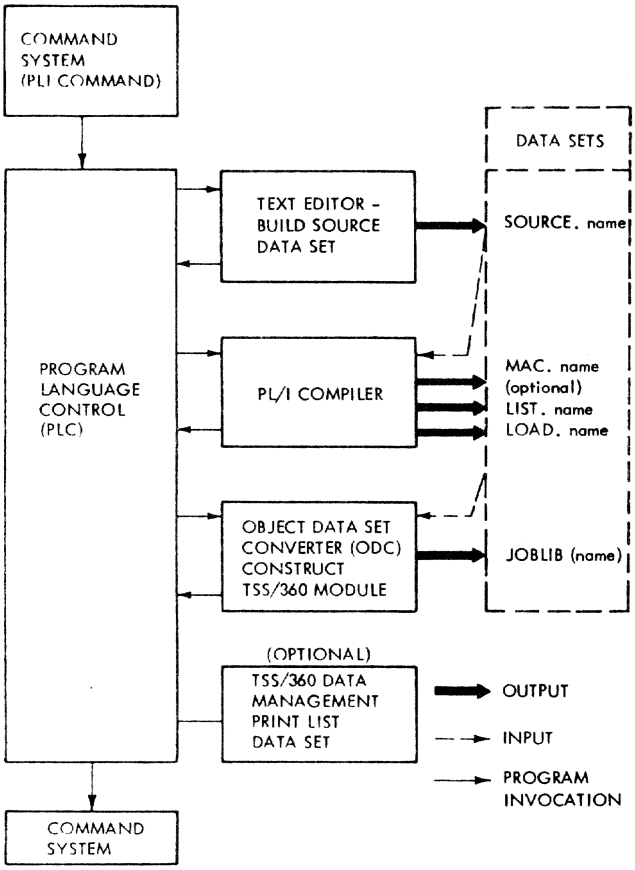


Figure 4. PLC Interfaces

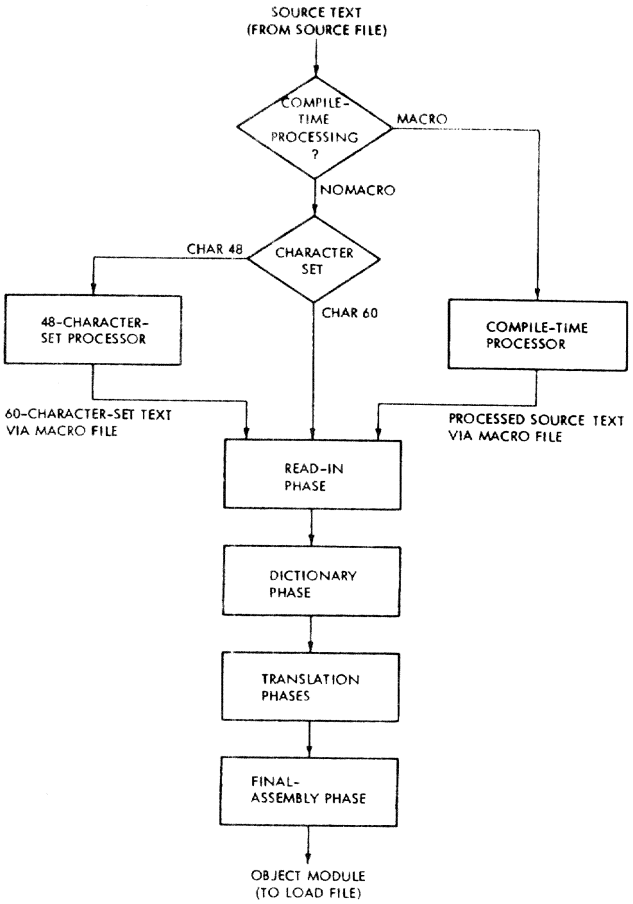


Figure 5. TSS/360 PL/I Compiler: Simplified Flow Diagram

trol its optional features, such as listings and diagnostics, that can be specified in suboperands of the PLI command.

COMPILER PHASES

The compiler comprises a control module and a series of subroutines (termed phases) that are executed under the supervision of the control module. Each phase performs a single function or a set of functions. The control module invokes the phases in accordance with the content of the source program and the optional compiler facilities that you select. Figure 5 is a simplified flow diagram of the compiler.

The PL/I compiler, unlike the TSS/360 Assembler and FORTRAN compiler, cannot function until the source data set, consisting of PL/I statements, has been fully entered. Therefore, when compilation is called for, the system searches your catalog for a source data set.

The system invokes the text editor if it does not find a data set with one of these names:

- SOURCE.name, where 'name' is specified in the NAME operand of the PLI command.
- 'source data set name', specified in the SOURCEDS operand of the PLI command.

The text editor prompts you with line numbers, and you enter your PL/I statements, typing an _END command when done. (For a fuller discussion of text editor facilities, see Part III. Any of the techniques shown there can be used to correct errors when compiling.) At this point, a source data set exists; control passes to the PL/I compiler.

The data that is translated by the compiler is known throughout all stages of the translation as text. Initially, the text comprises the PL/I source statements that you submit. At the end of compilation, it comprises the machine instructions that the compiler has substituted for the source statements, plus some information for reference by ODC.

The source statements can be either prestored in the system or entered via a terminal or a high-speed card reader. The source statements are passed to the read-in phase either directly or by means of one or two preprocessor phases:

1. If the source statements are in the PL/I 48-character set, the 48-character set processor translates them into the 60-character set. You must indicate the need for translation by specifying the CHAR48 option.[1]

2. If the source statements contain preprocessor statements, the compile-time processor executes the preprocessor statements in order to modify other source statements or introduce additional statements. The compile-time processor includes a facility for translating statements written in the 48-character set into the 60-character set.

To request the services of the compile-time processor, specify the MACRO option.

Both preprocessors place the translated source statements in the data set named MAC.name(0).

The read-in phase takes its input either from MAC.name(0), from SOURCE.name, or from a user-specified data set. This phase checks the syntax of the source statements and removes comments and nonsignificant blank characters.

After read-in, the dictionary phase of the compiler creates a dictionary that contains entries for all the identifiers in the source statements. The compiler uses the dictionary to communicate descriptions of the elements of the source program and the object program between phases. The dictionary phase of the compiler replaces all identifiers and attribute declarations in the source statements with references to dictionary entries.

Translation of the source statements into machine instructions involves several compiler phases. The sequence of events is:

1. Rearrangement of the source statements to facilitate translation (for example, by replacing array or structure assignments with DO loops that contain element assignments).

2. Conversion of the statements from the PL/I syntactic form to an internal syntactic form termed text.

3. Mapping of arrays and structures to ensure correct boundary alignment.

4. Translation of text into a form similar to machine instructions; this form is termed pseudo-code.

5. Storage allocation: the compiler provides storage for STATIC variables and generates code to allow AUTOMATIC storage to be allocated during execution of the object module. (The PL/I library subroutines handle the allocation of storage during execution of the object module.)

The final-assembly phase translates the pseudo-code into machine instructions, and then creates the external symbol dictionary (ESD) and relocation dictionary (RLD). The external symbol dictionary is a list that includes the names of all subroutines that are referred to in the object module but are not part of the module; these names, termed external references, include the names of all the PL/I library subroutines that will be required when the object module is executed. The relocation dictionary contains information that enables absolute storage addresses to be assigned to locations within the object module when it is loaded for execution.

HOW TO INVOKE THE COMPILER

You invoke the PL/I compiler by issuing a PLI command. This command allows you to:

1. Compile a prestored source data set.

2. Create a PL/I source data set and have it compiled.

3. Convert a PL/I object module into TSS/360 code.

4. Perform multiple compilations with a single command.

5. Have compiler-generated listings printed.

The format of the PLI command is shown in Table 2.

NAME --
The name by which the program will be known. It consists of one to eight alphameric characters, the first being alphabetic. If the name is omitted, PLC assumes that it is identical to the name of the source data set if that is in the correct form. If neither NAME nor the name of a source data set is provided, no compilation takes place and PLC processes the merge list or goes on to the next set of PLI parameters.

---

[1]The compiler options are discussed in Appendix G.

Table 2. PLI Command

Operation	Operand
PLI	[NAME=module name] [,PLIOPT=compiler option list] [,PLCOPT=language controller options][,SOURCEDS=source data set name] [,MERGELST=converter input list][,MERGEDS=converter input data set] [,MACRODS=intermediate data set name] [,EXPLICIT= {name / (name,...) / *ALL[(name[,...])]}] [,XFERDS=data set name]

PLIOPT --
The list of options to be used by the compiler. It is considered to be one parameter, and the list of compiler options following the equal sign in the PLIOPT parameter must therefore be enclosed in parentheses unless only one value is given; the separate options are separated by commas. The compiler options are described in Appendix G.

PLCOPT --
The list of options external to the PL/I compiler that affects the compilation's progression through TSS/360. These options must be enclosed in parentheses unless only one value is given. The options and the standard default for each are shown in Table 3.

Table 3. PLC Options

PLC Option	Standard Default
NOPRINT|PRINT|PRERASE	NOPRINT
DIAG|NODIAG	DIAG
NOCONT|CONT	NOCONT
LISTDS|LISTOUT	LISTDS
NOCONV	compile/convert
LIMEN= BREVITY=	system defaults

- NOPRINT or PRINT or PRERASE: specifies whether the listing data set produced by the compiler is to be printed on a high-speed printer. NOPRINT indicates that the data set is not to be printed as a part of the compilation. You can at some later time issue a PRINT command as follows:

  ```
  PRINT LIST.name(0),,,EDIT
  ```

  where 'name' is the module name given in the NAME operand. PRINT indicates that PLC should issue the print request automatically. PRERASE indicates that PLC should cause the data set to be printed and erased after printing; this is equivalent to:

  ```
  PRINT LIST.name(0),,,EDIT,ERASE
  ```

  If LISTOUT is specified, the data normally written into the list data set is directed to SYSOUT and no print request is appropriate. In this case, the NOPRINT option is assumed, whether or not it was specified.

- DIAG or NODIAG: specifies whether diagnostics are to be directed to SYSOUT. (This option has meaning only if LISTDS is specified. If LISTOUT is specified, then all compiler diagnostics appear on SYSOUT as a part of the listing data.) If DIAG is specified, then the diagnostics that appear on SYSOUT are controlled by two command-system operands, LIMEN and BREVITY, which control the severity and length of the PL/I diagnostics selected for printing on SYSOUT. LIMEN and BREVITY are explained later in this section.

  The format of the diagnostic message is:

  ```
  x IEMnnnnI statement no.  line no.
  text
  ```

  where x is the severity of the diagnostic and nnnn is the diagnostic number. For example:

  ```
  S IEM0182I  15  1600 TEXT BEGINNING
  'KEYFROM CK' SKIPPED IN OR FOLLOWING
  STATEMENT NUMBER 15
  ```

- NOCONT or CONT: specifies whether additional programs are to be compiled before return to the command system. NOCONT indicates that there is no continuation of compilation. If CONT is specified, then PLC prompts for a new module name by typing PLI when the

first compilation is complete; compilations can continue in this manner indefinitely. To end the prompting, enter an underscore with a command, or default by pressing the RETURN Key.

- NOCONV: Specifies that you want EXPLICIT name transformation only. There is to be no compilation or conversion; PLC will not call the compiler or ODC.

- LISTDS or LISTOUT: specifies whether the compiler is to construct a data set to contain the listing(s) it produces. LISTOUT indicates that a separate listing is unnecessary and that the listings can be placed in SYSOUT. In nonconversational mode, use of the LISTOUT option reduces the load on the system. In conversational mode, placing the listing on SYSOUT means that the system will type it at the terminal. Only in most urgent circumstances should you consider this alternative.

- LIMEN=: LIMEN is the operand name in the user profile for message-severity codes; it controls the severity of diagnostic messages printed on SYSOUT. If specified in the PLI command, it applies only to PL/I diagnostics. (See DIAG, above.) If not specified, the current value in the system profile is used.

LIMEN Value	Lowest Level Diagnostic Issued
I (information)	Warning messages
W (warning)	Error messages
N (normal error)	Severe error messages
X (extreme error)	Termination error message
T (termination error)	None will be shown

Note: The LIMEN PLC option cannot suppress messages issued by the compiler prior to the read-in phase (i.e., during the initialization phase).

- BREVITY=: BREVITY is the operand name in the user profile for message-length codes; it controls the length of diagnostic messages printed on SYSOUT. If specified in the PLI command, it applies only to PL/I diagnostics. (See DIAG, above.) If not specified, the current value in the system profile is used.

BREVITY Value	Action
M (message ID)	Message ID only is printed
S (standard)	Full text of message given
E (extended text)	Full text of message given

Note: Both LIMEN= and BREVITY= must be followed by only one character. If the equal sign is not the next-to-last character, the option is ignored. Thus:

```
LIMEN=I      is valid
LIMEN=INFO   is invalid
LIMEN= I     is invalid
```

SOURCEDS --
The fully qualified name of the data set from which the PL/I source statements are to be obtained. Any valid line data set is allowable. Examples:

1. ABLE
2. A.B.C.D
3. A.B(C)
4. A.B.G0000V00
5. A.B(0)
6. A.B(0)(C)

If the NAME operand is omitted, the SOURCEDS name is used as the name of the object module. Therefore, if the NAME operand is omitted and a TSS/360-executable object module is to be generated, the source data set must not be in the last-defined job library, since the object module will be stored in that library; TSS/360 does not allow a library to contain duplicate entry names.

If SOURCEDS is omitted, the name assumed for the source data set is SOURCE.name, where 'name' is the value you gave for the NAME operand.

If neither NAME nor SOURCEDS is given, it is assumed that no compilation is to take place for this iteration of PLC. Other functions involving ODC may be indicated.

MERGELST --
The names, separated by commas, of previously compiled modules to be converted by ODC for execution with the module being compiled. Each of these modules should still exist as unconverted modules, that is, as data sets named LOAD.name(0), where 'name' is the name given by you, or by default, in the NAME operand. (Initially, the compiler creates all modules as LOAD.name(0) data sets; you should not erase these data sets until you are sure that you have all needed copies

of the converted object module.) Modules that have been stored in job libraries after processing by ODC cannot be used in a MERGELST (merge list).

If the MERGELST operand is omitted but the LOAD option is indicated in the PLIOPT list, the PL/I compiler still generates a merge list containing the name of the compiled program.

MERGELST is similar to the NAME cards generated by OBJNM=aaaaaaaa in IBM System/360 Operating System PL/I. The merge list can be a single program name:

```
BAKER
```

or a list of program names enclosed in parentheses:

```
(FOX, GEORGE, HOW)
```

The list must not exceed 253 characters, including blanks and commas.

Duplicate program names in the list cause reprocessing of those programs. The only penalty is added processing time.

If no value is supplied for MERGELST, then a null string is assumed.

MERGEDS --
Allows you to name a data set as the source of the merge list. This can be in lieu of MERGELST or as a supplement to it. If this data set's organization is VS or VI, it is assumed that each record contains from 0 to 15 program names separated by commas. Spaces are immaterial. PLC and ODC assume that all programs named in the MERGEDS for which a LOAD.name(0) data set exists are to be combined into a single JOBLIB. Duplicate names cause duplicate processing but otherwise do not hurt.

If the data set organization is VP, then it is assumed that all the member names for which a LOAD.name(0) data set exists are to be combined into a JOBLIB. If the current active JOBLIB has the same name as MERGEDS, then all modules in the POD for which a PL/I LOAD.name(0) data set exists are to be reprocessed.

If no value is supplied, no data set is assumed for MERGEDS.

MACRODS --
The data set name to be associated with the intermediate text. If no name is given and either CHAR48 or MACRO options are specified, the compiler creates a data set named

```
MAC.name(0)
```

where 'name' is the user-supplied module name. This data set is normally erased when the compilation is completed. (See MACDCK, in Appendix G.) If you specify a value for MACRODS, that name is used instead of MAC.name(0) for the data set, and it is retained permanently if MACDCK is specified, with a compiler-generated source margin of 2 to 72. If a value is given for MACRODS but neither CHAR48 nor MACRO is specified, the value is ignored and does not affect compilation.

Note: When using this data set for recompilation, a source margin of 2 to 72 must be specified in the SORMGIN option of the PLI command's PLIOPT parameter.

EXPLICIT --
Specifies the entry names of procedures to be loaded on an as needed basis.

The implementation maximum for the cumulative length of all the PR entries in the ESD (see "External Symbol Dictionary," later in this section) is 4096 bytes. This normally results in a maximum of approximately 974 object modules per program. However, a larger program can be executed if one or more of its procedures are called dynamically, that is, only when actually needed during execution.

Dynamic calls can also be used to avoid unnecessary overhead in the invoking of a program. When normal calls are used, subroutines (that is, all routines for which there are CALL statements) are loaded and linked to the calling routines whether or not they will be needed during execution. On the other hand, loading and linking of dynamically called procedures is deferred until execution of the CALL statements.

Names specified in the EXPLICIT operand are padded on the left with @. (You can specify a different pad character by issuing the command DEFAULT PADCHAR=pad-character.) The pad characters are inserted into the object module, not into the source data set. Calls to the padded names function as calls to a transfer module that calls the original names dynamically. You must create the transfer

module yourself; Example 18, in "Part III: Examples" shows how.

name

(name,...)
Specifies one or more entry names to be padded. These names must appear in CALL statements in the module being processed.

*ALL[(name[,...])]
Specifies that all called names except those within parentheses and except names beginning with IHE (that is, names of PL/I library modules, for which dynamic calls are automatically generated) will be padded.

Names unacceptable to basic assembler language or to the PL/I compiler will not be padded.

Using the default value of @ for PADCHAR:

Original Name	Padded Name
PROC	@PROC
@PROC	@@PROC
NO_GOOD	unacceptable to basic assembler language; not padded
NOT2BIG	@NOT2BIG
ONE2MANY	unacceptable to PL/I compiler; not padded.

You should avoid starting a procedure name with the pad character that you use. If procedures PROC and @PROC were in the same program and their names were padded to form @PROC and @@PROC, as in the above example, procedure @PROC could not execute.

If you issue the command DEFAULT MAP=Y, the system reports on the results of EXPLICIT processing. A sample report:

```
MAP FOLLOWS FOR MODULE XXX
ABC  GENERATED AS @ABC
XYZ  GENERATED AS @XYZ
DOIT RESOLVED TO #DOIT
ABC  RESOLVED TO @ABC
DOIT RESOLVED TO #DOIT
```

In this example, you have used the XFERDS operand to specify a transfer data set (the main part of the transfer module), and you are updating the transfer data set with new entries. "GENERATED AS" means that the name was padded and entered in the transfer data set. "RESOLVED TO" means that the name was padded and was already in the transfer data set.

The EXPLICIT operand is used in conjunction with the UPDTXFER default value. Refer to the following description of the XFERDS operand for an explanation of UPDTXFER.

XFERDS --
The name of the transfer data set, the core of the transfer module. The transfer data set is generated as a line data set of the form:

0-7	line-number
8	X'00'
9-16	generated name
17	blank
18-24	PLICALL
25-27	blank
28-35	entry-name

If you include the XFERDS operand, the system creates/maintains the transfer data set for you. If you omit this operand, you must create/maintain your own transfer data set.

A default value, UPDTXFER, can be used with the EXPLICIT and XFERDS operands to tell the system exactly how to generate explicit calls. See Table 4. The IBM-supplied value for UPDTXFER is N.

HOW TO STOP THE COMPILER

In most cases, the simplest way to stop a blundered-into compilation is to press the attention key. If you have been entering source statements and have not yet typed _END, instead of pressing the attention key you can type _|END. (The vertical bar signals that compilation is not wanted.) Like the attention interruption, the _|END command returns your task to the command mode instead of passing control to the PL/I compiler; in addition, it closes the source data set and makes a clean exit from the text editor. Use this command if you want to save the source data set for a processor other than the text editor.

Source data sets should be erased when they are no longer needed.

DATA SETS ACCESSED BY THE COMPILER

The compiler accesses several standard data sets, the number depending on the optional facilities that you request. These are shown in Table 5. You do not have to issue DDEF commands for these files, unless you specify an INCLUDE library. See "Invoking the Preprocessor," in this section.

Table 4. Dynamic Calls -- Padding and Entering of Entry Names by the System

	UPDTXFER=Y	UPDTXFER=N
	name is...	name is...
EXPLICIT=name name in transfer data set	padded in module	padded in module
EXPLICIT=name name not in transfer data set	padded in module entered in transfer data set	padded in module not entered in transfer data set
EXPLICIT omitted or EXPLICIT=(name$_1$,...) where name not in list name in transfer data set	padded in module	ignored
EXPLICIT omitted or EXPLICIT=(name$_1$,...) where name not in list name not in transfer data set	ignored	ignored
EXPLICIT=*ALL(name) name in transfer data set	ignored	ignored
EXPLICIT=*ALL(name) name not in transfer data set	ignored	ignored

Table 5. Standard Data Sets for Compilation

Purpose	DSNAME	DDNAME	Associated Option
Primary input (PL/I source statements)	SOURCE.name or user-specified	PLIINPUT or user-specified	-
Object code data set output (will be converted by ODC for TSS/360 use)	LOAD.name(0)	PLILOAD	PLIOPT=LOAD, or PLIOPT=DECK
Storage for: 1. Translated source statements when 48-character set used 2. Source statements generated by preprocessor	MAC.name(0) or user-specified	PLIMAC	 PLIOPT=CHAR48 PLIOPT= MACRO, COMP
Listing	LIST.name(0)	PLILIST	PLCOPT=LISTDS
Library containing source statements for insertion by preprocessor	USERLIB or user-specified	SYSULIB or user-specified	PLIOPT=MACRO
Note: 'name' is the module name.			

CONTENTS OF THE SOURCE DATA SET AND THE OBJECT MODULE

A PL/I object module contains one, and only one, external procedure. Consequently, a source data set must not be structured like this:

```
A:  PROCEDURE;
      .
      .
      .
    END;

B:  PROCEDURE;
      .
      .
      .
    END;
```

There are two alternatives:

- Create two separate source data sets, one for procedure A and one for procedure B, for compilation into two separate object modules, or
- Nest one of the procedures within the other.

FORMAT OF SOURCE LINES

Lines of the source data set can be up to 100 characters long. Margins can be set within this range, using the SORMGIN option of the PLI command. If you set margins, you can put information such as comments, card numbers, etc., outside the margins.

CHARACTER SETS -- KEYBOARD FORMAT

KA and KB commands are used to specify the character set to be used during keyboard input. KA specifies the full EBCDIC character set during input. KB specifies that the lowercase characters (a-z and !") are translated into their uppercase equivalents (A-Z and $#@ respectively).

ENTRY OF KEYBOARD SOURCE STATEMENTS FOR LATER PUNCHING AND RECOMPILATION

Entry of source statements so that they can be later punched out and reentered in card format is governed by the following considerations:

1. Source lines reside in a line data set in which the initial input source line is preceded by eight characters -- a 7-byte zoned-decimal key and a character specifying to TSS/360 that the source line was entered in card form or at the terminal keyboard.
2. A continued line (hyphen preceding the carriage return) when punched and reentered in card format retains the hyphen unless precautions are taken to remove it.
3. Keyboard input positioning requirements are much more flexible than for card input.

LISTING

During compilation, the compiler generates a listing that contains information about the compilation and about the source and load modules. It places this listing in the list data set (defined as DSNAME=LIST.module name), if the PLCOPT operand of the PLI command contains the suboperand LISTDS, or if that suboperand is defaulted. If the listing suboperand is specified as LISTOUT, the listing is written to SYSOUT and the PRINT suboperand is ignored. SYSOUT is the terminal in conversational mode, and a printer in nonconversational mode.

The following description of the listing refers to its appearance on a printed page.

The listing comprises a small amount of standard information that always appears, together with those items of optional information requested or applied by default in the PLIOPT operand of the PLI command. Table 6 lists the optional components of the listing and the corresponding compiler options.

The first page of the compiler listing is identified by a heading giving the date and time, the title "TSS/360 PL/I Compiler", and the version number. Starting with this page, all the pages of the listing are numbered sequentially in the top right-hand corner. Page 1 also includes a list of the options specified for the compilation, exactly as they are written in the PLIOPT operand of the PLI command.

The listing always ends with a statement that no errors or warning conditions were detected during the compilation, or with one or more diagnostic messages. Appendix H lists all compiler messages.

The following paragraphs describe the optional parts of the listing in the order of appearance on the listing.

OPTIONS USED FOR THE COMPILATION

If the option OPLIST applies, a complete list of the options for the compilation, including the default options, follows the statement of the options specified in the PLIOPT Parameter of the PLI command.

Table 6. Optional Components of Compiler Listing

Listings	Option Required
Options for the compilation	OPLIST
Preprocessor input	SOURCE2
Source program	SOURCE
Statement nesting level	NEST
Attribute table	ATR
Cross-reference table	XREF
Aggregate-length table	ATR or XREF
External symbol dictionary	EXTREF
Object module	LIST
Diagnostic messages for severe errors, errors, and warnings	FLAGS, FLAGE, FLAGW

PREPROCESSOR INPUT

If both the options MACRO and SOURCE2 apply, the compiler lists the input statements to the preprocessor, one record per line. The records are numbered sequentially at the left.

If the compiler detects an error or the possibility of an error during the preprocessor phase, it prints a message on the page or pages following the listing of preprocessor input. The format and classification of the error messages are exactly as described for the compilation error messages, under "Diagnostic Messages", below.

SOURCE PROGRAM

If the option SOURCE applies, the compiler lists the source program input, one record per line; if the input statements include carriage control characters, the lines are spaced accordingly. The statements in the source program are numbered sequentially by the compiler, and the number of the first statement in the line appears to the left of each line in which a statement begins.

Between the statement number and the source line, appears a seven-character VI line number. This is incremented by 100 for each line.

If the source statements were generated by the preprocessor, columns 73-80 contain the following information:

Column

73-77 Input record number from which the source statement was generated. This number corresponds to the record number in the preprocessor input listing.

78,79 Two-digit number giving the maximum depth of replacement for this line. If no replacement occurred, the columns are blank.

80 'E' signifies that an error occurred while replacement was being attempted. If no error occurred, the column is blank.

STATEMENT NESTING LEVEL

If the options SOURCE and NEST apply, the block level and the DO level are printed to the right of the statement number under appropriate headings:

```
STMT  LEVEL  NEST
 1                    000100  A:  PROC OPTIONS
                                  (MAIN);
 2       1            000200  B:  PROC (L)
 3       2            000300  DO I=1 to 10;
 4       2      1     000400  DO J=1 to 10;
 5       2      2     000500  END;
 6       2      1     000600  BEGIN;
 7       2      1     000700  END;
 8       2      1     000800  END B;
 9       1            000900  END A;
```

ATTRIBUTE AND CROSS-REFERENCE TABLE

If the option ATR applies, the compiler prints an attribute table containing an alphameric list of the identifiers in the program together with their declared and default attributes. If the option XREF applies, the compiler prints a cross-reference table containing an alphameric list of the identifiers in the program together with the numbers of the statements in which they appear. If both ATR and XREF apply, the two tables are combined.

Except for file attributes, the attributes printed are those assigned after conflicts have been resolved and defaults ap-

plied. Since the file attributes are not analyzed until the attribute list has been prepared, the attributes listed for a file are those supplied by you, regardless of conflicts.

If either the ATR or the XREF option applies, the compiler also prints an aggregate-length table that gives, where possible, the lengths in bytes of all major structures and all non-structured arrays in the program.

Attribute Table

If an identifier was declared explicitly, the number of the DECLARE statement is listed under the heading DCL NO. The statement numbers of statement labels and entry labels are also given under this heading.

The attributes INTERNAL and REAL are never included; they can be assumed unless the conflicting attributes EXTERNAL and COMPLEX appear.

For a file identifier, the attribute EXTERNAL appears if it applies; otherwise, only explicitly declared attributes are listed.

For an array, the dimension attribute is printed first; the bounds are printed as in the array declaration, but expressions are replaced by asterisks.

For a character string or a bit string, the length preceded by the word STRING is printed as in the declaration, but an expression is replaced by an asterisk.

Cross-Reference Table

If the cross-reference table is combined with the attribute table, the numbers of the statements in which an identifier appears follow the list of attributes for that identifier. The number of a statement in which a based-variable identifier appears is included, not only in the list of statement numbers for that variable, but also in the list of statement numbers for the pointer associated with it.

Aggregate Length Table

Each entry in the aggregate length table consists of an aggregate identifier preceded by a statement number and followed by the length of the aggregate in bytes.

The statement number is the number either of the DECLARE statement for the aggregate or, for a CONTROLLED aggregate, of an ALLOCATE statement for the aggregate. An entry appears for every ALLOCATE statement involving a CONTROLLED aggregate, since such statements have the effect of changing the length of the aggregate during execution. Allocation of a BASED aggregate does not have this effect, and only the entry for the DECLARE statement appears.

The length of an aggregate may be unknown at compilation, either because the aggregate contains elements having adjustable lengths or dimensions, or because the aggregate is dynamically defined. In these cases, the word 'ADJUSTABLE' or 'DEFINED' appears in the LENGTH IN BYTES column.

An entry for a COBOL mapped structure, that is, for a structure into which a COBOL record is read or from which a COBOL record is written, has the word '(COBOL)' appended, but such an entry appears only if the structure does not consist entirely of:

1. doubleword data, or
2. fullword data, or
3. halfword binary data, or
4. character string data, or
5. aligned bit string data, or
6. a mixture of character string and aligned bit string data.

If a COBOL entry does appear, it is additional to the entry for the PL/I-mapped version of the structure.

STORAGE REQUIREMENTS

If the option SOURCE applies, the compiler lists the following information under the heading STORAGE REQUIREMENTS on the page following the end of the aggregate-length table:

1. The storage area in bytes for each procedure.
2. The storage area in bytes for each BEGIN block.
3. The storage area in bytes for each ON-unit.
4. The length of the text control section. (The machine instructions in the object module are grouped in blocks called control sections, or CSECTs.) The text control section contains the executable instructions.
5. The length of the static internal control section. This control section contains all storage for variables declared STATIC INTERNAL.

TABLE OF OFFSETS

If the options SOURCE, NOSTMT, and NOLIST apply, the compiler lists, for each primary entry point, the offsets at which the various statements occur. This information is found, under the heading TABLE OF OFFSETS AND STATEMENT NUMBERS WITHIN PROCEDURE, following the end of the storage requirements table.

EXTERNAL SYMBOL DICTIONARY

If the option EXTREF applies, the compiler lists the contents of the external symbol dictionary (ESD) for the object module. The ESD is a table containing all the external symbols that appear in the module. (An external symbol is a name that can be referred to in a control section other than the one in which it is defined.) The information appears under the following headings:

SYMBOL
An 8-character field that identifies the external symbol.

TYPE
Two characters from the following list to identify the type of ESD entry:

- SD - Section definition: the name of a control section within this module.
- CM - Common area: a type of control section that contains no executable instructions. The compiler creates a common area for each non-string element variable declared STATIC EXTERNAL without the INITIAL attribute.
- ER - External reference: an external symbol that is not defined in this module.
- PR - Pseudo-register: a field in a communications area, the pseudo-register vector (PRV), used by the compiler and the library subroutines.
- LD - Label definition: the name of an entry point to the external procedure other than that used as the name of the program control section.

ID
Four-digit hexadecimal number: the entries in the ESD are numbered sequentially, commencing from 0001.

ADDR
Hexadecimal representation of the address of the symbol: this field is not used by the compiler, since the address is not known at compile time.

LENGTH
The hexadecimal length in bytes of the control section (SD, CM, and PR entries only).

Standard ESD Entries

The external symbol dictionary always starts with seven standard entries (Table 7).

1. Name of the text control section (the control section that contains the executable instructions). This name is the first label of the external procedure statement.
2. Name of the static internal control section (which contains storage for all variables declared STATIC INTERNAL). This name is the first label of the external procedure statement, padded on the left with asterisks to seven characters if necessary, and extended on the right with the character A.
3. IHEQINV: pseudo-register for the invocation count (a count of the number of times a block is invoked recursively).
4. IHESADA: entry point of the library routine that obtains automatic storage for a block.
5. IHESADB: entry point of the library routine that obtains automatic storage for variables whose extents are not known at compile time.

Table 7. Typical Standard ESD Entries

SYMBOL	TYPE	ID	ADDR	LENGTH
FIG5	SD	0001	000000	00033A
***FIG5A	SD	0002	000000	00005F
IHEQINV	PR	0003	000000	000004
IHESADA	ER	0004	000000	
IHESADB	ER	0005	000000	
IHEQERR	PR	0006	000000	000004
IHEQTIC	PR	0007	000000	000004

6. IHEQERR: pseudo-register used by the library error-handling routines.

7. IHEQTIC: pseudo-register used by the library multitasking routines.

Other ESD Entries

The remaining entries in the external symbol dictionary vary, but generally include the following:

1. Section definition for the 4-byte control section IHEMAIN, which contains the address of the principal entry point to the external procedure. This control section is present only if the procedure statement includes the option MAIN.

2. Section definition for the control section IHENTRY (always present). Execution of a PL/I program always starts with this control section, which passes control to the appropriate initialization subroutine of the PL/I library; when initialization is complete, control passes to the address stored in the control section IHEMAIN. (Initialization is required only once during the execution of a PL/I program, even if it calls another external procedure; in such a case, control passes directly to the entry point named in the CALL statement, and not to IHENTRY.)[1]

3. LD-type entries for all names of entry points to the external procedure except the first.

4. A PR-type entry for each block in the compilation. The name of each of the pseudo-registers comprises the first label of the external procedure statement, padded on the left with asterisks to seven characters if necessary, and extended on the right with an eighth character selected from one of two tables used by the compiler. If the number of blocks exceeds the number of characters in the first table, the first character of the pseudo-register name is replaced by a character taken from the second table, and the last character recycles. If the first character thus overwritten is the start of the external procedure name rather than an asterisk, the compiler issues a warning message (since identical pseudo-register names could be generated from different procedure names).

   These pseudo-registers are termed <u>display pseudo-registers</u>.

   <u>Example</u>:

```
X:  PROC;
      Y:  PROC;
            Z:  BEGIN;
      END X;
```

   The display pseudo-registers for X, Y, and Z would have the names:

```
******XB
******XC
******XD
```

5. ER-type entries for all the library routines and external routines called by the program. The list includes the names of library routines called directly by compiled code (first-level routines), and the names of routines that are called by the first-level routines.

6. CM-type entries for non-string element variables declared STATIC EXTERNAL without the INITIAL attribute.

7. SD-type entries for all other STATIC EXTERNAL variables and for EXTERNAL file names.

8. PR-type entries for all file names. For EXTERNAL file names, the name of the pseudo-register is the same as the file name; for INTERNAL file names, the compiler generates names as for the display pseudo-registers.

9. PR-type entries for all controlled variables. For external variables, the name of the variable is used for the pseudo-register name; for internal variables, the compiler generates names as for the display pseudo-registers.

OBJECT MODULE

If the option LIST applies, the compiler generates a map of the static internal control section and lists the machine instructions of the object program in a form similar to System/360 assembler language. The machine instructions are described in <u>IBM System/360: Principles of Operation</u>. The following descriptions of the object module listings include many terms that can be properly defined only in the context of an explanation of the mechanism of compilation and the structure of the object program; such an explanation is beyond the scope of this manual.

---

[1]Although IHEMAIN and IHENTRY are produced by the compiler as described, they are combined into a single section during ODC conversion.

Both the static internal storage map and the object program listings start on a new page. If the LINECNT option specifies 72 or fewer lines per page and the number of lines to be printed (including skips) exceeds the specified line count, double-column format is used. If the LINECNT option specifies more than 72 lines per page or the number of lines to be printed (including skips) is less than the specified line count, single-column format is used.

Static Internal Storage Map

The first 52 bytes of the static internal control section are of a standard form and are not listed. They contain the following information:

```
DC F'4096'
DC AL4(SI.+X'1000')
DC AL4(SI.+X'2000')
DC AL4(SI.+X'3000')
DC AL4(SI.+X'4000')
DC AL4(SI.+X'5000')
DC AL4(SI.+X'6000')
DC AL4(SI.+X'7000')
DC VL4(IHESADA)
DC VL4(IHESADB)
DC A(DSASUB)
DC A(EPISUB)
DC A(IHESAFA)
```

SI. is the address of the static internal control section, and IHESADA, IHESADB, and IHESAFA are library subroutines. DSASUB and EPISUB are compiler routines for getting and freeing dynamic storage areas (DSAs).

The remainder of the static control section is listed, each line comprising the following elements:

1. Six-digit hexadecimal offset.
2. Up to eight bytes of hexadecimal text.
3. Comment indicating the type of item to which the text refers; a comment appears against the first line only of the text for an item.

The following abbreviations are used for the comments (xxx indicates the presence of an identifier):

DED FOR TEMP or DED	Data element descriptor for a temporary or for a programmer's variable.
FED	Format element descriptor.
DV..xxx	Dope vector for a static variable.
DVD..	Dope vector descriptor.
D.V. SKELETON	Dope vector skeleton for an automatic or controlled variable.
RDV..	Record dope vector.
A..xxx	Address of external control section or entry point, or of an internal label.
ARGUMENT LIST	Argument list skeleton.
CONSTANTS	Constants.
SYMTAB	Symbol table entry.
SYM..xxx	Symbolic name of label constant or label variable.
FILE..xxx	File name.
ON..xxx	Programmer-declared ON-condition.
ATTRIB	File attributes.
xxx	Static variable. If the variable is not initialized, no text appears against the comment, and there is also no static offset if the variable is an array. (This can be calculated from the dope vector if required.)

Object Program Listing

The object program listing includes comments of the following form as an aid to identification of the functions of the components of the program:

- STATEMENT NUMBER n - identifies the start of the code generated for source listing statement number n.
- PROCEDURE xxx - identifies the start of the procedure labeled xxx.
- REAL ENTRY xxx - heads the initialization code for an entry point to a procedure labeled xxx.
- PROLOGUE BASE - identifies the start of the initialization code common to all entry points to that procedure.
- PROCEDURE BASE - identifies the address loaded into the base register for the procedure.

- APPARENT ENTRY xxx - identifies the point of entry into the procedure for the entry point labeled xxx.
- END PROCEDURE xxx - identifies the end of the procedure labeled xxx.
- BEGIN BLOCK xxx - indicates the start of the begin block with label xxx.
- END BLOCK xxx - indicates the end of the begin block with label xxx.
- INITIALIZATION CODE FOR xxx - indicates that the code following performs initial value assignment or dope vector initialization for the variable xxx.

Wherever possible, a mnemonic prefix is used to identify the type of operand in an instruction, and where applicable this is followed by a source program identifier. The following prefixes are used:

A..	Address constant.
AE..	Apparent entry point (point in the procedure to which control passed from the prologue).
BLOCK.	Label created for an otherwise unlabeled block (followed by the number of the block).
C..	Constant (followed by a hexadecimal dictionary reference).
CL.	A label generated by the compiler (followed by a decimal number identifying the label).
DED..	Data element descriptor.
DV..	Dope vector.
DVD..	Dope vector descriptor.
FVDED..	Data element descriptor of function value.
FVR..	Function value.
IC.	Invocation count pseudo-register.
ON..	ON-condition name.
PR..	Pseudo-register.
RDV..	Record dope vector.
RSW..	Return switch.
SI.	Address of static internal control section.
SKDV..	Skeleton dope vector, followed by hexadecimal dictionary reference.
SKPL..	Skeleton parameter list, followed by hexadecimal dictionary reference.
ST..	Symbol table entry.
SYM..	Symbolic representation of a label.
TCA..	Temporary control area: a word containing the address of the dope vector of the specified temporary.
TMP..	Temporary, followed by hexadecimal dictionary reference.
TMPDV..	Temporary dope vector, followed by hexadecimal dictionary reference.
VO..	Virtual origin.
WP1. WP2. WS1. WS2. WS3.	Workspace, followed by decimal number of block which allocates workspace.

A listing of the various storage areas is not produced, but the addresses of variables can be deduced from the object program listing.

Example: A=B+10E1; in the source program produces:

```
0002CA  78 00 B 058  LE    0,B

0002CE  7A 00 B 064  AE    0,C..08F4

0002D2  70 00 B 060  STE   0,A
```

A and B are STATIC INTERNAL variables at an offset of X'60' and X'58', respectively, from the start of the control section.

DIAGNOSTIC MESSAGES

The compiler generates messages that describe any errors or conditions that may lead to error that it detects during compilation. Messages generated by the preprocessor appear in the compiler listing immediately after the listing of the statements processed by the preprocessor; all other messages are grouped together at the end of the listing. The messages are graded according to their severity:

A warning message calls attention to a possible error, although the statement to which it refers is syntactically valid.

An error message describes an attempt made by the compiler to correct an

erroneous statement (although it may not specify the corrective action).

A severe error message specifies an error that cannot be corrected by the compiler. The incorrect statement or part of a statement is deleted, but compilation continues. However, if a severe error is detected during the preprocessor stage, compilation is terminated after the compiler has listed the source program.

A termination error message describes an error that forces the termination of the compilation.

The compiler lists only those messages with a severity equal to or greater than that specified by the FLAG compiler option:

Type of Message	Option
warning	FLAGW
error	FLAGE
severe error	FLAGS
termination error	Always listed

Each error message is identified by an 8-character code:

1. The first three characters are IEM, which identify the message as emanating from the PL/I compiler.
2. The next four characters are a 4-digit message number.
3. The last character is the letter I, which is the code for an informative message.

Appendix H lists all the compiler messages in numeric sequence.

Messages issued by PLC and ODC are identified by the prefixes CFBAA and CFBAB. These messages are explained in the System Messages publication.

MULTIPLE COMPILATIONS

The multiple compilation facilities of the TSS/360 PL/I compiler allow you to compile more than one object module in a single execution of the compiler. Multiple compilation can increase compiler throughput by reducing system overhead.

Two forms of multiple compilations are possible: the CONT option of the PLCOPT operand of the PLI command, and the *PROCESS statement. Both forms can be used in a single execution of the compiler.

CONT OPTION

By specifying the CONT option in the PLCOPT parameter, you notify Program Language Control (PLC) that a new compilation or series of compilations is to be initiated after the current compilation or series of compilations is completed; PLC will prompt you for the new module name by typing PLI. The PLCOPT and PLIOPT parameters are unchanged from the first compilation.

PLC treats the new compilation(s) as an entirely separate unit; in effect, PLC acts as if a new PLI command has been issued with a new module name, the only difference being that no exit is taken from PLC to the Command System and the user, when the compilation is complete. Each compilation or series of compilations initiated by either the PLI command itself or by the CONT option is called an iteration of the PLI command. Each iteration of PLI invokes the PL/I compiler (and the text editor, if required) and then passes the programs that were compiled in that iteration together with all compiled modules listed in the merge list or merge data set for that iteration, to the Object Data Set Converter (ODC) for processing. ODC is invoked only once in each iteration; when ODC has finished its processing, PLC starts on the next iteration, if CONT was specified.

There is no limitation on the number of iterations of the PLI command that can be initiated through CONT options.

THE *PROCESS STATEMENT

A number of source programs can be compiled consecutively within one iteration of PLI. This is accomplished by following the first program in that iteration with a *PROCESS statement. The *PROCESS statement informs the compiler that a new program is to be compiled. The statement is followed by a PL/I option list, identical in every respect with the PLIOPT parameter of the PLI command except that the name you give to the new program must be specified in the option OBJNM=aaaaaaaa where aaaaaaaa stands for the program name of one to eight alphameric characters, the first of which is alphabetic.

Since only PLIOPT options can be specified in this option list, all the PLC options specified in the parameters of that iteration apply unchanged. All batched compilations to be initiated with *PROCESS must therefore reside in the same source data set, and all those specified with the LOAD option are placed in the same merge list.

Output listings from one iteration of PLI will be directed to one listing data set where they will be formatted consecutively. The name associated with this listing data set is the name of the first program in the iteration. Compiled modules in the form LOAD.name(0) are, however, separate, and each such module has associated with it the program name specified by the NAME parameter for the first compilation and by the OBJNM option for subsequent compilations. If the name given for OBJNM repeats a name already used once as a program name in that iteration, the module compiled earlier is erased and replaced by the new module.

Format of the *PROCESS Statement

The format of the *PROCESS statement is:

```
*PROCESS ('options');
```

where 'options' indicates a list of compiler options exactly as specified in the PLIOPT operand of the PLI command; the list of options must be enclosed within apostrophes. The asterisk must be in the first byte of the record, and the keyword PROCESS may follow in the next byte (column) or after any number of blanks. Blanks are also permitted between:

1. The keyword PROCESS and the option-list delimiter (left parenthesis).
2. The option-list delimiters and the start or finish of the option list.
3. The option-list delimiter and the semicolon.

The options in the option list can include any of those described in Appendix G. The options must be separated by commas, and there must be no embedded blanks. The options apply to the compilation of the source statements between that *PROCESS statement and the next *PROCESS statement. If you omit any of the options, the default values apply; there is no carryover from the preceding *PROCESS statement. The number of characters is limited only by the length of the record. If you do not want to specify any options, code

```
*PROCESS;
```

THE OBJNM OPTION: The OBJNM option is used in the option string of a *PROCESS statement to give a module name to the new module. It is specified as OBJNM=a, where 'a' is a one-to-eight character alphameric name.

COMPILE-TIME PROCESSING

The MACRO option or the CHAR48 option in the PLIOPT options cause compile-time processing to be initiated. You can save the output from compile-time processing by specifying MACDCK in the PLIOPT operand. If you would like your own DSNAME for this macro file, you can specify it in the MACRODS operand. You can then use the intermediate data set, which will have the name given by MACRODS, as input in future compilations, saving time by bypassing compile-time processing.

The compile-time facilities of the TSS/360 PL/I compiler are described in PL/I Language Reference Manual. These facilities allow you to include in a PL/I program statements that, when they are executed by the preprocessor stage of the compiler, modify your source statements or cause source statements to be included in your program from a library.

INVOKING THE PREPROCESSOR

The preprocessor stage of the compiler is executed only if you specify the option MACRO. If no value is specified for the MACRODS parameter, the compiler creates the data set MAC.name(0) to hold the preprocessed source statements until compilation begins.

The term MACRO owes its origin to the similarity of some applications of the compile-time facilities to the macro language available with such processors as the System/360 assembler. Such a macro language allows you to write a single instruction in your program to represent a sequence of instructions that have previously been defined.

Three other compiler options, MACDCK, SOURCE2, and COMP, are meaningful only when you also specify the MACRO option. All are described earlier in this section.

The %INCLUDE Statement

PL/I Language Reference Manual describes how to use the %INCLUDE statement to incorporate source statements from a partitioned data set into a PL/I source program. (A partitioned data set is used for the storage of other data sets, termed members. Thus, a set of source statements that you may want to insert into a source program by means of a %INCLUDE statement must exist as a data set (member) within a partitioned data set.)

The %INCLUDE statement includes one or more pairs of identifiers. Each pair of identifiers specifies the DDNAME operand of

a DDEF command that defines a library and, in parentheses, the name of a member of the library. For example, the statement:

```
%INCLUDE DD1 (INVERT),DD2(LOOPX)
```

specifies that the source statements in member INVERT of the library defined DDNAME=DD1, and those in member LOOPX of the library defined DDNAME=DD2, should be inserted into the source program. The task must include appropriate DDEF commands.

If you omit the DDNAME from any pair of identifiers in a %INCLUDE statement, the preprocessor assumes USERLIB; no DDEF command is then required.

Source statements in a library must be in the form of a virtual index sequential (VI) line data set. The source margin for input records specified by the SORMGIN option applies equally to source statements inserted by a %INCLUDE statement.

SECTION 6: STORING AND INVOKING THE MODULE

At the beginning of Section 5, an object module was defined as the output of a single compilation, exclusive of the listing; an object module corresponds to a single external procedure and any procedures nested within it. A PL/I program is one or more object modules consisting of a main procedure and any subroutines that it requires.

A program library is a set of object modules that is treated as a single data set in relation to the system catalog and access devices. All programs are stored in object module form in program libraries. (Exception: Your installation may provide for storing of object modules in initial virtual storage. See "Sharing," later in this section.) The system provides you with two program libraries: a user library and a copy of the system library. Using the DDEF command, you can define additional program libraries.

Using the linkage editor,[1] the merging facilities of the PLI command, or the CDS (copy data set) command, you can move object modules from one library to another.

PROGRAM LIBRARY LIST CONTROL

Each program library is a partitioned data set. There are four types of program libraries:

1. System library (SYSLIB)
2. User library (USERLIB)
3. User-defined job libraries (JOBLIBs)
4. Other user-defined program libraries.

SYSTEM LIBRARY

The system library contains some service routines, provided by the installation, and the PL/I library subroutines. It is accessible to all users on a read-only basis. You need not define or catalog this library.

USER LIBRARY

The user library is the private library assigned to you the first time you log on. It is kept in public storage, and, hence, located on a direct access device. This library is automatically defined and made a part of your catalog by the system; it is thus available each time you log on. If you do not specify a job library, the object modules resulting from use of the PL/I compiler are placed in your user library.

Note: TSS/360 does not allow a library to contain more than one noncontextual declaration of an entry name. For PL/I, this applies to module names and entry names of external procedures; entry names of internal procedures are excepted.

Since a library must not contain more than one declaration of any entry name, you may want to restrict your user library to programs that you rely on for getting work done. Test versions of these programs can be placed elsewhere. The program library list makes it possible for you to control the contents of your user library. This list is a defined hierarchy of program libraries; it is initialized at log-on time and consists of your user library and a copy of the system library. The library at the top of the list (initially the user library) automatically receives all object modules resulting from language processing. In addition to using the program library list to store object modules, the system uses this list to control its order of search when looking for programs that must be loaded at execution time. The library at the top of the list is searched first, then the next-to-the-top library, etc. The user library and the system library are searched after any other libraries on the list. If no job libraries are defined, the library at the top of the list is always the user library.

JOB LIBRARIES (JOBLIBS)

You can specify that a job library be added to the program library list to receive the output of the language processors by issuing a DDEF command defining that job library and containing the operand OPTION=JOBLIB. When this command is

---

[1]The TSS/360 linkage editor is an optional service program for connecting and editing object modules that have been assembled or compiled separately, and for moving object modules from one library to another. Refer to IBM System/360 Time Sharing System: Linkage Editor, GC28-2005, and to "Notes on Link-Editing PL/I Control Sections," later in this section.

executed, the name of that job library is added to the top of the program library list. That library then receives all subsequent module output of the language processors until another job library is defined (and is thus at the top of the list), until a RELEASE command is issued for that job library, until a JOBLIBS command moves another library to the top of the program library list, or until you log off. To be used in subsequent tasks, the job library must be redefined. A job library must always have a VP data set organization; it can be defined on a public or private volume.

Note: A job library can contain a mixture of PL/I-compiled members and members created in any other way, but unless a number of requirements are met, only PL/I modules can be executed together. See "Section 11: Interface between PL/I and Assembler-Language Programs."

The following types of job libraries are available:

- Private-volume job library
- Public-volume job library

Private-Volume Job Library

You can create a library for infrequently used modules by issuing a DDEF command for a cataloged job library that resides on a private removable disk pack. When using a private job library in a nonconversational task, you must request (with the SECURE command) a device for that job library. Modules can be entered in such a library:

- Automatically, if the library is the last one defined in the session.
- By link editing it from the user library, a session job library, or a public-volume job library, and specifying to the linkage editor the desired private-volume job library as the output destination.

Public-Volume Job Library

This type of library is useful for frequently used programs whose names and external symbols conflict with other programs in the user library. By defining it at the beginning of a task, as the only job library in the task, you can also use it to contain all modules compiled during the task. All job libraries residing on public volumes are automatically cataloged and can be shared among users.

To obtain a list of the job libraries for which you have issued DDEF commands in the current session, issue the command

```
DDNAME? JOBLIB=Y
```

The system responds by typing each job library's DDNAME and DSNAME, in the order that the job libraries appear on the program library list.

To facilitate the maintenance of programs within job libraries and the user library, the POD? command is available. POD? enables you to obtain on SYSOUT a list of the member names (and optionally the alias names and other data) of modules in USERLIB or a job library.

OTHER USER-DEFINED PROGRAM LIBRARIES

You can define a program library without making it a job library; simply issue a DDEF command for a VP data set, omitting OPTION=JOBLIB, and use that VP data set to store object modules. To get object modules into that program library from USERLIB or from a job library, use:

- A CDS command, or
- An LNK command with a linkage editor INCLUDE statement.

Such a library can be referred to by subsequent CDS commands or linkage editor INCLUDE statements. However, it is not in the program library list, and hence is not included in the linkage editor's automatic search and is not available to the dynamic loader. (The dynamic loader is a required service program for allocating virtual storage to user-selected object modules that reside in external storage.)

MULTIPLE VERSIONS OF OBJECT MODULES

If you have only one version of an object module and you want to replace it with a new version, simply modify the source data set with the desired changes and recompile the module, causing it to be placed in the same library. The old version disappears, and the new version takes its place.

But if you want to create another version of the same module and keep both versions, using the same module name or an identical entry name in each module, you must place each version in a different library.

This restriction occurs because a library cannot contain more than one declaration of any external-procedure entry name, nor can it contain two modules with the same name.

For examples of how to store two or more versions of the same module, see Example 2 of "Part III: Examples."

USER-ASSIGNED NAMES

Table 8 shows the restrictions on duplication of user-assigned external names. For a description of how these names are stored internally, see "External Names," later in this section.

The POD? command can be used to list external names in a library, thus assisting you to avoid duplication. You can always have the same name in different libraries.

Reserved Names

You can never assign an external name beginning with the characters SYS; names beginning with these letters are reserved for certain system programs. Any module starting with these symbols can never be retrieved from a user library or job library for execution, since resolution of SYS symbols for loading and running is always attempted from the system library. In addition, a diagnostic is issued if a module loaded by another name contains an external symbol beginning with SYS.

To avoid accidentally duplicating the names of IBM-supplied subprograms, do not use external symbols starting with the characters IHE or any PL/I library subroutine entry point name (that is, SIN, COS, etc.), unless you want to substitute for such a program one of your own.

PL/I CONTROL SECTIONS

A control section (CSECT) is the smallest relocatable part of an object module; the PL/I compiler subdivides each object module into a standard set of CSECTs. Certain CSECTs, especially those frequently used, will require less storage space, execution time, or compilation time if you help the system manage them.

TYPES OF PL/I CONTROL SECTIONS

INITIALIZATION CSECT: This CSECT is entered when the external procedure is called by module name. Initialization is required for all PL/I programs. See "PL/I Subroutines Called from Assembler Programs," in Section 12.

For subroutine procedures, the initialization CSECT issues an error message; only the main procedure should be called by module name.

STATIC INTERNAL CSECT: This CSECT contains parameter lists, save areas, and any variables declared static internal.

TEXT CSECT: This CSECT contains executable code that is never modified by the program.

FILE CSECT: This CSECT contains the file declare block, which is never modified by executable code. A file CSECT is generated for each file.

STATIC EXTERNAL CSECT: A static external CSECT is generated for each variable declared static external. This CSECT has the COMMON attribute (see Linkage Editor) if it is declared without the INITIAL attribute, and if it contains no string items.

Table 8. Restrictions on Assigning External Names

Type of Name	Program*	Program Library
module name	Must be specified in PLI command; must not be declared; must not duplicate other external names.	No duplicates allowed.
name of external procedure	Must be declared; must not duplicate other external names.	
ENTRY name of external procedure	If declared, must not duplicate other external names.	
file name or name of static external variable	If declared, other procedures can have duplicate declarations. Attributes in duplicate declarations must be identical.	Duplicates allowed.

*Restrictions for the object module are the same as for the program.

LINK-EDITING

Why Link-Edit?

In TSS/360, use of the linkage editor is optional. If the linkage editor has not linked a calling object module to a called object module, the dynamic loader links them automatically, when it loads the calling object module for execution. However, you may want to use the linkage editor for any of these purposes:

- Link the output of separate compilations into one object module. If one of the included modules references another implicitly (that is, not explicitly), this pre-execution linkage reduces the time it will take to load and execute the program.
- Combine the CSECTs of an object module into a single CSECT, thus reducing the amount of virtual and external storage required. Normally, the CSECTs are combined when they are created, since the IBM-supplied value for PLIPACK is P. See "Packing," later in this section.
- Delete, substitute, or rename CSECTs and delete or rename ENTRY names without recompiling.
- Change the attributes of CSECTs. See "Sharing," later in this section.
- Move object modules from one program library to another.
- Prepare a list of unresolved external references, distinguishing those that will be resolved out of SYSLIB from those that will be resolved out of the JOBLIBs and USERLIB in use at execution time.
- Prepare a listing of an object module's program module dictionary.

IBM System/360 Time Sharing System: Linkage Editor, GC28-2005, describes the program module dictionary and explains how to use the linkage editor.

External Names

For some uses of the linkage editor, you must know what the external names of a module are, and what CSECTs they are in. External names that you assign are the names of the module, the external procedure, ENTRY statements in the external procedure, files, and static external variables. External names that the system assigns are the names of the initialization CSECT and the static internal CSECT. The module name and external ENTRY-statement names are the only external names that are not also CSECT names.

MODULE NAME: The module name is specified by the NAME, MERGELST, or MERGEDS operand of the PLI command. This name always qualified the name of a load data set -- that is, LOAD.name(0).

Note: Load data sets cannot be link-edited.

INITIALIZATION CSECT: ODC generates the name of the initialization CSECT by padding the procedure name on the right with percent signs (%) to eight characters.

STATIC INTERNAL CSECT: The PL/I compiler generates the name of the static internal CSECT by adding an A to the right of the procedure name and padding on the left with asterisks (*) to eight characters.

TEXT CSECT: The name of the text CSECT is the procedure name.

FILE CSECT: The name of the file CSECT is the file name.

STATIC EXTERNAL CSECT: The name of the static external CSECT is the name specified in the declaration of the static external variable.

NAMES OF ENTRY STATEMENTS: Names of ENTRY statements are kept in the text CSECT.

Names of the text CSECT, file CSECT, static external CSECT, and ENTRY statements are truncated to seven characters by the PL/I compiler. The truncation consists of using the first four and last three letters of the name. Names of seven characters or less are not truncated.

Rules for Link-Editing PL/I Modules

The following rules have special importance in relation to PL/I modules. Some of these rules, and other required information for linkage-editor users, are given in IBM System/360 Time Sharing System: Linkage Editor, GC28-2005.

- If you link a module containing a main procedure with modules containing subroutine procedures, you should link the module containing the main procedure first (in a linkage-editor INCLUDE statement) so that the main entry point of the output module will be to the main procedure.
- Within an object module, external names must be unique. After a PL/I module is link-edited, its external names are known to the linkage editor; any attempt to include external names that

duplicate external names already in the output module is rejected. Therefore, all references to a specific file name or static-external-variable name are resolved to the first file CSECT or static external CSECT that is link-edited for that name.

- Some CSECTs should not be given a PUBLIC or READONLY attribute. See "Attributes of Shared CSECTs," later in this section.
- Dynamic calls are not known to the linkage editor, and all dynamic linkage is completed by the dynamic loader, during execution.
- Load data sets cannot be link-edited.
- If a PL/I module is linked to a non-PL/I module, the non-PL/I module must be adjusted to the PL/I environment. See "Section 12: Interface between PL/I and Assembler-Language Programs."
- Packed CSECTs appear as a single CSECT to the linkage editor. See "Packing," later in this section.

SHARING

It is possible for PL/I CSECTs to be shared among two or more TSS/360 users. The installation must provide for the loading of preselected PL/I CSECTs into initial virtual storage (initial virtual memory, or IVM) when the system is generated. The CSECTs to be loaded must be specified by means of an LLIST macro instruction or DC instructions at system-generation time. IBM System/360 Time Sharing System: System Programmer's Guide, GC28-2008, tells how to specify the CSECTs to be loaded, and IBM System/360 Time Sharing System: System Generation and Maintenance, GC28-2010, describes the overall process of loading CSECTs into IVM.

Note: The SHARE and PERMIT commands have no application to PL/I object modules.

Linkage Involving Shared CSECTs

- If a CSECT in IVM calls a CSECT not in IVM, the call must be dynamic. Refer to the description of the PLI command's EXPLICIT and XFERDS operands, in Section 5.
- If a CSECT in IVM calls a CSECT in IVM, the call must not be dynamic. Since the compiler generates explicit calls to PL/I library modules, PL/I library modules must be placed in IVM.
- If a control section not in IVM calls a control section in IVM, the call can be dynamic or static.
- A program contained wholly or partially in IVM is still limited to a maximum of 4096 bytes for PR-type ESD entries and may require dynamic calls if it has many hundreds of subroutines. See the description of the EXPLICIT and XFERDS operands, in Section 5.

Attributes of Shared CSECTs

- Public CSECTs -- that is, PL/I CSECTs that are link-edited to give them the PUBLIC attribute -- must be executed (and loaded) in IVM.
- All sharers of a public CSECT in IVM reference the same copy of that CSECT.
- If a CSECT in IVM is not public, all sharers reference separate copies.
- Link-editing a CSECT to give it the READONLY attribute ensures that the shared code will not modify itself during execution. If a public CSECT is not read-only, it is the responsibility of each user to ensure the integrity of the CSECT at any stage of execution, preventing mutual interference.
- You can conserve external paging storage by giving the PUBLIC and READONLY attributes to CSECTs that are to be shared.
- Text CSECTs can always be given the READONLY attribute. Initialization, static external, and file CSECTs can be made read-only if the external procedure was coded with the REENTRANT option. A static internal CSECT can be made read-only if (1) the procedure does not contain an assignment statement that assigns a value to a static internal variable, and (2) the external procedure was coded with the REENTRANT option.
- If a PL/I CSECT to be stored in IVM can be made read-only, it can be made public.

PACKING

Without CSECT packing, at least one page (4096 bytes) is assigned to each CSECT. A CSECT may require more than one page; however, many PL/I CSECTs require much less than one page. A file CSECT, for example, is 56 bytes long. If an entire page is assigned to a 56-byte CSECT, this results in less than 2% storage utilization.

When you compile, ODC checks the value of PLIPACK, in your user profile. If PLIPACK=N, one or more pages are assigned to each CSECT. If PLIPACK=Y, CSECTs are packed. Packing consists of combining CSECTs into contiguous storage, retaining doubleword boundaries for CSECT origins. The name of the initialization CSECT is retained as a CSECT name, and other CSECT names are transformed into entry-point names. In effect, the CSECTs are combined into a single CSECT. If PLIPACK=P, ODC packs all CSECTs except static external CSECTs that have the TSS/360 COMMON attribute or are more than 4096 bytes long. This is generally more efficient than PLIPACK=Y, since COMMON CSECTs are null CSECTs and they are mapped onto external storage only if they are packed. The IBM-supplied default for PLIPACK is P.

Notes:

- The fifth operand of the LOGON command specifies whether all CSECTs are to be packed at execution time. PLIPACK specifies whether PL/I CSECTs are to be always packed -- on external storage, as well as at execution time. Although PLIPACK requires additional compilation time, it saves external storage and avoids the overhead of packing during the loading process.

- CSECTs that are not already packed are automatically packed when they are loaded into IVM.

- REJMSG, another value in your user profile, controls the dynamic loader's output of duplicate-name messages. The command DEFAULT REJMSG=Y causes the dynamic loader to not issue such a message whenever it encounters a duplicate CSECT or entry-point name, and rejects a CSECT or entry point, during the loading process. If REJMSG=N, duplicate-name messages are issued. (There may be many of these messages if your CSECTs are packed.) The IBM-supplied default for REJMSG is N.

INVOKING THE MODULE

A PL/I module invoked from the command mode must be invoked by its module name (that is, the name in the NAME operand of the PLI command). A PL/I module called from a PL/I procedure must be invoked by its procedure name (that is, the name in the PROCEDURE statement).

If the invoked procedure expects to receive parameters, these parameters can be passed following the name of invocation.

For information on how to pass parameters in a PL/I CALL statement, see PL/I Language Reference Manual. For information on how to pass parameters from the command stream, see Command System User's Guide.

If the invoked module has close to a thousand subroutines, the dynamic loader may not be able to load all the program at one time. In this case, some or all of the subroutine calls must be via a transfer module, which calls the subroutines dynamically; in addition, the transfer module should selectively unload subroutines whenever necessary to make room for the dynamically called subroutines. See the discussions of the EXPLICIT and XFERDS operands, in Section 5, and Example 18 in Part III.

RECOVERING FROM ERRORS WHEN DYNAMICALLY LOADING

The dynamic loader notes all of the external calls in a module that is explicitly loaded or invoked and resolves them by searching the program library list. While the loader is linking the object modules into virtual storage, diagnostic messages may be issued indicating error conditions that can affect the eventual execution of the program.

- Name to be loaded or run not found in library - While in command mode, you either specified the wrong module name or forgot to define the job library containing the object module. In the latter case, if you are operating conversationally, you can enter the DDEF command defining the job library and reissue the LOAD command or module name.

- Unresolved references - You executed an object module that refers to a subroutine that cannot be located in any of the libraries in the program library list. A diagnostic message is issued specifying the name that was used in the reference. Further linking of other object modules is not suspended, however, so that the main program and possibly other subprograms are placed in virtual storage.

- Duplicate names - The dynamic loader does not load control sections with duplicate names; only the first one encountered is loaded. If a call by module name, a call by procedure name, or an attempt by the dynamic loader to satisfy an external call results in an attempt to load a module containing the second occurrence of an external name, the dynamic loader rejects the CSECT with the duplicate name and issues a

diagnostic message. (Exception: If there is only a duplicate ENTRY name, the control section is loaded but the duplicate ENTRY name is removed.) Therefore, (1) Static external variables should not have conflicting initial values; only the first initial value encountered is loaded. (2) Only one file control section is loaded for each file; conflicting attributes will arise when a file is opened for INPUT but the file control section is for an OUTPUT file, etc.

Note: Ordinarily, a program that is called by simply issuing the name of the main module (rather than issuing a CALL command or LOAD and GO) is not automatically unloaded after execution. The UNLOAD command can be used to unload it from virtual storage.

If you want to execute the program regardless of the error, retype the module name. You must, however, repeat the name originally specified. This is necessary to define the point at which execution is to be initiated.

If you anticipate that an object module will have unresolved references, first issue a LOAD command naming the module; then issue the module name itself. This method is recommended for a nonconversational task, since execution is initiated regardless of unresolved references.

If you do not want to execute the version of the program that has been put into storage, issue an UNLOAD command. (This answers that you will not attempt to load modules with duplicate names.) You can then issue a DDEF command defining the job library that was missed in the first LOAD attempt; if the job library is already defined, but not at the top of the program library list, you can put it there by issuing the appropriate RELEASE or JOBLIBS command. A LOAD command or module name issued at this point causes the entire linking process to be redone.

TSS/360 PL/I contains facilities for writing programs that interact with the terminal user. There are two classes of interactive statements: the DISPLAY statement and STREAM I/O statements.

DISPLAY

The DISPLAY statement is the simplest type of terminal I/O, and is ideally suited to character-string transmission, although it can do some data conversion on output. A disadvantage is that it lacks formatting facilities.

A message can be displayed in either of two forms:

1. Without the REPLY option:

```
DISPLAY(element-expression);
```

2. With the REPLY option:

```
DISPLAY(element-expression)
REPLY(character-variable)
[EVENT(event-variable)];
```

General rules:

- Execution of the DISPLAY statement causes the element expression to be evaluated and, where necessary, converted to a varying character string of maximum length 126 characters. This character string is printed at the terminal, starting at the beginning of a new line.
- There is also a return to the next line after the message has been printed, unless the last character of the message is a colon. In that case, the colon is not printed, but causes the print head to stay where it is, positioned ready to print immediately after the displayed message.
- Thus successive DISPLAY messages are double-spaced unless they end with colons, in which case they are single-spaced.
- The character variable, specified in the REPLY option, receives a message that you enter; that is, a string of up to 126 characters.
- Without the REPLY option, execution continues uninterrupted. If REPLY is specified, execution is suspended until the REPLY character string is received.
- If the EVENT (event-variable) option is given, the completion part of the event variable is not set to '1'B until a WAIT statement naming the event is executed.
- You start typing the REPLY character string wherever the last terminal operation has left the print head. Terminate the reply by pressing RETURN; the RETURN is not part of the reply. If you want to continue the reply on the next line, type a hyphen and press RETURN; neither character is part of the reply. To cancel the reply and start again, press # followed by RETURN.

Examples:

```
DISPLAY('THIS MESSAGE AND RETURN');

DISPLAY('DON'T RETURN:');

DISPLAY(CHARVAR);

DISPLAY('N HAS VALUE '|| N);

DISPLAY('ENTER NAME:')REPLY(CHARVAR);

DISPLAY(MSGVAR)REPLY(ANSVAR)EVENT
(EVTVAR);
```

STREAM I/O

All three forms of STREAM I/O (that is, data-directed, list-directed, and edit-directed) have the same PL/I language facilities for both terminal and non-terminal I/O; even rules regarding delimiters are the same. Although the same terminal can be used for both input and output, there are at least two separate, unconnected files involved.

INPUT USING 'GET'

Input is requested from your terminal on execution of a conversational mode GET statement that refers to SYSIN. The reference to SYSIN can happen in three ways:

1. The file name, omitted from the GET statement, becomes SYSIN by default (this causes a warning message during compilation):

```
GET LIST(A,B,C);
```

2. The file name SYSIN is used explicitly:

```
GET FILE(SYSIN) LIST(A,B,C);
```

3. A file name other than SYSIN is used, but no DDEF command is issued for that file prior to execution.

Prompting Action

When a GET statement is executed, the buffer is examined to see whether the request can be satisfied from data already read in. If it can, the buffer pointer is simply advanced, and execution continues without external data input.

If there is no data in the buffer, or not enough to satisfy the GET, then a prompting signal (a carriage return followed by a colon) is sent to the terminal and the keyboard is unlocked. At this point, you should enter the data required for the GET. If you press RETURN without having entered enough to satisfy the GET, then the system prompts again. (This allows you to enter each item on a new line, if you want.)

You need not stop after providing enough input for the GET. If you know what data will be required next, you can enter it ahead of time, and thus "prime" the buffer. As with DISPLAY REPLY, if you want to continue on another line, you must enter a hyphen and press RETURN; to cancel a line and restart, enter # and press RETURN. By using successive continuations, you can enter up to 32,760 characters for one prompt.

Note: There is no connection between STREAM I/O and DISPLAY. For example, it is impossible to use excessive STREAM input for a GET to satisfy a subsequent DISPLAY REPLY.

SKIP Option

As implied above, there is no direct correspondence between execution of GET statements and prompting for input. You can obtain synchronization by using the SKIP option alone; e.g.:

```
GET SKIP;
```

or in an input request; e.g.:

```
GET LIST(A,B,C)SKIP;
```

The SKIP option is executed first. The system ignores anything already in the input buffer and points to the beginning of a new buffer. It then executes the GET statement by prompting for input; any data entered at this point will be used by the next GET to be executed. If there is no data in the current buffer, the system prompts for the data that is to be ignored and prompts again when it has gotten a new buffer.

It does not make sense to use successive SKIP options in conversational mode, because the first SKIP causes prompting for data that will be discarded by the second.

COPY Option

The COPY option causes any assigned input data to be written on SYSOUT exactly as it was entered.

- Only assigned data is copied, not input that is skipped.
- Delimiters are included.
- Each item is written on a new line.

Delimiters

DATA-DIRECTED INPUT: Each item but the last must be followed by a blank, a comma, or a carriage return. If one or more of the items in the data list is omitted from the input list, the last item in the input list must be followed by a semicolon. Example:

```
GET DATA(A,B,C,D);
   .
   .
   .
A = 32,B = 'MESSAGE', C=0.001;
```

LIST-DIRECTED INPUT: Each item including the last must be followed by a blank, a comma, or a carriage return. Example:

```
32, 'MESSAGE' 0.001
```

EDIT-DIRECTED INPUT: In general, there are no input delimiters, because the layout of the data is defined by a format list. However, a carriage return delimits an incompletely entered item:

- If the target item is a varying string, the input is transmitted as is; no extra blanks are inserted.
- If the target item is not a varying string, the input is padded on the right with blanks to give it the necessary field width.

End-of-File

When you are prompted at the beginning of a GET operation, you can indicate end-of-file by pressing RETURN, thus entering a null line. This causes the ENDFILE condi-

tion to be raised; if your program contains an ON ENDFILE(SYSIN) statement, that statement is executed.

End-of-file can take effect when a SKIP is being executed. Once end-of-file is recognized, it remains effective until the file is closed.

OUTPUT USING 'PUT'

When a PUT statement that refers to the file SYSOUT is executed in conversational mode, output is sent to the terminal. The reference to SYSOUT can happen in these ways:

1. The file name, omitted from the PUT statement, becomes SYSOUT by default. Example:

   ```
   PUT LIST(A,B,C);
   ```

2. The file name SYSOUT is used explicitly:

   ```
   PUT FILE(SYSOUT) LIST(A,B,C);
   ```

3. The file name SYSPRINT is used explicitly:

   ```
   PUT FILE(SYSPRINT) LIST(A,B,C);
   ```

4. A file name other than SYSOUT or SYSPRINT is used, but no DDEF command is issued for the file prior to execution. To be compatible with SYSOUT, the named file should have the PRINT attribute.

Use of a data list with a PUT DATA statement is optional. Execution of a PUT DATA statement with a data list causes the system to write each specified variable and its value on SYSOUT, in assignment-statement form. Execution without a data list causes the system to write the contents of each variable in your program. In conversational mode, such a storage dump may be impractical, since the terminal does not function as a high-speed printer. However, you can interrupt the printout and return to the command mode by pressing the attention key.

Buffering

The PUT operation uses buffers in the form of output lines. The default line size is 120 characters, but you can change this up to a maximum of 130 characters by using an OPEN statement with the LINESIZE option. Example:

```
OPEN FILE(SYSOUT)LINESIZE(130)OUTPUT;
```

Operation of the PUT Statement

Execution of a PUT statement causes associated data to appear at the terminal. The print head does not return to start a new line, but remains positioned immediately after the data just presented. In the absence of control items, successive PUTs fill up the current line buffer, and then begin a new line. This allows you to see the results of each PUT without having to wait for the whole buffer to fill.

There is sometimes an extra line of spacing when a PUT operation synchronizes with the beginning of a new line (for example, when you use the SKIP option, explained below). A RETURN is sent to the terminal preceding the data for the current PUT; however, if the last operation was a DISPLAY or input for a GET or REPLY, the print head is already on a new line, and the effect is two RETURNs.

Note: If you use GET and PUT statements interchangeably, positioning of the print head will also be affected by execution of GET statements; in addition, the terminal sheet will show a mixture of SYSIN and SYSOUT. SYSOUT doesn't know what SYSIN is doing.

Print Control Options

Three options of the PUT statement control the spacing of lines on a printed page:

```
PAGE
LINE(expression)
SKIP(expression)
```

They all take effect before any data is sent to the terminal.

General rules:

- PAGE and LINE are intended primarily for printer output and are not recommended for terminal output, since a terminal has no way of identifying physical page boundaries. If they are used on the terminal, for example in conversationally checking out a program intended to be run nonconversationally, their effects are restricted. Both PAGE and LINE cause three RETURNs. Even if both PAGE and LINE are used in the same PUT statement, the effect is still three lines of spacing. If an expression is associated with the LINE option, it is ignored.

- SKIP is used to cause line spacing. The expression is evaluated to an integer; if there is no expression, the integer 1 is assumed. If the integer is 1, single spacing occurs; that is,

the print head returns to the beginning of the next line. If the integer is 2, double spacing results. If it is 3 or more, triple spacing results. If the integer is negative or zero,

1. For nonconversational output, the print head is returned to the beginning of the current line. This allows overprinting of that line -- useful for underscoring, obliterating confidential data, slashing zeros, drawing pictures, etc.

2. For conversational output, the SKIP option is ignored.

Format Items

For data- and list-directed output, there are no format items; the SKIP option affords the only easy print control. In edit-directed output, however, you can exercise minute control over the appearance of data at the terminal by using format items in format lists.

General rules:

- PAGE, LINE, and SKIP operate the same way in format lists and as options, except that as options they operate before the system types any data. All format items are processed sequentially, so that line spacing can occur between items. However, it is not possible to have line spacing after the last data item, because once the data list is exhausted any other items in the format list are ignored. Thus,

```
PUT
EDIT(A,B,C)(A(6),F(8.2),A(2),SKIP);
```

  will not return to a new line after typing A, B, and C. To achieve that result, you might:

  1. Follow the above statement with a PUT SKIP; statement.

  2. Include a SKIP option in the next PUT statement that transmits data.

  3. Include a SKIP as the first format item if the next PUT is edit-directed.

- The X(expression) format item inserts blanks between data items. The expression is evaluated as an integer. If the integer is negative or zero, the item has no effect; otherwise, the specified number of blanks is typed by the system. If there are enough blanks to cause overflow of the line buffer, a SKIP to the next line results.

- COLUMN(expression) makes it unnecessary to calculate the number of blanks required. The expression is evaluated as an integer, and the system types enough blanks to bring the print head to the specified character position in the current line. If the current print position is ahead of the specified one, a SKIP is made to the next line, and blanks are typed to bring the print head to the required displacement from the beginning of the line. If the integer is negative, zero, or greater than the line size, position 1 (the beginning of the line) is assumed; this causes a SKIP to a new line, unless a SKIP was just done.

- The format items A, B, C, E, F, and P, which relate to the external representation of internal data, and R, which specifies a remote format list, are not limited to conversational I/O. See the PL/I Language Reference Manual for detailed descriptions.

Layout of Data- and List-Directed Output

Data items are automatically aligned on preset tab positions. PL/I has the positions 1, 25, 49, 73, 97, and 121; if you want, you can change these tab settings by following the instructions given under "Tab Control Table," in Section 8.

This section explains how data sets vary in:

- Location (see "Storing and Manipulating Data Sets")
- Availability to nonowners (see "Protecting and Sharing Data Sets")
- Record format (see "Record Formats")
- Overall organization (see "Data Set Organizations")
- The way they are handled by the PL/I library subroutines (see "Types of PL/I Data Transmission").

This section also discusses the DDEF command, which identifies data sets and describes them to the system.

STORING AND MANIPULATING DATA SETS

VOLUMES

A data set resides on one or more volumes. A volume is a standard unit of external storage that can be written on or read by an I/O device (for example, a reel of tape or a disk pack); a unique serial number identifies each volume.

A magnetic-tape or direct access volume can contain more than one data set; conversely, a single data set can span two or more such volumes.

Some direct access volumes are public, meaning that they are permanently mounted while the system is running, and they can be accessed by all users. Some direct access volumes, and all magnetic-tape volumes, are private. This means that they are not mounted on the system until needed, they are dismounted when no longer needed, and they can be used by only one user at a time.

Volume Allocation

The system assumes that you want storage on a public volume unless you specifically ask for storage on a private volume by specifying VOLUME=PRIVATE in the DDEF command. (See Appendix D.) When it is necessary to retain the data set in the system, it is more convenient to store it on a public volume. Public volumes are automatically available for allocation to your task, within the limits of public allocation established by your installation.

If you use private volumes, you may need to wait for device availability; in any case, you must wait for the operator to mount the volume on the device. Each time a request is made for a device on which to mount a private volume, the system must determine whether or not it can honor the request, based on current requirements throughout the system for that type of device. If the system cannot allocate a private device to your task, one of two actions occurs, depending upon the operational mode:

- In a conversational task, if a device is not available, you are asked to either wait for an available device or cancel the DDEF command. If your device ration is exceeded or a specified device cannot be found, the system cancels the DDEF command, returns control to the terminal, and awaits another command.
- A nonconversational task is either terminated by the operator or queued until the required private devices are mounted. You must include a SECURE command to reserve all devices required for private volumes during the execution of a nonconversational task. Only one SECURE command is allowed for each task. It is recommended that the SECURE command appear immediately after the LOGON command. The devices specified for private volumes are reserved so that the task can be executed without waiting for I/O devices; any waiting that may be necessary to reserve the devices occurs at SECURE time rather than during execution time. The SECURE command is never used in a conversational task; it is mandatory only in nonconversational tasks that include references to private volumes.

SYSTEM CATALOG

The cataloging facility of TSS/360 aids you in referring to data sets by their names alone, without specifying their physical locations. Since it contains your data-set-naming structure, the system catalog is an index, like the catalogs used in libraries, that points to items residing elsewhere; see Figure 6 for a simplified view. Altogether, the system catalog records:

- Where the data set is physically located -- the catalog associates its name with the serial numbers of its volumes.
- Who can use the data set.
- How the data set can be used -- read only, read and write, or unlimited access.

The structure of the catalog protects your data from being read or written into by other users, except those that you specifically permit to share the data.

Figures 7 and 8 show more details of the system catalog:

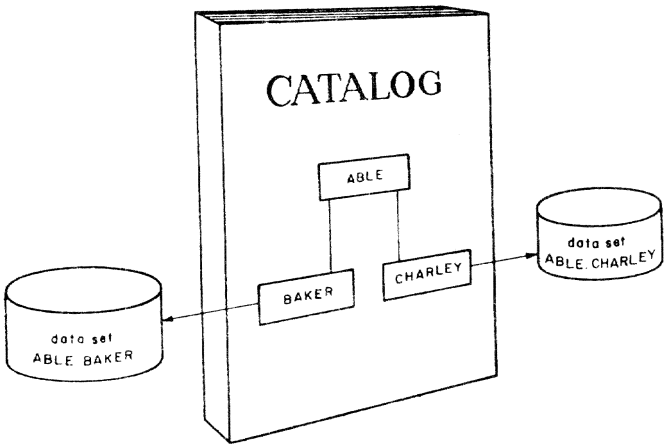


Figure 6. System Catalog

- The system catalog consists of a master index and sets of subordinate catalog entries. It is, in effect, a collection of separate catalogs. The system has its own catalog and each user has his own catalog.
- Each catalog is an index of the data sets associated with it.

When the system was generated at your installation, catalog entries were created for all system data sets, including SYSLIB, which contains the system routines that are loaded on demand -- for example, the PL/I compiler.

When the system manager or administrator joins you to the system, your user identification is placed in the master index and you are given your own user catalog. When you log on for the first time, special entry is created in your catalog for a data set called USERLIB. USERLIB is your own private library for object programs.

Except for USERLIB, you control all entries in your catalog by the way you name data sets and by the way you use the cataloging and uncataloging facilities of the system. Some of these facilities are for entering, removing, and renaming catalog entries. Others are for indicating which data sets can be shared by others and to what extent. The key points are:

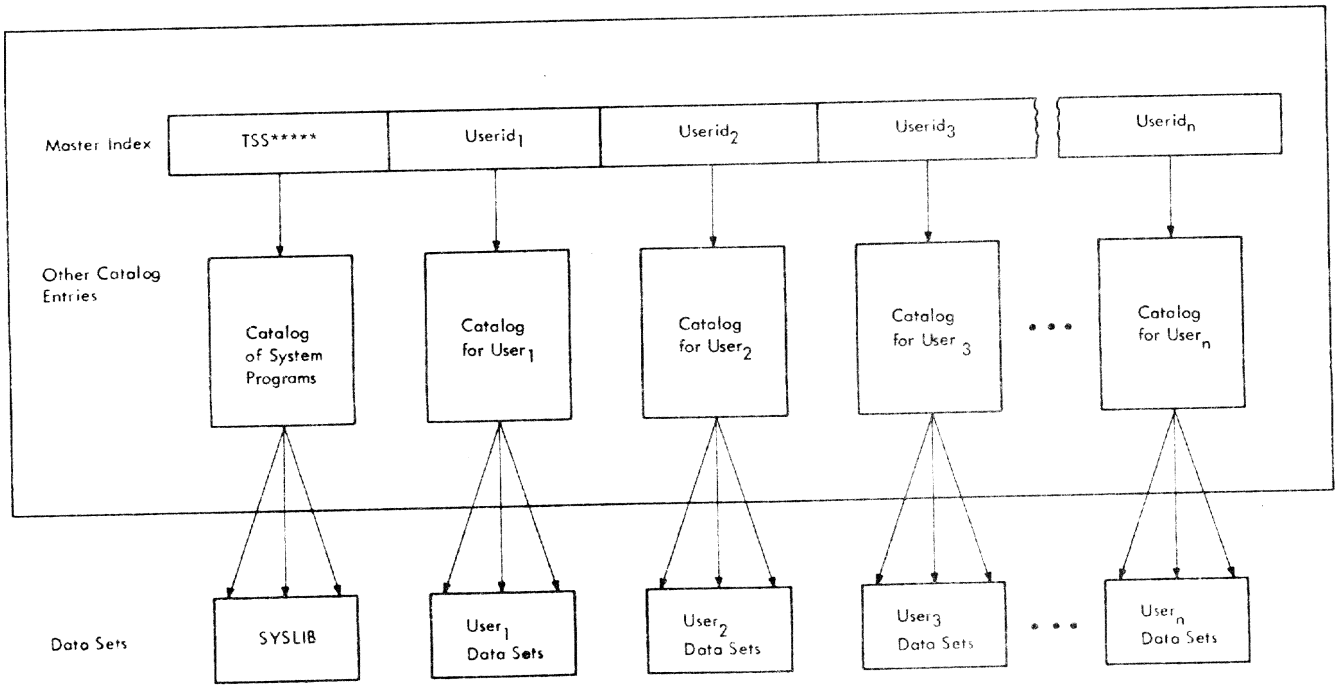


Figure 7. Catalog Organization

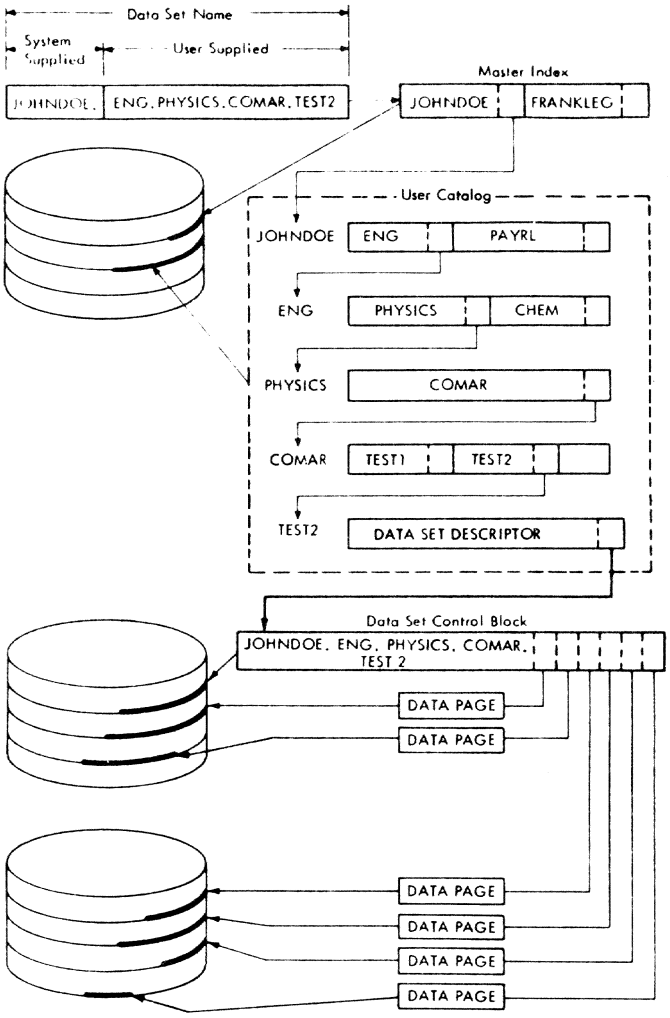


Figure 8. Locating a Data Set

- Your catalog exists in the system from the time you are joined until the time your access privilege is withdrawn.

- Data sets on public volumes are automatically cataloged for you; thus, they are available from session to session.

- You can share your programs and data with others, if you want.

Generation Data Groups

The cataloging facilities of TSS/360 provide an option that assigns numbers to individual data sets in a sequentially ordered collection, thereby allowing you to catalog the entire collection under a single name. You can use the numbers to distinguish among successive data sets in the collection without assigning a new name to each data set. Because each data set is normally an update of the data set created on the previous run, the new data set is called a generation, and the number associated with it is called a generation number. The entire structure of data sets of the same name is called a generation data group (GDG).

Each data set in a generation data group has the same name qualified by a unique parenthesized generation number (for example, STOCK(0), STOCK(-1,) STOCK(-2). The most recently cataloged data set is generation 0, and the preceding generations are -1, -2, and so on. You specify the number of generations to be saved when you establish the generation data group.

For example, consider a generation data group that contains a series of data sets used for weather reporting and forecasting; the name of the group is WEATHER. The generations for the group (assuming that three generations are to be saved) are:

```
WEATHER (0)

WEATHER (-1)

WEATHER (-2)
```

(The numbers in parentheses are relative generation numbers. You could also use absolute generation numbers:

```
WEATHER.G0002V00

WEATHER.G0001V00

WEATHER.G0000V00
```

where WEATHER G0000V00 is the first version recorded.)

When WEATHER is updated, the new data set is specified as WEATHER (+1). When the new data set is cataloged, the system changes the name WEATHER (+1) to WEATHER (0), WEATHER (0) to WEATHER (-1), the former WEATHER (-1) to WEATHER (-2), and deletes the former WEATHER (-2).

To establish a data set as a generation data group, you must catalog it using the GDG form of the CATALOG command, before any DDEF command is issued for it. For an example of how to catalog a generation data group, refer to the description of the CATALOG command in Command System User's Guide.

Catalog Maintenance

If you want to refer to a data set without keeping track of its physical location, or if you want to share the data set with others, you must catalog it. In addition, many TSS/360 commands that relate to

data sets require that the data sets be cataloged.

Data sets that are to be cataloged must reside on one or more direct access or magnetic tape volumes. Data sets on either public or private volumes can be cataloged.

Most TSS/360 data sets are cataloged automatically when they are created. The CATALOG command need only be used to:

- Catalog a data set formatted for magnetic tape (known as a physical sequential data set).
- Alter the entry of a previously cataloged data set; for example, change the catalog index structure for a renamed data set or change the version number of a generation data group member.
- Structure the catalog for an entire generation data group. You can indicate the number of generations to be retained, as well as the disposition of old generations when the specified number of retentions is exceeded.
- Catalog a data set as a new generation of an existing generation data group.

Note: Catalog control of the generations of a generation data group can be exercised only by the owner of the generation data group. (Refer to "Protecting and Sharing Data Sets," in this section).

The EVV (enter VAM volumes) command is used to catalog existing data sets that (1) reside on private volumes, and (2) are not physical sequential.

You can use the DELETE command to remove a catalog entry for a data set if:

1. You want to remove the catalog entry of a data set from the catalog but not erase it, and the data set resides on a private volume.
2. You want to remove the catalog entry of someone else's data set that you are sharing (because you no longer have a need to share that data set).

The ERASE command can also be used for uncataloging. ERASE removes the catalog entry, and erases the data set if it resides on a direct access volume. (Erasing means making the storage space of the data set available for other use.)

So that you can specify whether you want to be given one data set name at a time when you enter a partially qualified name, or no name at all, as the operand of either the ERASE or DELETE command, provision is made to set the value of DEPROMPT (a value contained in your user profile)[1] to either Y (yes) or N (no). If the value is set to Y, you are given one data set name at a time for disposition. If the value is set to N, all data sets grouped under this partially qualified data set name are erased or deleted without individual presentation. If you specify a fully qualified name, the data set is erased or deleted no matter what was specified for DEPROMPT.

Note: When deleting a shared data set, you must specify the fully qualified data set name; you will not be prompted for individual data sets under a partially qualified name.

You have the option in certain commands, as PRINT and PUNCH, if a cataloged data set is involved, of specifying whether it is to be erased or not after the output operation.

PLANNING I/O

The simplest way to handle TSS/360 I/O is to read input data from the terminal and write output data to the terminal. However, for data sets that are not small, it is more efficient to use external storage as follows:

1. Prior to program execution, store input data in the system on a direct access volume. If the data is the output of a previously executed program, you can simply write it on a direct access volume during that program; the system automatically catalogs it to retain it for subsequent use. The data could also be created using a conversational or nonconversational EDIT or DATA command. In addition, you can prestore the data using operator procedures, involving your card input deck or magnetic tape volume.
2. During program execution, read input data from the direct access volume on which you stored it; write output data to a direct access volume (for offline output, following execution). You also have facilities for I/O from and to tape devices. However, in most

---

[1]The system maintains a special data set called a user profile, which contains information about the user. When you log on, the prototype user profile in the system library (SYSLIB) is copied into your virtual storage where it resides during the task. The values in this copy of the user profile can be altered by the DEFAULT, SET, and SYNONYM commands.

TSS/360 installations, no problem program communication with unit record devices (card reader/punches and printers) is possible during execution.

3. Following execution, you can print out or punch on cards the program output you stored on a direct access device, using the PRINT and PUNCH commands. You could also produce a magnetic tape for subsequent printing by issuing a WT (write tape) command.

Since you can communicate with your programs during their execution, you can design programs that mix external-storage I/O with terminal I/O. For example, a program can read input from the terminal and write output to a direct access volume. Or a program can be designed so that when predetermined events occur, intermediate results are printed at your terminal. You can then decide how you want to proceed: supply additional or different data at that time; change the sequence of program execution; stop the program; or examine key final results prior to initiating their final printout.

COPYING, MODIFYING, AND ERASING DATA SETS

The CDS command copies any existing data set to which you have access. You can also use it to renumber the lines of a line data set as it is copied. The original data set must be defined in your task or cataloged. The VV, VT, and TV commands copy data sets formatted for interface with the TSS/360 virtual access method (VAM) data management routines. The VV command copies a VAM data set (or program library) in direct access storage. The VT command copies a VAM data set to nine-track magnetic tape as a physical sequential data set; used with the TV command, VT allows you to store VAM data sets on magnetic tape and retrieve them at a later time. The TV command retrieves and writes onto a direct access volume a data set previously written on magnetic tape by the VT command.

The MODIFY command inserts, deletes, replaces, or inspects records of a VAM data set that is indexed by keys (for example, line numbers). You must identify the record to be modified, by its key. You can review corrected lines for confirmation of your changes.

You can use the ERASE command to erase data sets that you own. See "Catalog Maintenance," earlier in this section.

If you are sharing someone else's data set, you can remove its entry from your catalog by issuing the DELETE command. See "Catalog Maintenance," earlier in this section.

See Command System User's Guide for a complete list of rules concerning the above commands.

PROTECTING AND SHARING DATA SETS

You cannot access a data set you don't own unless you have system authorization to do so, or unless the owner of the data set has permitted you to share it.

A shared data set is cataloged and the owner has issued a PERMIT command for it. It belongs to one user, but can be shared with other users in any of the following ways:

1. Read-only access: The sharer can read the data set, but cannot change it in any way.

2. Read-and-write access: The sharer can both read and write to the data set, but he cannot erase it.

3. Unlimited access: The sharer can treat the data set as his own; he can even erase it.

A PERMIT command designates which data sets are to be shared, the users who can share them, and the level of access those users have. You can also use the PERMIT command to withdraw from previously authorized sharers the right to continue sharing your data. Each time you issue a PERMIT command, information on who can share which of your data sets is updated in your catalog.

If you have been named in another user's PERMIT command, you must issue a SHARE command before you can actually access the data sets he has authorized you to use. To see how this command is used, assume that a sharer's user identification is JONES and that he has been permitted to share one data set. The data set is owned by user SMITH, and is cataloged under the fully qualified name ENG.PHYSICS.COMAR.TEST. Assume also that the sharer wants to name the data set ENG.CHEM.NOTAR.TEST1. He would then issue the SHARE command shown at the top of Figure 9. In response to that command, the system would search the owner's catalog to see if the prospective sharer is authorized. If he is not, the system issues a diagnostic; if he is authorized, the system places the owner's (complete) name for the data set in the sharer's catalog with a pointer back to the master index. Whenever the sharer subsequently refers to the data set by the name he gave, the system locates the data set by the search procedure shown in Figure 9.

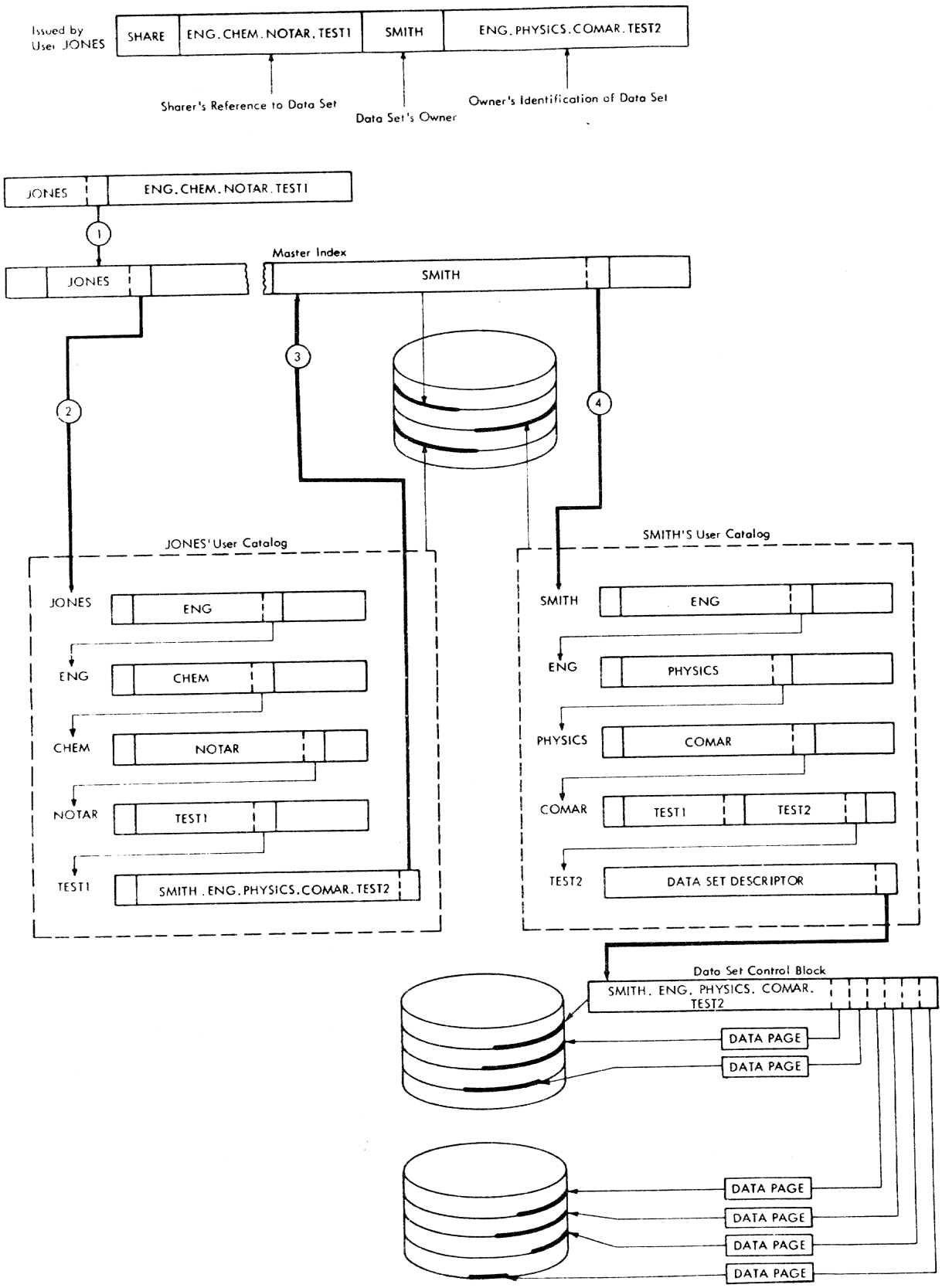


Figure 9. Sharing of Cataloged Data Sets

To be concurrently accessible by more than one task, a data set must be a VAM data set.

Table 9 explains the commands applicable to shared data sets.

DATA SET ORGANIZATIONS

Two basic types of data sets can be used in TSS/360:

- Virtual access method (VAM) data sets
- Sequential access method (SAM) data sets, also known as physical sequential (PS) data sets.

VAM data sets are formatted for use with direct access devices and for interface with the VAM data management routines. PS data sets are formatted for use with magnetic tape (although they can also be stored on direct access devices formatted for use by the sequential access methods discussed in Section 10, under "Physical Sequential Data Sets") or for communication between TSS/360 programs and programs on

Table 9. Shared Data Set Commands

Command	By Owner	By Sharer
PERMIT	Allowed.	Not allowed. A user cannot permit access to a data set that he does not own.
SHARE	Not allowed.	Must be issued prior to any other references to the data sets. Once issued, the sharer can access the data set until he issues an ERASE or DELETE. The SHARE command places an entry in the sharer's catalog, so that a CATALOG command is not necessary.
ERASE	The owner can only erase a member (object module) from his job library or erase the entire library when no sharer is accessing that member at the time the ERASE command is issued. If he erases the job library, the entry in the sharer's catalog is not removed. The sharer must issue a DELETE command to remove the entry from his own catalog.	A sharer can erase only if he has been granted unlimited access. If he then erases an object-module neither sharer's or owner's catalog is affected. If he erases the entire job library, both his catalog entry and the owner's are removed.
DELETE	The owner can delete a library or group of libraries from his catalog. An object module alone cannot be deleted. When the owner deletes a shared job library, the sharer's catalog entry is not removed.	A sharer can delete his catalog entry for a job library without affecting the owner's catalog. The sharer must reissue a SHARE command if he again wants to refer to the data set that has been deleted.
CATALOG	The owner can catalog a fully qualified data set name. If that name is a component of a partially qualified name that the owner has permitted to be shared, all sharers have immediate access to the newly cataloged data set. If an owner changes the name of a single data set to which he permitted access using a fully qualified name, each sharer must delete his catalog entry and reissue the SHARE command with the owner's new name.	A sharer who has been granted unlimited access can change or add entries to the owner's catalog. If he is permitted to share a group of data sets, he can catalog a new data set into the group, but he must include as part of the name the paritally qualified name that he used in the SHARE command. If he changes the name of one of the data sets in the group, the new name must still contain the partially qualified name. A sharer who has been granted unlimited access to an individual data set can never change the data set name.

the IBM System/360 Operating System or on the Model 44 Programming System.

The system organizes all VAM data sets into pages; i.e., blocks of 4096 bytes, and stores them on direct access volumes; for PS data sets, you must specify your own block size and your own direct access or magnetic tape volume. For more information on blocking, refer to Appendix D and to the subject "Consecutive Files," in Section 10.

Since VAM data sets are specifically formatted for TSS/360, they can use an installation's supply of public volumes. In order to access a PS data set, you must be qualified to receive a private volume from the installation, and the installation may require you to supply the tape reel or disk.

VAM data sets can be copied onto magnetic tape, but they must reside on direct access volumes if they are to be accessible to the VAM data management routines.

VAM DATA SETS

The types of VAM data sets are:

- Virtual sequential (VS)
- Virtual index sequential (VI)
- Virtual partitioned (VP)

Virtual Sequential (VS)

If described in the PL/I ENVIRONMENT attribute, a VS data set would be referred to as CONSECUTIVE. In a VS data set, the order of the logical records is determined solely by the order in which the records were created, and not by the content of the records. You can read back records in the order of their creation by merely requesting one record after the other. Since this is generally a simpler method of access, the virtual sequential organization is intrinsically more efficient for most applications where you don't need to access records at random.

A special type of VS data set, for PL/I users, is a list data set. List data sets, produced during each PL/I compilation, contain the listings; each print line is a record.

Example 12 in "Part III: Examples" shows the creation of a VS data set.

Virtual Index Sequential (VI)

If described in the PL/I ENVIRONMENT attribute, a VI data set would be referred to as INDEXED. VI records are similar to VS records, with the addition of an extra field called the key. The key can be anywhere in the record; however, if the key is embedded in the record it is transmitted as part of the record. (Refer to "Initial and Embedded Keys" in "Section 10: Record-Oriented Transmission.") All keys within the same data set must be placed similarly. The records in the data set are ordered by ascending sequence of the key field, and the records are accessed by key.

There are two special types of VI data sets -- line data sets and region data sets.

A line data set is indexed by line number, where each line is a record and is prefixed with the line number as its key. Source programs are line data sets. You can display all or part of a line data set using the LINE? command. Other commands enable you to effect replacements, insertions, and deletions on line data sets.

A region data set is indexed by both line number and region name; region names, arranged alphabetically, divide the data set into regions; line numbers index the elements of each region. See Part III, Example 10.

PL/I programs can process a VI data set either sequentially (by key) or nonsequentially (by ignoring the keys).

Examples 14 and 15 in "Part III: Examples" show the creation and updating of a VI data set.

Virtual Partitioned (VP)

A VP data set combines other data sets into a single data set. Each data set in the VP data set is called a member, and each member is identified by a unique name. A program module library is an example of a VP data set. Your USERLIB is organized in this way, and the compiled program modules you store in USERLIB are its members.

The partitioned organization allows you to refer to either the entire data set (by the VP data set's name) or to any member of that data set (by a name consisting of the name of the data set qualified by the member name in parentheses).

The VP data set can be composed of VS or VI members or a mixture of both. Individual members, however, cannot be of mixed organization.

PL/I I/O statements cannot be used directly on any VP data set. However, a CDS command can copy a VS or VI data set out of a VP data set, thus making a copy that is accessible to a PL/I program, or

insert a new, PL/I-processed member into the VP data set.

You will also use VP data sets if you define job libraries for storing object modules (see "Program Library List Control," in Section 6), or if you prestore source statements that are to be included in source programs by means of the %INCLUDE statement (see "Invoking the Preprocessor," in Section 5).

PS DATA SETS

The records in a PS data set are arranged strictly in the order of their creation. When these records are processed in TSS/360, the block is used as the unit of transfer to and from the I/O device; a block can consist of one or more logical records. You will use PS data sets each time you process magnetic tape in your programs. PS data sets are discussed further in Sections 9 and 10 and in Appendix D.

RECORD FORMATS

A record is the unit of information transmitted to or from a program; it is a set of contiguous bytes. TSS/360 recognizes three basic record formats:

- Format F, for records of a fixed length.
- Format V, for records of varying length.
- Format U, for records of undefined length.

FORMAT F

If a data set is made up of records that are all of the same length, it is format F, for fixed length. There are no special restrictions on the contents of a format-F record; however, record length is limited to 32,760 bytes in VS data sets and 4,000 bytes in VI data sets. Format-F records are not allowed in line or region data sets.

An example of a format-F data set is a data set where each record represents the contents of a punched card; each of these records would be 80 bytes long.

FORMAT V

If a data set is made up of records that are of varying length, it is format V, for variable length. The first four bytes of each record must contain a length indicator. The maximum record length permitted, including the length indicator, is 32,760 bytes in VS data sets and 4,000 bytes in VI data sets. Format-V records are best suited for data sets whose records are intrinsically varying in length, as is likely in the case of a data set consisting of lines typed at the terminal. Either format-V or format-F records can occur in any TSS/360 data set except data sets on ASCII tape (see Appendixes D and E), although they cannot be mixed together in the same data set unless they are in separate members of a VP data set.

FORMAT U

A third class of record is format-U, for undefined length. Format-U records are not allowed in VI data sets; in VS data sets, their length is always considered to be a multiple of a page (4096 bytes). The maximum record length permitted is 1,048,576 bytes.

The system stores object code as format-U records.

TYPES OF PL/I DATA TRANSMISSION

I/O statements that cause data transmission, that is, a transfer of data, are either STREAM I/O statements or RECORD I/O statements. STREAM I/O statements are GET and PUT; RECORD I/O statements are READ, WRITE, REWRITE, LOCATE, and DELETE.

There are two important differences between STREAM transmission and RECORD transmission. In STREAM transmission, each data item is treated individually, whereas RECORD transmission is concerned with collections of data items (records) as a whole. In STREAM transmission, each item may be edited and converted as it is transmitted; in RECORD transmission, the record on the external medium is an exact copy of the record as it exists in internal storage, with no editing or conversion performed.

As a result of these differences, record-oriented transmission is particularly applicable for processing large files that are written in an internal representation, such as in binary or packed decimal. Stream-oriented transmission can be used for processing typed (or keypunched) data and for producing readable output, where editing is required. Since files for which stream-oriented transmission is used tend to be smaller, the larger processing overhead can be ignored.

Table 10. Relationship Between PL/I Files and TSS/360 Access Methods

TYPE	ORGANIZATION	ACCESS	MODE	BUFFERING	RECORD FORMATS	ACCESS METHODS
STREAM	CONSECUTIVE	SEQUENTIAL	INPUT OUTPUT	BUFFERED	-	system*
					F,V,U	VSAM
					All	QSAM
RECORD	CONSECUTIVE	SEQUENTIAL	INPUT OUTPUT UPDATE	BUFFERED	F,V,U	VSAM
					All	QSAM
				UNBUFFERED	F,V,U	BSAM
	INDEXED	SEQUENTIAL	INPUT OUTPUT UPDATE	BUFFERED	F,V	VISAM
		DIRECT	INPUT OUTPUT UPDATE	UNBUFFERED	F,V	VISAM

*"system" means input from SYSIN and output to SYSOUT.

ACCESS METHODS

The system routines that process data sets with VAM or PS organizations are termed access methods. The access methods used by the PL/I library are:

VSAM: Virtual Sequential Access Method
VISAM: Virtual Index Sequential Access Method
} for VAM data sets

BSAM: Basic Sequential Access Method
QSAM: Queued Sequential Access Method
} for physical sequential data sets

The Virtual Partitioned Access Method (VPAM), available in TSS/360, is not used by the PL/I library. However, the PL/I library can use VSAM or VISAM on a data set that has been copied out of a VP data set by the CDS command.

The PL/I library subroutines use VSAM or QSAM for all stream-oriented transmission. They implement PL/I GET and PUT statements by transferring the appropriate number of characters from or to the data management buffers, and use GET and PUT macro instructions in the locate mode to fill or empty the buffers. Table 10 shows the relationship between PL/I files and TSS/360 access methods.

BASIC DDEF COMMAND

A DDEF command describes a data set to the system, and is a request to the system for the allocation of I/O resources. The DDEF command gives the data set's name; it can also describe the data set's organization, the attributes of the data itself (record format, etc.), and the data set's location (for example, the volume serial number and identification of the unit on which the volume will be mounted).

In a conversational task, the system analyzes the data set's requirements at the time the DDEF command is issued. It then attempts to allocate the required resources, and issues any mounting messages that are required, at that time. If there is no device available, you are asked to either wait for an available device or cancel the DDEF command.

Each TSS/360 task must include a DDEF command for each data set that is processed by the task, unless the data is read from SYSIN, written to SYSOUT, or defined in another command; (for example, the PLI command can define the source, object, and list data sets).

A DDEF command can be issued at any time during the task prior to execution of the program in which the data set is to be used. Each DDEF command is valid only during the task in which it is issued; previously defined data sets must be redefined in every task that refers to them. A DDEF

Table 11. Basic DDEF Command for the PL/I User

Operation	Operand
DDEF	DDNAME=data definition name[,DSORG={VI|VS|VP}],DSNAME=data set name [,DISP={OLD|NEW}][,DCB=([RECFM={F|V|U}],[LRECL=integer] [,KEYLEN=integer][,RKP=integer])]

command that has been entered can be canceled by a RELEASE command.

The DDEF command enables you to write PL/I source programs that are independent of the data sets and I/O devices they will use. You can modify the parameters of a data set or process different data sets without recompiling your program; for example, you can modify a program that originally read from a direct access device so that it will accept input from magnetic tape merely by changing the DDEF command.

Normally, PL/I users require only basic DDEF commands, defaulting most of the operand fields. In some cases, DDEF commands themselves can be defaulted; this causes the system to choose SYSIN for input, or SYSOUT for output.

COMMAND FORMAT

Table 11 shows the format of the PL/I user's basic DDEF command. For information on the full DDEF command, see Appendix D.

Note: This section and Appendix D present shortened forms of the DDEF command that eliminate operands not useful to you. As a result, the description of the portion of the DDEF command that follows DSNAME is positionally inaccurate. If you specify DDEF operands that follow DSNAME, you should give them in the keyword form shown in this manual, not in positional form.

DDNAME: Specifies the data definition name.

Specified as one to eight alphameric characters; the first character must be alphabetic. DDNAME must not begin with SYS, because these characters are reserved to prefix system-generated data definition names.

Since DDNAME is a required parameter, it cannot be defaulted.

DSORG: Specifies the data set organization. The default value differs from installation to installation.

DSNAME: The DSNAME parameter specifies the name of the data set. This is the name under which the data set is to be cataloged or referred to by other commands during the session. It contains one or more simple names, each simple name having one to eight alphameric characters, the first of which must be alphabetic. A period is used as separator between simple names. The maximum number of characters, including periods, is 35. The maximum number of simple names is 18.

In most cases, the DSNAME need only be one simple name such as:

```
DDEF  DDNAME=NAME,DSORG=VS,DSNAME=OUTPUT
```

A DSNAME may be of value in describing the contents of the data set. Thus, a program that generates a table of random numbers and a table of square roots might employ the DDEF commands:

```
DDEF  DDNAME=NAME1,DSORG=VS,-
 DSNAME=TABLE.RANNUM

DDEF  DDNAME=NAME2,DSORG=VS,-
 DSNAME=TABLE.SQRROOTS
```

The DSNAME can contain a generation number in either absolute or relative form.

Examples:

PAYROLL(0)	Means the most recent generation of PAYROLL
PAYROLL(-1)	The last generation.
PAYROLL(+1)	The next generation.
PAYROLL.G0005V00	Fifth absolute generation.

If a DSNAME is to contain a generation name, the DSNAME proper is limited to 26 characters, including periods. Prior to use of the generation name, you must set up a generation data group with the CATALOG command. (See Command System User's Guide.)

Since DSNAME is a required parameter, it cannot be defaulted.

DISP: Specified as OLD or NEW. OLD means that the data set is being redefined and is supposed to exist; NEW means that the data set is being defined for the first time and

that no data set should already exist under the specified DSNAME.

DISP=OLD and DISP=NEW do not affect the data set being defined. DISP=OLD guards against accidental creation and use of a new data set; DISP=NEW guards against use of an existing, forgotten data set. It is recommended that you use the DISP parameter habitually. If the DISP specification disagrees with the actual state of the named data set, then:

- In conversational mode, the user receives a diagnostic message so that he can correct this error.
- In nonconversational mode, the task is abnormally terminated.

When unspecified, DISP defaults to NEW if the system does not find the DSNAME in the catalog, to OLD if the DSNAME is found.

DCB: A data control block (DCB) is one of the major control tables for communication between TSS/360 data management and any program requiring control of a data set. The PL/I library I/O routines build a DCB whenever a DSNAME is encountered for the first time in executing the object program. Sources of information for the DCB are the DDEF command, file attributes declared explicitly or implicitly in the PL/I program, and, if the data set already exists, the data set label.

In case of conflict, information that you specify in the DDEF command is given first priority.

In the DDEF command, the DCB parameters of critical interest to the PL/I programmer are RECFM, LRECL, KEYLEN, and RKP. KEYLEN and RKP apply only to VI data sets.

RECFM:
RECFM specifies the format or character of the records in the data set. This format is:

F -- fixed-length records
-- maximum record length is 32,756 bytes for VS, and 4,000 bytes for VI

V -- variable-length records
-- each record contains in the first four bytes a binary count of the number of bytes in the record
-- maximum record length is 32,756 bytes for VS, and 4,000 bytes for VI

U -- undefined-length records
-- record length always a multiple of a page (4096 bytes)
-- maximum record length is 1,048, 576 bytes

The default value is V.

LRECL:
LRECL specifies the length in bytes of a logical record. For format-F records, this operand specifies the length of each record in the data set. For format-V and -U records, it specifies the maximum expected length. The maximum acceptable record lengths are given in Appendix D.

If record length information is given in the ENVIRONMENT attribute, the LRECL operand of the DDEF command is ignored.

KEYLEN and RKP:
If DSORG=VI and DISP=NEW, you must specify key length (KEYLEN) and relative key position (RKP).

KEYLEN is the length in bytes of the key associated with a record. The maximum value is 255.

RKP specifies the displacement of the key field from the first byte of the logical record. (See "Indexed Files," in Section 10.)

The CDD Command

The DDEF commands used in the task need not be issued directly. One, or more, or all, of the DDEF commands needed can be made available by using the CDD (call data definition) command. The CDD command is used to retrieve one or more DDEF commands from a line data set;[1] you must supply the name of the data set. If this is all you specify, the system assumes that you want to use all the DDEF commands in the data set. If you want to use only selected DDEF commands, you identify each by its DDNAME (data definition name). You should prestore frequently used DDEF commands in a data set and call them in this fashion wherever possible.

FILES AND DATA SETS

When you write a PL/I program, you do not need to know which data sets you will use or where the volumes that contain them will be mounted. PL/I uses a conceptual 'file' as a means of accessing a data set. When an OPEN statement is executed, the file is associated with a data set through the TITLE option, which refers to the name

---

[1]Such a data set can be created using a DATA or EDIT command.

of the DDEF command (data definition name, or DDNAME) that describes the data set; if the OPEN statement does not include the TITLE option, the compiler takes the data definition name from the first eight characters of the file name, padding it with blanks if necessary.

The OPEN statement indicates the DDNAME of the DDEF command that describes the data set to be associated with the file that is being opened; the DDEF command specifies the type of device that will access the data set, the serial number of the volume that contains the data set, and the name of the data set (DSNAME). See Figure 10. If the DDEF command refers to a cataloged data set, it need supply only the DDNAME and the DSNAME; the system can use the DSNAME to obtain unit and volume information from the system catalog.

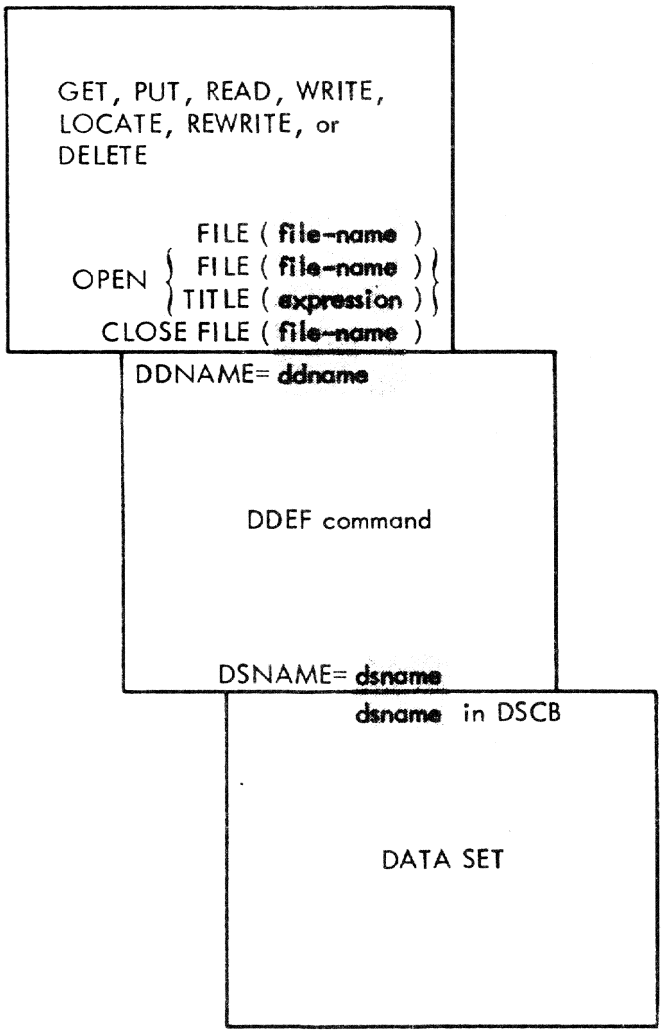


Figure 10. Associating a File with a Data Set

Since the link between the PL/I file and the data set exists only while the file is open, the same file can be associated with different data sets during the execution of a single program; and the same data set can be accessed through different files. Furthermore, the use of a DDEF command to define the data set, the volume that contains it, and the device on which they will be placed, enables you to defer your choice until execution time; and you can use the same program to process different data sets on different devices without recompiling the program.

OPENING A FILE

The execution of a PL/I OPEN statement associates a file with a data set. This requires the merging of the information describing the file and the data set. If any conflict exists between file attributes and data set characteristics the UNDEFINEDFILE condition will be raised.

It should be noted that the omission of a DDEF command for a RECORD file causes the UNDEFINEDFILE condition to be raised. A STREAM file, on the other hand, defaults to SYSIN or SYSOUT, so that PL/I does not raise the UNDEFINEDFILE condition merely because the STREAM file has no corresponding DDEF command. (However, it is still possible to have it raised because of attribute conflicts.)

The data management subroutines of the PL/I library create a skeleton data control block (DCB) for the data set, and use the file attributes from the DECLARE and OPEN statements, and any attributes implied by the declared attributes, to complete the DCB as far as possible. They then issue a data management OPEN macro instruction, which calls the system data management routines to check and complete the DCB. System routines examine the DCB to see what information is still needed and then look for this information, first in the DDEF command, and finally, if the data set already exists, in a control block called the data set control block (DSCB). The DSCB is on the volume containing the data set and describes the data set. PS data sets on magnetic tape don't have DSCB's; however, if they are labeled, similar information is contained in the tape labels.

Neither the DDEF command nor the data set label can override information provided by the PL/I program; nor can the data set label override information provided by the DDEF command.

When the DCB fields have been filled in from these sources, control returns to the

PL/I library subroutines. If any fields have still not been filled in, the PL/I OPEN subroutine provides default information for some of them; for example, if LRECL has not been specified, it is now provided from the value given for BLKSIZE.

CLOSING A FILE

The execution of a PL/I CLOSE statement dissociates a file from the data set with which it was associated. The PL/I library subroutines first issue a data management CLOSE macro instruction and then, when control returns from the system data management routines, release the DCB that was created when the file was opened. The data management routines complete the writing of labels for new data sets and update the labels of existing data sets.

SUMMARY

To specify problem program I/O activity, you must consider:

1. Use of I/O statements in the source program to indicate data transfer or I/O control functions.

2. Use of (or omission of) file declarations in the source program to indicate the usage, function, access, buffering, scope, etc., of the data sets associated with the I/O statements.

3. Use of (or omission of) DDEF (define data) commands to identify the name, location, organization, etc., of the data sets associated with the I/O statements.

To be processed by a PL/I program, a data set other than SYSIN or SYSOUT must be identified by a DDEF command; the basic purpose of the DDEF command is to specify the data set's name and organization.

Table 12 summarizes the TSS/360 data set organizations. Related access methods are also shown. Normally, the access method is of no concern to you; the system automatically uses the other information that you give to choose the correct access method.

Table 12. Types of Access Methods and Data Set Organizations

This fundamental type of access method and data set organization,	includes these access methods:	and these data set organizations:
virtual access method: VAM	virtual sequential access method: VSAM	virtual sequential: VS
	virtual indexed sequential access method: VISAM	virtual indexed sequential: VI special VI types: line data sets region data sets list data sets
	virtual partitioned access method: VPAM	virtual partitioned: VP (combines VS and VI data sets)
sequential access method: SAM	basic sequential access method: BSAM	physical sequential: PS
	queued sequential access method: QSAM	

Stream-oriented transmission allows a PL/I program to ignore block and record boundaries and treat a data set as a continuous stream of data items in character form. For output, the data management subroutines of the PL/I library convert the data items from the program variables into character form if necessary, and build the stream of characters into records for transmission to the data set. For input, the library subroutines take records from the data set and separate them into the data items requested by the program, converting them into the appropriate form for assignment to the program variables. Because stream-oriented transmission always treats data as a continuous stream, it can be used only to process data sets with CONSECUTIVE organization.

Under TSS/360, stream I/O files can operate in either of two ways:

1. By use of the system I/O files SYSIN/SYSOUT. This mode is used automatically by all stream files for which no corresponding DDEF command has been issued prior to execution.

2. By accessing a data set that has been defined previously by a DDEF command. The data set can have either virtual sequential (VS) or physical sequential (PS) organization.

SYSTEM FILES

There are two system files for any TSS task -- SYSIN for input and SYSOUT for output.

Note: At execution time, the standard PL/I file SYSIN becomes the system file SYSIN, and the standard PL/I file SYSPRINT becomes the system file SYSOUT.

Any stream file for which no corresponding DDEF command has been issued automatically defaults to SYSIN or SYSOUT, depending on whether it is opened for input or output; it need not have the file name SYSIN, SYSOUT, or SYSPRINT. System files differ in one respect from other files that may be accessed by a PL/I program: No record can be accessed more than one time by the task. Thus, closing the SYSIN file and reopening it does not affect the sequence of records read from the file. Similarly, once a record is written to SYSOUT, it is inaccessible to the program.

SYSTEM INPUT FILE -- SYSIN

The records from the SYSIN file can come from several alternative sources.

Conversational Mode

In conversational mode, records can be entered:

- At the terminal keyboard.
- From a terminal card reader.

If the records are typed at the terminal, a prompting character (:) appears at the terminal as each record is required. For example; on execution of the PL/I statement:

```
GET DATA(A,B,C);
```

the system prompts with a colon and you enter the data; if you don't enter data for all the items in the data list, indicate the end of the data with a semicolon:

```
: A=5.3, B=6.0 ;
```

Alternatively, the above data could be entered like this:

```
: A=5.3
: B=6.0 ;
```

In this case, since the terminating semicolon did not appear in the first line, another prompting character is sent to the terminal.

If your terminal is connected to an IBM 1056 card reader, you can designate the card reader as the SYSIN device by typing a C command at any time during the task when the system expects to receive a command. You can return control to the keyboard by including a K command in the card-input stream, wherever the command mode is in effect. When the system reads the K command, the keyboard becomes the SYSIN device again; and the task continues uninterrupted. (For more information on the C and K commands and related commands, see Terminal User's Guide publication.) The cards to be read can contain commands, as well as SYSIN data for PL/I programs.

Note: To transfer control to the card reader during program execution, you must press the attention key; after the system prompts you with an exclamation mark (!), type a C command. After this, a card that

contains a GO command will cause the program to resume execution.

Nonconversational Mode

Data for a program executed as part of a nonconversational task can be entered in one of four ways:

1. As a card deck supplied for input through the system card reader.

2. As a card deck entered through the card reader at a remote station (IBM 2780 Data Transmission Terminal) that is connected to a TSS/360 installation. (See IBM System/360 Time Sharing System: Remote Job Entry.)

3. As part of a data set that is the subject of an EXECUTE command.

4. As part of a data set that is the subject of a BACK command.

In cases 3 and 4, the data must be in a VS data set or a VI line data set.

Data Contained Within Command Procedures

By use of the SYSINX parameter to the DEFAULT command, it is possible to read data from within a command procedure (PROCDEF). Given this command procedure,

```
PROCDEF  ABC

PLIPROG                 PL/I program

1.0  2.0  3.0,  }
4.0  5.0  6.0,  }       PL/I data
etc.            }

_END                    End of procedure
```

the following commands cause the program PLIPROG to be executed with data taken from within the PROCDEF;

```
DEFAULT SYSINX=E     Set SYSINX

ABC                  Invoke procedure

DEFAULT SYSINX=G     Reset SYSINX
```

ENDFILE Condition for SYSIN

A null record, that is, a record of zero length, is interpreted by the stream I/O routines as an ENDFILE condition. In conversational mode, a zero-length record is formed by pressing RETURN after the system types the prompting character. In nonconversational mode, a zero-length record is formed by an EOB character in the first position of the record. This program would continue processing until it read a zero-length record:

```
EXAMPLE:  PROC OPTIONS(MAIN);

          ON ENDFILE (SYSIN) GO TO END;

LOOP:     GET DATA;
          .
          .
          .
          GO TO LOOP;
END:      END;
```

SYSTEM OUTPUT FILE -- SYSOUT

At execution time, the standard PL/I file SYSPRINT automatically becomes the TSS/360 output file SYSOUT; therefore, you do not have to supply a DDEF command for SYSPRINT.

Conversational Mode

In conversational mode, all data sent to SYSOUT appears at the terminal device. The output is formatted in the normal way, with the exception that page skips do not occur; instead, a maximum of three line feeds is used.

Nonconversational Mode

In nonconversational mode, all records sent to SYSOUT during a task are placed in a data set in strict sequential order. This data set is printed and erased automatically. All formatting of data is accepted in the normal way, including page skips.

SYSPRINT Attributes

If you do not declare the file SYSPRINT, the compiler gives it the attribute PRINT in addition to the normal default attributes; thus, the complete implicit declaration is SYSPRINT FILE STREAM OUTPUT PRINT EXTERNAL. Since SYSPRINT is a PRINT file, the compiler also supplies a default line size of 120 characters and a format-V record.

You can override the attributes given to SYSPRINT by the compiler by explicitly opening the file. If you do so, bear in mind that this file is also used by the error-handling routines of the compiler, and that any change you make in the format of the output from SYSPRINT also apply to the format of execution-time error messages. When an error message is printed, eight blanks are inserted at the start of each line except the first; consequently, if you specify a line size of less than nine characters (or a block size of less than ten bytes for format-F or format-U records, or less than 18 bytes for format-V records), the second and successive lines will not be printed, and the error-message routine will be locked in a permanent loop.

USER-SPECIFIED DATA SETS

As an alternative to using the system files SYSIN and SYSOUT, any STREAM I/O file can be made to access a VS or PS data set that you name in a DDEF command. To do this, you must issue the DDEF command before the file is opened; the DDNAME parameter must be the same as either the file name or the name specified in the TITLE option of an OPEN statement issued against the file. Examples:

```
DECLARE  XYZ  FILE STREAM ...
         ↓    ↑
DDEF  XYZ,VS,DATASET,...
```

Or

```
DECLARE ABC FILE STREAM
                                  ⎫
OPEN FILE(ABC) TITLE('XYZ')       ⎬ PL/I
     ↓                       ↑    ⎭ statements
DDEF XYZ,VS,DATASET,...
```

In the second example, the use of the TITLE option overrides the filename ABC.

Note: DDNAMES beginning with SYS are reserved for use by system programs. Thus, it is not possible to issue a DDEF command with a DDNAME of SYSIN, SYSOUT, or SYSPRINT. If a file name begins with SYS, the TITLE option is the only way you can make the file access a data set other than SYSIN or SYSOUT. For example, the statement:

```
GET DATA(A,B,C);
```

becomes by implication:

```
GET FILE(SYSIN) DATA(A,B,C);
```

If you include the statement:

```
OPEN FILE(SYSIN) TITLE('NOTSYS');
```

before the first input statement, then a DDEF command:

```
DDEF NOTSYS,PS,....etc.
```

can be issued against the file.

VIRTUAL SEQUENTIAL DATA SETS (DSORG=VS)

Under the TSS/360 virtual access method, all blocking of logical records is controlled by the system routines; hence, a VS data set is easier to use than a PS data set. Required DCB parameters for a VS data set are record format (RECFM) and logical record length (LRECL). These parameters can be supplied in any of four ways:

1. From the DSCB for old (DISP=OLD) data sets.
2. In the DDEF command.
3. In the ENVIRONMENT attribute of the file declaration. See IBM System/360 Time Sharing System: PL/I Language Reference Manual.
4. By default if 1, 2, and 3 do not apply. The default values are RECFM=V, LRECL=132.

Example 12 in "Part III: Examples" illustrates the use of a VS data set. Table 13 shows the relationship between the RECFM and LRECL parameters and the LINESIZE option of the OPEN statement, for VS PRINT and non-PRINT files.

PHYSICAL SEQUENTIAL DATA SETS (DSORG=PS)

Physical sequential data sets are always private and unsharable. They can reside on disk or tape devices. The DCB subparameters RECFM (record format) and LRECL (logical record length), required for VS data sets, are also required for PS data sets. You control the blocking of records for a PS data set; hence, an additional DCB subparameter, BLKSIZE (physical block size) is required. The records may be unblocked (that is, one logical record per physical record) or blocked (that is, more than one logical record per physical record). Refer to Appendix D for ways of specifying the DCB subparameters relating to data set residence (UNIT, VOLUME, and LABEL) and data set protection (PROTECT).

The required DCB parameters, RECFM, LRECL, and BLKSIZE, can be specified in any of three ways:

1. In the data set control block (DSCB) or data set label for old (DISP=OLD) data sets.
2. In the DDEF command.
3. In the ENVIRONMENT attribute of the file declaration.

Example 12 in "Part III: Examples" illustrates the use of a PS data set. Table 13 shows the relationship between the RECFM, LRECL, and BLKSIZE parameters and the LINESIZE option of the OPEN statement, for PS PRINT and non-PRINT OUTPUT files.

PRINT FILES

Both TSS/360 and PL/I include features that facilitate the formatting of printed output. TSS/360 allows you to use the first byte of each record for a printer control character; the control characters, which are not printed, cause the printer to skip to a new line or page. In PL/I, the use of a PRINT file provides a convenient

Table 13. Relationship of LINESIZE Option with RECFM, LRECL, and BLKSIZE Parameters for STREAM OUTPUT Files

Type of Data Set	OPEN Option	DCB Subparameter		
	LINESIZE	RECFM	LRECL	BLKSIZE
SYSOUT (non-PRINT)	L	N/A	N/A	N/A
SYSOUT (PRINT)	L	N/A	N/A	N/A
VS (non-PRINT)	L L 4096*N	F V U	L L+4 4096 * Nax N	N/A N/A N/A
VS (PRINT)	L L (4096*N)-1	F V U	L+1 L+5 4096 * Max N	N/A N/A N/A
PS (non-PRINT, Unblocked)	L L L	F V or D U	L L+4 L	L L+8 Max L
PS (non-PRINT, Blocked)	L L	FB VB or DB	L L+4	B*L B*(L+4)+4
PS (PRINT, Unblocked)	L L L	F V or D U	L+1 L+5 L+1	L+1 L+9 (Max L)+1
PS (PRINT, Blocked)	L L	FB VB or DB	L+1 L+5	B*(L+1) B*(L+5)+4

Notes: B = blocking factor
D and DB are RECFM values for tapes in ASCII format (see Appendix E)
L = specified linesize
Max N = number of pages occupied by largest record
N = positive integer
N/A = not applicable

means of inserting printer control characters; the compiler automatically inserts them in response to the PAGE, SKIP, and LINE options and format items.

You can apply the PRINT attribute to any STREAM OUTPUT file, even if you do not intend to print the associated data set directly. When a PRINT file is associated with a magnetic tape or direct access data set, the control characters have no effect on the layout of the data set, but appear as part of the data in the records.

The first byte of each record transmitted by a PRINT file is reserved for an American National Standard FORTRAN control character (hereinafter referred to as FORTRAN control character), and the appropriate character is inserted automatically. Appendix C discusses the FORTRAN control characters; a PRINT file uses only the following five characters:

New Page	1
Single line space	b (blank)
Double line space	0
Triple line space	-
Suppress space	+

The PL/I library handles the PAGE, SKIP, and LINE options or format items by padding the remainder of the current record with blanks and inserting the appropriate control character in the next record. If SKIP or LINE requests more than a triple line space, the library inserts sufficient blank records with appropriate control characters to accomplish the required positioning. In the absence of a printer control option or format item, where a record is full the library inserts a blank code (single line space) in the first byte of the next record. For a PRINT file directed to the terminal, the PAGE option results in a three-line SKIP, and a SKIP option specifying a spacing greater than three lines results in a three-line skip.

RECORD FORMAT

You can limit the length of the printed line produced by a PRINT file either by specifying a record size in the ENVIRONMENT attribute or in a DDEF command, or by giving a line size in an OPEN statement. The record size must include the extra byte for the printer control character, that is, it must be one byte larger than the length of the printed line (five bytes larger for format-V records). The value you specify in the LINESIZE option refers to the number of characters in the printed line; the PL/I library adds the control bytes.

The blocking of records has no effect on the appearance of the output produced by a PRINT file, but it does result in more efficient use of storage space when the file is associated with a data set on magnetic tape or a direct access device. If you use the LINESIZE option, ensure that your line size is compatible with your block size: for format-F records, blocksize must be an exact multiple of (line size + 1); for format-V records, blocksize must be at least nine bytes greater than line size.

Although you can vary the line size for a PRINT file during execution by closing the file and opening it again with a new line size, you must do so with caution if you are using the PRINT file to create a data set or magnetic tape or a direct access device; you cannot change the record format established for the data set when the file is first opened. If the line size specified in an OPEN statement conflicts with the record format already established, the UNDEFINEDFILE condition will be raised; to prevent this, either specify format-V records with a block size at least nine bytes greater than the maximum line size you intend to use, or ensure that the first OPEN statement specifies the maximum line size. (Output destined for the printer is temporarily stored on a direct access device, even if you intend it to be fed directly to the printer.)

Since PRINT files have a default line size of 120 characters, you need not give any record format information for them. In the absence of other information, the compiler assumes format-V records; the complete default information is:

```
BLKSIZE=129
LRECL=125
RECFM=V
```

Example 13 in "Part III: Examples" illustrates the use of PRINT files and stream-oriented transmission. Table 13 shows the relationship between the RECFM, LRECL, and BLKSIZE parameters and the LINESIZE option of the OPEN statement, for PRINT and non-PRINT files.

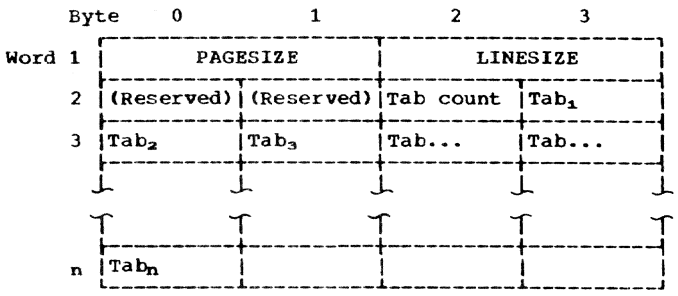
TAB CONTROL TABLE

Data-directed and list-directed output to a PRINT file is automatically aligned on preset tabulator positions; the tab settings are stored in a table in the PL/I library module IHEWTAB (Figure 11). (IHEWTAB is contained in module CFBAJ, in SYSLIB.) The functions of the fields in the table -- Figure 11(a) -- are:

PAGESIZE
: halfword binary integer that defines the default page size.

LINESIZE
: halfword binary integer that defines the default line size.

Reserved Bytes
: reserved for left and right margin facilities.



(a) Tab control table

60		120	
0	0	5	25
49	73	97	121

(b) Standard form of table

```
IHETAB    CSECT
          ENTRY   IHETABS
IHETABS   DC      AL2(60)     DEFAULT PAGE SIZE
          DC      AL2(120)    DEFAULT LINE SIZE
          DS      X           RESERVED
          DS      X           RESERVED
          DC      AL1(5)      NO. OF TAB POSITIONS
          DC      AL1(25)     TAB POSITIONS
          DC      AL1(49)
          DC      AL1(73)
          DC      AL1(97)
          DC      AL1(121)
          END
```

(c) Control section IHETAB

Figure 11. Tabular Control Table (Module IHEWTAB)

Tab count
number of tab position entries in table (maximum 255). If tab count = 0, the tab positions are not used: each data item is put out as if a PRINT file were not being used.

Tab_1 - Tab_n
tab positions within the print line. The first position is numbered 1, and the highest position is numbered 255. The value of each tab should be greater than that of the tab preceding it in the table; otherwise, it will be ignored. The first data field in the printed output begins at the left margin (position 1), and thereafter each field begins at the next available tab position.

You can alter the tab control table by changing the values in the assembler language control section -- Figure 11(c). There are three ways to do this:

1. The installation can assemble a new version into SYSLIB.

2. You can assemble your own version into a private job library.

3. During a task, you can change items by using PCS. For example, to change tab settings to 1, 41, 81, and 121:

```
display ihetab
IHETAB VERSION ID 09/14/70 01:17:10
00000000 003C0078 00000519 31496179
set ihetab.(6,6)=x'032951790000'
```

Note that position 1 does not count as a tab setting.

SUMMARY OF STREAM-ORIENTED TRANSMISSION

Figure 12 shows the types of TSS/360 data that can be accessed by a STREAM file.

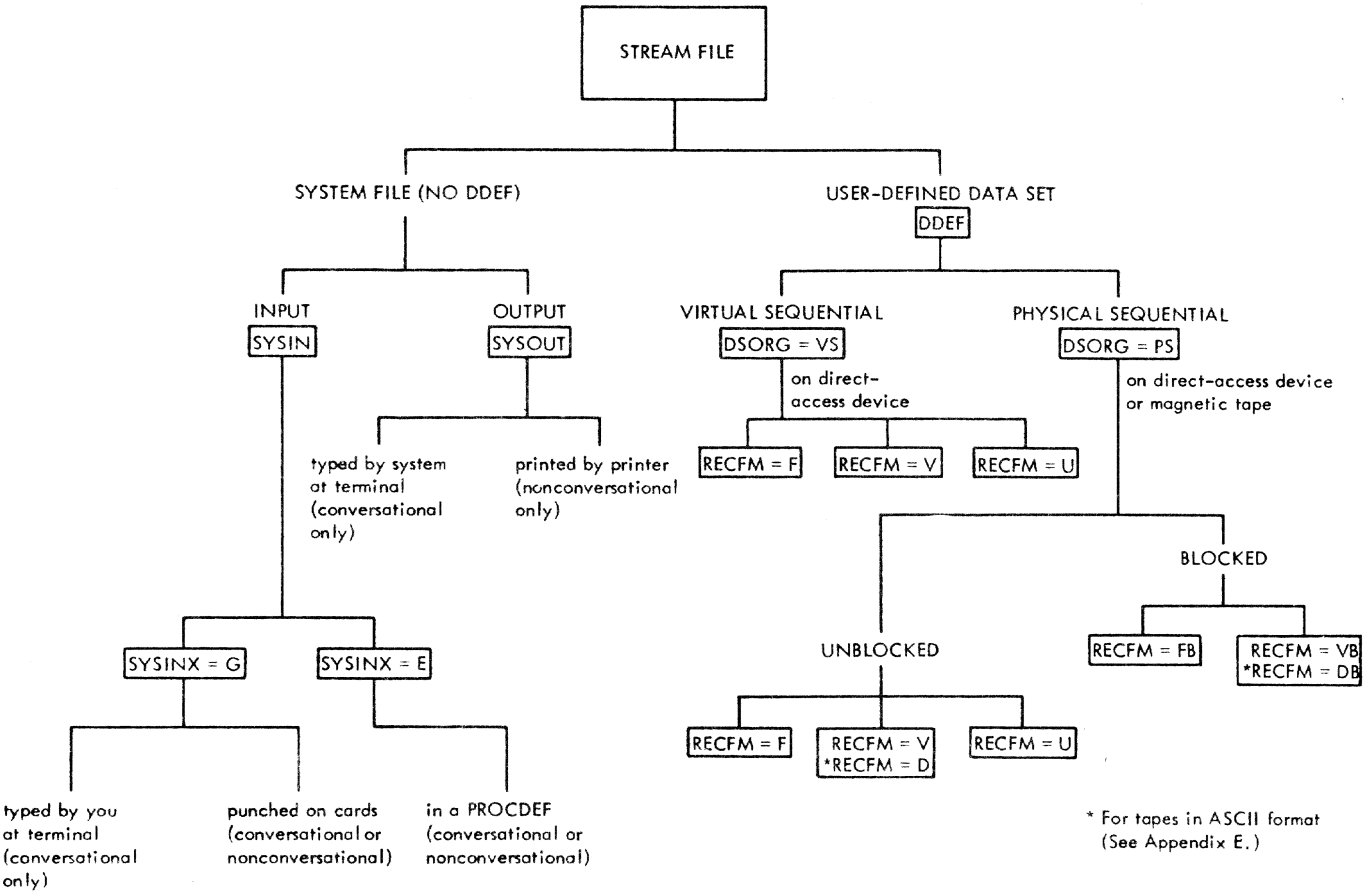


Figure 12. Relationship Between a STREAM File and TSS/360 Data

In record-oriented transmission, data is transmitted to and from auxiliary storage exactly as it appears in the program variables; no data coversion takes place. In most cases, the data contained in a logical record of a data set corresponds to a variable in the program. Usually data management control information (for example, block and record lengths for format-V records) is removed before assigning the data to a variable, or inserted on output.

Normally, format-V records are read into and written from strings or aggregates of varying length. Format-F records can be used for fixed-length variables.

Record-oriented transmission cannot be used for accessing the system files SYSIN and SYSOUT. A corresponding DDEF command must be issued for any record file that is to be opened during execution. Failure to do this causes an UNDEFINEDFILE condition to be raised against the file.

Figure 13 shows the relationship between the attributes of a RECORD file and the types of TSS/360 data sets that it can access. From the table, these points can be seen:

1. RECORD files can have only CONSECUTIVE or INDEXED organization. REGIONAL and TRANSIENT organizations are not supported by TSS/360.

2. CONSECUTIVE files can only be accessed in SEQUENTIAL mode.

3. INDEXED files can be accessed in either SEQUENTIAL or DIRECT mode. The default mode is SEQUENTIAL.

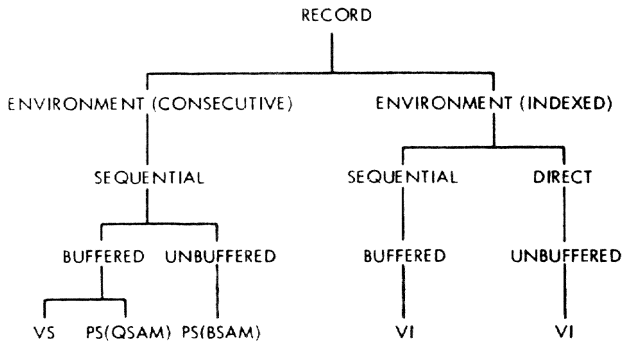


Figure 13. Access of RECORD Files to TSS/360 Data Sets

4. CONSECUTIVE files can have the BUFFERED or UNBUFFERED attribute.

5. CONSECUTIVE BUFFERED files can be used for data sets having either virtual sequential (VS) or physical sequential (PS) organization. The organization must be specified in the DSORG parameter of the DDEF command.

6. CONSECUTIVE UNBUFFERED files can only be used for PS data sets.

7. INDEXED files can only be used for data sets having virtual indexed sequential (VI) organization.

8. INDEXED files that are given the SEQUENTIAL attribute are automatically given the BUFFERED attribute; hence, both move- and locate-mode I/O statements can be used.

9. INDEXED files that are given the DIRECT attribute are automatically given the UNBUFFERED attribute; hence, they can use only the move mode of access.

CONSECUTIVE FILES

Table 14 shows the types of CONSECUTIVE files that you can specify and the I/O statements that you can use with them. CONSECUTIVE files can be given the BUFFERED or UNBUFFERED attributes. (Note: The default attribute is BUFFERED.) The BUFFERED attribute allows the file to access either VS or PS data sets without any program changes. The type of data set is determined by the DSORG parameter, which must be included in the DDEF command issued for the file. If the file is given the UNBUFFERED attribute, it can only be used to access PS data sets using the Basic Sequential Access Method (BSAM). (For further information on BSAM, see IBM Time Sharing System: Data Management Facilities, GC28-2056.) The following pages discuss the use of CONSECUTIVE files to access:

- Virtual Sequential Data Sets
- Physical Sequential Data Sets

VIRTUAL SEQUENTIAL DATA SETS

Virtual sequential data sets are the simplest and most efficient way of storing

Table 14. Characteristics of CONSECUTIVE Files

Organization	Access	Buffering	Mode	Statement	Options	Access Method	Record Formats
ENVIRONMENT (CONSECUTIVE)	SEQUENTIAL	BUFFERED	INPUT	READ	INTO IGNORE SET	VSAM or QSAM	F, V or D, U
			OUTPUT	WRITE	FROM		F, V, U
				LOCATE	SET		FB, VB
			UPDATE	READ	INTO IGNORE SET		or DB
				REWRITE	FROM		
		UNBUFFERED	INPUT	READ	INTO IGNORE EVENT	BSAM only	F, V or D, U
			OUTPUT	WRITE	FROM EVENT		
			UPDATE	READ	INTO IGNORE EVENT		
				REWRITE	FROM EVENT		

Note: The D and DB record formats are for tapes in ASCII format (see Appendix D).

data in TSS/360. Records are automatically blocked into page-size physical records and there is no need for the user to provide blocking information. They are stored on public volumes and cataloged automatically at creation time. When you have them read or updated, you supply the data set name (DSNAME) and data definition name (DDNAME).

Example 12 in "Part III: Examples" shows the creation of a VS data set.

Creating a Virtual Sequential Data Set

To create a VS data set using record-oriented transmission, certain essential information must be supplied to the system's data management routines. This information is taken from the following sources:

- The PL/I program (the ENVIRONMENT option of the file declaration).
- The DDEF command.
- By default, if not specified in the program or in the DDEF command.

Default values may not always be adequate for correct execution of the PL/I program.

Table 15 details the parameters which are always required for a new VS data set and lists alternative ways they may be supplied.

Table 15. Specification of VS Data Set Characteristics

Required Parameter	DDEF Command	PL/I Program	Default Value
DDNAME	file-name title-name	file-name title-name	none
DSORG=	VS	--	none
DSNAME=	DATA SET NAME	--	none
DISP=	NEW	--	See Note
DCB=			
RECFM=	F, V, or U	ENVIRONMENT option	V
LRECL=	length		132

Note: DISP= defaults to OLD if DSNAME is in catalog, to NEW if DSNAME is not in catalog.

Accessing a Virtual Sequential Data Set

Since VS data sets are cataloged automatically when they are created, a minimum of information is required when they are accessed at a later time. The essential information required in the DDEF command is

DDNAME = file or title-name

DSNAME = data set name

DISP = OLD

The remaining information that is required by the data management routines is obtained from the catalog entry.

Note: It is not possible for the programmer to alter the existing data management parameters (for example, RECFM, LRECL). If they are supplied, they must be the same as the existing values.

The existing data set can be accessed in three ways; the associated file can be opened for INPUT, OUTPUT or UPDATE.

INPUT: The file is positioned at the first record in the data set. The records are presented in sequence; after the last existing record is read, a further read statement causes an ENDFILE condition to occur for the file.

OUTPUT: The file is positioned after the last existing record; new records are added to the end of the data set.

UPDATE: The file is positioned at the beginning of the data set. Records that are replaced cannot have their length altered.

PHYSICAL SEQUENTIAL DATA SETS

Physical sequential data sets can reside on tape or disk devices and are always private and non-sharable. They can be accessed using either the Queued Sequential Access Method (QSAM) or the Basic Sequential Access Method (BSAM), depending upon whether the associated files are given the BUFFERED or UNBUFFERED attribute. When QSAM is used, the records are blocked and deblocked automatically by the data management routines; hence, the PL/I program is simpler to write and has better safeguards against errors. Although the use of BSAM involves you in working with blocked records, it has the advantage that input and output can be overlapped with other processing by the use of EVENT variables. The following paragraphs discuss the creation and accessing of PS data sets using QSAM. Most of the information given also applies when using BSAM; for a discussion of the special considerations concerning BSAM, see "Physical Sequential Data Sets (BSAM Access)," in this section. For further discussion of QSAM and BSAM, see IBM System/360 Time Sharing System: Data Management Facilities, GC28-2056.

Creation of Physical Sequential Data Sets

The information required for the creation of a PS data set is more extensive than for a VS data set. The user must supply information concerning

- Device Type
- Volume Serial Number
- Labeling (tape only)
- Record Format
- Record Length
- Block Sizes

Additional information is required if the user wishes to override standard system values (for example, space allocation, tape density, etc.).

Some of the information can be supplied by the PL/I program (ENVIRONMENT attribute of the file declaration), implied from other values (for example, for unblocked records, the record length can be implied by the format and block-size), or supplied by the PL/I library routines by default.

Table 16 specifies possible sources of essential information required for the creation of a new PS data set using QSAM. For a complete discussion of the DDEF command, see Appendix D.

Accessing a Physical Sequential Data Set (QSAM)

The data management information required for accessing a PS data set is considerably simplified if the data set is cataloged after creation. Cataloging a data set preserves information concerning:

- Device Type
- Volume Serial Number
- Labeling and Density (tape only)
- Record Format
- Record Length
- Block Size

If the data set is uncataloged, information concerning the location of the data set is

Table 16. Specification of PS (QSAM) Data Set Characteristics

Characteristic	DDEF Command	PL/I Program	Default Value
DDNAME=	File-Name Title-Name	File-Name Title-Name	Name
DSORG=	PS	--	None
DSNAME=	Data Set Name	--	None
DISP=	OLD	--	None
DCB=			
RECFM	{F, V or D, U FB, VB or DB}	ENVIRONMENT option	U
LRECL	value		None
BLKSIZE	value		None
For Disk			
UNIT=	(DA, {2311 2314})	--	
VOLUME	(PRIVATE ,serial no)	--	System or User-Defined Defaults
For Tape			
UNIT=	(TA, {9 7})	--	
VOLUME=	(Private , serial no)	--	
LABEL=	(file-no, {NL SL SUL})	--	

always required (that is, UNIT, VOLUME). Further information is then required only in the case of unlabeled tapes. Data sets on direct access devices or labeled tapes have format information contained in control blocks (DSCBs or header labels) kept with the data.

The existing data set can be accessed in several ways, depending upon the way the associated file is opened.

INPUT: When opened for INPUT, the file is positioned at the first record and the records are presented in sequence. After the last record is read, a further read causes an ENDFILE condition to be signaled for the file.

For tape devices only, it is possible to read the records in reverse order. The file must be opened and processed beforehand, and the LEAVE option must be specified in its ENVIRONMENT attribute. When reopened for input, the BACKWARDS attribute must be specified. This method of tape operation is applicable for all record formats.

OUTPUT: When opening an existing PS data set for OUTPUT, the disposition DISP=MOD should be specified if you want to add records at the end of the data set. Failure to do this causes the data set to be overwritten.

Note: For tape data sets, only the last file on the tape can be extended in this manner. Otherwise any following file becomes unreadable.

UPDATE: Only data sets on direct access devices can be opened for update. The records must be read and rewritten without changing their length.

Track Overflow

If the volume is on a direct access device, the blocksize can be greater than the size of a track. In such a case, it is necessary to specify track overflow for the data set. (Note: Track overflow requires a hardware feature that may not be available on the access devices at your installation.) Track overflow can be specified in the ENVIRONMENT attribute (TRKOFL) or in the record format (RECFM) DCB subparameter. Examples:

```
RECFM=UT    format-U with track overflow

RECFM=FBT   format-FB with track overflow
```

If the blocksize is smaller than the track size, the use of track overflow is optional.

Accessing a Physical Sequential Data Set (BSAM)

When a CONSECUTIVE file is given the UNBUFFERED attribute, it can only be used to access PS data sets using the BSAM access method. The use of BSAM causes complete physical records to be assigned to the program variables on input and inserted on output. Any blocking or deblocking of the records must be done by the PL/I program. The use of BSAM allows the possibility of overlapping the I/O operations with other processing. See "Synchronous I/O Using BSAM," in this section. Except for these considerations, there are no major differences between QSAM and BSAM.

SYNCHRONOUS I/O USING BSAM: When using BSAM to access a PS data set, it is possible to overlap I/O with other processing. This overlapped processing occurs when an EVENT variable is specified in the I/O statement of the PL/I program. (For a complete discussion of the use of EVENT variables in I/O processing, see PL/I Language Reference Manual.) The maximum number of I/O events that can be outstanding for a file at any instant must be supplied as an additional data management parameter. This value is called NCP (number of channel programs). It can be specified in the ENVIRONMENT attribute of the PL/I Program (NCP(n)) or in the DCB parameter of the DDEF command (NCP=n). A similar rule applies for the BUFNO data management parame-

ter. This specifies the number of buffers allocated to the file. Like NCP, this value must not be less than the maximum number of outstanding I/O events that can occur. BUFNO can be specified in the ENVIRONMENT attribute, in the form BUFFERS (n), or in the DDEF command, in the form BUFNO=n.

INDEXED FILES

Table 17 shows the types of INDEXED files that you can specify and the I/O statements that can you can use with them. A RECORD file declared to have INDEXED organization can only be used to access VI (virtual indexed sequential) data sets. Each record in the data set is identified by a key that is recorded with it. A key is a string of not more than 255 characters; all the keys in a data set must have the same length and the same relative position in the record. The records are arranged according to the collating sequence of their keys, which facilitate the direct (nonsequential) retrieval, addition, and deletion of the records.

INDEXED files can be accessed by the PL/I program in two alternative modes, SEQUENTIAL or DIRECT. SEQUENTIAL processing of INDEXED files is similar to the processing for CONSECUTIVE files. Records can be read in ascending key sequence from the first record onwards without specifying any key information; or the data set can be positioned to a particular record (using the KEY option) and then further records read in sequence. To update an INDEXED file sequentially, each record must first be read and then rewritten. There is no restriction for format-V records that the new record must be the same size as the old record. For DIRECT processing of an INDEXED file, there are no restrictions on the use of I/O statements; records can be added using the WRITE statement or replaced using the REWRITE statement, without previous issuance of a READ statement. To simplify programming for INDEXED files, the PL/I library routines treat indexed files

Table 17. Characteristics of Indexed Files

Organization	Access	Buffering	Mode	Statement	Options	Access Method	Record Format
ENVIRONMENT (INDEXED)	SEQUENTIAL	BUFFERED	INPUT	READ	INTO|SET IGNORE KEY|KEYTO	VISAM only	F, V
			OUTPUT	WRITE	FROM KEYFROM		
				LOCATE	SET KEYFROM		
			UPDATE	READ	INTO|SET IGNORE KEY|KEYTO		
				REWRITE	FROM		
				DELETE			
	DIRECT	UNBUFFERED	INPUT	READ	INTO KEY		
			OUTPUT	WRITE	FROM KEYFROM		
			UPDATE	READ	INTO KEY		
				WRITE	FROM KEYFROM		
				REWRITE	FROM KEY		
				DELETE	KEY		

having initial keys (see Appendix D, record format diagrams) as different from files having embedded keys.

Initial and Embedded Keys

When the keys are positioned at the beginning of the record (i.e., RKP=0 for format-F records or RKP=4 for format-V records), they are initial keys. In all other cases, they are embedded keys.

INITIAL KEYS: On output the PL/I library routines automatically concatenate the key variable (KEYFORM option) and the data variable (FROM option) before the record is written out to the data set. Thus, you can consider the key and data as completely independent variables. Conversely, on input, the data section of the record is assigned to the data variable (INTO or SET option) and the key is assigned to the key variable (KEYTO option).

EMBEDDED KEYS: For this form of processing, you are responsible for inserting the key at the appropriate position in the record before issuing a write statement. The PL/I library routines then compare the insertion with the specified key (KEYFROM option). Inequality results in a KEY error. Or the KEYTO option can be used to extract a copy of the key.

Creating an Indexed Data Set

The associated file can be opened for SEQUENTIAL OUTPUT or DIRECT OUTPUT. Sequential processing is more efficient than DIRECT, but the records must be presented in ascending key sequence. Failure to do this raises the KEY condition. For DIRECT OUTPUT processing, the records can be presented in any order; the only error that can occur, besides a key-specification error, is supplying of a duplicate key.

The data management parameters required when creating a new VI data set are shown in Table 18.

Table 18. Specification of VI Data Set Characteristics

Characteristic	DDEF Command	PL/I Program	Default Value
DDNAME=	file-name title-name	file-name title-name	None
DSORG=	VI	--	See Note
DSNAME=	Data Set name	--	None
DCB=			
RECFM=	F,V	ENVIRONMENT	V
LRECL=	value	option	132(V) 128(F)
RKP=	value	--	4(V) 0(F)
KEYLEN=	value	--	7 (V or F)
DISP=	NEW	--	See Note

Note: DSORG defaults to OLD if the DSNAME is in the system catalog, and to NEW if the DSNAME is not in the system catalog.

Accessing an Indexed Data Set

The data management information required for accessing an existing indexed data set is

DDNAME = file or title-name

DSNAME = data set name

DISP = OLD (optional)

No further information need be supplied. If other data management parameters are supplied, they must be the same as the existing values; no change can be made to the values set at creation time.

Example of Indexed Data Set

Examples 14 and 15 in "Part III: Examples" illustrate the creation and updating of an INDEXED data set.

Programs can be checked out using the program control system or the debugging facilities provided by PL/I.

PROGRAM CONTROL SYSTEM

The program control system (PCS) is a subset of the TSS/360 command system. Table 19 summarizes the PCS facilities available to the PL/I user.

Table 20 lists the restrictions on using PCS that apply only to you as a PL/I user or are of special interest to you. For further information, see Command System User's Guide.

Example:

```
LOAD    SIMPLE
```

The system loads module SIMPLE and modules invoked by SIMPLE, but does not initiate execution.

```
AT    PROC5;STOP
```

The system types the statement number assigned to the AT command.

```
SIMPLE
```

The system initiates execution of SIMPLE, and notifies you when control arrives at external procedure PROC5.

```
DISPLAY    0:15R
```

The system displays the contents of your general-purpose registers at your terminal.

Table 19. Program Control Commands and Their Functions

Command	Function
LOAD	Places a program in your virtual storage without initiating execution.
UNLOAD	Removes specified program from your virtual storage.
CALL	Loads and passes parameters to a program and execute.
GO	Resumes execution of previously interrupted program. See Appendix B.
REPEAT	After attention interruption, repeats last nonprompting message. See Appendix B.
BRANCH	Dynamically changes control path of program or resumes execution at a different location. (Resembles the PL/I GOTO statement.)
AT	Informs you when execution of program has reached designated instruction location, or designates instruction location at which rest of command statement is to be executed.
REMOVE	Selectively deletes previously entered command statements that include AT.
IF	Makes following command statement conditional.
SET	Changes contents of machine registers, values of program variables, virtual storage locations, or command symbols. (Command symbols are explained in Command System User's Guide.)
DISPLAY	Presents values of variables, contents of machine registers, and specified virtual storage locations to your SYSOUT.
DUMP	Presents values of variables, contents of machine registers, and specified virtual storage locations to task's PCSOUT data set.
STOP	Interrupts execution of your program and displays instruction location where interruption was handled. (Used only in a command statement that includes an AT command.)

Table 20. Rules for Using Program Control Commands

Commands	Rules
LOAD	Specify a module name, external-procedure name, or external-procedure ENTRY name. The module and those of its subroutines that are not called explicitly will be loaded.
CALL	Specify the module name for the main procedure.
LOAD, CALL	A CSECT in the program is rejected if it has the name of a CSECT that is already loaded; an attempt by the program to execute the rejected CSECT executes the CSECT that is actually loaded. Modules in IVM, however, are "forgotten" if you logged on with an IVM code (the 8th LOGON parameter) of Y, and other modules that duplicate their CSECT names can be loaded or executed.
BRANCH, AT, IF, SET, DISPLAY, DUMP	A location in the program must be specified as an external name, external name with offset, or an actual address. If the CSECT is in IVM, and you logged on with an IVM code of Y, you must specify an actual address.
UNLOAD	Specify a module name or the name of any loaded CSECT; the entire module, as well as subroutines not shared by another loaded module, will be unloaded. UNLOAD cannot unload modules from IVM.
AT, SET	AT or SET cannot reference a public CSECT.
SET	SET cannot reference a read-only CSECT.
LOAD, GO	See Appendix B.

```
DDEF   PCSOUT,VI,DSNAME=DUMPDS

DUMP   PROC5

RELEASE   PCSOUT

PRINT   DUMPDS,,,EDIT
```

The system dumps the contents of PROC5 into DUMPDS and prints DUMPDS at the installation's printer.

Accessing Static Internal Control Sections

You can obtain the address of a static internal control section with the command:

```
DISPLAY procedure-name.(X'10',4)
```

Then you can examine and change the static internal control section's contents, using the DISPLAY, DUMP, and SET commands.

PL/I DEBUGGING FACILITIES

Certain language features are provided in PL/I to assist you in debugging your program. The facilities include:

- Control over interruptions and error handling
- The ability to obtain a trace of active procedures
- Symbolic output
- Communication with the program during execution

The use of specific language features provides the debugging facilities; in addition, you can use your own techniques, such as inserting PUT statements at selected points.

When you are satisfied that your program is working correctly, you should remove debugging statements from your source program and then recompile to produce an optimum object program ready for execution.

CONTROL OF INTERRUPTION AND ERROR HANDLING

Some conditions can be enabled or disabled by means of the condition prefix, full details of which can be found in IBM System/360 Time Sharing System: PL/I Language Reference Manual.

In addition, you can specify your own exit (to be taken when a particular condition occurs), or you can cause an interruption by means of the SIGNAL statement. In particular, attention should be paid to the

CHECK condition, as this enables you to maintain a close watch on any variables you want to nominate.

If you want to exercise control of a more general nature, you can make use of the ERROR condition and, in an "ON-unit" further analyze the program by means of the ONCODE and ONLOC functions.

Standard system action for an ERROR condition causes the FINISH condition to be raised.

ON-CODES

The ONCODE built-in function can be used in any ON-unit to determine the nature of the error or condition that caused entry into that ON-unit.

An ON-unit, which has been established by the execution of an ON statement, is entered when the associated ON-condition is raised during execution of PL/I-compiled code or of a PL/I library module. Thus, for example, a FIXEDOVERFLOW ON-unit would be entered whenever any of the conditions occur for which the language demands the raising of the FIXEDOVERFLOW condition.

Two ON-conditions, ERROR and FINISH, require special explanation. The ERROR condition is raised:

1. Upon execution of a SIGNAL ERROR statement.

2. As a result of system action for those ON-conditions for which the language specifies system action to be "comment and raise the ERROR condition."

3. As a result of an error (for which there is no ON-condition) occurring during program execution.

The FINISH condition is raised:

1. Upon execution of a SIGNAL FINISH, STOP, or EXIT statement.

2. Upon normal completion of the MAIN procedure of a PL/I program.

3. Upon completion of the action associated with the raising of the ERROR condition, except when a GO TO statement in the ON ERROR unit has resulted in transfer of control out of that unit.

As a general rule, the value of the ON-code returned by the ONCODE function is that of the specific condition that caused entry into the ON-unit. Thus, in an ON CONVERSION unit, you can expect an ON code corresponding to one of the conversion conditions that cause the CONVERSION condition to be raised in PL/I. However, this is not necessarily true when executing an ON ERROR or an ON FINISH unit; the values are as follows:

1. When entered as a result of a SIGNAL ERROR or a SIGNAL FINISH, STOP or EXIT statement, or as a result of normal termination, the ON code values are those of ERROR or FINISH respectively.

2. When entered for any other reason, the ON code value is that associated with the error or condition that originally caused the ERROR condition to be raised.

Several separate but related occurrences can cause a particular PL/I ON-condition to be raised. For example, the TRANSMIT condition can be raised:

1. By execution of a SIGNAL TRANSMIT statement

2. By occurrence of an input TRANSMIT error

3. By occurrence of an output TRANSMIT error

Although it is often useful to know precisely what caused an ON-condition to be raised, at times it is sufficient simply to know which ON-condition was raised. This applies particularly if the ONCODE function is used in an ERROR ON-unit after system action has occurred for an ON-condition. The ON codes have therefore been grouped, each group containing codes associated with a particular ON-condition.

From time to time it may become necessary or desirable to add new ON-codes into a group. Perhaps a group containing only one ON-code may be expanded. This fact must be remembered when the ONCODE function is used to determine if a particular PL/I ON-condition has been raised. It is important to test to see whether the ON-code is within the range specified, even if there is only one ON-code in the range; otherwise, when a new set of library modules is used, it may become necessary to recompile the program.

When a group contains only one ON-code value, it is impossible to test specifically for the signaled condition. With more than one ON-code in the group, the first in the group represents the signaled condition.

The ON-code groups and their ranges are shown in Tables 21 and 22. (Language ON-conditions are shown in capitals, others in lowercase letters.)

Table 21. Main ON-Code Groupings

Range	Group
3-5	As for 1000-9999
10-199	I/O ON-conditions
300-399	Computational ON-conditions
500-549	Program checkout conditions
600-899	Conversion conditions
1000-9999	Error conditions (also 3-5)

Table 22. Detailed ON-Code Groupings

Range	Group
0	ONCODE
3	Source program error
4	FINISH
9	ERROR
10-19	NAME
20-39	RECORD
40-49	TRANSMIT
50-69	KEY
70-79	ENDFILE
80-89	UNDEFINEDFILE
90-99	ENDPAGE
100-299	(Unallocated)
300-309	OVERFLOW
310-319	FIXEDOVERFLOW
320-329	ZERODIVIDE
330-339	UNDERFLOW
340-349	SIZE
350-359	STRINGRANGE
360-369	AREA
370-499	(Unallocated)
500-509	CONDITION
510-519	CHECK
520-529	SUBSCRIPTRANGE
530-599	(Unallocated)
600-899	CONVERSION
900-999	(Unallocated)
1000-1199	I/O errors
1200-1499	(Unallocated)
1500-1699	Data processing errors
1700-1999	(Unallocated)
2000-2099	Unacceptable statement errors
2100-2999	(Unallocated)
3000-3499	Conversion errors
3500-3799	(Unallocated)
3800-3899	Structure and array errors
3900-3999	Tasking errors
4000-8090	(Unallocated)
8091-8199	Program interrupt errors
8200-8999	(Unallocated)
9000-9999	System errors

The ON-codes and their associated conditions and errors are shown below.

Code	Condition/Error
0	ONCODE function used out of context
3	Source program error
4	FINISH
9	ERROR
10	NAME
20	RECORD (signaled)
21	RECORD (record variable smaller than record size)
22	RECORD (record variable larger than record size)
23	RECORD (attempt to write zero length record)
24	RECORD (zero length record read)
40	TRANSMIT (signaled)
41	TRANSMIT (output)
42	TRANSMIT (input)
50	KEY (signaled)
51	KEY (keyed record not found)
52	KEY (attempt to add duplicate key)
53	KEY (key sequence error)
54	KEY (key conversion error)
55	KEY (key specification error)
56	KEY (keyed relative record/track outside data set limit)
57	KEY (no space available to add keyed record)
70	ENDFILE
80	UNDEFINEDFILE (signaled)
81	UNDEFINEDFILE (attribute conflict)
82	UNDEFINEDFILE (access method not supported)
83	UNDEFINEDFILE (blocksize not specified)
84	UNDEFINEDFILE (file cannot be opened, no DDEF command)
85	UNDEFINEDFILE (error initializing REGIONAL data set)
90	ENDPAGE
300	OVERFLOW
310	FIXEDOVERFLOW
320	ZERODIVIDE
330	UNDERFLOW
340	SIZE (normal)
341	SIZE (I/O)
350	STRINGRANGE
360	AREA raised in ALLOCATE statement
361	AREA raised in assignment statement
362	AREA signaled
500	CONDITION
510	CHECK (label)
511	CHECK (variable)
520	SUBSCRIPTRANGE
600	CONVERSION (internal) (signaled)
601	CONVERSION (I/O)
602	CONVERSION (transmit)
603	CONVERSION (error in format-F input)
604	CONVERSION (error in format-F input) (I/O)
605	CONVERSION (error in format-F input) (transmit)
606	CONVERSION (error in format-E input)
607	CONVERSION (error in format-E input) (I/O)
608	CONVERSION (error in format-E input) (transmit)
609	CONVERSION (error in format-B input)
610	CONVERSION (error in format-B input) (I/O)
611	CONVERSION (error in format-B input) (transmit)
612	CONVERSION (character string to arithmetic)
613	CONVERSION (character string to arithmetic) (I/O)

614	CONVERSION (character string to arithmetic) (transmit)
615	CONVERSION (character string to bit string)
616	CONVERSION (character string to bit string) (I/O)
617	CONVERSION (character string to bit string) (transmit)
618	CONVERSION (character to picture)
619	CONVERSION (CHARACTER TO PICTURE) (I/O)
620	CONVERSION (character to picture) (transmit)
621	CONVERSION (format-P input - decimal)
622	CONVERSION (format-P input - decimal) (I/O)
623	CONVERSION (format-P input - decimal) (transmit)
624	CONVERSION (format-P input - character)
625	CONVERSION (format-P input - character) (I/O)
626	CONVERSION (format-P input - character) (transmit)
627	CONVERSION (format-P input - sterling)
628	CONVERSION (format-P input - sterling) (I/O)
629	CONVERSION (format-P input - sterling) (transmit)
1000	Attempt to read output file
1001	Attempt to write input file
1002	GET/PUT string length error
1003	Unacceptable output transmission error
1004	Print option on non-print file
1005	Message length for DISPLAY statements zero
1006	Illegal array item for data-directed input
1007	REWRITE not immediately preceded by READ
1008	GET STRING -- unrecognizable data name
1009	Unsupported file operation
1010	File type not supported
1011	Inexplicable I/O error
1012	Outstanding READ for update exists
1013	No completed READ exists - incorrect NCP value
1014	Too many incomplete I/O operations
1015	EVENT variable already in use
1016	Implicit-OPEN failure - cannot proceed
1017	Attempt to REWRITE out of sequence
1018	ERROR condition raised if end-of-file is encountered before the delimiter while scanning list-directed or data-directed input, or if the field width in the format list of edit-directed input would take the scan beyond the end-of-file.
1019	Attempt to close file not opened in current task
1500	Short SQRT error
1501	Long SQRT error
1504	Short LOG error
1505	Long LOG error
1506	Short SIN error
1507	Long SIN error
1508	Short TAN error
1509	Long TAN error
1510	Short ARCTAN error
1511	Long ARCTAN error
1514	Short ARCTANH error
1515	Long ARCTANH error
1550	Invalid exponent in short float integer exponentiation
1551	Invalid exponent in long float integer exponentiation
1552	Invalid exponent in short float general exponentiation
1553	Invalid exponent in long float general exponentiation
1554	Invalid exponent in complex short float integer exponentiation
1555	Invalid exponent in complex long float integer exponentiation
1556	Invalid exponent in complex short float general exponentiation
1557	Invalid exponent in complex long float general exponentiation
1558	Invalid argument in short float complex ARCTAN or ARCTANH
1559	Invalid argument in long float complex ARCTAN or ARCTANH
2000	Unacceptable DELAY statement
2001	Unacceptable use of the TIME built-in function
3000	Format-E conversion error
3001	Format-F conversion error
3002	Format-A conversion error
3003	Format-B conversion error
3004	Format-A input error
3005	Format-B input error
3006	Picture character string error
3798	ONSOURCE or ONCHAR out of context
3799	Improper return from CONVERSION ON-unit
3800	Structure length ≥ 16**6 bytes
3801	Virtual origin of array ≥ 16**6 or ≤ -16**6
3900	Attempt to wait on inactive and incomplete event
3901	Task variable already active
3902	Event already being waited on
3903	Wait on more than 255 incomplete events
3904	Active event variable as argument to COMPLETION pseudo-variable
3905	Invalid task variable as argument to PRIORITY pseudo-variable
3906	Event variable active in assignment statement
3907	Event variable already active
3908	Attempt to wait on an I/O event in wrong task
8091	Invalid operation
8092	Privileged operation
8093	EXECUTE statement executed
8094	Protection violation
8095	Addressing interruption
8096	Specification interruption
8097	Data interruption
9000	Too many active ON-units and entry parameter procedures
9002	Invalid free storage (main procedure)

TRACE OF ACTIVE PROCEDURES

A trace of active procedures can be obtained by use of the SNAP option in an ON statement. However, this technique has the limitation that it records only procedures active at the time when the condition occurs, because of the use of dynamic storage; when the storage is released it is immediately available for some other use, and so cannot be used to maintain a full trace. If a full flow trace is required, then this should be programmed, either by means of the SIGNAL statement in association with an ON statement and ON-unit, or by specifying all procedure names in a CHECK list with the appropriate action in an ON-unit.

The format of the SNAP output is either of the following:

1. CONDITION xxxx OCCURRED AT OFFSET ± hhhhh FROM ENTRY POINT E1

2. CONDITION xxxx OCCURRED AT OFFSET ± hhhhh FROM ENTRY POINT OF xxxx ON-UNIT

followed by

CALLED FROM PROCEDURE WITH ENTRY POINT E2
CALLED FROM PROCEDURE WITH ENTRY POINT E3
etc., etc.

If the statement number compiler option is specified, the SNAP output message also contains IN STATEMENT nnnnn immediately following the word OCCURRED in the first line, or after the word CALLED in subsequent lines. The notation nnnnn gives the number of the statement in which the condition occurred.

The characters that replace xxxx are an abbreviated form of the name of the ON-condition that has occurred (the abbreviations are given in Table 23). hhhhh is a hexadecimal offset; E1, E2, etc., are entry point names indicating the actual entry points used to enter the procedure in which the condition occurred, or from which the next named entry point was called.

If a condition occurs in an ON-unit, then the entry point name in the second line will be that of the precedure from which the ON-unit was entered, not necessarily the procedure in which the ON-unit is situated.

If SNAP SYSTEM has been specified in an ON statement, the system action message is printed, followed by the trace of active procedures:

Table 23. Abbreviations for ON-Conditions

Condition	Abbreviation
OVERFLOW	OFL
SIZE	SIZE
FIXEDOVERFLOW	FOFL
SUBSCRIPTRANGE	SUBRG
CHECK	CHCK
CONDITION	COND
FINISH	FIN
ERROR	ERR
ZERODIVIDE	ZDIV
UNDERFLOW	UFL
STRINGRANGE	STRG
NAME	NAME
RECORD	REC
TRANSMIT	TMIT
KEY	KEY
ENDFILE	ENDF
UNDEFINEDFILE	UNDF
CONVERSION	CONV
ENDPAGE	ENDP

CALLED FROM PROCEDURE WITH ENTRY POINT E2
etc.

The one exeception is the case of SNAP SYSTEM for a CHECK condition. In this case a standard SNAP message is written, followed by the standard system action printout for the CHECK condition.

COMMUNICATION WITH THE PROGRAM

Symbolic Output Using GET and PUT Statements

Section 7 explains how the GET and PUT statements can be used to direct I/O to SYSIN/SYSOUT.

Execution of a GET DATA statement causes the system to read from SYSIN. SYSIN should provide a series of assignment statements that assign values to variables declared within the block containing the GET statement. Methods of entering a SYSIN data set in nonconversational mode are described under "System Input File -- SYSIN" in Section 8, and in Part III, Examples 3 and 4.

Execution of a PUT DATA[data list] statement causes the system to write on SYSOUT. Use of a data list with a PUT DATA statement is optional. Execution of a PUT DATA statement with a data list causes the system to write each specified variable and its value on SYSOUT, in assignment-statement form. Execution without a data list causes the system to write the contents of each static variable in your program.

For examples of terminal I/O using the simple GET and PUT statements, see "Section 3: Basic Data Manipulation." For an example of nonconversational input using the simple GET statement, see Example 4 in "Part III: Examples."

You can also use the data-directed I/O feature with files other than SYSIN and SYSOUT. This feature can be used instead of, or in addition to, the CHECK condition handling. Refer to PL/I Language Reference Manual for a full description of this feature.

The DISPLAY Statement

The DISPLAY statement provides an additional means of communicating with the program while it is being executed.

A message can be displayed in either of two forms:

1. Without the REPLY option, which gives the specified character string unaltered.

2. With the REPLY option, which gives the specified character string preceded by a two-digit code generated by the system. Use this code as a prefix to the reply message. The reply message can be given in either a conversational or nonconversational task; it must come from the SYSIN data set.

In TSS/360, the EVENT option of the DISPLAY statement can be used only to suppress flagging of the event variable as complete.

User-Requested Dump

An additional debugging feature is the ability to obtain a storage dump at any point in the program. A dump is obtained by one of the statements:

```
CALL IHEDUMC[(argument)];
 dump and continue execution

CALL IHEDUMP[(argument)];
 dump and terminate execution
```

The argument is optional; if used, it must be declared as ENTRY(FIXED BIN(31,0)). This integer appears in the heading of the dump as ID = n. It must be in the range 0 through 27.

The operation of the dump routines depends upon whether the invoking task is in conversational or nonconversational mode.

CONVERSATIONAL DUMPS: The following information is printed at the terminal:

- Dump header.
- Addresses of the DCB (data control block) and certain other PL/I library control blocks, for all opened files and for the current file.
- Addresses of all save areas currently in use, starting with the last used. For each save area, the calling and return addresses of the routines are given.

At this point, the message

PAUSE

is printed at the terminal and the task is put into command mode. You can now use PCS commands to obtain any further information you require.

To continue execution, issue the GO command; the dump routine will cause the program to either continue or terminate, depending on whether IHEDUMC or IHEDUMP was called.

NONCONVERSATIONAL DUMP: The system prints the same information that it prints for a conversational dump; in addition, it gives dumps of all pages containing the supplied addresses.

RETURN CODES

Return codes are set by use of the statement CALL IHESARC.

SECTION 12: INTERFACE BETWEEN PL/I AND ASSEMBLER-LANGUAGE PROGRAMS

You may want to write subroutines in assembler language and then call them from a PL/I program -- for example, to provide a new function. You may want to write your own versions of PL/I library subroutines. More rarely, you may want to call a PL/I subroutine from an assembler program.

For any but simple subroutines, it is recommended that you become familiar with IBM System/360 Time Sharing System: PL/I Subroutine Library Program Logic Manual, GY28-2052, hereinafter referred to as the library PLM. The general notes given here may help you decide whether your subroutine is simple.

ASSEMBLER SUBROUTINES CALLED FROM PL/I PROGRAMS

ABSENCE OF PSECTS

There [illegible] in PL/I, and no R-type addre[illegible] constants. You cannot write a subroutine with a PSECT, because the PL/I calling program will not supply the PSECT address in word 19 of its save area. All PL/I control sections are CSECTs.

ENTRY TO THE SUBROUTINE

At entry to a subroutine called from PL/I, these facts are known:

- Register 15 contains the address of the subroutine entry point, and can therefore be used as a base register.
- Register 14 contains the address of the return point in the PL/I calling program.
- Register 13 contains the address of the (PL/I) calling program's save area, in which the subroutine should save the general purpose registers on entry.
- Register 12 contains the address of the PRV communication area. This register must have the same value on return as it had on entry.
- Register 1 contains the address of the parameter list, if any.

Format of Parameter List

The parameter list built by the PL/I calling program consists of one word for each parameter specified in the CALL statement. Each word contains a three-byte address. (Remember that PL/I works only in a 24-bit addressing mode.) The last word in a parameter list contains an X'80' flag in the first byte, so that varying numbers of parameters can be used.

Depending on the attributes of each parameter, the corresponding address can be that of the parameter itself, or that of a control block containing information that describes the parameter's characteristics; for example, consider the PL/I statements:

```
DCL  PAR1 CHAR(8),PAR2 FIXED DECIMAL,
PAR3 BINARY,PAR4(5);
   .
   .
   .
CALL  SUBR (PAR1,PAR2,PAR3,PAR4);
   .
   .
   .
```

The CALL statement is expanded to the equivalent of this assembler code:

```
LA      8,DV..PAR1

ST      8,WS1.1

LA      8,PAR2

ST      8,WS1.1+4

LA      8,PAR3

ST      8,WS1.1+8

LA      8,DV..PAR4

ST      8,WS1.1+12

OI      WS1.1+12,X'80'

LA      1,WS1.1

L       15,A..SUBR

BALR    14,15
```

(The parameter list is built in a location WS1.1 in the calling program's dynamic storage area.)

PAR1 is defined as a character string. In PL/I a string item is described by a control block called a string dope vector. The address loaded into the parameter list for this call is that of the dope vector for this string, DV..PAR1. PAR2 is a

simple arithmetic item and is addressed directly. So is PAR3. PAR4 is an array, and this too is addressed through a control block, in this case an array dope vector.

Other types of data may have control blocks, and reference should be made to the library PLM for a complete description. When in doubt, it is useful to write a dummy PL/I calling program with the parameters declared fully, and examine the listing to see how the parameter list is built.

Note that a hex '80' is OR'd into the last item in the parameter list to indicate that it is the last parameter.

DATA REPRESENTATION

PL/I has its own way of representing arithmetic scalars, strings, arrays, and structures. Representation of an arithmetic data item depends upon its scale and base. A string is addressed through its string dope vector (SDV), an array or structure is addressed through its array dope vector (ADV), and a structure is addressed through its structure dope vector. See Section III of the library PLM for the format of each of these dope vectors.

ENVIRONMENT

24-Bit Addressing

A PL/I program cannot work with an address that is more than three bytes long. An assembler subroutine called from a PL/I program must take this into account.

Storage Management

If you want the subroutine to be shareable, all working must be done in registers or in dynamic virtual memory; that is, without a PSECT. Rather than do your own GETMAIN, you should use a piece of virtual memory that has already been obtained by PL/I and is being controlled by a library routine. The way to do this is to code:

```
IHEPRV  VDA,BR

BALR    LR,BR
```

IHEPRV is a PL/I macro, defined in TSS/360's assembler macro library (ASMMAC), that puts the contents of a particular pseudo-register into the specified real register. In the instruction above, it puts the contents of IHEQVDA into register 15. IHEQVDA is defined in part of the expansion of the PL/I macro IHELIB, which contains a lot of useful definitions (for example, the PL/I standard names for registers and control blocks). It is recommended that you include IHELIB, preferably as the first instruction, in the assembler subroutine. IHEQVDA is a slot or pseudo-register in a dynamic communication area called the pseudo-register vector (PRV). It contains the address of the entry point to a routine that obtains a block of virtual memory.

At entry, register 0 must contain the number of bytes of virtual memory required. On return, register 1 contains the address of the beginning of the assigned variable data area (VDA). To return such storage a different pseudo-register, IHEQFVD, is used to free the VDA. Thus:

```
IHEPRV  FVD,BR

BALR    LR,BR
```

frees the latest VDA. The library PLM describes the format of a VDA, the first eight bytes of which are reserved for flags, length, and chaining.

SAVE AREAS: Should the assembler subroutine require a save area of its own, library workspace can be used. There are different levels of library workspace called LW0, LW1 through LW4. If the subroutine is called only directly from compiled code, level zero should be used. If you are replacing an existing library routine, the level used should be one higher than the highest used in any calling library routine. (The library PLM shows the calling relationships of the library routines.) The address of the workspace can be obtained by using the library macro IHESDR; for example:

```
IHESDR  LW0,WR
```

puts the address of the level zero workspace in register WR.

PSEUDO-REGISTERS: The macro IHELIB defines the standard pseudo-registers used by the PL/I library, and you can examine or change these by using the macro IHEPRV; for example:

```
IHEPRV  CFL,RA
```

loads the address of the current file's pseudo-register into register RA. To indicate an error condition,

```
IHEPRV  ERR,RA,OP=ST
```

stores the contents of register RA into the pseudo-register IHEQERR.

Nonstandard pseudo-registers defined by compiled code can be addressed from

assembler language subroutines through the use of DXD instructions and Q-type address constants.

The standard pseudo-registers and library macro instructions are described in appendixes of the library PLM.

Interruption Handling

Initialization of interruption handling is automatic for PL/I main programs and their subprograms. An interruption of the assembler language subroutine will be fielded by the PL/I library.

EXAMPLE: This is a small example to show how assembler language subroutines can be accessed from a PL/I main program. The module shown is designed to accept a character string argument of any size and write it in the SYSPRINT data set. The actual output operation is done by the PL/I library module IHEPRTB, which is invoked by the assembler language subroutine (PRINTER).

Part 1

PRINTER is invoked by a standard CALL statement in PL/I, using the entry name to which control is to be passed.

```
TEST:  PROCEDURE OPTIONS(MAIN);

       DECLARE A1 CHAR(35);

          A1 is the variable to be written on SYSPRINT.

       C = 2;

       D = 4;

       A1 = (B*(C**2))/D;

          The last assignment statement calculates a value in floating
          point and converts it to a character string.

       CALL PRINTER(A1);

          To use facilities provided by the library routine to put out
          error messages, it is necessary to call the assembler subrou-
          tine PRINTER, shown later in this section.

       END TEST;
```

Part 2

This subroutine illustrates some of the functions necessary in order to communicate between a PL/I main program and an assembler subprogram. The comments included explain fully what has been done, and indicate what might have been done.

While this is a trivial example, it demonstrates most of the linkage problems. Note that the standard SAVE macro could have been used in this subroutine.

This subroutine is limited to extracting the address and current length of a character string from its dope vector, and presenting these items as arguments to a library print routine.

```
PRINTER   CSECT

          USING *,15

          BC    15,PRINT1

          DC    AL1(7)
```

Length of the character string that follows. Its purpose is to enable the PL/I SNAP option to print a trace if required.

```
          DC    C'PRINTER'
PRINT1    STM   14,11,12(13)
```

Saves registers in the caller's save area. Note that this subroutine is prepared to preserve register 12. In the event of an interruption, the PL/I execution error package would be invoked.

```
          DROP  15
          BALR  10,0
          USING *,10
```

The last two instructions establish addressability for the rest of the control section. The same system is used by the compiler-produced object code.

Use is now made of the PL/I library in order to obtain a save area. This is done dynamically, since the same is done by PL/I object code. If there is no requirement for this code to be reentrant or recursive, then storage could be reserved for it by means of DC's or DS's.

```
          LA    0,100     LENGTH OF DYNAMIC SAVE AREA
          L     15,ADDR1  GET ADDRESS OF LIBRARY GETDSA
          BALR  15,14     ROUTINE, AND BRANCH TO IT.
```

It is now necessary to initialize the save area. Not much work is done in this example, but PL/I object code usually performs many more functions. It is not absolutely necessary to do more than is indicated here, but if you wish to observe all the PL/I conventions then considerably more code would be required.

```
          MVI   0(13),X'80'
```

This instruction moves in the flag byte as required by the library FREEDSA routine.

At this stage, the saving conventions have been dealt with and attention can be given to parameters.

```
          L     14,0(0,1)
```

Gets the address of the argument -- note that this is not the string itself but its dope vector.

```
          L     1,0(0,14)
```

Gets the address of the string from the dope vector.

```
          LA    2,6(0,14)
```

Gets the address of the current length of the string from the dope vector.

```
          L     15,ADDR2  GET ADDRESS OF LIBRARY PRINT
          BALR  14,15     MODULE AND BRANCH TO IT.
```

Upon return, this subroutine has completed its task and now makes use of the library FREEDSA routine in order to release its dynamic storage (used as a save area), and to return to its caller.

```
         L      15,ADDR3    GET ADDRESS OF LIBRARY FREEDSA
         BCR    15,15     MODULE AND BRANCH TO IT
ADDR1    DC     V(IHESADA)     ADDRESS OF LIBRARY GETDSA RTN
ADDR2    DC     V(IHEPRTB)     ADDRESS OF LIBRARY PRINT RTN
ADDR3    DC     V(IHESAFA)     ADDRESS OF LIBRARY FREEDSA RTN
         END
```

PL/I SUBROUTINES CALLED FROM ASSEMBLER PROGRAMS

Generally speaking, a PL/I program (even a subroutine) expects to operate in a PL/I environment. This implies that initialization has been done, that data is in PL/I format, etc. This is particularly vital if the subroutine should call PL/I library routines.

If the assembler calling program has itself been called by a PL/I main program, or if you have called the PL/I initialization routine as shown below, then initialization will have been done. Where initialization is not done, you should not attempt to call any but the most straightforward PL/I [illegible] without a thorough knowledge of the library PLM.

INITIALIZATION ROUTINE

If you have not initialized PL/I but want to call a PL/I subroutine from assembler, you can call the initialization procedure yourself. The following routine is that actually used by PL/I compiled code to initialize execution. It is written as a separate CSECT.

```
         USING *,15
INIT     ST     1,REG1     SAVE PARAMETER LIST ADDRESS
         LA     1,REG1     SET UP SAPC PARAMETERS
```

The PL/I initialization routine is in the PL/I library, in module IHEWSAP at entry point IHESAPC. It expects register 1 to contain the address of a five-word parameter list, of which the first word (REG1) is a pointer to the parameter list for the program being initialized. At entry to the INIT routine, register 1 should contain this address.

```
         L      15, VSAPC    GET ADDRESS OF IHESAPC
         BR     15     ENTER INITIALIZATION ROUTINE
REG1     DC     F'0'
         DC     A      (return.) or DC V (RETURN POINT)
```

This is the address the initialization routine should go to when it is done. The address can be internal or external to your current CSECT; if external, a V-type address constant must be used.

```
         DC     V(IHEWCVC)
```

This is the address of the nonsharable portion of the PL/I library, which must be given to the initialization routine in order for it to function properly.

```
          CXD
VSAPC     DC    V(IHESAPC)     ENTRY POINT OF INITIALIZATION ROUTINE
```

EXAMPLE: This program consists of two modules. The first, ASMMOD, is an assembler language routine that calls the initialization routine, then calls a PL/I procedure and passes parameters. ASMMOD could do more work than this; its function here is only to show how the linkages to the initialization and PL/I routines are performed. The second module, PLIMOD, comprises the PL/I procedure.

```
INITIAL   CSECT                  MODULE IS 'ASMMOD'
          USING *,15
          ST  1,REG1             SAVE PARAMETER POINTER
          LA  1,REG1             ADDRESS PARAMETER LIST
          L   15,=V(IHESAPC)     ADDRESS INITIALIZATION ROUTINE
          BR  15                 SET UP PL/I ENVIRONMENT
          SPACE
REG1      DC  F'0'               REGISTER 1 SAVE AREA
          DC  A(LINK)            ENTRY POINT FOR RETURN FROM INITIALIZA-
                                 TION ROUTINE
          DC  V(IHEWCVC)         PL/I LIBRARY ADCON MODULE
          CXD
          LTORG
          EJECT
```

The assembler language routine that links to the PL/I procedure does not have to be in the same CSECT as the call to the initialization routine, or even in the same module. However, it is probably more convenient to put it in the same module.

First, the assembler language linking routine establishes addressability and deals with saving conventions as in Part 2 of the example under "Assembler Subroutines Called from PL/I Programs," in this section. There is no requirement to save register 12, since it is not modified by PL/I programs.

```
LINK      STM     14,11,12(13)     SAVE REGISTERS
          BASR    10,0             ESTABLISH BASE REGISTER
          USING   *,10             *
          LA      0,100            LENGTH OF DYNAMIC SAVE AREA (100 IS
                                   MINIMUM SIZE)
          L       15=V(IHESADA)    DSA ALLOCATION PROGRAM
          BASR    14,15            ALLOCATE DSA
          MVI     0(13),X'80'      SET FLAGS
```

```
          SPACE   5

          LA      1, PLILIST      ADDRESS PARAMETER LIST

          L       15,=V(PROG)     ADDRESS PL/I EXTERNAL PROCEDURE
          BASR    14,15           ENTER PL/I PROCEDURE
          SPACE
```

If the PL/I procedure executes a RETURN or END statement, control will be returned to the next instruction:

```
          GATWR   MESG,LEN        VERIFY RETURN TO THIS ROUTINE
          SPACE   5
          L       15,=V(IHESAFA)  PL/I END PROCESSOR
          BR      15              END OF JOB
          LTORG
LEN       DC      A(L'MESG)       MESSAGE LENGTH
PLIST     DC      A(FIRST)
          DC      A(SECOND)
MESG      DC      C'*LINK*        RETURN FROM PL/I PROGRAM *PROG*'
FIRST     DC      H'49'           FIRST PARAMETER
SECOND    DC      H'150'          SECOND PARAMETER
          END     INITIAL
PROG:     PROC(I,K);              /*MODULE IS 'PLIMOD'*/
          DCL(J,L)BINARY FIXED(15,3);
          J=SQRT(I);
          L=SQRT(K);
          PUT DATA(J,L);
          IF I=0 THEN STOP;
          RETURN;
          END;
```

Program execution:

```
ASMMOD

J=   7.0    L=   12.1;*LINK* RETURN FROM PL/I PROGRAM
                                                  *PROG*

QUALIFY ASMMOD;SET FIRST=150,SECOND=32768
```

The value assigned to SECOND is too large for the corresponding PL/I parameter, K, which has a default precision of 15. In this case, the sign bit is set; K will look like a negative number to the PL/I library, which will issue an error message.

```
ASMMOD
IHE200I IHESQS X LT 0 IN SQRT (X) IN STATEMENT 00003 AT OFFSET +000E8
   FROM ENTRY POINT PROG

SET SECOND =32767

ASMMOD

J=     12.1    L=    181.0;*LINK* RETURN FROM PL/I PROGRAM
                                                    *PROG*

DISPLAY 181*181

32761

SET FIRST=0
```

On the next invocation of the program, the PL/I STOP statement is executed and there is no return to entry point LINK.

```
ASMMOD

 J=    0.0    L=    181.0
```

The system prompts you for another command.

Notes on Passing Parameters

- Each parameter passed by the assembler language routine must have the internal form of the corresponding PL/I parameter. For example, if the PL/I parameter is BINARY FIXED(3), the PL/I program expects it to be a fixed-point halfword; if the PL/I parameter is DECIMAL FLOAT(6), it must be passed as a normalized short floating-point parameter. The internal forms of PL/I data are described in Section 3 of PL/I Language Reference Manual.

- When a character string is passed, the address in the parameter list points to a string dope vector. See Section III of the library PLM, or part two of the previous example in this section, for the content of this dope vector. The corresponding PL/I parameter should be declared as a varying-length string.

PART III: EXAMPLES

Part III is devoted to examples in which the dialog between you and the system appears (along with explanatory comments) as it would at the terminal. They are typical, but not exact, examples of system use. Unlike the examples throughout Part II, the examples in this part have not been system-tested. You may, therefore, observe certain minor differences between an example's description in Part III and the printout you obtain if you run the example itself. Use the examples, therefore, only as a learning device, and as models for designing your own work.

Commands and concepts are presented in an ordered sequence: the most necessary and basic ones appear first, and are reviewed in subsequent examples. The examples are designed so that the beginner should read them in sequence. Those familiar with the commands and concepts can use the examples for reference.

In these examples, lines typed by the system are headed SYS, lines you enter are headed YOU. Lines in which both the system and you enter something are headed S,Y.

EXAMPLE 1: INITIATING AND TERMINATING A CONVERSATIONAL TASK

In this example, you initiate a simple conversational task and then terminate it. The description of the example explains the keyboard entries required to converse with the system.

To begin a conversational task, make sure that the terminal is properly prepared (refer to instructions provided by your installation or to Section 4 of this publication). When you dial up the system or press the attention key for the first time in your task, the system assumes a log-on operation and responds with the current date and time. You then complete your log-on procedure by entering the operands of the LOGON command. During your dialog with the system, commands are not actually entered into the system until you press the return key at the end of the line containing the command.

```
YOU:  (press attention key or dial up system)
```

Note: From this point on, pressing the attention key halts current activity in most situations. Consult Appendix B for the specific action taken in each situation.

```
YOU:  LOGON ADUSERID,MYPASS*,24,ADACCT24,A
                                       9,A,,P
```

While typing the LOGON operands, you realize that you have entered your charge number incorrectly. Therefore, you backspace three characters, move the paper up one line by hand to avoid overtyping, and reenter the corrected portion of the charge number. You then complete the LOGON operands. If you wanted, you could have cancelled the entire line by typing a pound sign (#) and immediately pressing RETURN; then you would reenter the correct line.

```
SYS:  TSS/360 LEVEL 8.1
      LOGON OF TASKID=0020 IS AT 10:26 ON 3/31/71
```

After the system responds with the taskid assigned to this task and the current date and time, you can communicate with the system by entering commands.

Explanation of LOGON Operands

ADUSERID

First Operand -- User Identification

This operand is your full identification. It was assigned to you when you were joined to the system. Its first two characters identify the administrator who authorized your access to the system.

MYPASS*

Second Operand -- Password
This operand is a user-assigned code that provides protection against unauthorized use of your user identification. In conversational mode, you must supply a password if one has been assigned to you.

24

Third Operand -- Addressing
Specifies whether 24-bit or 32-bit addressing is to be used for this task. Note: To compile or run PL/I programs, you must log on with 24-bit addressing.

ADACCT29

Fourth Operand -- Charge Number
This operand is your charge or account number that was assigned to you by your administrator. The first two characters of your charge number also identify your administrator.

A

Fifth Operand -- Control Section Packing
This operand specifies whether control sections are to be packed (that is, not placed on separate pages), and the manner of packing to be used. The codes and their meanings are:

Code	Meaning
A	Pack all control sections.
P	Pack all prototype control sections (PSECTs).
O	Pack all control sections having neither public nor prototype attributes.
X	Pack all control sections except prototype control sections.
N	No packing; if the operand is defaulted, this code is assumed.

Sixth Operand -- Maximum External Storage
This operand specifies the maximum amount of external storage to be allocated to your task; you default this operand and use the installation default value.

P

Seventh Operand -- Pristine Mode
This operand allows you to log on with only the system-supplied defaults, synonyms, procdefs, and Character and Switch Table. Since you specified this operand as P, your user library is defined; if you had specified it as X, your user library would not be defined.

After logging you on, the system prints a single underscore and then backspaces; this is the standard signal that it is ready to receive your next command on the same line.

You decide to conclude your session, so you log off.

S,Y: LOGOFF

SYS (confirms your log-off request)

EXAMPLE 2: CREATING MULTIPLE VERSIONS OF THE SAME PROGRAM

Sometimes, you may want to keep more than one version of the same program, without attempting to give a unique module name for the second

version and unique names to all of its entry points. For example, you may create a new version of a program that you think is an improvement over the old version, yet you want to keep the old version until you are sure that the new one works. If the second version contains any names in common with the first version, you must place each version in a different library.

In this example, you compile a stand-alone module (a module that does not call another module and is not called by another module); then you create a slightly different version of this module, without erasing the first version.

Note the difference between the terms module and procedure. A module is the routine identified by the NAME operand of the PLI command; it contains one external procedure with up to 254 procedures. To change the operation of any of the procedures in the module, you must recompile the entire module. In this example, each of the modules contains only one procedure.

Having logged on, you create two versions of a stand-alone module named MOD, which contains a procedure named PROG.

PLI MOD

You invoke the PL/I compiler, indicating that the module name is to be MOD. The compiler, in turn, invokes the text editor, which prompts you with line numbers. Following the line numbers, you enter your PL/I statements.

```
0000100 PROG:   PROCEDURE;

0000200         DCL A FIXED INIT (0);

   .
   .
   .

0001400         IF A=B THEN IF A=C THEN D=E;

0001500              ELSE F=G;

0001600         ELSE F=A;

   .
   .
   .

0002500         END PROG;

0002600 _END
```

The module compiles without errors, and the compiler creates the following for you:

- A load data set named LOAD.MOD, which consists of the MOD object module in card-image format.
- An object module named MOD, which is executable with other TSS/360 PL/I programs. Since you have not yet defined any library in this session, MOD is automatically placed in your user library.

In addition, the text editor has created a line data set named SOURCE.MOD, which contains your PL/I statements for MOD. S3

Now you decide to change the statement ELSE F=G to a dummy statement and recompile MOD, while keeping the first version.

DDEF DDNAME=SCRATDD,DSORG=VP,DSNAME=SCRATCH,OPTION=JOBLIB
The DDEF command defines a job library, to receive the second version. You now modify line number 1500 of the source data set.

MODIFY SOURCE.MOD

#1500, ELSE;

#%E

You now recompile MOD.

PLI MOD,SOURCEDS=SOURCE.MOD
The compiler completes the compilation and places the object module in the job library named SCRATCH. Until you log off and log on again, release SCRATCH with a RELEASE command, or use the JOBLIBS command to move USERLIB (DDNAME=SYSULIB) to the top of the program library list, any invocation of the procedure PROG will invoke the version that was compiled last.

EXAMPLE 3: CONVERSATIONAL INITIATION OF NONCONVERSATIONAL TASKS

It is often more convenient to have your programs run after you have left the terminal, that is, to have them run in nonconversational mode. Two ways of doing this are shown in this example.

In Part 1, you begin your task conversationally and then use the BACK command to switch its execution to nonconversational mode. The BACK command names a prestored command sequence that becomes the SYSIN data set for the nonconversational portion of the task. When BACK is issued, control of the task is passed to the new SYSIN data set, effectively logging you off at the terminal. The nonconversational portion of the task takes its commands from the SYSIN data set named in the BACK command. This data set should conclude with a LOGOFF command; if not, the system performs the LOGOFF operation and issues a diagnostic message.

In Part 2, you construct a nonconversational task and then use the EXECUTE command to have it executed at a later time. The data set named in the EXECUTE command becomes the SYSIN data set for the nonconversational task; this task is completely independent of the task that issues EXECUTE. The EXECUTE command differs from the BACK command as follows:

1. EXECUTE requests an independent nonconversational task; BACK changes the user's conversational task to nonconversational mode.

2. The data set named in the EXECUTE command must contain LOGON and LOGOFF commands; the data set specified in the BACK command need only conclude with the LOGOFF command.

3. EXECUTE is accepted by the system even if no task space is currently available; the task will be created later. If task space is not available when the BACK command is issued, the command is ignored, and the user continues processing as though he had not issued the command.

Note: The output resulting from either a BACK- or EXECUTE- invoked task will appear on your installation's high-speed printer with the batch sequence number that is printed at your terminal.

Part 1: The BACK Command

After logging on, you build the SYSIN data set (named PROC12A) that will provide input to your task after you have switched to nonconversational mode. You issue a DATA command to build this data set; you do not issue a DDEF command for the data set, so default data set organization VS is assumed. The DATA command accepts each line as a string of

characters. Any mistakes you make while creating the data set will not be detected until the BACK command is executed.

S,Y: DATA DSNAME=PROC12A
The system prompts (with a #) for commands to be put into the data set named PROC12A. You want to execute program MAIN12, which is stored in your user library. MAIN12 reads data from the input stream (which you enter as part of this SYSIN data set), and creates an output data set (which you name SPRING).

S,Y: #DDEF DDNAME=OUTDD,DSORG=VS,DSNAME=SPRING

S,Y: #MAIN12

S,Y: #(data to be read by MAIN12)

.
.
.

S,Y: #
By merely pressing the RETURN key, you indicate end-of-file for MAIN12.

S,Y: #LOGOFF
You have entered all the commands and data for the SYSIN data set-;now enter a %E record to indicate that there are no more input records for the DATA command.

S,Y: #%E
Now that the SYSIN data set is complete, you can have it executed by issuing a BACK command that names the data set created by the DATA command. The system prompts you by printing an underscore.

S,Y: BACK DSNAME=PROC12A

SYS: (informs you that your BACK command has been accepted and what batch sequence number has been assigned)

Your BACK command has been accepted, and the task will be continued immediately as a nonconversational task beginning with the DDEF command. (Note that DDEF commands for private volumes must be given prior to issuing the BACK command.) If you want to cancel the task, issue a CANCEL command that specifies the batch sequence number.

Now you can depart and let the task run, since PROC12A is now its SYSIN and includes a LOGOFF command for task termination. If you want to initiate another task, you must log on again.

Part 2: The EXECUTE Command

After logging on, you build a SYSIN data set to perform the same functions as the SYSIN data set in Part 1. The LOGON command is the only difference between this SYSIN data set and the SYSIN data set in Part 1.

S,Y: DATA DSNAME=PROC12
The system prompts you with a #

S,Y: #LOGON PLIUSER,MYPASS,24,PLIACT70

S,Y: #DDEF DDNAME=OUTDD,DSORG=VS,DSNAME=SPRING

S,Y: #MAIN12

S,Y: # (data to be read by MAIN12)

.
.
.

S,Y: #
By merely pressing the RETURN Key, you indicate end-of-file for MAIN12.

S,Y: #LOGOFF
You have entered all the commands and data for the SYSIN data set; now enter a %E record to indicate that there are no more input records for the DATA command.

S,Y: #%E
Now that the SYSIN data set is complete, you can have it executed by issuing an EXECUTE command that names the data set created by the DATA command. The system prompts you by printing an underscore.

S,Y: EXECUTE DSNAME=PROC12

SYS: (accepts nonconversational task and assigns batch sequence number)

Your request for a nonconversational task has been accepted by the system, and will be executed when system resources are available. The SYSOUT of this task will consist of system messages and any output to SYSOUT generated by your executing programs.

Because the terminal is active (you are still logged on) after an EXECUTE is issued, another command sequence can be entered. In fact, another sequence similar to the one illustrated could be issued to create other tasks.

EXAMPLE 4: PREPARING A JOB FOR NONCONVERSATIONAL PROCESSING

In this example, you put a series of commands, PL/I source statements, and input data on cards. You will then send the cards directly to the system operator who will store the information from the cards into a data set that can be executed by the system. The data set becomes the SYSIN for a nonconversational task (described in the cards) and will be queued for execution.

After logging on, a job library named SCRATCH is defined. This library will contain the object module named ROOTS that you create with the PLI command. After the PLI command, you enter the PL/I source statements for ROOTS; the procedure uses the quadratic formula to find the roots. Data for the procedure is contained in the input stream following the command that calls for execution of ROOTS, since you have not included a DDEF command for the input data. Since you have also omitted a DDEF command for the output of ROOTS, output data will appear on your SYSOUT listing.

CARDS
When entered on a card, the LOGON command must start in the third card column, and the first two columns must be blank. All required LOGON operands must be included in the same card; these operands are the same as those required for a conversational log-on.

LOGON PLIUSER,,24,PLIACT70
You now specify a job library named SCRATCH that is to contain the object module created by the compiler. SCRATCH is automatically cataloged and available for use in any task that contains a DDEF command defining it.

DDEF DDNAME=SCRATDD,DSORG=VP,DSNAME=SCRATCH,OPTION=JOBLIB

```
PLI NAME=ROOTS
```
The PLI command invokes the PL/I compiler. You specify the name of the object module to be produced: ROOTS; this name must not be the name of the main procedure in the program. Since you do not specify the name of the source data set (SOURCEDS operand), the name SOURCE.ROOTS is assigned to the source data set, and this data set is cataloged under this name.

Your PL/I source statements follow. Since the GET DATA and PUT DATA statements do not specify a file name, SYSIN and SYSOUT are assumed; no DDEF commands are required. Data to be used by the object program must be in the input stream immediately following the command that calls for execution of the object program.

```
ROOT:  PROCEDURE OPTIONS (MAIN);

           ON ENDFILE GO TO EXIT;

       LOOP:  GET DATA;

              ROOT1=(-B+SQRT(B**2-4*A*C))/2*A;

              ROOT2=(-B-SQRT(B**2-4*A*C))/2*A;

              PUT DATA(A,B,C,ROOT1,ROOT2);

              GO TO LOOP;

       EXIT:  END;
```

After entering your PL/I source statements, you must enter an _END command to indicate the end of the source statements. Compilation then begins.

```
_END
```

After compilation and conversion, the object module resides in the library at the top of your program library list -- the job library SCRATCH in this example. You do not issue a PRINT command in this task; however, the listing data set is retained as the latest generation of LIST.ROOTS, and you can later print it by issuing: PRINT LIST.ROOTS(0), PRTSP=EDIT.

You now want to execute the object module; type the name of the module.

```
ROOTS
```

You now supply the cards containing data for your object program.

```
A=1     B=5     C=1;

A=2     B=4     C=0;

A=1     B=2     C=1;

A=2     B=7     C=5;

¢
```

The ¢ acts as an end-of-block character.

The last card in the deck is a LOGOFF command, which terminates the task. When punched on a card, the LOGOFF command must begin in the third column.

```
LOGOFF
```
When preparing a task for nonconversational execution, remember

that any errors in preparing the deck will probably terminate the task prematurely, since the system cannot prompt you for corrections.

EXAMPLE 5: STORING DDEF COMMANDS FOR LATER USE

In Part 1 of this example you create a data set containing DDEF commands for frequently used data sets. In Part 2, you cause them to be issued with a CDD command.

Part 1. Storing DDEF Commands

After logging on, you issue the EDIT command to invoke the Text Editor and enter DDEF commands. The DDEF commands are to be stored in a data set named DDPACK.PROG14. The commands are stored as character strings but are interpreted as commands when they are later retrieved by the CDD command.

S,Y: EDIT DSNAME=DDPACK.PROG14
The system will prompt you for each line by issuing a line number. You want to enter DDEF commands for these data sets: a job library (PLIPROGS) that contains compiled PL/I programs, two input data sets (DATA1 and DATA2), and an output data set (OUTPUT). The DDEFs do not have to be stored in any special order in the data set, but their ddnames must be unique.

S,Y: 0000100 DDEF DDNAME=YOURLIB,DSORG=VP,DSNAME=PLIPROGS,OPTION=JOBLIB

S,Y: 0000200 DDEF DDNAME=INPUT1,DSORG=VI,DSNAME=DATA1

S,Y: 0000300 DDEF DDNAME=INPUT2,DSORG=VS,DSNAME=DATA2

S,Y: 0000400 DDEF DDNAME=OUTPUT1,DSORG=VS,DSNAME=OUTPUT
You have entered all of the DDEF commands; when the system prompts with the next line number, enter a command system break character (underscore) followed by the END command. This terminates Text Editor processing of data set DDPACK.PROG14.

S,Y: 0000500 _END
The job library containing PROG14 and the DDEF commands associated with PROG14 are ready for use. You now check them out, to be sure there are no errors.

Part 2. Retrieving Stored DDEF Commands

In this part you retrieve the DDEF commands stored in Part 1. You want to first retrieve the job library definition and then the definitions for PROG14's data sets.

S,Y: CDD DSNAME=DDPACK.PROG14,YOURLIB
The system executes the specified DDEF command and prints it at your terminal, prefixing four zeros to distinguish it on your SYSOUT listing from those DDEF commands entered directly through SYSIN.

SYS: 0000 DDEF DDNAME=YOURLIB,DSORG=VP,DSNAME=PLIPROGS,OPTION=JOBLIB
Any diagnostic would be printed at this point, as the system is now analyzing the character string as a DDEF command. You now want to retrieve the remaining data definitions.

S,Y: CDD DSNAME=DDPACK.PROG14,(INPUT1,INPUT2,OUTPUT1)
The system retrieves the three specified DDEF commands and prints them at your terminal, prefixing each with four zeros.

SYS: 0000 DDEF DDNAME=INPUT1,DSORG=VI,DSNAME=DATA1

SYS: 0000 DDEF DDNAME=INPUT2,DSORG=VS,DSNAME=DATA2

SYS: 0000 DDEF DDNAME=OUTPUT1,DSORG=VS,DSNAME=OUTPUT
Now that all data definitions have been executed, you call for execution of PROG14.

S,Y: PROG14

SYS: (indicates that execution is complete)
The output of PROG 14 is the data set output; you request printing of OUTPUT

S,Y: PRINT DSNAME=OUTPUT,PRTSP=EDIT

SYS: (acknowledges your request and informs you of batch sequence number assigned to PRINT task)

EXAMPLE 6: MANIPULATION OF SEVERAL FORMS OF A PROGRAM

In this example you examine a previously cataloged program named MAIN19 to determine whether you want to retain it in the system. If you do not want to retain it, you will punch the source data set onto cards and then eliminate all forms of the program (source program, listing data set, and object program) from the system.

You first review some lines from the source program to determine whether it is to be eliminated. You issue a print request for the source data set and then a punch request in which you specify that the data set is to be erased after punching. You then decide that a listing of the source data set is not needed, since the deck will be punched, so you cancel the print request. You then erase the two versions of the listing data set from the system and erase the object module from your user library.

After logging on, you want to eliminate from the system any forms of a program named MAIN19 that you no longer need. You want to punch the source data set onto cards, but first you must determine whether such cards can be used as compiler input. To examine the source data set, you issue the following command.

S,Y: LINE? SOURCE.MAIN19,(1,5000)
In this command you specify the name of the data set and the lines to be displayed (lines 1 through 5000).

SYS: (informs you that the first line number in the data set is 000100 and then proceeds with the actual listing)

```
0000100COUTPUT:  PROCEDURE OPTIONS(MAIN);
0000200C         DCL  A  FIXED,
0000300C              B  FLOAT,
0000400C              COMP1  FLOAT,COMPLEX,
0000500C              STRING  BIT(8);
0000600C         A,B=6701;
0000700C         COMP1=A+6701I;
0000800C         STRING='110
```

YOU: (press attention button)

Satisfied that the program is the one that you want to eliminate, you press the attention button to terminate the LINE? command. The C following the line numbers of the source data set indicates that the statements were originally entered via a card reader. This means that, if you punch the source data set, the cards can later be used for compiler input. (See Section 5 for a more detailed description of compiler input format requirements.)

In order to determine the size of the data set, you request the maximum possible line number in the LINE? command; this causes the last line of the line data set to be printed.

S,Y: LINE? SOURCE.MAIN19,9999999

SYS: (issues the last line in the data set: 0013700C END)
You want to obtain a printout of the source data set for program MAIN19.

S,Y: PRINT DSNAME=SOURCE.MAIN19

SYS: (informs you that batch sequence number 0375 is assigned to the print task.)

The system establishes a separate nonconversational task to print the data set.

You now want to punch the data set SOURCE.MAIN19. Since the first eight characters of each record are a line number and input key, you want punching to start with the ninth character. Since the original records (without line number and key) were 80 characters long, you want to punch through byte 88 of the record that includes the key. You must use the ERASE operand of the PUNCH command rather than a separate ERASE command, since the system rejects an ERASE command if the data set referred to has an associated print or punch task pending. You should not insert the ERASE operand until the last print or punch request in any sequence that refers to the same data set is completed. It is possible for the first request, for example, PRINT above, to be executed in less time than it takes to type in the next command; therefore, if the ERASE operand had been specified in the PRINT command, it could take effect before the PUNCH command could be executed.

S,Y: PUNCH DSNAME=SOURCE.MAIN19,STARTNO=9,ENDNO=88,ERASE=Y

SYS: (informs you that batch sequence number 0376 is assigned to the task)
The system establishes a separate nonconversational task to punch the data set.

After issuing the PUNCH command, you decide that a listing of the source data set is superfluous (since you will have a source deck), so you cancel the print task, referring to it by the batch sequence number assigned by the system.

S,Y: CANCEL BSN=0375
Two versions of the listing data set are maintained for each cataloged program in the system; these versions are stored in a generation data group. You now erase both versions of the listing data set.

S,Y: ERASE LIST.MAIN19(0); ERASE LIST.MAIN19(0)
You now want to erase the object module from your user library.

S,Y: ERASE USERLIB(MAIN19)
All forms of MAIN19 are now removed from the system, so you decide to log off.

S,Y: LOGOFF

SYS: (acknowledges your request)

EXAMPLE 7: SURVEY OF SYSTEM FACILITIES AND SOME HOUSEKEEPING METHODS

In this example, you examine your data sets stored in the system and eliminate those that you no longer need. Three commands are available to allow you to examine your data sets -- PC? (present catalog), DSS? (present data set status), and POD? (present partitioned organization directory). The USAGE command enables you to find out the amount of system facilities allocated to you, and the EXHIBIT command enables you to inquire about your nonconversational tasks that are running or about to run. The KEYWORD command causes the system to tell you what PROCDEFs you have. (See Example 11 for a description of the PROCDEF facility.)

After logging on, you issue a PC? command to present the name, access, and, for shared data sets, the owner's identification of one or more cataloged data sets. You want information about all your cataloged data sets, so you issue the PC? command with no operands. If the name of one or more data sets is specified as the operand of a PC? command, only information about the specified data sets is presented.

```
S,Y:  PC?

SYS:  DATA SETS IN CATALOG WITH QUALIFIER PLIUSER

        PLIUSER.USERLIB           ACCESS:RW

        PLIUSER.JOBLIBA,          ACCESS:RW

        PLIUSER.M22OUT,           ACCESS:RW

        PLIUSER.PROJECT.A,        ACCESS:RW

        PLIUSER.PROJECT.B,        ACCESS:RW

        PLIUSER.PROJECT.Z,        ACCESS:RW

        PLIUSER.PROJECT.Z2,       ACCESS:RW

        PLIUSER.SOURCE.MATRIX7,   ACCESS:RW

        PLIUSER.SOURCE.TRIALX,    ACCESS:RW

        PLIUSER.VERSION5,         ACCESS:RO,OWNER:OTHERGUY
```

You want more information about data set SOURCE.MATRIX7, so you issue a DSS? command specifying SOURCE.MATRIX7 as an operand. The DSS? command presents more detailed information than does the PC? command. If DSS? is specified without an operand, detailed status information for all your cataloged data sets is presented.

```
S,Y:  DSS? SOURCE.MATRIX7

SYS:  PLIUSER.SOURCE.MATRIX7

        ACCESS:  RW

        VOLUME:  232323(2311)

        DS ORGANIZATION:  VI          PAGES:  002

        REFERENCE DATE:  103/70       CHANGE DATE:  103/70

        RECORD FORMAT:  V             RECORD LENGTH:  00132

        KEY LENGTH:  00007            RELATIVE KEY POSITION:  00004
```

You no longer need this data set, so you erase it.

S,Y: ERASE SOURCE.MATRIX7
You now want to examine the data set SOURCE.TRIALX to determine whether you want to erase it. The LINE? command can be used to list an entire data set or selected lines. Since you want your listing to start from the beginning of the data set, you do not specify line numbers.

S,Y: LINE? SOURCE.TRIALX

SYS: 0000100 TRIALX: PROCEDURE OPTIONS(MAIN)

0000200 /*ROUTINE ANALYZES TEST DATA*/

The first two lines printed out is sufficient for you to recognize this as an old program that you no longer need. You halt further printing by pressing the attention key.

YOU: (press attention key)
Pressing the attention key returns you to command mode so that you can enter commands. You decide to erase SOURCE.TRIALX and the data sets of numeric data associated with it. The data sets of numeric data all have the partially qualified name PROJECT. You can specify this partially qualified name; the system prints the individual names one by one for a decision regarding their disposal. You have the option of erasing (E) or retaining (R) the individual data sets cataloged under the generic name PROJECT, or erasing all of them (A). You decide to erase all but PROJECT.B.

S,Y: ERASE SOURCE.TRIALX

S,Y: ERASE PROJECT

SYS: PROJECT.A

YOU: E

SYS: PROJECT.B

YOU: R

SYS: PROJECT.Z

YOU: A
By typing A, you cause the system to erase PROJECT.C and those data sets whose names would follow if prompting continued (in this case, just PROJECT.Z2).

After the system has presented the names of all data sets cataloged under the partially qualified name PROJECT, it prints an underscore; you can then enter another command. You decide to erase data set VERSION5, which you no longer need.

S,Y: ERASE VERSION5

SYS: (informs you that VERSION5 is not yours to erase and ignores the command)
VERSION5 is a shared data set for which you do not have unlimited access; therefore, you cannot erase it. You can issue a DELETE command to remove only your catalog entry for VERSION5 without affecting the data set or the owner's catalog.

S,Y: DELETE DSNAME=VERSION5
You now issue a POD? command to request a list of each object module in your user library.

S,Y: POD?

SYS: MAIN7

MAIN10

SUBMATRX

After listing the modules in your library, the system prints an underscore so you can enter a command. You can erase modules from the library without erasing the entire library. You decide to erase modules MAIN7 and SUBMATRX.

S,Y: ERASE USERLIB(MAIN7); ERASE USERLIB(SUBMATRX)
You now want to examine library JOBLIBA to see if it is still needed.

S,Y: POD? JOBLIBA

SYS: PROG14

MAIN12

.
.
.

After listing the modules in JOBLIBA, the system prints an underscore so that you can enter a command. You no longer need JOBLIBA, so you erase it.

S,Y: ERASE JOBLIBA
You enter the USAGE command to inquire about the amount of system resources you have used. Two totals are presented: 1) the amount of resources allocated, and 2) the total amount of resources used since you were joined and since your present LOGON.

The following resources are accounted for: permanent storage, temporary storage, direct access devices, magnetic tapes, printers, card reader-punches, bulk input, bulk output, TSS/360 tasks, total time that your terminal was connected to the system, and CPU time used.

S,Y: USAGE

SYS: (prints tabulation of system resources charged to you)

You issue an EXHIBIT command to inquire about the status of your uncompleted nonconversational tasks. For a complete explanation of the EXHIBIT command, see Command System User's Guide.

S,Y: EXHIBIT OPTION=BWQ

SYS: (describes each of your uncompleted nonconversational tasks, including each task's batch sequence number and status in the system)

S,Y: KEYWORD

SYS: (types the names of all your PROCDEFs)

EXAMPLE 8: TRANSFERRING VIRTUAL STORAGE DATA SETS BETWEEN DISK AND TAPE

In this example, you copy VAM data sets from disk to tape, tape to disk, and from disk to disk. The following commands are used to transfer the data sets: VT (VAM to Tape), TV (Tape to VAM), and VV (VAM to VAM). The VAM data sets to be copied are ORIGIN1, ORIGIN2, ORIGIN3, and ORIGIN4.

After logging on, you copy data set ORIGIN1 onto tape; the data set on tape is to be named COPY1. Before the first VT command in a task, a DDEF command must be issued for the tape data set, with the DDNAME of DDVTOUT and the DSNAME of the first data set to be copied onto that tape.

```
S,Y:  DDEF DDVTOUT,PS,COPY1,UNIT=(TA,9),LABEL=(,SL,RETPD=12),-
      VOLUME=(PRIVATE),DISP=NEW
```

The data set organization must be PS, and the tape must be a nine-track tape.

```
S,Y:  VT DSNAME1=ORIGIN1
```

Since you omit the DSNAME2 operand, the system will assign to the copy data set the DSNAME given in the DDEF command (that is, COPY1).

Note: If you do not want the copy data set to be the first data set on the tape, you must specify the file sequence number in the LABEL operand of the DDEF command. This prevents overlaying of data sets.

SYS: (performs the copy operation, indicates the names of the original and copy data sets, and indicates the file sequence numbers and volume serial numbers used for the copy data set)

Now you write ORIGIN2 onto the same tape volume.

```
S,Y:  VT DSNAME1=ORIGIN2,DSNAME2=COPY2
```

SYS: (copies ORIGIN2 onto the tape with the data set name COPY2; if you had omitted the DSNAME2 operand, the system would have assigned the name TA000001.ORIGIN2)

To copy the tape copy of the data set back onto disk so that it can be used, issue a TV command. In this command, the name of the tape data set (COPY2) and the name of the disk copy (COPYBACK) are specified.

```
S,Y:  TV COPY2,COPYBACK
```

SYS: (copies the data set just written on a nine-track tape back onto disk and informs you that the operation has been performed)
You now want to copy data set ORIGIN3, which is on a disk, to produce a copy named COPY3, which also resides on disk.

```
S,Y:  VV ORIGIN3,COPY3
```

SYS: (informs you that the copy operation has been performed)
You now want to copy a VAM data set onto a private disk whose serial number is 333333.

```
S,Y:  DDEF PRIVDD,VI,COPY4,UNIT=(DA,2311),VOLUME=(,333333)

S,Y:  VV ORIGIN4,COPY4
```

SYS: (informs you that the copy operation has been performed)

EXAMPLE 9: THE TEXT EDITOR FACILITY

In this example, you use the text editor to create and edit data sets. The example illustrates only the more basic facilities of the text editor.

To invoke the text editor, issue the EDIT command. The operand of this command is the name of the data set to be created or edited.

S,Y: EDIT DSNAME=EX9
Since no DDEF command was issued for EX9, the following data set attributes are assumed: virtual index sequential organization, format-V records, maximum logical record length of 132 bytes, a key length of 7 bytes, and a relative key position of 4.

The system responds with the first line number.

S,Y: 0000100 THESE ARE LINES
0000200 OF A LINE DATA SET
0000300 SIMPLE DATA IS USED HERE. I TYPED THIS
You enter the first three lines of EX9. Each time you press RETURN, the text editor responds with a new line number.

To tell the system that you want to enter a command now instead of a line of data, you enter a break character (_) after the line number, followed by a command.

S,Y: 0000400 _INSERT 200,50
You begin the line with an underscore to inform the system that a command, not text-editor data, is being entered. The INSERT command tells the system that you want to insert one or more lines after line 200, and that the first inserted line should have the line number 250. The system responds by prompting you with the line number 250, and you enter the line to be inserted.

S,Y: 0000250 FOR ILLUSTRATIVE PURPOSES,
The system will not prompt you with line number 300, since there is already a line numbered 300 in the data set. The system will now prompt you for another command. Perhaps you'd like to remove line 300 and replace it with two lines:

S,Y: REVISE 300

S,Y: 0000300 SIMPLE DATA IS USED HERE.
0000400 i typed this in lower case letters

You can avoid constant use of the shift key by typing data in lowercase letters. The system normally translates the lowercase letters into capital letters. (However, if you issue a KA command, you can store data in mixed lowercase-uppercase form.)

You decide to stop editing this data set. When the system prompts with a line number, enter a break character followed by the END command.

S,Y: 0000500 _END
This terminates text editing. You could also have issued another EDIT command, with a different data set name; this would have terminated editing of the first data set and allowed you to start editing the next one.

The system types an underscore to resume prompting for commands. You review the data set EX9 and see that there are no mistakes.

S,Y: LINE? EX9

SYS: 0000100 THESE ARE LINES
0000200 OF A LINE DATA SET
0000250 FOR ILLUSTRATIVE PURPOSES,
0000300 SIMPLE DATA IS USED HERE.
0000400 I TYPED THIS IN LOWERCASE LETTERS.

Later you return to the data set EX9. You issue the EDIT command:

S,Y: EDIT EX9

```
S,Y:  UPDATE
      The UPDATE command tells the system that you want to change or add
      lines.

SYS:  (unlocks keyboard).

      You now add lines 230 and 90 to your data set.  After typing each
      line number, you skip a space and type the data.

YOU:  230 CREATED THROUGH THE TEXT EDITOR.

SYS:  (unlocks keyboard)

YOU:  90 THE TEXT EDITOR:

SYS:  (unlocks keyboard)

YOU:  _EXCISE N1=400
      Line 400 is deleted.
```

EXAMPLE 10: THE TEXT EDITOR FACILITY

In this example, you make more extended use of the updating capabilities of the text editor. The example is probably more complex than you might expect for a single terminal session, but it shows the flexibility of the commands available.

```
S,Y:  EDIT EX10,RNAME=REGION1,REGSIZE=8
      The data set attributes are the same as for EX9, in the previous
      example, except that the key length is increased to 15 bytes to
      allow 8 bytes for the region name, and the record length is
      increased accordingly.

      You enter the data lines for the region named REGION1.

S,Y:  0000100 LINEAE
      0000200 LINEB
      0000300 LINEC
      0000400 LINED
      0000500 LINEE
      0000600 LINEF
      0000700 LINEG
      0000800 LINEH
      0000900 LINEI
      0001000 _NUMBER N1=300,N2=500,BASE=300,INCR=50

      You renumber lines 400 and 500 as lines 350 and 400.  The system
      prompts you for another command, and you issue a LIST command to
      see how the data set EX10 looks.

S,Y:  LIST

SYS:  REGION1 0000100 LINEAE
      REGION1 0000200 LINEB
      REGION1 0000300 LINEC
      REGION1 0000350 LINED
      REGION1 0000400 LINEE
      REGION1 0000600 LINEF
      REGION1 0000700 LINEG
      REGION1 0000800 LINEH
      REGION1 0000900 LINEI

S,Y:  REGION RNAME=REGION2
      You start a new region.
```

S,Y: 0000100 _DEFAULT TRANTAB=Y
This optional command makes the following data set changes provisional. These changes can be made permanent by issuing an ENABLE command after you are satisfied all changes were entered properly.

DEFAULT is not a text-editing command, but activation of the text editor does not limit you to text-editing commands.

S,Y: EXCERPT DSNAME=EX9,N1=100,N2=300
You excerpt lines 100 to 300 from EX9, created in the previous example, and insert these lines into the current data set, at the beginning of REGION2.

S,Y: CONTEXT N1=100,N2=300,STRING1=DATA,STRING2=XXXX
The current region is searched for all occurrences of the character string DATA in lines 100 to 300. Wherever it is found, DATA is replaced by XXXX.

Note: This facility is useful for symbol replacement in source language data sets.

S,Y: LIST

```
SYS: REGION1 0000100 LINEAE
     REGION1 0000200 LINEB
     REGION1 0000300 LINEC
     REGION1 0000350 LINED
     REGION1 0000400 LINEE
     REGION1 0000600 LINEF
     REGION1 0000700 LINEG
     REGION1 0000800 LINEH
     REGION1 0000900 LINEI
     REGION2 0000100 THESE ARE LINES
     REGION2 0000200 OF A LINE XXXX SET
     REGION2 0000230 CREATED THROUGH THE TEXT EDITOR.
     REGION2 0000250 FOR ILLUSTRATIVE PURPOSES,
     REGION2 0000300 SIMPLE XXXX IS USED HERE.
```

S,Y: ENABLE
Up to this point, the revisions made since DEFAULT TRANTAB=Y was issued have been temporary. With the issuance of the ENABLE command, these revisions will be permanent as soon as the next line-modifying command is executed. A STET command, on the other hand, would have deleted all changes made since TRANTAB was set to Y.

The ENABLE command has a continuing effect; until a DISABLE command is executed, only the latest line modification can be revoked by a STET command.

S,Y: REGION REGION1
You return to the region named REGION1.

S,Y: CORRECT N1=100
You want to remove a single character from a line. Standard correction characters are assumed, by default.

SYS: LINEAE

YOU: * %
The result is LINEA. The ENABLE command now takes effect, but the correction to line 100 is still revocable.

S,Y: POST
With this command, you make permanent the correction to line 100. (Since the ENABLE command is in effect, POST and STET affect only the latest text-editing change.) If this were a STET command, the correction to line 100 would be revoked.

S,Y: LOCATE STRING=LINEF
The entire data set EX10 is searched for the character string LINEF.

SYS: (types the line in which LINEF is first discovered)

S,Y: LIST N1=100,N2=400

SYS: (displays lines 100 to 400 of the current region, REGION1)

S,Y: END
You terminate text editor processing.

EXAMPLE 11: USE OF COMMAND PROCEDURE (PROCDEF)

In this example, you create a command procedure (PROCDEF) to be called at a later time just as if it were a system-supplied command. This PROCDEF defines the input and output data sets for a PL/I program; it then executes the program, providing some of the input data, and prints the output data set.

The PL/I program has the module name TEST and includes the following statements:

```
PROG:  PROC OPTIONS(MAIN);
         .
         .
         .
       ON ENDFILE(IN) GO TO FINISH;
       GET DATA(B,C,D);
       DO J=1 TO 1000;
          DO I=1 TO 6;
          GET FILE(IN) LIST(A(I,J));
          F=A(I,J)*B;
          G=A(I,J)*C;
          H=A(I,J)*D;
          PUT FILE(OUT) LIST(F,G,H);
          END;
       END;
FINISH:  END PROG;
```

You issue a PROCDEF command, to create a PROCDEF named PLIPROG.

S,Y: PROCDEF PLIPROG
The text editor prompts you with line numbers, and you type the commands to be contained in the command procedure PLIPROG.

S,Y: 0000100 DDEF IN,VS,INDATA,DISP=OLD
S,Y: 0000200 DDEF OUT,VS,OUTDATA,DISP=OLD
S,Y: 0000300 DEFAULT SYSINX=E
The SYSINX operand of the DEFAULT command determines from which source the system expects SYSIN data when PROCDEF is invoked. Normally, SYSINX is defaulted to G, and the system expects conversational SYSIN data to come from the terminal. When SYSINX is set to E, the system reads SYSIN data from the PROCDEF itself.

S,Y: 0000400 TEST
S,Y: 0000500 B=245.6,C=.4329,D=984;
When PLIPROG invokes TEST, it will feed TEST (that is, PROG) the SYSIN data in line 500.

S,Y: 0000600 DEFAULT SYSINX=G
You must remember to return the value of SYSINX to G.

S,Y: 0000700 PRINT OUTDATA,ERASE=Y
S,Y: 0000800 _END

If you log off and log on again, PLIPROG is still effective. When you want to invoke PLIPROG, you type PLIPROG as a command:

```
S,Y:  PLIPROG
```

PROG is executed; it reads the data in PLIPROG, and creates the output data set. The output data set is printed on a high-speed printer and erased from its direct access device.

EXAMPLE 12: CREATING A CONSECUTIVE DATA SET

The program in this example merges the contents of two existing physical sequential data sets on tape, DS1 and DS2, and writes them onto a new virtual storage data set, DS3. Each of the original data sets contains 15-byte, fixed-length records arranged in EBCDIC collating sequence. The two input files, IN1 and IN2, have the default attribute BUFFERED, and locate mode is used to read records from the associated data sets into the respective buffers. The output file, OUT, is not buffered, allowing move mode to be used to write the output records directly from the input buffers.

After logging on you invoke the compiler and enter source statements for program MERGE. (The line numbers that the text editor types before each line of the source statements are not shown in this or in following examples.)

```
S,Y:  PLI NAME=MERGE
      MERG:  PROC OPTIONS (MAIN);
         DCL (IN1,IN2,OUT) FILE RECORD SEQUENTIAL,
             (ITEM1 BASED(A),ITEM2 BASED(B)) CHAR(15);
             ON ENDFILE(IN1) BEGIN;
       ON ENDFILE(IN2) GO TO FINISH;
NEXT2:  WRITE FILE(OUT) FROM(ITEM2);
        READ FILE(IN2) SET(B);
       GO TO NEXT2;
       END;
       ON ENDFILE(IN2) BEGIN;
       ON ENDFILE(IN1) GO TO FINISH;
NEXT1:  WRITE FILE(OUT) FROM(ITEM1);
        READ FILE(IN1) SET(A);
        GO TO NEXT1;
              END;
              OPEN FILE(IN1) INPUT,
                   FILE(IN2) INPUT,
                   FILE(OUT) OUTPUT;
               READ FILE(IN1) SET(A);
               READ FILE(IN2) SET(B);
       NEXT:  IF ITEM1>ITEM2 THEN DO;
               WRITE FILE(OUT) FROM(ITEM2);
               READ FILE(IN2) SET(B);
               GO TO NEXT;
               END;
              ELSE DO;
               WRITE FILE(OUT) FROM(ITEM1);
               READ FILE(IN1) SET(A);
               GO TO NEXT;
               END;
       FINISH:  CLOSE FILE(IN1),FILE(IN2),FILE(OUT);
                END MERG;
```

This begin block will be activated when the ENDFILE condition is raised for IN1.

This begin block will be activated when the endfile condition is raised for IN2.

After entering your source statements, type a command-prefix character (underscore) and enter an END command. The underscore designates the line as a command rather than PL/I source statements.

```
YOU:  _END
SYS:  (informs you that compilation is completed)
```

Before calling for execution of MERGE, you must define the two

input data sets and the output data set. The input data sets have physical sequential organization and reside on tape volumes 33731 and 987655.

```
S,Y:  DDEF DDNAME=IN1,DSORG=PS,DSNAME=DS1,DISP=OLD,-
          UNIT=(TA,9),VOLUME=(,033731)
S,Y:  DDEF DDNAME=IN2,DSORG=PS,DSNAME=DS2,DISP=OLD,-
          UNIT=(TA,9),VOLUME=(,987655)
      The output data set DS3 will reside on public storage and have
      virtual sequential organization.

S,Y:  DDEF DDNAME=OUT,DSORG=VS,DSNAME=DS3,DISP=NEW,-
          DCB=(RECFM=F,LRECL=15)
      You can now execute your program.

S,Y:  MERGE
```

EXAMPLE 13: USING A PRINT FILE

This example illustrates the use of a PRINT file and the options of the stream-oriented transmission statements to format a table and write it onto public storage for subsequent printing. The table comprises the natural sines of the angles from 0° to 359° 54' in steps of 6'.

The statements in the ENDPAGE ON-unit insert a page number at the bottom of each page and set up the headings for the following page. After the last line of the table has been written, the statement PUT FILE(TABLE) LINE(54) causes the ENDPAGE condition to be raised to ensure that a number appears at the foot of the last page; the preceding statement sets the flag FINISH to prevent a further set of headings from being written.

The DDEF command that defines the data set that this program creates includes no record-format information; the compiler infers the following from the file declaration and the line size specified in the statement that opens the file TABLE:

Record format = V (the default for a PRINT file)

Record size = 98 (line size + one byte for printer control character + four bytes for record control field)

Block size = 102 (record size + four bytes for block control field)

You invoke the PL/I compiler to enter and compile your source statements; you name the program SINE.

```
S,Y:  PLI NAME=SINE
      You now enter source statements for SINE.  (The line numbers typed
      by the text editor are not shown.)

SINPROC:  PROC OPTIONS(MAIN);
          DCL TABLE FILE STREAM OUTPUT PRINT,
          TITLE CHAR (13) INIT('NATURAL SINES'),
          HEADINGS CHAR(90) INIT('      0       6       12       1
          8       24       30       36       42       48       54'),
          PGNO FIXED DEC(2) INIT(1),
          FINISH BIT(1) INIT('0'B),
          VALUES(0:359,0L9)FLOAT DEC(6);
          ON ENDPAGE(TABLE)BEGIN;
          PUT FILE(TABLE) EDIT('PAGE',PGNO)(LINE(55),COL(87),A,F(3));
          IF FINISH='0'B THEN DO;
          PGNO=PGNO+1;
          PUT FILE(TABLE) EDIT(TITLE ' (CONT''D)',HEADINGS)
                               (PAGE,A,SKIP(3),A);
          PUT FILE(TABLE) SKIP(2);
```

```
        END;
        END;
        DO I=0 TO 359;
        DO J=0 TO 9;
        VALUES(1,J)=I=J/10;
        END;
        END;
        VALUES=SIND(VALUES);
        OPEN FILE(TABLE) PAGESIZE(52) LINESIZE(93);
        PUT FILE(TABLE) EDIT(TITLE,HEADINGS)(PAGE,A,SKIP(3),A);
        DO I=0 TO 71;
        PUT FILE(TABLE) SKIP(2);
        DO J=0 TO 4;
           K=5*I+J;
           PUT FILE(TABLE) EDIT(K,VALUES(K,*))(F(3),10 F(9,4));
           END;
        END;
        FINISH='1'B;
        PUT FILE(TABLE) LINE(54);
        CLOSE FILE(TABLE);
        END SINPROC;
     END
```

After your program is compiled, you issue a DDEF command to define the data set associated with the PL/I file TABLE.

```
S,Y:  DDEF DDNAME=TABLE,DSORG=VS,DSNAME=SINES
```

You can now execute program SINE.

```
S,Y:  SINE
SYS:  (executes SINE and creates data set SINES)
```

You now want to print data set SINES, which you just created on public storage. Since each record contains a printer control character, you specify printer spacing as EDIT. Each page of your PRINT file contains 54 lines, the default value in the PRINT command; therefore, you do not have to specify the number of lines per page in the command.

```
S,Y:  PRINT DSNAME=SINES,PRTSP=EDIT
SYS:  (sets up a separate nonconversational task to print the data set
      and informs you of the batch sequence number assigned)
```

EXAMPLE 14: CREATING AN INDEXED DATA SET

This example illustrates the creation of a simple INDEXED data set. The data set contains a telephone directory, using the subscribers' names as keys to the telephone numbers. Since the file is being created the file attribute SEQUENTIAL is declared. You are going to enter the data records from your terminal during program execution. The DDEF command for the data set specifies the PL/I file name as the data definition name (DIREC), the data set organization as virtual index sequential (VI), and the name of the data set (TELNO).

```
S,Y:  DDEF DDNAME=DIREC,DSORG=VI,DSNAME=TELNO
```

You now invoke the PL/I compiler to enter and compile source statements for program TELNOS.

```
S,Y:  PLI NAME=TELNOS
```

The text editor prompts with line numbers (not shown).

```
      TELNUM:  PROC OPTIONS(MAIN);
               DCL DIREC FILE RECORD SEQUENTIAL KEYED ENV(INDEXED),
                   CARD CHAR(80),
                   NAME CHAR(20) DEF CARD,
                   NUMBER CHAR(3) DEF CARD POS(21),
                   IOFIELD CHAR(3);
               ON ENDFILE(SYSIN) GO TO FINISH;
```

```
            OPEN FILE(DIREC) OUTPUT;
      NEXTIN: GET FILE(SYSIN) EDIT(CARD)(A(23));
            IOFIELD=NUMBER;
            WRITE FILE(DIREC) FROM(IOFIELD) KEYFROM(NAME);
            GO TO NEXTIN;
      FINISH: CLOSE FILE(DIREC);
      END TELNUM;
S,Y:  END
SYS:  (compiles your program and informs you when compilation is
      complete)
S,Y:  TELNOS

During execution of TELNOS, you are prompted to enter input data.
     :BAKER,R.          152
     :BRAMLEY,O.H.      248
     :CHEESEMAN,I.      141
     :CORY,G.           336
     :ELIOTT,D.         875
     :FIGGINS,S.        413
     :HARVEY,C.D.W.     205
     :HASTINGS,G.M.     391
     :KENDALL,J.G.      294
     :LANCASTER,W.R.    624
     :MILES,R.          233
     :NEWMAN,M.W.       450
     :PITT,W.H.         515
     :ROLF,D.E.         114
     :SHEERS,C.D.       241
     :SUTCLIFFE,M.      472
     :TAYLOR,G.C.       407
     :WILTON,L.W.       404
     :WINSTONE,E.M.     307
     :             (NULL LINE)
```

EXAMPLE 15: UPDATING AN INDEXED DATA SET

This example updates the data set created in Example 14 and prints out the new contents of the data set. The input data includes codes to indicate the operation required:

```
A:  Add a new record
C:  Change an existing record
D:  Delete an existing record
```

You must define the data set created in Example 14.

```
S,Y:  DDEF DDNAME=DIREC,DSORG=VI,DSNAME=TELNO,DISP=OLD
      You now invoke the PL/I compiler.

S,Y:  PLI NAME=DIRUPDT
      The text editor prompts with line numbers (not shown).
      DIRUPD:  PROC OPTIONS(MAIN);
               DCL DIREC FILE RECORD KEYED ENV(INDEXED),
                   NUMBER CHAR(3),
                   NAME CHAR(20),
                   CODE CHAR(1);
               ON ENDFILE(SYSIN) GO TO PRINT;
               ON KEY(DIREC) BEGIN;
                 IF ONCODE=51 THEN PUT FILE(SYSPRINT) SKIP EDIT
                                   ('NOT FOUND:',NAME)(A(15),A);
                 IF ONCODE=52 THEN PUT FILE(SYSPRINT) SKIP EDIT
                                   ('DUPLICATE:',NAME)(A(15),A);
                 END;
               OPEN FILE(DIREC) DIRECT UPDATE;
      NEXT:    GET FILE(SYSIN) EDIT(NAME,NUMBER,CODE)(A(20),A(3),X(3),A(1));
               IF CODE='A' THEN WRITE FILE(DIREC) FROM(NUMBER) KEYFROM(NAME);
```

```
            ELSE IF CODE='C' THEN REWRITE FILE(DIREC) FROM(NUMBER)
                                                          KEY(NAME);
                    ELSE IF CODE='D' THEN DELETE FILE(DIREC) KEY(NAME);
                         ELSE PUT FILE(SYSPRINT) SKIP EDIT('INVALID CODE:',
                                                 NAME)(A(15),A);
            GO TO NEXT;
PRINT:      CLOSE FILE(DIREC);
            PUT FILE(SYSPRINT) PAGE;
            OPEN FILE(DIREC) SEQUENTIAL INPUT;
            ON ENDFILE(DIREC) GO TO FINISH;
NEXTIN:     READ FILE(DIREC) INTO (NUMBER) KEYTO(NAME);
            PUT FILE(SYSPRINT) SKIP EDIT(NAME,NUMBER)(A);
            GO TO NEXTIN;
FINISH:     CLOSE FILE(DIREC);
            END DIRUPD;

S,Y:  _END
SYS:  (compiles your program and informs you when compilation is ready)
      You now call for execution of DIRUPDT

S,Y:  DIRDUPDT
      During execution of the program, you enter your input data.

      :NEWMAN,M.W.          516450   C
      :GOODFELLOW,D.T.      889      A
      :MILES,R.                233   D
      :HARVEY,C.D.W.        209      A
      :BARTLETT,S.G.        183      A
      :CORY,G.                 336   D
      :READ,K.M.            001      A
      :PITT,W.H.               515
      :ROLF,D.F.               114   D
      :ELLIOTT,D.           291875   C
      :HASTINGS,G.M.           391   D
      :BRAMLEY,O.H.         439248   C
      :          (NULL LINE)
```

EXAMPLE 16: BATCH PROCESSING

This example illustrates the use of a single invocation of the PL/I compiler to compile four procedures with three external names and then execute them.

```
S,Y:  PLI NAME=PGM1
      The text editor prompts with line numbers (not shown).

      FIRST:    PROC OPTIONS(MAIN);
                 DO I=1250 TO 1500 BY 50;
                   DO J=10, 15, 20;
                    K=SQRT(I/J);
                    PUT SKIP(2) DATA;
                 END FIRST;
      *PROCESS ('EXTREF,N=PGM2');
      SECOND:   PROC OPTIONS(MAIN);
                DCL PRINT ENTRY EXT,
                    A(5) INIT(1,2,4,8,16),
                    B(5) INIT(3,5,7,9,11),
                    C(5,5);
                DO I=1 TO 5;
                  DO J=1 TO 5;
                    C(1,J)=12*A(I)/B(J);
                    END;
                END;
                CALL PRINT (A,B,C);
                END SECOND;
      *PROCESS ('N=PGM3');
```

```
PRINT:    PROC(THOR,TVERT,ARRAY);
          DCL THOR(*),TVERT(*),ARRAY(*,*);
          I=DIM(THOR,1);
          PUT EDIT (THOR) (X(7), (I) F(7,2));
          DO J=1 TO DIM(TVERT,1);
            PUT SKIP EDIT(TVERT(J), (ARRAY(J,K) DO K=1 TO I))(F(7,2));
          END PRINT;
*PROCESS ('N=PGM4,FE');
THIRD:    PROC OPTIONS(MAIN);
            ON ENDFILE(SYSIN) GO TO FINISH;
NEXT:     GET DATA(A,B);
            C=A+8*B**2/3;
            PUT SKIP DATA;
            GO TO NEXT;
FINISH:   END THIRD;

S,Y:  END
```

You now want to call for execution of the programs. You can call for their execution separately or with a single command statement of the form:

```
S,Y:  PGM1; PGM2; PGM3;
SYS:  (executes the programs and prompts you to enter the input data
      required in PGM3)
S,Y:  :A=27, B=42; A=39, B=17; A=15, B=19; A=12, B=7;
SYS:  (prints output data from SYSOUT)
```

EXAMPLE 17: THE OBEY FACILITY

TSS/360 provides the facility for high-level language users to execute the assembler language OBEY macro. This macro enables you to specify a character string consisting of one or more TSS/360 commands, and have these commands obeyed during execution of a PL/I program. The syntax is:

$$\text{CALL SYSOBP}\left(\left\{\begin{array}{l}\text{'command character string'}\\ \text{character string variable}\\ \text{expression}\end{array}\right\}\right)$$

The argument within parentheses must be a character string consisting of the command to be obeyed. It can be literal (in quotes), the name of a character string variable that contains the command, or an expression that produces the character string.

The following program shows some uses of this facility.

```
OBEYTST:  PROC  OPTIONS(MAIN);
          DCL  DDEF CHAR(5) INIT('DDEF'),
               RELS CHAR(8) INIT('RELEASE'),
               ERAS CHAR(6) INIT('ERASE'),
               DSS  CHAR(5) INIT('DSS?'),
               STOP CHAR(5) INIT('STOP'), ;
          DCL  (PARAM,RECORD,TSTREC) CHAR(120) VARYING;
          DCL  (IN,OUT) FILE RECORD;
     /*
     */   PARAM = 'OUT,VS,TESTDATA,DCB=(RECFM=V,LRECL=124),DISP=NEW';
          CALL SYSOBP(DDEF  PARAM);

               DDEF||PARAM is an expression that produces a DDEF command.

          RECORD = 'ABCDE';
          DO I=1 TO 10;
          RECORD = RECORD||'ABCDE';
          WRITE FILE(OUT) FROM (RECORD);
          END;
          CLOSE FILE(OUT);
```

```
          CALL SYSOBP(RELS||'OUT');
          CALL SYSOBP(DDEF||'IN,,TESTDATA');
          TSTREC = 'ABCDE';
          DO I=1 TO 10;
          TSTREC = TSTREC||'ABCDE';
          READ FILE(IN) INTO (RECORD);
          IF RECORD ¬= TSTREC THEN GO TO ERROR;
          END;
          CLOSE FILE(IN);
          CALL SYSOBP(RELS||'IN');
          CALL SYSOBP(DSS||'TESTDATA');
          CALL SYSOBP('PRINT TESTDATA,ERASE=Y');
          CALL SYSOBP('DISPLAY ''OBEYTST SUCCESSFUL''');
          GO TO FINISH;
ERROR:    DISPLAY('OBEYTST FAILED');
FINISH:   END OBEYTST;
```

EXAMPLE 18: DYNAMIC CALLS

This example shows how to prepare a very large PL/I program (that is, a PL/I program containing more than 4096 bytes of PR entries, or approximately 970 subroutines) for execution.

If a PL/I main module has close to a thousand subroutines, it may be necessary to load and unload some of the subroutines dynamically to avoid an overflow of the PRV. (See "External Symbol Dictionary," in Section 5.) You can code an assembler-language module that will provide this dynamic interface between your PL/I program and the system.

```
S,Y:  PLI  MAIN,EXPLICIT=*ALL,XFERDS=CALLDS

S,Y:  0000100 PROCESS:  PROC OPTIONS (MAIN);
                        .
                        .
                        .
      0000900     CALL SUBA; /*SUBA HAS 500 SUBROUTINES*/
                        .
                        .
                        .
      0001500     CALL SUBB; /*SUBB HAS 500 SUBROUTINES*/
                        .
                        .
                        .
      0002000     CALL SUBC; /*SUBC HAS 500 SUBROUTINES*/
                        .
                        .
                        .
      0003000     END PROCESS;
      0003100_END
```

MAIN is compiled, and a transfer data set named CALLDS is created. (Assume that no data set named CALLDS existed previously.) CALLDS is shown later in this example as the second component of the transfer module.

The calls to SUBA, SUBB, and SUBC become, in the object module, calls to @SUBA, @SUBB, and @SUBC. @SUBA, @SUBB, and @SUBC are the intermediate entry points; they are in the transfer module. You could have omitted the EXPLICIT operand and entered the @s yourself, while typing the source data set, but then the XFERDS operand would have had no effect and you would have had to create the transfer data set yourself.

Note: EXPLICIT is also useful when the PLI command relates to an already existing source data set that contains calls with unpadded names.

The components of the transfer module are:

1. A PLICALL macro and a CSECT instruction
2. A transfer data set
3. Finishing touches, including an END instruction.

Before assembly of the transfer module, a source data set for each component exists as a line data set (to aid readability, line numbers are not shown):

```
         MACRO
&P0      PLICALL &P1,&TYPE,&CLEANUP
         AIF   ('&TYPE' EQ 'N').MEXIT   NAMING ENTRY ONLY
         ENTRY &P0
         USING &P0,15
         AIF   ('&TYPE' EQ 'O').ETYPE
         AIF   ('&TYPE' EQ 'E').ETYPE
.*
.*       IMPLICIT LINKAGE REQUIRED
.*
&P0      L     15,EXA&SYSNDX
         BR    15
EXA&SYSNDX DC  V(&P1)
.MEXIT   MEXIT
.*
.*       EXPLICIT LINKAGE REQUEST
.*
.ETYPE   ANOP
&P0      CLI   ACG&SYSNDX,X'0A'         SVC STILL ON?
         BNE   PRV&SYSNDX               NO, LOAD NOT REQUIRED
LDR&SYSNDX STM 14,3,12(13)              SAVE REGS AROUND CALL
         BAS   3,SKP&SYSNDX             BRANCH AROUND ADCON GROUP
         USING *,3
ERV&SYSNDX DC  A(ABEND)
ACG&SYSNDX ADCON LOAD,EP=&P1,LDERR=CODE
SKP&SYSNDX EX  0,ACG&SYSNDX             DYNAMIC LOAD
         CLI   ACG&SYSNDX,X'0A'         SUCCESSFUL LOAD?
         BE    ERR&SYSNDX               NO, ABEND
         LM    14,3,12(13)              RESTORE REGISTERS
         B     LNK&SYSNDX               GO PERFORM LINK
         DROP  3
PRV&SYSNDX LOAD 1,=V(SYSTCM)
         USING CHATCM,1
         LOAD  3,TCMCXD                 GET CURRENT CXD VALUE
         DROP  1
         CLI   0(3),X'1000'             PRV OVERFLOW?
         BNH   LNK&SYSNDX               NO, GO PERFORM LINK
         LR    1,3
         CALL  &CLEANUP                 CALL CLEANUP ROUTINE

         The cleanup routine is a routine that you provide for
         selectively unloading subroutines.

LNK&SYSNDX L   15,ACG&SYSNDX+12         V-CON
         BR    15                       ENTER ROUTINE
ERR&SYSNDX LM  14,3,12(13)              RESTORE REGISTERS
         L     15,ERV&SYSNDX            ERROR ADDRESS
         BR    15
         MEND
TRANSFER CSECT
```

component 1 (data set name is PREFIX)

Note:

1. All registers except register 15 are passed to the called PL/I module exactly as they are received by the transfer module.

2. The called module will not return to the transfer module. The return will be to the module containing the CALL statement.

```
        @SUBA   PLICALL   SUBA
        @SUBB   PLICALL   SUBB
        @SUBC   PLICALL   SUBC
```

component 2 --created by system in this example (data set name is CALLDS)

X'00' follows each line number in component 2.

```
        EJECT
        USING ABEND,15
ABEND   DS      0H
        ABEND 1,'*ABEND* TRANSFER LOAD ERROR.'
        COPY CHATCM
        END
```

component 3 (data set name is SUFFIX)

The above transfer module is only an example; you should vary it to fit your needs.

Now you gather the components into a single source data set. The name of the transfer module will be DYNAMO.

```
S,Y:  EDIT SOURCE.DYNAMO
S,Y:  0000100 _EXCERPT PREFIX
S,Y:  EXCERPT CALLDS
S,Y:  EXCERPT SUFFIX
S,Y:  END
```

(Now you assemble the transfer module.)

```
S,Y:  ASM  DYNAMO,Y
```

If you update the transfer module in the future, you must reassemble it.

You are now ready to execute the PL/I program. Calls to the dummy names will cause module DYNAMO to be loaded when module MAIN is loaded. Whenever MAIN branches to DYNAMO, DYNAMO will dynamically load the invoked subroutine, and all subroutines which that subroutine calls non-dynamically, and perform error checking.

PART IV: APPENDIXES

A PL/I user who is switching from IBM System/360 Operating System (OS/360) to IBM System/360 Time Sharing System (TSS/360) should know the differences in the ways that the two systems implement PL/I. In particular, he should understand:

- The TSS/360 command system
- Interchange of data between OS/360 and TSS/360
- Data set positioning and DISP=NEW
- Raising of UNDEFINEDFILE condition for STREAM files
- Compiler options not supported by TSS/360
- TSS/360 language restrictions.

TSS/360 COMMAND SYSTEM

All functions performed by job control language (JCL) in OS/360 are performed by the command system in TSS/360. TSS/360 commands perform all the functions necessary to compile and execute PL/I programs.

This manual serves as an introduction to the command system for the PL/I user. The definitive manual on the command system is Command System User's Guide.

INTERCHANGE OF DATA BETWEEN OS/360 AND TSS/360

Interchange of data between OS/360 and TSS/360 must be by cards or by CONSECUTIVE PS data sets on tape or disk. TSS/360 cannot read INDEXED data sets produced by OS/360; they must be rewritten on tape or disk as CONSECUTIVE PS data sets.

DATA SET POSITIONING AND DISP=NEW

In TSS/360 if the DDEF command specifies DISP=NEW and the file is opened for output, closed, and then reopened, the read-write mechanism is positioned after the last record in the data set. In OS/360, in similar circumstances, the read-write mechanism is positioned before the first record in the data set.

RAISING OF UNDEFINEDFILE CONDITION FOR STREAM FILES

Omission of the DDEF command for a RECORD file causes the UNDEFINEDFILE condition to be raised. A STREAM file, on the other hand, defaults to SYSIN or SYSOUT so that PL/I does not raise the UNDEFINEDFILE condition merely because the STREAM file has no corresponding DDEF command. (It is still possible to have the UNDEFINEDFILE condition raised because of attribute conflicts.) Note that omission of the DD statement for a STREAM file in OS/360 does cause the UNDEFINEDFILE condition to be raised.

COMPILER OPTIONS NOT SUPPORTED BY TSS/360

The options SIZE, M91 or NOM91, and EXTDIC or NOEXTDIC can be specified in the PLIOPT operand of the PLI command, but they are ignored during execution. These options are discussed further under "Dummy Options," in Appendix G.

TSS/360 LANGUAGE RESTRICTIONS

The following PL/I language features are not supported by TSS/360. Statements containing these features can be issued and compiled correctly, but at execution time these features are rejected as described below.

1. Multitasking features -- Already inherent in TSS/360; must be handled at the command system level. If a request for multitasking is incorporated in a PL/I program, it will cause the execution of that program to be terminated when the request is encountered during TSS/360 execution. A CALL statement that contains a multitasking option (TASK, EVENT, or PRIORITY) will prevent the entire object module containing the CALL statement from executing on TSS/360.
2. SORT -- An attempt to execute a call to the SORT routine results in an error message; execution is then terminated, and the user's task reverts to the command mode.
3. CHECKPOINT/RESTART -- An attempt to execute a call to the CHECKPOINT routine results in an error message; execution then continues as though the call had not been made. An attempt to

execute a call to the RESTART routine results in an error message, followed by termination of execution.

4. REGIONAL I/O -- Raises UNDEFINEDFILE condition.

5. TRANSIENT files, PENDING condition, ENVIRONMENT options G and R -- Raise UNDEFINEDFILE condition.

6. UNLOCK statement, NOLOCK option of READ statement, ENVIRONMENT options GENKEY, INDEXAREA, and NOWRITE -- Ignored.

7. Block-size specifications in ENVIRONMENT options F, V, and U, for VAM data sets -- The system ignores any attempt to specify a block size, and groups all VAM records into page-size blocks (4096 bytes). If both block size and record size are given, the block-size operand is ignored. If only one size is specified, it is interpreted as the record size.

8. EXCLUSIVE attribute -- need not be declared, since record locking is automatic and cannot be suppressed by a NOLOCK option.

The REDUCIBLE and IRREDUCIBLE attributes cause no action in the TSS/360 PL/I compiler other than to imply the ENTRY attribute.

APPENDIX B: ATTENTION INTERRUPTIONS

An attention interruption is generated by pressing the ATTN key on a 2741 Communications Terminal, or the ATTENTION key on a 1050 Data Communications system. This key can be pressed at any time -- it cannot be locked out. The system, if running, always responds, though perhaps not immediately; the response depends on what was happening when the interruption occurred.

Table 24 shows the system's responses to attention interruptions and to subsequent actions of the user.

Note: If a LOAD command is issued while a module is interrupted, it is not possible to resume execution at the point of interruption with any assurance that all conditions have been properly restored.

It a data set is opened prior to the interruption, a DDEF or RELEASE command cannot be issued against it until the interrupted module is unloaded and reloaded. The module must be unloaded before it is reloaded, since the system never loads a module that is already loaded.

If the interrupted module was inadvertently loaded from the wrong library, the user must:

1. Unload the interrupted module.
2. Put the correct library on top of the program library list by issuing RELEASE commands for the libraries above it or by issuing a JOBLIBS command for the correct library.
3. Load or execute the correct module.

If the user does not resume the interrupted module's execution with the GO command, and if the module may have left a data set open, he should issue a CLOSE command to ensure that the data set is closed.

Levels of Interruption

The status (registers and PSW) of each interrupted nonprivileged program is saved in a table called the stack table, whenever another nonprivileged program is invoked without resuming the interrupted program. (The interruption can be caused by an AT command, a call to IHEDUMC, a PL/I call to SYSOBP, or a program interruption, as well as by pressing the attention key.) When a program's status is saved in the stack table, it is said to remain activated, although it is not executing. The status is removed when the interrupted program again receives control. The RTRN, PUSH, and EXIT commands can be used to manipulate the stack table as described in Table 24.

The current level of interruption is an indicator of how much of the stack table is in use. One level is taken every time a program's status is saved; the level is freed when the interrupted program regains control. Ten levels are available. The STACK command displays the names of all activated programs, from the current level on down.

Table 24. Attention Interruptions (Part 1 of 2)

	System Status when Attention Key is Pressed		
	A nonprivileged program[1] is in operation.	A privileged program[2] is in operation and the current command is the only one or the last one in its command statement.	A privileged program[2] is in operation and there are additional commands in the command statement.
	System Response		
	!(If program was for a command that terminated without completing, issue appropriate message.)	_(If current command, other than DISPLAY, terminates without completing, issue appropriate message.)	*(If current command, other than DISPLAY, terminates without completing, issue appropriate message.)
Subsequent User Actions	System Responses		
	type 1	type 2	type 3
press attention key again	!	_	*
press RETURN	Resume interrupted program; after completion, response type =2.	_ Response type still =2.	Resume command statement; after completion, response type =2.
GO command	Resume interrupted program; after completion, response type =2.	Resume nonprivileged program that is at current level of interruption, forgetting command statements subsequent to its interruption. After completion of nonprivileged program, response type =2.	Resume nonprivileged program that is at current level of interruption, forgetting command statements subsequent to its interruption. After completion of nonprivileged program, response type =2.
REPEAT function	Repeat interrupted message, if any. Response type =1.	Repeat interrupted message, if any. Response type =2.	Repeat interrupted message, if any. Response type =3.
STRING function[3]	Display unprocessed portion of interrupted command statement. Response type =1.	Error message. Response type =2.	Display unprocessed portion of interrupted command statement. Response type =3.
EXIT command	End[4] and deactivate interrupted program; resume command statement, if unfinished. After completion of command statement, response type =2.	End[4] and deactivate nonprivileged program at current level of interruption; resume its command statement, if unfinished. After completion of command statement, response type =2.	End[4] and deactivate nonprivileged program at current level of interruption; resume its command statement, if unfinished. After completion of command statement, response type =2.
RTRN command	Deactivate nonprivileged programs at all levels of interruption; cancel any unprocessed portions of associated command statements. Go to response type 2.	Deactivate nonprivileged programs at all levels of interruption; cancel any unprocessed portions of associated command statements. Response type =2.	Deactivate nonprivileged programs at all levels of interruption; cancel any unprocessed portions of associated command statements. Go to response type 2.

Table 24. Attention Interruptions (Part 2 of 2)

Subsequent User Actions	System Response		
	type 1	type 2	type 3
PUSH command	Save status of interrupted program.[5] Go to response type 2.	Save status of currently interrupted nonprivileged program.[5] Response type =2.	Save status of currently interrupted nonprivileged program.[5] Go to response type 2.
Any command except those above	Execute command; after execution, response type =2.	Execute command; after execution, response type =2.	Execute command, canceling unprocessed portion of interrupted command statement. After execution, response type =2.

[1]A user-written program, the text editor, the PL/I compiler, PLC, ODC, or the PL/I library. Note: A nonprivileged program can call a privileged program, thus causing a privileged program to be in operation when the attention interruption occurs.
[2]A command-system program for other than a text editor or language processing command, or some other privileged system program called by one of the nonprivileged programs.
[3]STRING is valid only if it is the first user action after the attention.
[4]As if control were transferred to an END statement within the program.
[5]Normally, the status of the interrupted program is not saved until another nonprivileged program is invoked without resumption of the interrupted program. However, if the user wants to use PCS to change the contents of his registers, the PUSH command allows him to save the status immediately.

APPENDIX C: PRINTER AND PUNCH CONTROL CHARACTERS

All record formats can optionally include a control character in each logical record. This control character is recognized and processed if a data set is being written to a printer or punch. For format-F and -U records, this character is the first byte of the logical record. For format-V records, it must be the fifth byte of the logical record, immediately following the logical record length field.

Two alternatives are available; that is, FORTRAN control characters or machine-code control characters. If either option is specified, the character must appear in every record. Use of FORTRAN control characters is specified by the A option of the RECFM DCB subparameter to the DDEF command, or by the CTLASA option of the ENVIRONMENT attribute. Use of machine code is specified by the M option of the RECFM subparameter or by the CTL360 option of the ENVIRONMENT attribute.

FORTRAN control characters are usually preferred, since the PL/I library automatically inserts them in each record of a STREAM PRINT file. FORTRAN control characters are given in IBM System/360 Time Sharing System: Command System User's Guide. FORTRAN control characters for PRINT files are listed under "PRINT files," in Section 9.

The machine-code options can be ignored by most users; they should never be specified unless the user has coded hexadecimal data into the first byte of every record. The user-supplied byte must contain the bit configuration specifying a write and the desired carriage or stacker-select operation. Only those control characters that include a write-specification are permitted; the independent carriage and stacker-select operations are excluded. A list of the machine codes appears in Command System User's Guide.

Those portions of a DDEF command that are applicable to determine or specify the characteristics of a data set operated on by PL/I programs are presented in Figure 14. Other parameters and options of the general DDEF command, as described in the publication Command System User's Guide, are not given because they are ignored or overridden by the PL/I I/O routines.

Specification of DDEF commands for peripheral devices of the CPU is also described in the publication Command System User's Guide.

The DDEF command that defines a cataloged data set is brief and simple. The only required operand fields are DDNAME and DSNAME. Other operand fields are unnecessary since other information about the data set is described in its catalog entry. For a cataloged data set if SPACE, UNIT, LABEL, or VOLUME operands are entered, diagnostics will be displayed as appropriate. However, the associated fields will be taken correctly from the existing catalog entry.

DDEF commands that define uncataloged data sets can be divided into two groups: (1) those defining new data sets (data sets that are to be generated during the run but do not yet exist) and (2) those defining old (already existing, but uncataloged) data sets. These old, uncataloged data sets can exist only on private volumes.

Operation	Operand
DDEF	DDNAME=data definition name [,DSORG={PS|VI|VP|VS}] ,DSNAME=[{data set name *data set name}] [UNIT=({DA[,direct access device type] TA[,tape type] AFF=data definition name})] [,SPACE=({CYL|TRK|record length},primary [,secondary][,HOLD])] [,VOLUME=([PUBLIC PRIVATE volume sequence number] ,[volume serial number,...],)] [,LABEL= ([file sequence number][,{NL|SL|AL}] [,RETPD=retention period])] [,DISP={OLD|NEW|MOD}] [,OPTION={CONC|JOBLIB}] [,RET=retention code] [PROTECT={Y|N}] [,DCB= ([,DSORG=code] [,RECFM=code] [,LRECL=integer] [,BLKSIZE=integer][,BUFOFF=integer] [,KEYLEN=integer] [,RKP=integer] [,PAD=integer] [,DEVD=code] [,DEN=integer] [,TRTCH=code] [,BUFNO=integer] [,OPTCD={W|A}] [,IMSK=code] [,NCP=integer])]

Figure 14. Full DDEF Command for the PL/I User

To define a new data set that is to be written on a public volume, the user can use the DDNAME, DSNAME, SPACE, DSORG, and LABEL operand fields. Exactly which fields he uses other than DDNAME and DSNAME, which are required, depends on the character of his particular data set. To define a new data set that is to be written on a private volume, the user must give DDNAME, DSNAME, UNIT, and VOLUME operands. If he wishes, he can also furnish DSORG, SPACE, LABEL, and DISP fields.

The user defines an old, uncataloged data set by specifying the DDNAME, DSNAME, VOLUME, and UNIT fields. The remaining fields can be defaulted for all data sets except unlabeled tapes.

The description of the basic DDEF command given in Section 8 also applies to the full DDEF command. If the data set is old, the full DDEF command can be used to override data set specifications already given in the standard label; however, the user is cautioned that to do this may cause errors in processing the data.

New data sets can differ radically from the standard data set resulting from the basic DDEF command. In particular, the user can define output data sets that are compatible with other systems.

DDNAME

Specifies the data definition name.

Specified as one to eight alphameric characters; the first character must be alphabetic. DDNAME must not begin with SYS, because these characters are reserved to prefix system-generated data definition names.

Since DDNAME is a required parameter, it cannot be defaulted.

DSORG

In the basic DDEF command this is virtual sequential (VS), virtual index sequential (VI), or virtual partitioned (VP).[1] The other option is physical sequential (PS).

The PS option must be used for tapes or disks that originate outside the TSS/360 environment and for tapes or disks that are to be written under TSS/360 and then transferred to other systems for processing.

DSNAME

See the discussion under "Basic DDEF Command," in Section 8.

No more than one member of a partitioned data set can be processed at one time.

The data set name can optionally be specified within apostrophes. In this case, the name need not consist of alphameric characters.

The *data set name option of the full DDEF command is needed only when processing tape or disk data sets written by OS/360 with 44-character data set names. Therefore, this option is used only with a dsorg of PS. Subsequent references to the name do not include the asterisk prefix.

UNIT

This is required only for uncataloged data sets.

$$\text{UNIT}=\left(\text{DA},\left\{\begin{matrix}2311\\2314\end{matrix}\right\}\right)$$

Specifies direct access (either a 2311 Disk Storage Drive or a 2314 Multi-disk Storage Drive).

UNIT=(TA,{7|7DC|9})

Specifies that a tape unit (7-track, 7-track with data conversion, or 9-track) is required for the data set. If given, it should agree with the DEVD parameter in the DCB field.

UNIT=(AFF=data definition name)

Specifies unit affinity. The data set being defined is to be assigned the same device reserved for the data set identified by symbol, which is the data definition name of a previously issued DDEF command. This subfield cannot be used if the data set is new and is to be on a direct access device. This subfield can be specified only for PS data sets.

SPACE

The SPACE parameter is never required for existing data sets. It can be used for new virtual data sets (DSORG is VI, VS, or VP) to request an initial allocation of public storage that is different from that specified at system generation time. Its function in this respect is of interest only if the expected size of the data set is either much larger or much smaller than

---

[1]The DSORG parameter is also present within the DCB sublist of the full DDEF command. This distinguishes between the different forms of VP, namely virtual index sequential partitioned (VIP) and virtual sequential partitioned (VSP), and identifies the organization of the partitioned data set member to be processed.

the standard system allocation. In these cases, it permits somewhat greater efficiency in storage allocation. Even if the storage required is greater than the standard allocation, additional storage is automatically issued so that the SPACE parameter is never critical for virtual data sets.

Form 1

```
SPACE=(,primary[,secondary][,HOLD])
```

This form is used to request allocation parameters for virtual data sets that differ from the system standard. Primary and secondary allocation are in space units of 4096 bytes (pages). Primary specifies the number of initial space units to be allocated to the data set. It is one to three digits. Secondary is the number of space units to be allocated each time the space allocated to the data set has been exhausted and more data is to be written. This allocation consists of a one- to three-digit decimal number.

The HOLD option within the SPACE parameter specifies that unused storage assigned to the data set is not to be released when the data set is closed.

Form 2

```
SPACE=({TRK|CYL|record length},primary
[,secondary][,HOLD])
```

This form is used for direct access devices where dsorg is PS. It allocates space in units defined by the first subparameter, namely tracks, cylinders, or record lengths.

VOLUME

Form 1

```
VOLUME=({PRIVATE                    })
       {,volume serial number[,...1]}
```

The volume parameter is required for old, uncataloged data sets that reside on private volumes. It can also be supplied for new data sets that are to reside on private volumes. Volume serial numbers can be one to six characters and should uniquely identify a particular disk pack or tape reel that is to be mounted. If any nonalphameric characters are used in the volume serial number, it must be enclosed in apostrophes. If PRIVATE is specified and the data set is new, the system obtains an available volume and informs the user of the volume selected.

In general, therefore, this form of the VOLUME field is needed only for data sets that are not cataloged. It applies mainly when dsorg is PS and an OS/360-generated disk pack or tape is to be read.

Form 2

```
VOLUME=(volume sequence number)
```

Where a data set extends over more than one volume, this form specifies the sequence number of the volume to be read or written. The number consists of one to four digits. This form is meaningful only if the data set has PS organization, is cataloged, and its earlier volumes are not to be processed.

Form 3

```
VOLUME=PUBLIC
```

This form is used for a new public data set if the user specifies a device type in the UNIT parameter. If PUBLIC is specified, the volume serial number is not recognized. PUBLIC is also assumed if the VOLUME parameter is not specified.

LABEL

This parameter applies only when the data set organization is PS. It is generally used only when magnetic tapes are to be processed, since all data sets on direct access volumes have labels known as Data Set Control Blocks (DSCBs). The RETPD subparameter, however, is applicable to all PS data sets.

If the entire label field is defaulted, the labeling conventions specified by the installation are assigned. However, if the data set is cataloged, label information is retrieved from the catalog.

Form 1

```
LABEL=(file sequence number)
```

Specifies the file sequence number of a data set on tape when multiple data sets are on one tape volume. This facility, therefore, permits the user to skip one or more data sets in order to find the one of interest and implies that the program should not issue a REWIND for that data set. The file sequence number is one or two decimal digits.

Form 2

```
LABEL=(,[NL|SL|AL],RETPD=days)
```

The options shown are NL for no labels, SL for standard labels, and AL for standard ASCII labels (see Appendix E). The exact meaning of standard labels is installation dependent. The NL option should not be

used for PL/I output data sets unless a definite reason exists, since a tape data set without labels requires a more complicated DDEF command when read back by a PL/I program than one with labels.

RETPD specifies the number of retention days and applies to output tapes with standard labels and to direct access output.

If defaulted, RETPD is set to zero to permit immediate rewriting of any tape or direct access data set.

DISP

DISP= {OLD | NEW | MOD}

Uses of DISP=OLD and DISP=NEW are described under "Basic DDEF command," in "Section 7: Data Sets and PL/I Files."

DISP=MOD applies only when the data set organization is PS and a private volume is being processed. This option causes logical positioning after the last record of the data set. Additional WRITE statements are then possible to expand the data set. This option applies mainly to magnetic tapes.

OPTION

OPTION=CONC

Specifies that a data set is being added to the concatenated data set named as ddname. The order of concatenated data sets is the same as the order in which they are defined. Only existing PS data sets can be concatenated.

OPTION=JOBLIB

Specifies that the data set is to be used as a job library. The data set name specified in the DSNAME field is entered into the program library list. The data set organization must be VP.

RET

The RET parameter allows the owner of a virtual storage data set to specify the storage type, and deletion and access attributes of a data set:

RET=([P|T][C|L][U|R])

Storage types [P|T]:

P -- permanent storage

T -- temporary storage
  -- if neither is specified, permanent storage (P) is assumed.

Deletion options [C|L]:

C -- delete at CLOSE

L -- delete at LOGOFF
  -- if not specified, deletion at LOGOFF (L) is assumed for a temporary (T) data set.
  -- a permanent data set (P) is not deleted automatically.

Access attributes [U|R]:

U -- read/write

R -- read-only
  -- if neither is specified, read/write (U) is assumed.

PROTECT

PROTECT= {Y|N}

Applicable to data sets on tape, this operand specifies whether file protection, i.e., no file protect ring, is required.

Y -- the tape will be mounted without a file protect ring, unless DISP is NEW or MOD, in which case the DDEF command is canceled. A tape already mounted with a file protect ring will be remounted in order to have the ring removed.

N -- the tape will be mounted with a file-protection ring, regardless of the DISP value. If the tape is already mounted without a ring, it will be remounted in order to have the ring put in.
  -- if neither Y nor N is specified, N is assumed if DISP is NEW or MOD. There is no IBM-supplied default for DISP=OLD; this default will be whatever is called for by the installation.

If the device is not magnetic tape, the PROTECT operand is ignored.

DCB

A data control block (DCB) is one of the major control tables used for communication between TSS/360 data management and any program requiring control of a data set. For every data set, the PL/I I/O routines build a DCB as it is encountered in executing the object program. The DCB is initially void, but can be filled from information in the DDEF commands, the file declaration, the OPEN options, or the input data set labels. Therefore, any required information not in the DDEF command is entered from one of these sources.

The only DCB suboperands of critical interest to the PL/I user are RECFM, LRECL,

KEYLEN, and RKP; KEYLEN and RKP apply only to VI data sets. Of the remaining DCB parameters, PAD applies only to VI data sets; BLKSIZE, BUFOFF, DEVD, DEN, TRTCH, BUFNO, NCP, OPTCD, and IMSK apply only to PS data sets.

DCB Suboperands - RECFM, LRECL, and BLKSIZE:

RECFM:

RECFM indicates the format of the records in the data set. Specified as:

For VAM data sets:

RECFM={F[A|M] / V[A|M] / U[A|M]}

Where the record format is:

F -- fixed-length records
-- maximum record length is 32,756 bytes for VS, and 4,000 bytes for VI

V -- variable-length records
-- each record contains in the first four bytes a binary count of the number of bytes in the record
-- maximum record length is 32,756 bytes for VS, and 4,000 bytes for VI

U -- undefined-length records
-- applicable to VS and VP data sets only
-- record length always a multiple of a page (4096 bytes)
-- maximum record length is 1,048,576 bytes

The default value is V.

Figure 15 shows the F, V, and U record formats for VS data sets. Figure 16 shows the F and V record formats for VI data sets

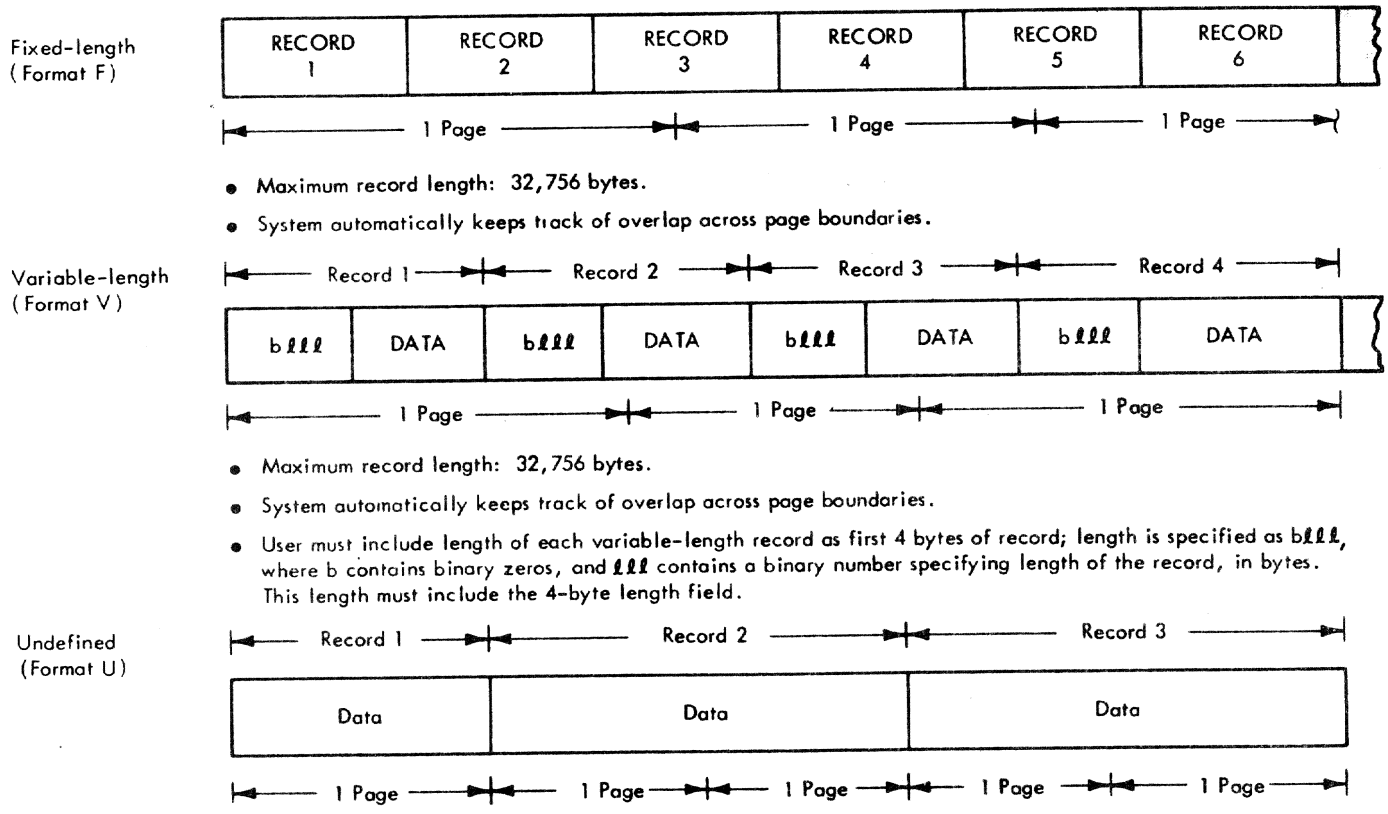
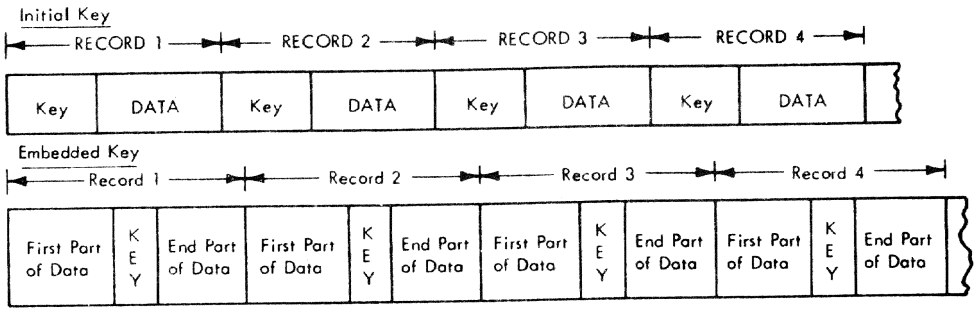
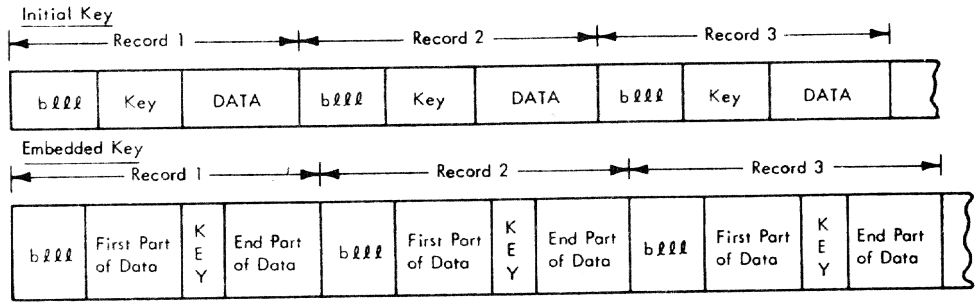


Figure 15. Record Formats -- VS Data sets

Fixed-length
(Format F)

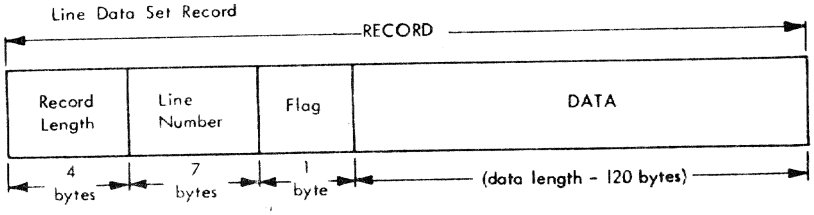


Variable-length
(Format V)



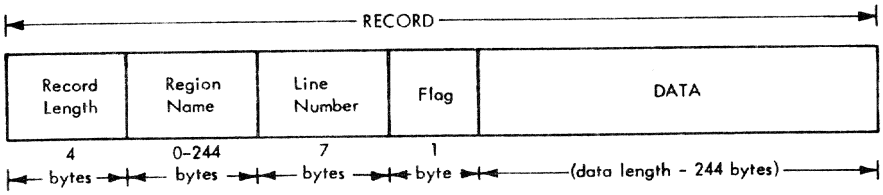
- Maximum logical record length: 4000 bytes.
- Maximum number of records per data page: 1300.
- Maximum key length: 255 bytes.
- Maximum number of data pages: 65,000.
- Maximum number of overflow pages: 240.
- Maximum number of records per overflow page: 255.
- Maximum number of directory pages: 255.
- User must include length of each variable-length record as first 4 bytes of record; length is specified as bℓℓℓ, where b contains binary zeros, and ℓℓℓ contains a binary number specifying length of the record, in bytes. This length must include the 4-byte length field.

Line Data Set Record



- Maximum record length: 132 bytes.
- Maximum data length: 120 bytes.
- Flag byte indicates whether record originally came from terminal keyboard (01) or card reader(00).

Region Data Set Record



- Maximum record length: 256 bytes
- Maximum data length: 244 bytes
- Flag byte indicates whether record originally came from terminal keyboard (01) or card reader (00).

Figure 16. Record Formats -- VI Data sets

Where the record contains:

A -- FORTRAN control character
M -- machine code control character

If A or M is not specified, no control character is assumed. Refer to Appendix C for a discussion of control characters.

For PS data sets:

```
        ⎧F[B|S|T|BS|BT|BST|ST][A|M]⎫
RECFM=  ⎪V[B|T][A|M]                ⎪
        ⎪U[T][A|M]                  ⎪
        ⎩D[B]                       ⎭
```

Where the record format is:

F -- fixed-length records
  -- maximum record length is 32,760 bytes

V -- variable-length records
  -- maximum record length is 32,756 bytes

D -- variable-length records used with tapes in ASCII format (see Figure 19, in this appendix, and "ASCII Tapes," in Appendix E)

U -- undefined-length records
  -- records have physical attributes (on a tape) and can vary in length (that is, in blocksize)

Figures 17, 18, and 19 show the possible record formats for PS data sets, where the physical attributes are:

B -- blocked records

S -- standard data set; no truncated blocks or unfilled tracks

T -- track overflow employed

Where the record contains:

A -- FORTRAN control character
M -- machine code control character

Refer to Appendix C for a discussion of control characters.

Notes on Record Format

1. Absence of any of the physical attribute mnemonics implies the opposite of that attribute. For instance, writing RECFM=V implies: variable-length, unblocked records, no control character, and no track overflow feature.

2. If the RECFM suboperand is omitted, record-format information can be supplied by the user's program or the data set label.

3. Figures 15-19 of this appendix illustrate external record formats -- the formats seen by the user. Internal formats may differ. For descriptions of internal record formats, see IBM System/360 Time Sharing System: Access Methods PLM, GY28-2016.

LRECL:

LRECL specifies record length. It must be four bytes greater than the largest logical record that is to be read or written.

For format-F records, LRECL must be specified. If defaulted for format-V records, it is assumed to be 4096 for VS, 4000 for VI, or 4096 for PS; if VS format-V records longer than 4096 bytes are to be accessed, LRECL must be specified. For format-U records, LRECL is ignored.

BLKSIZE:

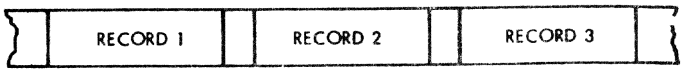
BLKSIZE is required only if RECFM is FB (fixed-length blocked records) and this option, in turn, is meaningful only if DSORG is PS. In this case, it must be a multiple of LRECL. Otherwise, any value given is ignored and replaced by LRECL.

Examples of how to use the RECFM, LRECL, and BLKSIZE parameters are shown below (they are not complete DDEF commands; only the DSORG and DCB portion is shown).

```
VS,DCB=(RECFM=F,LRECL=80)
       VS 80-character fixed-length records

VS,DCB=(RECFM=V)
       VS variable length records
       (standard)

VS,DCB=(RECFM=FA,LRECL=133)
       VS data set for listing

VI,DCB=(RECFM=V,RKP=4,KEYLEN=4)
       VI variable-length records with
       4-byte key in initial position
       (after 4-byte control word)

PS,DCB=(RECFM=FB,LRECL=100,BLKSIZE=1000)
       PS fixed-length blocked with 10
       records per block
```

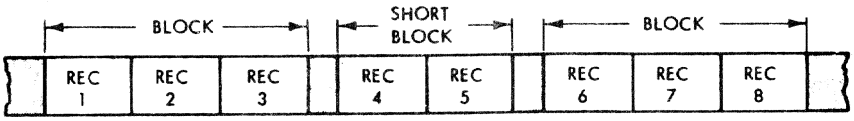
DCB Parameters -- VI Associated: If DISP=NEW and dsorg is VI, the user must specify record key position (RKP), key length (KEYLEN), and optionally padding percent (PAD). RKP specifies the displacement of the key field from the first byte of the logical record. For a full discussion of RKP refer to the subject "Initial and Embedded Keys," in Section 10. KEYLEN specifies the length of the key, in bytes. PAD specifies the percent of space (to a limit of 50 percent) to be left available

Fixed-length
(Format F)



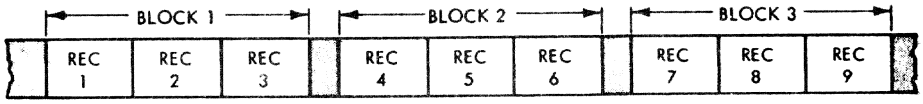
- Maximum record length – 32,760 bytes.
- Each block treated as a logical record.

Fixed-length
Blocked
(Format FB)



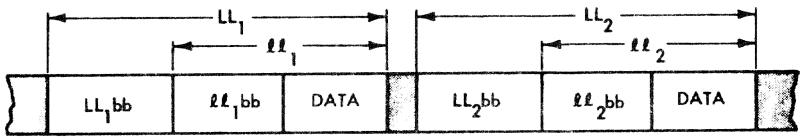
- Maximum block length – 32,760 bytes.
- Blocking factor is usually constant; however, data set may contain truncated or short blocks.

Fixed-length,
Blocked
Standard Blocking
(Format FBS)



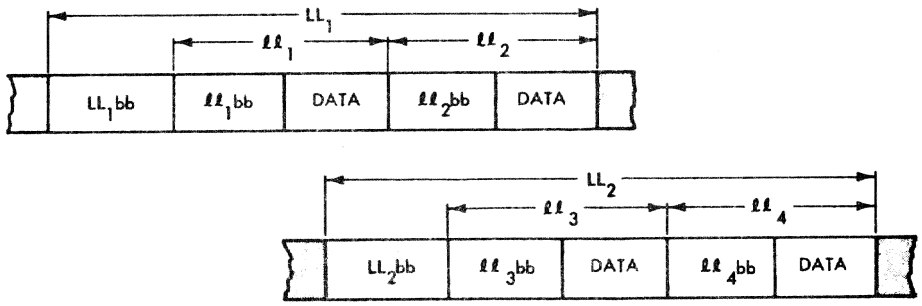
- Maximum block length – 32,760 bytes.
- Last block may be truncated; truncated block invokes end-of-volume routines.

Variable-length
(Format V)



- Maximum logical record length – 32,756 bytes.

Variable-length,
Blocked
(Format VB)



- Maximum logical record length – 32,763 bytes.
- Each logical record must describe its own length; this information must be included by user as first 4 bytes of each record:

  ll – Binary number specifying record length in bytes.

  bb – Binary 0s.

- System performs length checking of blocks containing Format-V records, based on user-supplied length information; when data sets with Format-V records (either blocked or unblocked) are created, a 4-byte control block is required in the form LLbb, where:

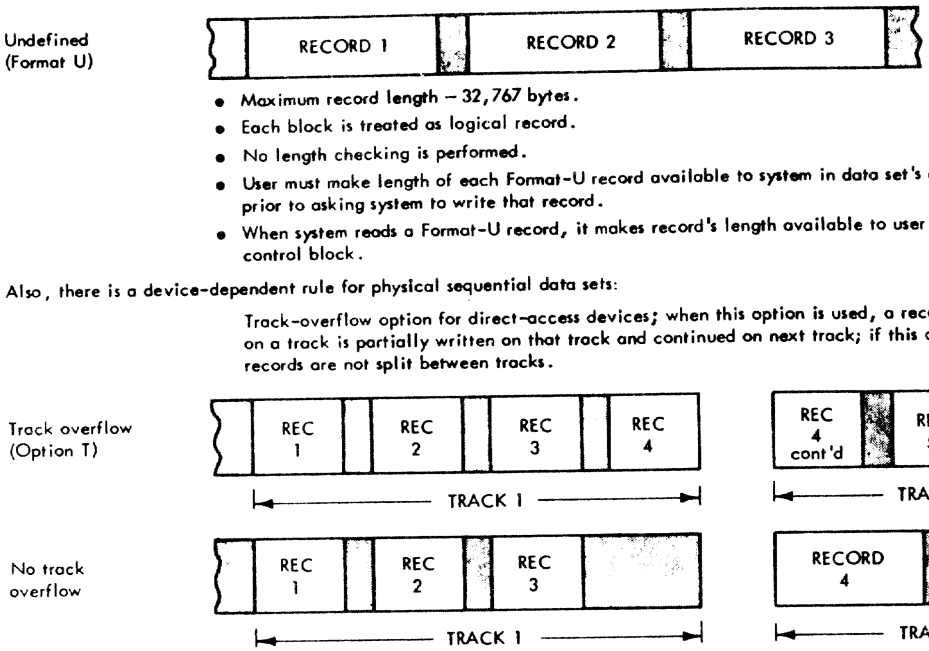
  LL – Binary number specifying block length in bytes.

  bb – Two bytes reserved for system use.

  Value of LL is determined by adding the ll s of the records within block and adding 4 bytes for the control field.

- Format-V and Blocked Format-V records cannot be processed on 7-track tape units without data conversion feature.

Figure 17. Record Formats -- Physical Sequential Data Set Without Keys (Part 1 of 2)

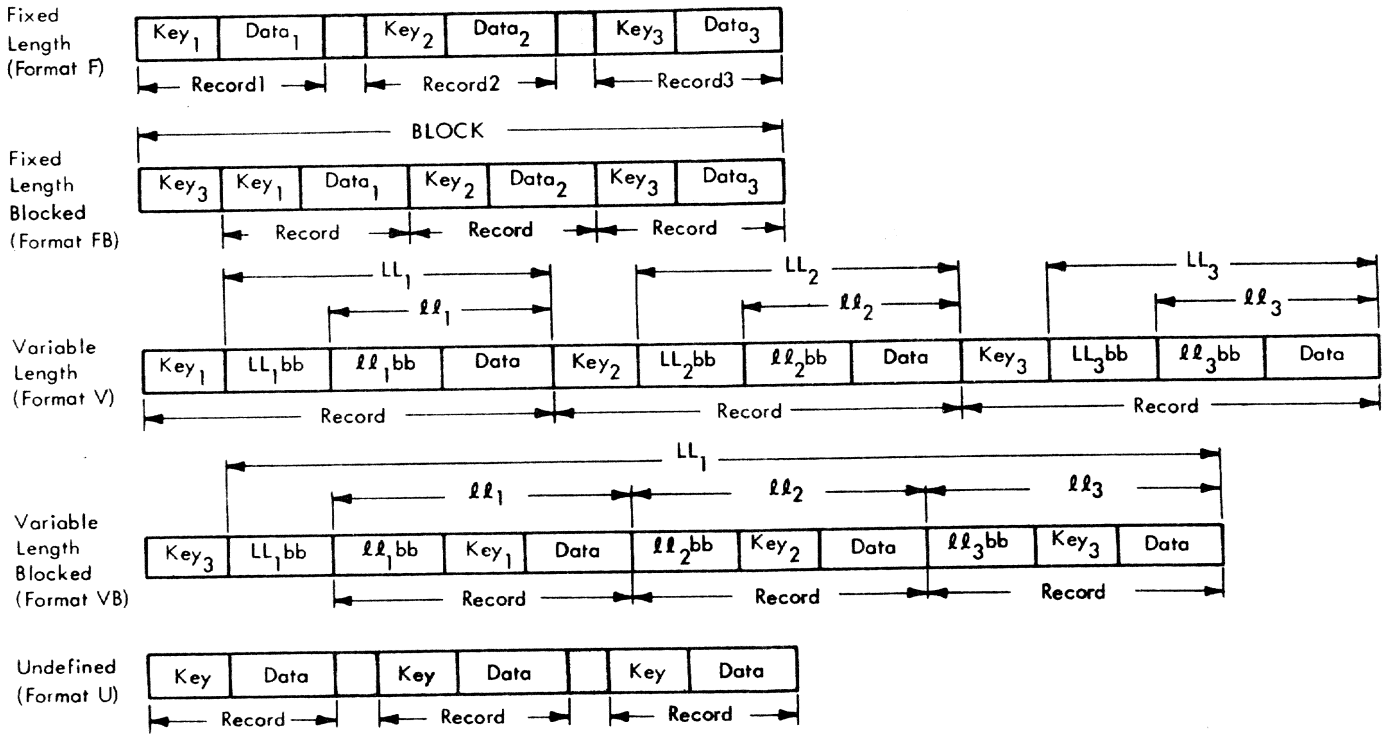


- Maximum record length – 32,767 bytes.
- Each block is treated as logical record.
- No length checking is performed.
- User must make length of each Format-U record available to system in data set's data control block, prior to asking system to write that record.
- When system reads a Format-U record, it makes record's length available to user in data set's data control block.

Also, there is a device-dependent rule for physical sequential data sets:

Track-overflow option for direct-access devices; when this option is used, a record that does not fit on a track is partially written on that track and continued on next track; if this option is not used, records are not split between tracks.

Figure 17. Record Formats -- Physical Sequential Data Set Without Keys (Part 2 of 2)



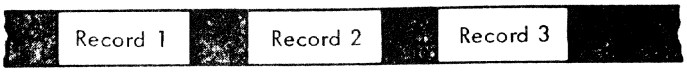
The same rules apply to physical sequential data sets with keys as for those without keys; also:

- All keys in data set must be the same length.
- Number of bytes transmitted in a READ or WRITE operation equals the key plus the data portion of record.

Note: Non-zero KEYLEN operand in DCB identifies data set with keys.

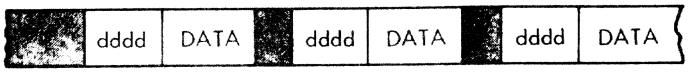
Figure 18. Record Formats -- Physical Sequential Data Sets With Keys

Fixed-length, Blocked and unblocked (Format F)



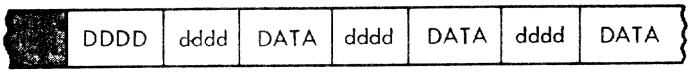
- Maximum record length - 32,760 bytes
- Buffer offset not supported
- Data in EBCDIC form is translated to ASCII

Variable-length, Unblocked (Format D)



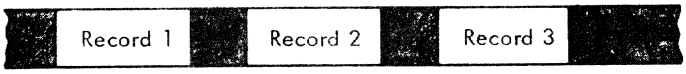
- Maximum logical record length - 32,756 bytes
- Block descriptor in example has been stepped over
- Each logical record must describe its own length; this information must be included as first four bytes of each record:
  dddd - unpacked decimal number specifying length in bytes
- dddd and DATA are translated to ASCII
- Buffer offset of 0 and 4 are supported

Variable-length Blocked (Format DB)



- Maximum logical record length - 32,763 bytes
- System performs length checking of blocks containing format-D records, based on user supplied length information; when data sets with format-D records (either blocked or unblocked) are created, a 4-byte control block in the form DDDD is required, where:
  DDDD - unpacked decimal number specifying block length in bytes
  Value of DDDD is determined by adding the dddd's of the records within the blocks and adding 4 bytes for the control field.
- DDDD, dddd, and DATA are translated to ASCII

Undefined (Format U)



- Maximum record length - 32,767 bytes
- Each block is treated as a logical record
- No length checking is performed
- User must make length of each format-U record available to system in data set's data control block
- Buffer offset not supported
- Format U is supported when 128 character set is used
- Data translated to ASCII

Note: This represents the output after the system has processed the infernal EBCDIC data format described in Figure 27.

Figure 19. Output Record Formats for ASCII Tapes

within the pages of a VI data set, thus providing for insertions within pages.

DCB Parameters -- PS Associated: If DSORG is PS, a large number of DCB parameters can be used that otherwise have no meaning. As previously discussed, the BLKSIZE parameter is required if RECFM=FB. In addition, the parameters listed below apply. Note: If the DDEF command is for an ASCII tape (see Appendix E), all relevant DCB parameters must be specified; there is no other source of information.

1. BUFOFF=integer: Length of buffer offset field (that is, block descriptor); used only with tapes in ASCII format. Acceptable buffer offset values are:

	RECFM		
	F	D	U
input	0-99	0-99	0-99
output	0	0 or 4	0

   If BUFOFF is specified, BLKSIZE must include the buffer offset field.

2. DEVD=Code: Specifies the device on which the data set resides. It is not required for cataloged data sets; it can be one of the following:

   a. DA specifies direct access (disk formatted in accordance with Operating System/360 conventions). In this case KEYLEN has a special meaning since it specifies how many of the initial bytes of each record are to be written on the disk (or read from it) as a key. This condition has no connection with VI and the key cannot be used for random access by the PL/I programmer. If all processing is to be done on TSS/360, it is not necessary to use it. However, if a data set is to be written on a disk pack for the purpose of being processed on another IBM system (for example, OS/360), the use of KEYLEN may be required.

   b. TA specifies magnetic tape. If 7-track tape is specified in the UNIT parameter, DEN is given a value of 0, 1, or 2 for recording density of 200, 556, or 800 bytes, respectively. If 7-track tape is to be read, TRTCH can be given as C for data conversion, E for even parity, and T for BCDIC to EBCDIC conversion. The defaults are odd parity and no translation.

3. BUFNO=1: Physical sequential I/O normally takes place with two buffers. The user can reduce space allocation requirements by specifying the number of buffers as only one. Any other value given to BUFNO is disregarded.

4. OPTCD=W|A: Specifies an optional service to be provided. OPTCD=W applies only for direct access output. It causes additional checking of all write operations; this increases execution time. OPTCD=A must be specified if an ASCII tape is being used.

5. IMSK=Code: Specifies a 4-byte hexadecimal number whose bit pattern indicates the system's error-handling procedures to be invoked. If FFFFFFFF is written, the system is to apply all optional error recovery procedures. This is the default condition. If 00000000 is written, the system is to apply none of its optional error recovery procedures. If any other 4-byte hexadecimal number is written, the system applies its error-recovery procedures wherever a bit is set to one in IMSK which corresponds to an error condition. The first two bytes correspond to the first two bytes of the channel status word, and the other two correspond to the first two sense bytes. Bit positions in each byte for specification of system error recovery procedure are in the following format:

   XXXXXXXB XCXXXXXD YEFGHIYY YYYYYYYY

   where a one bit in a given position indicates that the system is to handle the associated error condition.

   X = System never tests this bit to determine entry to retry routines
   Y = Device-dependent conditions
   B = Unit exception
   C = Incorrect length
   D = Channel chaining check
   E = Intervention required
   F = Bus-out parity
   G = Equipment check
   H = Data check
   I = Overrun

6. NCP=integer: Specifies the maximum number of I/O events that can be outstanding for a file at any instant. This number must not exceed 99. Use of the NCP subparameter is valid only with PS data sets accessed by BSAM (that is, CONSECUTIVE SEQUENTIAL UNBUFFERED files).

APPENDIX E: EXTERNAL STORAGE DEVICES

This appendix states the record formats that are acceptable for magnetic tapes and direct access devices, and summarizes the salient operational features of these devices.

MAGNETIC TAPE

Nine-track magnetic tape is standard in IBM System/360, but some 2400 series magnetic-tapes drives incorporate features that facilitate reading and writing 7-track tape. The translation feature changes character data from EBCDIC (the 8-bit code used in System/360) to BCD (the 6-bit code used on 7-track tape) or vice-versa. The data conversion feature treats all data as if it were in the form of a bit string, breaking the string into groups of eight bits for reading into storage, or into groups of six bits for writing on 7-track tape; the use of this feature precludes reading the tape backwards. If the user wants to employ translation or data conversion, he must include the TRTCH (tape recording technique) subparameter in the DDEF command.

The user can specify format-F, format-V, or format-U records for 9-track magnetic tape, but format-V records are acceptable on 7-track tape only if the data conversion feature is available. (The data in the control bytes of format-V records is in binary form; in the absence of the data conversion feature, only six of the eight bits in each byte are transmitted to 7-track tape.

The maximum recording density available depends on the model number of the tape drive used; single-density tape drive units have a maximum recording density of 800 bytes per inch, and dual-density tape drive units have a maximum of 1600 bytes per inch. For 9-track tape, a single-density drive offers only the 800 bytes per inch density; the standard density for a dual-density drive is 1600 bytes per inch, but the DEN (density) subparameter of the DDEF command can be used to specify 800 bytes per inch. For 7-track tape, the standard recording density for both types of drive unit is 200 bytes per inch; the user can employ the DEN subparameter of the DDEF command to select alternatives of 556 or 800 bytes per inch.

ASCII Tapes

The user can process and create magnetic tapes formatted in American National Standard Code for Information Interchange, ANSI X3.4-1967, hereinafter referred to as ASCII. The following restrictions apply to use of ASCII tapes:

- The DDEF command for an ASCII data set differs from that for an EBCDIC data set. See the discussions of DSNAME, LABEL, RECFM, BUFOFF, and OPTCD in Appendix D.
- The data management routines do not translate EBCDIC data to ASCII data if the EBCDIC data is in packed format; it must be expanded to character format.
- When using a CDS command with an ASCII tape as input, the block descriptor may be destroyed.
- ASCII tapes cannot be cataloged.

Figure 19 illustrates output record formats for data sets recorded on ASCII tapes; the Data Management Facilities publication illustrates ASCII tape labels.

DIRECT ACCESS DEVICES

Direct access devices accept format-F, format-V and format-U records. The storage space on these devices is divided into conceptual cylinders and tracks. A cylinder is usually the amount of space that can be accessed without movement of the access mechanism, and a track is that part of a cylinder that is accessed by a single read/write head. For example, a 2311 disk pack has ten recording surfaces, each of which has 200 concentric tracks; thus, it contains 200 cylinders, each of which includes ten tracks.

The following reference cards contain tables that will enable the user to determine the amount of space he will require:

2311 Disk Storage Drive, Form X20-1705

2314 Storage Facility, Form X20-1710

Operands not explained in this manual are explained in Command System User's Guide.

Operation	Operand
ABEND	
ASM	NAME=module name [,STORED={Y | N}] [,MACROLIB=({data definition name of symbolic portion, data definition name of index portion}[,...])] [,VERID=version identification][,ISD={Y|N}][,SYMLIST={Y|N}] [,ASMLIST={Y|N}][,CRLIST={Y|N}] [,STEDIT={Y|N}][,ISDLIST={Y|N}][,PMDLIST={Y|N}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]
AT	instruction location[,...]
BACK	DSNAME=data set name
BEGIN	application name[,application parameters-if required]
Note: For MT/T use only	
BRANCH	INSTLOC=instruction location
BUILTIN	NAME=command name[,EXTNAME=bpkd macro name] [,DSNAME=data set name]
CANCEL	BSN=batch sequence number
Form 1 CATALOG	DSNAME=data set name[,STATE={N|U}][,ACC={R|U}] [,NEWNAME=data set name]
Form 2 CATALOG	GDG=generation data group name,GNO=number of generations [,ACTION={A|O}][,ERASE={Y|N}]
CDD	DSNAME=data set name [, {data definition name | (data definition name[,...])}]
CDS	DSNAME1=input data set name[(member name[,...])], DSNAME2=copy data set name[(member name)] [,ERASE={Y|N}][, {BASE=first line number[,INCR=increment] | REPLACE={R|I}}]
CHGPASS	[PASSWORD]
CLOSE	[DSAME=data set name][,TYPE=T] [,DDNAME=data definition name]
CONTEXT	[N1=starting position][,N2=ending position], STRING1=search string[,STRING2=replacement string]
CORRECT	N1=starting line][,N2=ending line][,SCOL=first column] [CORMARK=replacement correction characters][,CHAR={C|M|H}]

Operation	Operand
DATA	DSNAME=data set name [,RTYPE={{I | LINE}[,BASE=first line number,INCR=increment] | FTN | CARD | S}]
DDEF (basic)	DDNAME=data definition name[,DSORG={VI|VS|VP}], DSNAME=data set name[,DISP={OLD|NEW}[,DCB=([RECFM={F|V|U}] [,LRECL=integer][,RKP=integer])]
DDNAME?	JOBLIB={Y|N}
DEFAULT	[operand=[value]}[,...]
DELETE	[DSNAME=data set name]
DISABLE	
DISPLAY	{data field name[,...] ID?data field name[,...]}
DSS?	[NAMES={data set name | (data set name[,...])}]
DUMP	data field name[,...] ID?data field name[,...]
EDIT	DSNAME=data set name[(member name)] [,RNAME=region name][,REGSIZE=region length]
ENABLE	
END	
ERASE	[DSNAME=data set name]
EVV	DEVICE=device type,VOLUME=(volume serial number[,...])
EXCERPT	DSNAME=data set name[(member name)][,RNAME=region name] [,N1=starting line[,N2=ending line]]
EXCISE	[N1=starting line][,N2=ending line]
EXECUTE	DSNAME=data set name
EXHIBIT	OPTION={BWQ[,TYPE={ALL | BSN.number}] | UID[,TYPE={ALL | CONV | BACK | UID.userid}]}
EXIT	[SIRTEST={Y | N}]
EXPLAIN	{MSGIDD | ORIGIN {WORD | TEXT | RESPONSE | MSGE | MSGS} [,message identification]}

Operation	Operand
FTN	NAME=module name[,STORED={Y|N}][,VERID=version identification] [,ISD={Y|N}][,SLIST={Y|N}][,OBLIST={Y|N}][,CRLIST={Y|N}] [,STEDIT={Y|N}][,MMAP={Y|N}][,BCD={Y|N}][,PUBLIC={Y|N}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]
GO	
IF	condition
INSERT	[N1=preceding line][,INCR=increment]
JOBLIBS	DDNAME=data definition name
KEYWORD	[COMNAME=procdef name]
LINE?	DSNAME=data set name [,{line number | (first line number,last line number)}[,...]]
LIST	[N1={starting position|CLP}][,N2=ending position] [,CHAR={C|H|M}]
LNK	NAME=module name[,STORED={Y | N}] [,LIB=data definition name of library] [,VERID=version identification][,ISD={Y|N}][,PMDLIST={Y|N}] [,LISTDS={Y|N}][,LINCR(first line number,increment)]
LOAD	[NAME=entry point name]
LOCATE	N1=starting position][,N2=ending position] [,STRING=character string]
LOGOFF	
LOGON see note	user identification,[password],[addressing],[charge number] ,[control section packing],[maximum auxiliary storage] ,[pristine],[user IVM code]
Notes: 1. Trailing commas can be omitted 2. Use positional format only (no keywords).	
MCAST	[EOB = {character | X'characters'}] [,CONT = {character | X'characters'}] [,CLP = {character | X'characters'}] [,TRP = {character | X'characters'}] [,RCC = {character | X'characters'}] [,SSM = {character | X'characters'}] [,USM = {character | X'characters'}] [,KC = {character | X'characters'}] [,RS = {character | X'characters'}] [,CP = {1 to 8 characters | X'2-16 characters'}]

Operation	Operand
MCASTAB	[INTRAN = {N | Y}] [,OUTRAN = {Y | N}]
MODIFY	SETNAME=data set name[,CONF=R][,LRECL=record length, KEYLEN=key length,RKP=key displacement,RECFM={V|F}] [,FTN={Y|N}]
NUMBER	[N1=starting line][,N2=ending line][,BASE=base number] [,INCR=increment]
PC?	[NAMES={data set name | (data set name[,...])}]
PERMIT	DSNAME={data set name|*ALL} [,USERID={(user identification[,...])|*ALL}] [,ACCESS={R|RO|RW|U}]
PLI	[NAME=module name][,PLIOPT=compiler option list] [,PLCOPT=language controller options] [,SOURCEDS=source data set name] [,MERGELST=converter input list] [,MERGEDS=converter input data set] [,MACRODS=intermediate data set name] [,EXPLICIT={name | (name,...) | *ALL[(name[,...])]}] [,XFERDS=dataset name]
POD?	PODNAME=data set name[,DATA=Y][,ALIAS=Y] [,MODULE={ALL|module name}]
POST	
PRINT	DSNAME=data set name[,STARTNO=starting position] [,ENDNO=ending position] [,PRTSP={EDIT | 1 | 2 | 3}[,HEADER=H][,LINES=lines per page][,PAGE=P]] [,ERASE={Y|N}][,ERROROPT={ACCEPT|SKIP|END}] [,FORM=paper form][,STATION=station id]
PRMPT	MSGID=message identification[,INSERTn=inserted characters [,...]]
PROCDEF	NAME=procedure name[,DSNAME=data set name]
PROFILE	[CSW={N|Y}]
PUNCH	DSNAME=data set name[,CBIN=BINARY] [,STARTNO=starting position][,ENDNO=ending position] [,STACK={1|2|3|EDIT}][,ERASE={Y|N}][,FORM=card form]
PUSH	[SIRTEST={Y | N}]
QUALIFY	MNAME={link-edited module name.}object module name
REGION	[RNAME=region name]

Operation	Operand
RELEASE	DDNAME=data definition name[,DSNAME=data set name] [,{SCRATCH|HOLD}]
REMOVE	{statement number[,...] {ALL}
RET	DSNAME=data set name,RET={P|T}{L|C}{U|R}
REVISE	[N1=starting line][,N2=ending line][,INCR=increment]
RTRN	
SECURE	{(TA=number of devices[,type of device]) {(DA=number of devices[,type of device]) [,...]
SET	data location=value[,...]
SHARE	DSNAME=data set name,USERID=owner's user identification [,OWNERDS={owner's data set name|*ALL}]
STACK	
STET	
STOP	
SYNONYM	{term=[value]}[,...]
TIME	[MINS=minutes]
TV	DSNAME1=tape data set name[,DSNAME2=VAM data set name]
UNLOAD	[NAME=entry point name]
UPDATE	
USAGE	
VT	DSNAME1=VAM data set name[,DSNAME2=tape data set name]
VV	DSNAME1=current data set name[,DSNAME2=new data set name]
WT	DSNAME=current data set name,DSNAME2=tape data set name [,VOLUME=tape volume number][,FACTOR=blocking factor] [,STARTNO=starting position][,ENDNO=ending position] [,PRTSP={EDIT|1|2|3}][,HEADER=H][,LINES=lines per page][,PAGE=P] [,ERASE={Y|N}]
ZLOGON	

APPENDIX G: PL/I COMPILER OPTIONS

The PLIOPT operand of the PLI command specifies a list of PL/I options to be used by the compiler. It is considered to be one parameter, and the list of compiler options following the equal sign in the PLIOPT operand must therefore be enclosed in parentheses unless only one value is given; the separate options must be separated by commas. For an option that includes a numeric specification (for example, SIZE or LINECNT), only significant digits need be specified. Futhermore, for an option that includes more than one numeric specification (for example, SORMGIN) the numbers must be enclosed in parentheses and separated by commas.

The compiler accepts the abbreviated option names given in Table 25 as alternatives to the longer mnemonics.

There is no required order for specifying the compiler options, but if conflicting options are specified, the last specification in the list is used. The standard defaults shown in Table 25 are used unless you specify an alternative.

To simplify understanding the options, they may be grouped under six headings.

1. Control options, which establish the conditions for compilation.
2. Preprocessor cptions, which request the services of the preprocessor and specify how its output is to be handled.
3. Input options, which specify the format of the input to the compiler.

Table 25. Compiler Options, Abbreviations, and Standard Defaults

Compiler	Options	Abbreviated Names	Standard Defaults
Control options	OPT=n STMT/NOSTMT OBJNM=aaaaaaaa SYNCHKE/SYNCHKS/SYNCHKT	O ST/NST N SKE/SKS/SKT	01 NOSTMT None conversational: SYNCHKS nonconversational: SYNCHKT
Preprocessor options	MACRO/NOMACRO COMP/NOCOMP MACDCK/NOMACDCK	M/NM C/NC MD/NMD	NOMACRO COMP NOMACDCK
Input options	CHAR60/CHAR48 BCD/EBCDIC SORMGIN=(mmm,nnn,[ccc])	C60/C48 B/EB SM	CHAR60 EBCDIC (1,100)
Output options	LOAD/NO LOAD DECK/NO DECK	LD/NLD D/ND	LOAD NODECK
Listing options	LINECNT=xxx OPLIST/NOOPLIST SOURCE2/NOSOURCE2 SOURCE/NOSOURCE NEST/NONEST ATR/NOATR XREF/NOXREF EXTREF/NOEXTREF LIST/NOLIST FLAGW/FLAGE/FLAGS	LC OL/NOL S2/NS2 S1NS NT/NNT A/NA X/NS E/NE L/NL FW/FE/FS	50 OPLIST SOURCE2 SOURCE NONEST NOATR NOXREF NOEXTREF NOLIST FLAGW
Dummy options	SIZE=yyyyyy/yyyK/999999/MAX M91/NOM91 EXTDIC/NOEXTDIC	SIZE M91/NOM91 ED/NED	MAX NOM91 ED

4. Output options, which specify the type of data set that will contain the object module.

5. Listing options, which specify the information to be included in the compiler listing.

6. Dummy options, which are included solely to give compatibility with the OS/360 PL/I (F) compiler.

CONTROL OPTIONS

OPT

The OPT option specifies the type of optimization required:

OPT=0 instructs the compiler to keep object-program storage requirements to a minimum at the expense of object-program execution time.

OPT=1 causes object-program execution time to be reduced at the expense of storage.

OPT=2 has the same effect as OPT=1, and in addition requests the compiler to optimize the machine instructions generated for certain types of DO-loops and expressions in subscript lists. PL/I Language Reference Manual includes a discussion of DO-loop and subscript-expression optimization.

There is little difference in compilation time for optimization levels 0 and 1, but specifying OPT=2 could result in a substantial increase in compile time.

STMT or NOSTMT

The STMT option requests the compiler to produce additional instructions that will allow statement numbers from the source program to be included in diagnostic messages produced during execution of the compiled program.

The use of this option causes degradation of execution time. However, you can get information about statement numbers and their associated offsets by referring to the table of offsets. (See "Listing" in Part II, Section 5).

OBJNM

This option has meaning only in a *PROCESS statement. At PLI command time, this option is ignored and the value is taken from the NAME parameter. See description of *PROCESS handling in Section 5.

The OBJNM option allows you to specify a name for successive compilations in a batched compilation. The format of the option is:

OBJNM = a

where 'a' represents a name comprising not more than eight alphameric characters.

SYNCHKE, SYNCHKS, or SYNCHKT

After execution of the compiler's dictionary phase, only about 25% of the compile time has elapsed, but almost all of the source-program syntax checking is complete. The syntax-check options govern termination or continuation of compilation at this stage, if errors of a specified severity have been detected. (For descriptions of error severities, see the discussions, below, of the FLAGW, FLAGE, and FLAGS options.)

The syntax-check options and their corresponding error severities are:

SYNCHKE	error
SYNCHKS	severe error
SYNCHKT	terminal error

If you specify SYNCHKE or SYNCHKS for conversational mode, and errors of the specified severity are detected, the compiler prompts you to indicate (Y or N) whether compilation is to continue beyond the dictionary phase. In nonconversational mode, compilation is terminated. If you specify SYNCHKT for either mode, compilation is terminated only if a terminal error is detected, and there is no prompting in conversational mode.

PREPROCESSOR OPTIONS

MACRO or NOMACRO

Specify MACRO when you want to employ the compiler preprocessor. The use of the preprocessor is described under "Compile-Time Processing," in Section 5.

COMP or NOCOMP

Specify this option if you want the PL/I source module produced by the preprocessor to be compiled immediately. The source module is then read by the compiler from the data set identified by the DDNAME PLIMAC.

MACDCK or NOMACDCK

Specify this option if you want to save the intermediate macro file that has the DDNAME of PLIMAC. NOMACDCK causes the file to be erased after compilation is complete.

INPUT OPTIONS

CHAR60 or CHAR48

If the PL/I source statements are written in the PL/I 60-character set, specify CHAR60; if they are written in the 48-character character set, specify CHAR48. PL/I Language Reference Manual lists both character sets. (Note that the compiler will accept source programs written in either character set if you specify CHAR48. However, CHAR48 is inefficient and should only be used when necessary.)

BCD or EBCDIC

The compiler will accept source statement in which the characters are represented by either of two codes; binary coded decimal (BCD) and extended binary-coded-decimal interchange code (EBCDIC). For binary coded decimal, specify the option BCD; for extended binary-coded-decimal interchange code, specify the option EBCDIC. Whenever possible, use EBCDIC since BCD requires translation and is therefore less efficient. PL/I Language Reference Manual lists the EBCDIC representation of both the 48-character set and the 60-character set.

SORMGIN

The SORMGIN (source margin) option specifies the extent of the part of each input record that contains the PL/I source statements. The compiler will not process data that is outside these limits. The option can also specify the position of a FORTRAN control character to format the listing of source statements produced by the compiler if you include the SOURCE option. However, if the MACRO option is in effect, the format of the SOURCE listing produced after preprocessing cannot be controlled and will always be single spaced. The format of the SORMGIN option is:

SORMGIN=(mmm,nnn[,ccc])

where

mmm represents the number of the first byte of the field that contains the source statements,

nnn represents the number of the last byte of the source statement field, and

ccc represents the number of the byte that will contain the control character.

The value mmm must be less than or equal to nnn, and neither must exceed 100. The value ccc must be outside the limits set by mmm and nnn. The valid control characters are:

b Skip one line before printing

0 Skip two lines before printing

- Skip three lines before printing

+ Suppress space before printing

1 Start new page

The carriage control character can be ignored by specifying zero. Zero is the standard default.

OUTPUT OPTIONS

DECK or NODECK

Specifying DECK causes the compiler to put the object code, in card image form, into a data set called LOAD.name(0), where 'name' is the object module name. This option should be considered in conjunction with the LOAD or NOLOAD option.

LOAD or NOLOAD

Specifying the LOAD option invokes the same action as the DECK option except that in addition, the load data set is presented to the Object Dataset Converter (ODC) to produce an executable module.

Note that the load data set is created as a data generation set of depth one. Each time the program is recompiled the last data set is erased and replaced with the one currently being generated.

LISTING OPTIONS

LINECNT

The LINECNT option specifies the number of lines to be included in each page of a printed listing, including heading lines and blank lines. Its format is:

LINECNT=xxx

OPLIST or NOOPLIST

The OPLIST option requests a list showing the status of all the compiler options at the start of compilation.

SOURCE2 or NOSOURCE2

The SOURCE2 option requests a listing of the PL/I source statements input to the preprocessor.

SOURCE or NOSOURCE

The SOURCE option requests a listing of the PL/I source statements processed by the compiler. The source statements listed are either those of the original source program of the output from the preprocessor.

NEST or NONEST

The NEST option specifies that the source program listing should indicate for each statement, the block level and the level of nesting of a DO-group.

ATR or NOATR

The ATR option requests the inclusion in the listing of a table of source program identifiers and their attributes. Attributes with a precision of fixed binary (15.0) or less are flagged '*********'. An Aggregate Length Table, giving the length in bytes of all major structures and non-structured arrays in the source program, will also be produced when the ATR option is specified.

XREF or NOXREF

The XREF option requests the inclusion in the listing of a cross-reference table that lists all the identifiers in the source program with the numbers of the source statements in which they appear. If you specify both ATR and XREF, the two tables are combined. An Aggregate Length Table will also be produced when the XREF option is specified.

EXTREF or NOEXTREF

The EXTREF option requests the inclusion of a listing of the external symbol dictionary (ESD).

LIST or NOLIST

The LIST option requests a listing of the machine instructions generated by the compiler (in a form similar to System/360 assembler language instructions).

FLAGW, FLAGE, or FLAGS

A high level of diagnostic capability is available in the compiler. Messages are listed in order of their occurrence on the user console and in order of their severity in the output listing. There are four classes of diagnostic messages, which are graded in order of severity:

A warning is a message that calls attention to a possible error, although the statement to which it refers is syntactically valid. In addition to alerting you, it may assist in writing more efficient programs in the future.

An error message describes an attempt to correct an erronious statement; you are informed of the correction. Errors do not normally terminate processing of the text.

A severe error message indicates an error that cannot be corrected by the compiler. The incorrect section of the program is deleted, but compilation is continued. Where reasonable, the ERROR condition will be raised at object time, if execution of an incorrect source statement is attempted. If a severe error occurs during compile-time processing, compilation will be terminated after the SOURCE listing has been produced.

A terminal error message describes an error that, when discovered, forces the termination of the compilation.

You can select the severity at and above which diagnostic messages appear on the output listing.

FLAGW List all diagnostic messages

FLAGE List all diagnostic messages except 'warning' messages

FLAGS List only 'severe' errors and 'termination' errors

For control of conversational diagnostics (see DIAG/NODIAG option in the PLCOPT parameter of the PLI command, Part II, Section 5 of this manual), the diagnostic messages are filtered through both the FLAG setting of the compiler options and the LIMEN setting of the PLCOPT.

DUMMY OPTIONS

SIZE

This option is used in the OS/360 compiler to specify the amount of main storage available. The option has no effect in TSS/360 PL/I and is included to give compatability with OS/360 PL/I. Since there is a standard default built into the compiler, you need never take account of this option.

M91 or NOM91

This option is used to indicate in OS/360 if the machine is a System/360 Model 91. It is included for the same reasons as given above for SIZE.

EXTDIC or NOEXTDIC

This option is ignored, since the TSS/360 PL/I compiler always takes the EXTDIC option.

APPENDIX H: PL/I DIAGNOSTIC MESSAGES

This appendix contains all of the diagnostic messages issued by the PL/I compiler and library under TSS/360. Messages issued by PLC and ODC, recognizable by their prefixes CFBAA and CFBAB, are explained in the System Messages publication.

Three classes of diagnostic messages are issued: source program, compile time, and object time.

Source-program diagnostic messages are identified by an eight-character code of the form IEMnnnnI, where "IEM" indicates that the message emanates from the PL/I compiler, "nnnn" is a four-digit decimal number in the range 0000-3999 that uniquely identifies each message, and "I" is a system-standard action code indicating an informative message for the programmer. All source-program diagnostic messages produced are written in a group following the source program listing and any other listings specified in the PLI command. If conversational diagnostics are elected in the PLI command, these messages will also appear at the user's terminal.

Compile-time diagnostic messages are identified by an eight-character code of the form IEMnnnnI, where "IEM" indicates that the message emanates from the PL/I compiler, "nnnn" is a four-digit decimal number in the range 4000-4999 that uniquely identifies each message, and "I" is a system-standard action code indicating an informative message for the programmer. All compile-time diagnostic messages are listed in the group following the SOURCE2 input listing and preceding the source program listing. If conversational diagnostics are elected in the PLI command, these messages will also appear at the user's terminal.

Object-time diagnostic messages are identified by a seven-character code of the form IHEnnnI, where "IHE" indicates that the message emanates from a PL/I library routine, "nnn" is a three-digit decimal number that uniquely identifies each message, and "I" indicates the informative nature of the message. The object-time diagnostic messages are printed on SYSOUT or on the output data set specified as SYSPRINT as the result of an exceptional or error condition occurring during the execution of a PL/I program.

In the following sections, messages are followed where necessary by an explanation, a description of the action taken by the system, and the response required from the user. "Explanation" and "System Action" are given only when this information is not contained in the text of the message. When no "User Response" is stated, the user should assume that he must correct the error in his source program and recompile unless the action taken by the system makes it unnecessary for him to do so. However, even when system action successfully corrects an error, the user should remember that if he subsequently recompiles the same program he will get the same diagnostic message again unless he has corrected the source error.

Severity of Source-Program and Compile-Time Diagnostic Messages

There are four types of source-program diagnostic messages: warning (W), error (E), severe error (S), and termination error (T).

A warning message calls attention to a possible error, although the statement to which it refers is syntactically valid. In addition to alerting the programmer, it may assist him in writing more efficient programs in the future.

An error message informs the programmer of a correction to an erroneous statement. Such errors do not normally terminate processing of the text.

A severe error message indicates an error that cannot be corrected by the compiler. The incorrect section of the program is deleted, but compilation continues. Where reasonable, the error condition will be raised at object time if execution of an incorrect source statement is attempted. If a severe error occurs during compile-time processing, compilation is terminated after the source listing is produced.

A termination error message describes an error which, when discovered, forces the termination of the compilation.

The choice of severity level at and above which diagnostic messages appear on the output is an option specified by the programmer in the PL/I command statement. W is the system-supplied default value.

In the source-program and compile-time diagnostic messages which follow, the abbreviations W, E, S, and T are used to indicate the severity of the message and appear immediately before the number of the message. The abbreviations do not appear in this way in the compiler output listings; instead, the messages are printed in separate groups according to severity.

E IEM0002I INVALID PREFIX OPERATOR IN STATEMENT NUMBER xxx. REPLACED BY PLUS.

E IEM0003I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

E IEM0004I OPERATOR .NOT. IN STATEMENT NUMBER xxx USED AS AN INFIX OPERATOR. IT HAS BEEN REPLACED BY .NE.

E IEM0005I RIGHT PARENTHESIS INSERTED AFTER SINGLE PARENTHESIZED EXPRESSION IN STATEMENT NUMBER xxx

E IEM0006I RIGHT PARENTHESIS INSERTED AT END OF SUBSCRIPT, ARGUMENT OR CHECK LIST IN STATEMENT NUMBER xxx

S IEM0007I IDENTIFIER MISSING IN STATEMENT NUMBER xxx . A DUMMY IDENTIFIER HAS BEEN INSERTED.

E IEM0008I RIGHT PARENTHESIS INSERTED AT END OF CALL ARGUMENT LIST OR OTHER EXPRESSION LIST IN STATEMENT NUMBER xxx

W IEM0009I A LETTER IMMEDIATELY FOLLOWS CONSTANT IN STATEMENT NUMBER xxx. AN INTERVENING BLANK IS ASSUMED.

E IEM0010I IMPLEMENTATION RESTRICTION. IDENTIFIER yyyy IN OR NEAR STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN SHORTENED.

Explanation: Implementation restriction. Identifiers may not exceed 31 characters in length.

System Action: Identifier has been shortened by concatenating first 16 characters with last 15.

W IEM0011I CONSTANT IMMEDIATELY FOLLOWS IDENTIFIER IN STATEMENT NUMBER xxx. AN INTERVENING BLANK IS ASSUMED.

E IEM0012I EXPONENT MISSING IN FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . ZERO HAS BEEN INSERTED.

E IEM0013I INTEGER yyyy TOO LONG IN STATEMENT NUMBER xxx . IT HAS BEEN TRUNCATED ON THE RIGHT.

E IEM0014I EXPONENT TOO LONG IN FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . IT HAS BEEN TRUNCATED.

E IEM0015I SOLITARY DECIMAL POINT FOUND IN OPERAND POSITION IN STATEMENT NUMBER xxx . A FIXED-POINT ZERO HAS BEEN INSERTED.

E IEM0016I FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED ON THE RIGHT.

E IEM0017I ZERO INSERTED IN FLOATING-POINT CONSTANT BEGINNING .E IN STATEMENT NUMBER xxx

E IEM0018I ZERO INSERTED IN PENCE FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0019I POUNDS FIELD IN STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

E IEM0020I ZERO INSERTED IN POUNDS FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0021I DECIMAL POINT IN EXPONENT FIELD OF CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . FIELD TRUNCATED AT DECIMAL POINT.

E IEM0022I DECIMAL PENCE TRUNCATED IN STERLING CONSTANT BEGINNING yyyy STATEMENT NUMBER xxx

E IEM0023I LETTER L MISSING FROM STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

System Action: None

E IEM0024I SHILLINGS FIELD TRUNCATED IN STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0025I ZERO INSERTED IN SHILLINGS FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0026I ILLEGAL CHARACTER IN APPARENT BIT STRING yyyy IN STATEMENT NUMBER xxx . STRING TREATED AS A CHARACTER STRING.

E IEM0027I FIXED-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED ON THE RIGHT.

S IEM0028I LABEL REFERENCED ON END STATEMENT NUMBER xxx CANNOT BE FOUND. END TREATED AS HAVING NO OPERAND.

S IEM0029I INVALID CHARACTER IN BINARY CONSTANT IN STATEMENT NUMBER xxx . CONSTANT TREATED AS DECIMAL CONSTANT.

S IEM0030I POINTER QUALIFIER FOLLOWS EITHER A SUBSCRIPT OF ANOTHER POINTER QUALIFIER IN STATEMENT NUMBER xxx.

System Action: As stated in a further message referring to the same statement.

S IEM0031I OPERAND MISSING IN OR FOLLOWING STATEMENT NUMBER xxx . DUMMY OPERAND INSERTED.

Explanation: Something invalid has been found in an expression, or where an expression was expected but not found. In order that further diagnosis can be made, the compiler has inserted a dummy operand. This may cause further error messages to appear for this statement.

T IEM0032I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG. COMPILATION TERMINATED AT THIS POINT.

User Response: Subdivide statement and recompile.

E IEM0033I AN INVALID PICTURE CHARACTER IMMEDIATELY FOLLOWS TEXT yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THIS POINT.

W IEM0034I A LETTER IMMEDIATELY FOLLOWS A CONSTANT AT nnnn SEPARATE POSITION(S) IN STATEMENT NUMBER xxx. AN INTERVENING BLANK HAS BEEN ASSUMED IN EACH CASE.

User Response: Check that the system action will have the required effect.

E IEM0035I LETTER F IS NOT FOLLOWED BY LEFT PARENTHESIS IN PICTURE IN STATEMENT NUMBER xxx. ONE HAS BEEN INSERTED.

E IEM0037I ZERO INSERTED IN SCALING FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx

E IEM0038I RIGHT PARENTHESIS INSERTED AFTER SCALING OR REPLICATION FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx

E IEM0039I NO CHARACTER FOLLOWS REPLICATION FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE REPLICATION FACTOR.

E IEM0040I A REPLICATION FACTOR OF 1 HAS BEEN INSERTED IN PICTURE yyyy IN STATEMENT NUMBER xxx

E IEM0044I IN STATEMENT NUMBER xxx PRECISION NOT AN INTEGER.

Explanation: Precision should be an unsigned integer.

System Action: The action taken depends on whether the precision is found in a DECLARE statement or a PROCEDURE statement. A further message will be produced.

E IEM0045I ZERO INSERTED IN FIXED PRECISION SPECIFICATION IN STATEMENT NUMBER xxx

E IEM0046I RIGHT PARENTHESIS INSERTED AFTER PRECISION SPECIFICATION IN STATEMENT NUMBER xxx

E IEM0048I RIGHT PARENTHESIS INSERTED IN FILE NAME LIST IN STATEMENT NUMBER xxx

E IEM0049I THE COMMENT FOLLOWING THE LOGICAL END OF PROGRAM HAS NOT BEEN TERMINATED.

Explanation: A /* was found following the logical end of the program and was interpreted as the start of a comment, but end-of-file was reached before the comment was terminated.

System Action: All text following the /* is read as a comment.

User Response: Check if this is a delimiter in the wrong column of the record.

S IEM0050I INVALID STATEMENT LABEL CONSTANT IN LABEL ATTRIBUTE IN STATEMENT NUMBER xxx. THE STATEMENT LABEL CONSTANT LIST HAS BEEN DELETED.

W IEM0051I MISSING RIGHT PARENTHESIS INSERTED FOLLOWING STATEMENT

LABEL CONSTANT IN LABEL ATTRIBUTE IN STATEMENT NUMBER xxx

S IEM0052I INVALID ATTRIBUTE IN RETURNS ATTRIBUTE LIST IN STATEMENT NUMBER xxx. THE INVALID ATTRIBUTE HAS BEEN DELETED FROM THE LIST.

W IEM0053I SURPLUS COMMA HAS BEEN FOUND IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx. THIS COMMA HAS BEEN DELETED.

S IEM0054I ILLEGAL FORM OF CALL STATEMENT. STATEMENT NUMBER xxx DELETED.

W IEM0055I LABEL OR LABELS ON DECLARE STATEMENT NUMBER xxx HAVE BEEN IGNORED.

E IEM0056I NULL PICTURE FORMAT ITEM IN STATEMENT NUMBER xxx. THE CHARACTER 9 HAS BEEN INSERTED IN THE PICTURE.

Explanation: The null picture may be the result of the compiler truncating an invalid picture.

E IEM0057I INVALID CHARACTER FOLLOWING ITERATION FACTOR IN PICTURE BEGINNING yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE ITERATION FACTOR.

E IEM0058I ITERATION FACTOR IN PICTURE BEGINNING yyyy NOT AN UNSIGNED INTEGER IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE ITERATION FACTOR.

E IEM0059I MISSING RIGHT PARENTHESIS INSERTED IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx.

E IEM0060I POSITION MISSING IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx. POSITION OF 1 INSERTED.

E IEM0061I MISSING LEFT PARENTHESIS INSERTED IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx.

W IEM0062I THE ATTRIBUTE 'PACKED' IN DECLARATION STATEMENT NUMBER xxx IS NOW OBSOLETE, AND HAS BEEN IGNORED.

Explanation: PACKED has been removed from the language; the complementary attribute to ALIGNED is now UNALIGNED.

System Action: Since PACKED applied only to arrays and major structures, the new alignment defaults will be compatible with those of earlier versions of the compiler, except for bit string arrays that are not members of structures.

User Response: Correct source, and recompile if necessary.

E IEM0063I MISSING LEFT PARENTHESIS INSERTED IN RETURNS STATEMENT NUMBER xxx.

S IEM0064I ILLEGAL STATEMENT FOLLOWS THE THEN IN STATEMENT NUMBER xxx. SEMICOLON HAS BEEN INSERTED AFTER THE THEN.

S IEM0066I TEXT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error is detailed in another message referring to the same statement.

E IEM0067I EQUAL SYMBOL HAS BEEN INSERTED IN DO STATEMENT NUMBER xxx

T IEM0069I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS.

System Action: Compilation is terminated.

User Response: Rewrite program with fewer blocks, or divide into more than one separate compilation.

T IEM0070I BEGIN STATEMENT NUMBER xxx IS NESTED BEYOND THE PERMITTED LEVEL. COMPILATION TERMINATED.

User Response: Reduce level of nesting of blocks to 50 or less.

T IEM0071I TOO MANY PROCEDURE, BEGIN, ITERATIVE DO, ON STATEMENTS IN THIS PROGRAM. COMPILATION TERMINATED.

Explanation: There is an implementation restriction on the number of blocks in a compilation.

User Response: Subdivide program into two or more compilations.

S IEM0072I DO STATEMENT NUMBER xxx REPLACED BY BEGIN STATEMENT.

E IEM0074I THEN INSERTED IN IF STATEMENT NUMBER xxx

S IEM0075I NO STATEMENT FOLLOWS THEN IN IF STATEMENT NUMBER xxx

S IEM0076I NO STATEMENT FOLLOWS ELSE IN OR FOLLOWING STATEMENT NUMBER xxx

S IEM0077I ELSE DELETED IN OR FOLLOWING STATEMENT NUMBER xxx

E IEM0078I IMPLEMENTATION RESTRICTION. TOO MANY CHARACTERS IN INITIAL LABEL ON STATEMENT NUMBER xxx. LABEL IGNORED.

Explanation: There is an implementation restriction on the number of characters in the subscript of a subscripted identifier. The maximum permissible number is 225.

E IEM0080I EQUAL SYMBOL HAS BEEN INSERTED IN ASSIGNMENT STATEMENT NUMBER xxx

S IEM0081I LABELS OR PREFIX OPTIONS BEFORE ELSE TRANSFERRED TO STATEMENT NUMBER xxx

Explanation: Labels or prefix options illegal before ELSE and therefore transferred to following statement.

S IEM0082I OPERAND MISSING IN CHECK LIST IN OR FOLLOWING STATEMENT NUMBER xxx. DUMMY INSERTED.

S IEM0083I ON-CONDITION INVALID OR MISSING IN STATEMENT NUMBER xxx. ON ERROR HAS BEEN ASSUMED.

System Action: ON ERROR inserted in place of invalid condition.

E IEM0084I THE I/O ON-CONDITION IN STATEMENT NUMBER xxx HAS NO FILENAME FOLLOWING IT. SYSIN IS ASSUMED.

E IEM0085I COLON MISSING AFTER PREFIX OPTION IN OR FOLLOWING STATEMENT NUMBER xxx. ONE HAS BEEN ASSUMED.

S IEM0086I COMPILER LIMIT EXCEEDED IN PHASE CI. NO MORE ERROR MESSAGES CAN BE ISSUED FOR STATEMENT NUMBER xxx.

User Response: Correct the statement as far as possible and recompile.

S IEM0089I A RECORD IN STATEMENT NUMBER xxx OR THE FOLLOWING STATEMENT EXCEEDS 100 CHARACTERS. THE EXCESSIVE CHARACTERS HAVE BEEN REJECTED.

T IEM0090I THERE ARE NO COMPLETE STATEMENTS IN THIS PROGRAM. COMPILATION TERMINATED.

W IEM0094I RECORD IN OR FOLLOWING STATEMENT NUMBER xxx IS SHORTER THAN THE SPECIFIED SOURCE START. THE OUTPUT RECORD HAS BEEN MARKED WITH AN ASTERISK AND IGNORED.

E IEM0095I LABEL ON STATEMENT NUMBER xxx HAS NO COLON. ONE IS ASSUMED.

Explanation: The compiler has encountered an identifier which appears to be a statement label, but without a colon.

System Action: A colon is inserted.

E IEM0096I SEMI-COLON NOT FOUND WHEN EXPECTED IN STATEMENT NUMBER xxx . ONE HAS BEEN INSERTED.

E IEM0097I INVALID CHARACTER HAS BEEN REPLACED BY BLANK IN OR FOLLOWING STATEMENT NUMBER xxx. THE CONTAINING OUTPUT RECORD IS MARKED BY AN ASTERISK.

S IEM0099I LOGICAL END OF PROGRAM OCCURS AT STATEMENT NUMBER xxx. THIS STATEMENT HAS BEEN IGNORED SO THAT SUBSEQUENT STATEMENTS MAY BE PROCESSED.

Explanation: Although the compiler has detected the end of the program, there is more text following it. The programmer appears to have made an error in matching END statements with PROCEDURE, BEGIN, DO, or ON statements.

System Action: The END statement is ignored.

S IEM0100I END OF FILE FOUND IN OR AFTER STATEMENT NUMBER xxx, BEFORE THE LOGICAL END OF PROGRAM.

System Action: If statement is incomplete, it is deleted. Whether or not the statement is incomplete, the required number

of END statements are added to the program so that compilation can continue.

User Response: Correct the source code. Possible causes of this error include:

1. Unmatched quotation marks.
2. Insufficient END statements.
3. Omission of final semicolon.

S IEM0101I PARAMETER MISSING IN STATEMENT NUMBER xxx . A DUMMY HAS BEEN INSERTED.

S IEM0102I LABEL MISSING FROM PROCEDURE STATEMENT NUMBER xxx. A DUMMY LABEL HAS BEEN INSERTED.

S IEM0103I LABEL MISSING FROM ENTRY STATEMENT NUMBER xxx

S IEM0104I ILLEGAL STATEMENT FOLLOWS ELSE IN STATEMENT NUMBER xxx

System Action: Null statement inserted.

S IEM0105I ILLEGAL STATEMENT FOLLOWS ON IN STATEMENT NUMBER xxx

System Action: Null statement inserted.

T IEM0106I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS.

System Action: Compilation is terminated.

User Response: Rewrite program with fewer blocks, or divide into more than one separate compilation.

T IEM0107I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG. THIS STATEMENT MAY CONTAIN UNMATCHED QUOTE MARKS.

User Response: Subdivide statement and recompile.

S IEM0108I ENTRY STATEMENT NUMBER xxx IN AN ITERATIVE DO GROUP HAS BEEN DELETED.

S IEM0109I TEXT BEGINNING yyyy IN OR FOLLOWING STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: Error detailed in another message referring to same statement.

S IEM0110I TEXT BEGINNING yyyy IN OR FOLLOWING STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: Error detailed in another message referring to same statement.

S IEM0111I FIRST STATEMENT NOT A PROCEDURE STATEMENT. A DUMMY PROCEDURE STATEMENT HAS BEEN INSERTED.

S IEM0112I ENTRY STATEMENT NUMBER xxx IN BEGIN BLOCK HAS BEEN DELETED.

S IEM0113I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Explanation: Parenthesized list in ON statement is either not closed or contains an error and has been truncated.

E IEM0114I RIGHT PARENTHESIS INSERTED IN PREFIX OPTION IN OR FOLLOWING STATEMENT NUMBER xxx

E IEM0115I LEFT PARENTHESIS INSERTED AFTER WHILE IN STATEMENT NUMBER xxx

E IEM0116I PREFIX OPTION FOLLOWS LABEL IN STATEMENT NUMBER xxx. PREFIX OPTION IS IGNORED.

W IEM0117I DATA ATTRIBUTE LIST IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx IS NOT PRECEDED BY RETURNS ATTRIBUTE AND IS NOT PARENTHESIZED. RETURNS AND PARENTHESES HAVE BEEN ASSUMED.

S IEM0118I OFFSET ATTRIBUTE NOT FOLLOWED BY PARENTHESIZED BASED VARIABLE IN STATEMENT NUMBER xxx. THE ATTRIBUTE IS IGNORED.

E IEM0119I THE RETURNS ATTRIBUTE IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx IS NOT FOLLOWED BY A PARENTHESIZED DATA ATTRIBUTE LIST. RETURNS HAS BEEN IGNORED.

E IEM0120I DATA ATTRIBUTE LIST FOLLOWING RETURNS IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx IS NOT PARENTHESIZED. PARENTHESES HAVE BEEN ASSUMED.

E IEM0121I THE ATTRIBUTE USES OR SETS IN STATEMENT NUMBER xxx IS OBSOLETE AND HAS BEEN IGNORED TOGETHER WITH ITS PARENTHESIZED ITEM LIST.

E IEM0122I THE ATTRIBUTE NORMAL OR ABNORMAL IN STATEMENT NUMBER xxx IS OBSOLETE AND HAS BEEN IGNORED.

E IEM0123I THE DATA ATTRIBUTE LIST IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx HAS NO CLOSING PARENTHESIS. ONE HAS BEEN ASSUMED.

S IEM0124I INVALID ATTRIBUTE IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx. ATTRIBUTE TEXT DELETED.

E IEM0125I INVALID USE OF LABEL yyy ON ON-UNIT BEGINNING AT STATEMENT xxx. LABEL HAS BEEN DELETED.

Explanation: An on-unit cannot be referenced by a label.

System Action: The label is ignored.

S IEM0126I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx HAS TOO MANY ERRORS TO BE INTERPRETED. THE STATEMENT HAS BEEN DELETED.

S IEM0127I INVALID TEXT IN PREFIX OPTIONS LIST. THE TEXT BEGINNING yyy TO THE END OF THE OPTIONS LIST HAS BEEN IGNORED.

S IEM0128I LENGTH OF BIT OR CHARACTER STRING MISSING IN STATEMENT NUMBER xxx. LENGTH 1 INSERTED.

S IEM0129I INVALID WAIT STATEMENT NUMBER xxx DELETED.

E IEM0130I OPERAND MISSING. COMMA DELETED IN WAIT STATEMENT NUMBER xxx

S IEM0131I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

S IEM0132I DUMMY OPERAND INSERTED IN STATEMENT NUMBER xxx

S IEM0134I IMPLEMENTATION RESTRICTION. TOO MANY LEVELS OF REPLICATION IN INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx. THE ATTRIBUTE HAS BEEN DELETED.

Explanation: There is an implementation restriction on levels of nesting.

User Response: Rewrite INITIAL attribute with lower level of replication.

E IEM0136I 'IN' CLAUSE IN STATEMENT NUMBER xxx HAS NO ASSOCIATED 'SET' CLAUSE.

Explanation: An IN clause must be accompanied by a SET clause in the same statement.

System Action: The IN clause is ignored.

E IEM0138I SOLITARY I FOUND WHERE A CONSTANT IS EXPECTED IN INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx. FIXED DECIMAL IMAGINARY 1I HAS BEEN ASSUMED.

Explanation: The programmer has initialized an element using the variable I where the constant 1I was expected.

System Action: 1I is assumed.

S IEM0139I TEXT IMMEDIATELY FOLLOWING yyyy IN INITIAL ATTRIBUTE IS ILLEGAL. INITIAL ATTRIBUTE DELETED IN STATEMENT NUMBER xxx

Explanation: A language feature has been used that is not supported. Although the message states that the error follows the quoted text, the quoted text may itself be invalid, and the compiler may have attempted to correct the source error. In this case, there will usually be another diagnostic message associated with the statement.

W IEM0140I NO IDENTIFIER FOUND IN DECLARE STATEMENT NUMBER xxx STATEMENT REPLACED BY NULL STATEMENT.

Explanation: Either no identifiers appear in the DECLARE statement or, as a result of previous compiler action, all identifiers have been deleted from the statement.

System Action: Null statement assumed.

S IEM0144I RETURNS ATTRIBUTE IS NOT FOLLOWED BY A DATA DESCRIPTION IN STATEMENT NUMBER xxx. THE RETURNS ATTRIBUTE HAS BEEN DELETED.

S IEM0145I DUMMY IDENTIFIER INSERTED IN GENERIC ATTRIBUTE LIST IN STATEMENT NUMBER xxx

S IEM0147I THE USE OF REFER IN STATEMENT NUMBER xxx IS EITHER INVALID OR IS NOT IMPLEMENTED IN THIS RELEASE.

Explanation: The implementation of the REFER option is restricted.

System Action: Ignores the REFER clause. A further message identifying the invalid text will usually accompany this message.

E IEM0148I LEFT PARENTHESIS MISSING IN STATEMENT NUMBER xxx

System Action: See further messages relating to this statement.

E IEM0149I COMMA HAS BEEN DELETED FROM LIST IN STATEMENT NUMBER xxx

E IEM0150I STATEMENT NUMBER xxx IS AN INVALID FREE STATEMENT. THE STATEMENT HAS BEEN DELETED.

Explanation: The format of the statement is invalid.

S IEM0151I SEMI-COLON INSERTED IN STATEMENT NUMBER xxx

S IEM0152I TEXT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error may be detailed in another message referring to the same statement.

E IEM0153I THE ATTRIBUTED BASED HAS BEEN ASSUMED IN STATEMENT NUMBER xxx WHERE CONTROLLED WAS SPECIFIED.

Explanation: The PL/I feature CONTROLLED (pointer) has been changed to BASED (pointer).

S IEM0154I IMPLEMENTATION RESTRICTION IN STATEMENT NUMBER xxx. BASED MUST BE FOLLOWED BY AN IDENTIFIER IN PARENTHESIS.

System Action: Text is deleted. See further error message for this statement.

User Response: Correct source statement.

E IEM0158I ZERO STRUCTURE LEVEL NUMBER DELETED IN DECLARE STATEMENT NUMBER xxx

Explanation: Zero level number not allowed.

E IEM0159I SIGN DELETED PRECEDING STRUCTURE LEVEL NUMBER IN DECLARE STATEMENT NUMBER xxx

Explanation: The level number must be an unsigned integer.

S IEM0163I FORMAT LIST MISSING, (A) INSERTED IN STATEMENT NUMBER xxx

S IEM0166I OPERAND MISSING IN GO TO STATEMENT NUMBER xxx. DUMMY IS INSERTED.

E IEM0172I LEFT PARENTHESIS INSERTED IN DELAY STATEMENT NUMBER xxx

Explanation: The expression in a DELAY statement should be contained in parentheses.

E IEM0180I EQUAL SYMBOL HAS BEEN INSERTED IN DO SPECIFICATIONS IN STATEMENT NUMBER xxx

E IEM0181I SEMICOLON INSERTED IN STATEMENT NUMBER xxx

Explanation: An error has been discovered. A semicolon is therefore inserted and the rest of the statement is skipped.

S IEM0182I TEXT BEGINNING yyyy SKIPPED IN OR FOLLOWING STATEMENT NUMBER xxx

Explanation: The source error is detailed in another message referring to the same statement.

S IEM0185I OPTION IN GET/PUT STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

S IEM0187I DATA LIST MISSING IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0191I DUMMY OPERAND INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

E IEM0193I RIGHT PARENTHESIS INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

E IEM0194I MISSING RIGHT PARENTHESIS INSERTED IN FORMAT LIST IN STATEMENT NUMBER xxx

S IEM0195I INVALID FORMAT LIST DELETED IN STATEMENT NUMBER xxx. (A) INSERTED.

S IEM0198I COMPLEX FORMAT ITEM yyyy IN STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

S IEM0202I DEFERRED FEATURE. STATEMENT NUMBER xxx NOT IMPLEMENTED IN THIS VERSION.

Explanation: The statement referred to is of a type not supported.

System Action: Compilation continues.

User Response: Rewrite source program avoiding use of unsupported feature.

E IEM0207I COMMA REPLACED BY EQUAL SYMBOL IN ASSIGNMENT STATEMENT NUMBER xxx

E IEM0208I LEFT PARENTHESIS INSERTED IN CHECK LIST IN STATEMENT NUMBER xxx

T IEM0209I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO COMPLEX.

Explanation: The level of nesting exceeds the implementation restriction.

System Action: Compilation is terminated.

User Response: Divide statement into two or more statements.

E IEM0211I LEFT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

E IEM0212I MULTIPLE TASK OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first.

E IEM0213I MULTIPLE EVENT OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first.

E IEM0214I MULTIPLE PRIORITY OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first.

E IEM0216I INVALID EVENT OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM0217I INVALID PRIORITY OPTION IGNORED IN STATEMENT NUMBER xxx

W IEM0218I REPETITION FACTOR MISSING AFTER ITERATION FACTOR IN STATEMENT NUMBER xxx. REPETITION FACTOR OF 1 INSERTED.

S IEM0219I KEYWORD 'CONDITION' NOT SPECIFIED IN SIGNAL STATEMENT NUMBER xxx

S IEM0220I IDENTIFIER MISSING OR INCORRECT AFTER OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0221I NUMBER OF LINES NOT GIVEN AFTER LINE OPTION IN STATEMENT NUMBER xxx. (1) INSERTED.

S IEM0222I DEFERRED FEATURE. THE IDENT OPTION ON OPEN/CLOSE STATEMENT NUMBER xxx IS NOT IMPLEMENTED BY THIS VERSION.

Explanation: A language feature has been used that is not supported.

System Action: Option ignored.

S IEM0223I EXPRESSION MISSING AFTER IDENT/TITLE/LINESIZE/PAGESIZE OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

Explanation: No left parenthesis found following keyword.

S IEM0224I INVALID OPTION DELETED IN I/O STATEMENT NUMBER xxx

S IEM0225I OPTION AFTER OPEN/CLOSE IN STATEMENT NUMBER xxx IS INVALID OR MISSING.

S IEM0226I EXPRESSION MISSING AFTER FORMAT ITEM IN STATEMENT NUMBER xxx. ITEM DELETED.

W IEM0227I NO FILE/STRING OPTION SPECIFIED IN ONE OR MORE GET/PUT STATEMENTS. SYSIN/SYSPRINT HAS BEEN ASSUMED IN EACH CASE.

Explanation: One or more GET or PUT statements have appeared in the program with no specified FILE option or STRING option.

System Action: The compiler has assumed the appropriate default file (SYSIN for GET, SYSPRINT, or SYSOUT, for PUT).

S IEM0228I EXPRESSION MISSING AFTER OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0229I FORMAT ITEM IN STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

S IEM0230I INVALID DATA LIST IN STATEMENT NUMBER xxx. STATEMENT DELETED.

E IEM0231I MISSING COMMA INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

Explanation: Comma missing between elements of a data list.

E IEM0232I KEYWORD DO MISSING IN DATA LIST IN STATEMENT NUMBER xxx. DO IS INSERTED.

S IEM0233I RETURN STATEMENT NUMBER xxx IS WITHIN AN ON-UNIT. IT IS REPLACED BY A NULL STATEMENT.

S IEM0235I ARGUMENT OMITTED FOLLOWING yyyy OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0236I THE OPTION yyyy IN STATEMENT NUMBER xxx IS UNSUPPORTED OR INVALID.

S IEM0237I INSUFFICIENT OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE STATEMENT HAS BEEN REPLACED BY A NULL STATEMENT.

S IEM0238I THE LOCATE-VARIABLE IN LOCATE STATEMENT NUMBER xxx IS OMITTED OR SUBSCRIPTED. THE STATEMENT HAS BEEN DELETED.

Explanation: The omission of the locate variable renders the statement meaningless. Subscripted locate variables are invalid.

System Action: Replaces invalid statement with a null statement.

T IEM0240I COMPILER ERROR IN PHASE CV. SCAN CANNOT IDENTIFY DICTIONARY ENTRY.

Explanation: The main scan of fifth pass of read-in has found something in the dictionary which it cannot recognize.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM0241I MULTIPLE USE OF A PREFIX OPTION HAS OCCURRED IN STATEMENT NUMBER xxx. THE LAST NAMED OPTION IS USED.

E IEM0242I PREFIX OPTION INVALID OR MISSING IN OR FOLLOWING STATEMENT NUMBER xxx. INVALID OPTION DELETED.

T IEM0243I COMPILER ERROR. PHASE CS HAS FOUND AN UNMATCHED END.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM0244I CHECK PREFIX OPTION IN STATEMENT NUMBER xxx IS NOT FOLLOWED BY A PARENTHESIZED LIST. THE OPTION HAS BEEN IGNORED.

E IEM0245I A CHECK PREFIX OPTION IS GIVEN FOR STATEMENT NUMBER xxx WHICH IS NOT A PROCEDURE OR BEGIN. THE OPTION HAS BEEN IGNORED.

S IEM0247I ALL SUBSCRIPTED LABELS PREFIXING PROCEDURE OR ENTRY STATEMENT NUMBER xxx HAVE BEEN IGNORED.

Explanation: Subscripted labels may not be used as prefixes on PROCEDURE or ENTRY statements.

T IEM0254I COMPILER UNABLE TO RECOVER FROM I/O ERROR - PLEASE RETRY JOB.

System Action: Compilation is terminated.

User Response: Reattempt compilation.

T IEM0255I THERE ARE NO COMPLETE STATEMENTS IN THIS PROGRAM.

Explanation: Compiler cannot reconcile END statements with stack entries. Usually caused by a program containing only comments.

System Action: Compilation is terminated.

User Response: Check source for completed statements. If these are present, save relevant data and inform the

system manager or administrator of the error.

S IEM0257I DATA DIRECTED I/O LIST IN STATEMENT NUMBER xxx CONTAINS BASED ITEM zzzz

System Action: Statement will be deleted by later phases.

S IEM0258I NUMBER OF SUBSCRIPTS SPECIFIED FOR zzzz IN STATEMENT NUMBER xxx CONFLICTS WITH DIMENSIONALITY. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases.

W IEM0510I THE TASK OPTION HAS BEEN ASSUMED TO APPLY TO THE EXTERNAL PROCEDURE STATEMENT NUMBER xxx

Explanation: TASK, EVENT, or PRIORITY options have been detected in a CALL statement, but the TASK option has not been specified in the external procedure.

System Action: The TASK option is correctly applied.

W IEM0511I OPTIONS MAIN AND/OR TASK ARE NOT ALLOWED ON THE INTERNAL PROCEDURE STATEMENT NUMBER xxx

System Action: The invalid options are ignored.

S IEM0512I IDENTIFIER yyyy IN STATEMENT NUMBER xxx IN INITIAL ATTRIBUTE LIST IS NOT A KNOWN LABEL CONSTANT AND HAS BEEN IGNORED.

System Action: Identifier changed to * in the list.

S IEM0513I REPEATED LABEL IN SAME BLOCK ON STATEMENT NUMBER xxx. LABEL DELETED.

Explanation: A label may not be used more than once in the same block.

S IEM0514I PARAMETER yyyy IN STATEMENT NUMBER xxx IS SAME AS LABEL. PARAMETER REPLACED BY DUMMY.

S IEM0515I IMPLEMENTATION RESTRICTION. CHARACTER STRING LENGTH IN STATEMENT NUMBER xxx REDUCED TO 32,767.

S IEM0516I ILLEGAL OPTIONS LIST ON STATEMENT NUMBER xxx. LIST IGNORED.

System Action: Compiler scans for next right bracket. If this is not the bracket closing the illegal options list, a compiler error will probably follow.

S IEM0517I CONFLICTING ATTRIBUTE DELETED IN STATEMENT NUMBER xxx

S IEM0518I IMPLEMENTATION RESTRICTION. PRECISION TOO LARGE IN STATEMENT NUMBER xxx. DEFAULT PRECISION GIVEN.

Explanation: If later a valid precision is given, this will be accepted in place of the default.

System Action: Attribute ignored. Attribute test mask restored so that later attribute will not be found to conflict with deleted one.

S IEM0519I ILLEGAL ATTRIBUTE ON STATEMENT NUMBER xxx IGNORED.

Explanation: Only data attributes allowed on procedure or entry statements. (No dimensions allowed.)

T IEM0520I COMPILER ERROR CODE nnnn

Explanation: A compiler error has occurred.

System Action: Compilation is terminated immediately.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

S IEM0521I INVALID STRING LENGTH IN STATEMENT NUMBER xxx. LENGTH OF 1 ASSUMED.

Explanation: Either no length has been given or string length * has been used in source code.

System Action: Assumes length of 1 and skips to next attribute.

S IEM0522I IMPLEMENTATION RESTRICTION. NUMBER OF PARAMETERS IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx TRUNCATED TO 64.

S IEM0523I PARAMETER zzzz IN STATEMENT NUMBER xxx APPEARS TWICE. SECOND ONE REPLACED BY DUMMY.

S IEM0524I IDENTIFIER yyyy IN LABEL LIST IN STATEMENT NUMBER xxx IS NOT A LABEL OR IS NOT KNOWN.

System Action: Ignores identifier.

T IEM0525I IMPLEMENTATION RESTRICTION. TOO MANY PAIRS OF FACTORED ATTRIBUTE BRACKETS FOR THIS SIZE OPTION.

Explanation: Factor bracket table has overflowed.

System Action: Compilation is terminated.

User Response: Recompile using a SIZE sufficient to provide a larger block size or reduce factoring by expanding declarations.

W IEM0526I OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE STATEMENT NUMBER xxx

S IEM0527I IMPLEMENTATION RESTRICTION. ARRAY BOUND IN STATEMENT NUMBER xxx IS TOO LARGE AND HAS BEEN REPLACED BY THE MAXIMUM PERMITTED VALUE (32767 OR -32768).

T IEM0528I COMPILER ERROR CODE nnnn IN STATEMENT NUMBER xxx

Explanation: Compiler error found in processing a DECLARE statement.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

S IEM0529I IMPLEMENTATION RESTRICTION. STRUCTURE LEVEL NUMBER IN STATEMENT NUMBER xxx REDUCED TO 255.

S IEM0530I IMPLEMENTATION RESTRICTION. TOO MANY LABELS IN LABEL LIST IN STATEMENT NUMBER xxx. THE LABEL zzzz AND ANY FOLLOWING IT HAVE BEEN IGNORED.

Explanation: There is an implementation restriction limiting the number of label constants following the LABEL attribute to 125.

S IEM0532I ILLEGAL ASTERISK AS SUBSCRIPT IN DEFINING LIST IN STATEMENT NUMBER xxx. LIST TRUNCATED.

System Action: Compilation continues with truncated iSUB list, possibly causing cascade errors.

S IEM0533I IMPLEMENTATION RESTRICTION. I-SUB VALUE IN STATEMENT NUMBER xxx TOO LARGE. REDUCED TO 32.

Explanation: There is an implementation restriction limiting the number of dimensions to 32.

S IEM0534I IMPLEMENTATION RESTRICTION. STRING LENGTH IN STATEMENT NUMBER xxx REDUCED TO 32,767.

S IEM0536I IDENTIFIER yyyy IN STATEMENT NUMBER xxx IS NOT A LABEL CONSTANT OR IS NOT KNOWN. IT IS IGNORED.

Explanation: Identifiers following the LABEL attribute must be LABEL constants and must be known.

S IEM0537I IMPLEMENTATION RESTRICTION. POSITION CONSTANT IN STATEMENT NUMBER xxx REDUCED TO 32,767.

E IEM0538I IMPLEMENTATION RESTRICTION. PRECISION SPECIFICATION IN STATEMENT NUMBER xxx TOO LARGE. DEFAULT PRECISION GIVEN.

E IEM0539I ILLEGAL NEGATIVE PRECISION IN STATEMENT NUMBER xxx . DEFAULT PRECISION GIVEN.

S IEM0540I * BOUNDS ARE MIXED WITH NON-*BOUNDS IN DECLARE STATEMENT NUMBER xxx . ALL THE BOUNDS ARE MADE *.

E IEM0541I LOWER BOUND GREATER THAN UPPER BOUND IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx . THE BOUNDS ARE INTERCHANGED.

S IEM0542I IMPLEMENTATION RESTRICTION. NUMBER OF DIMENSIONS DECLARED TRUNCATED TO 32 IN STATEMENT NUMBER xxx

T IEM0543I COMPILER ERROR. ILLEGAL STATEMENT FOUND IN THE DECLARE CHAIN.

Explanation: Compiler error found in scanning chain of DECLARE statements.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

T IEM0544I COMPILER ERROR. INITIAL CODE BYTE OF DECLARE STATEMENT IS NEITHER STATEMENT NUMBER NOR STATEMENT LABEL.

Explanation: Compiler error found in first byte of DECLARE statements.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

T IEM0545I COMPILER ERROR. ILLEGAL INITIAL CHARACTER TO DECLARED ITEM IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scanning start of declared item.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

T IEM0546I COMPILER ERROR. ILLEGAL CHARACTER FOUND AFTER LEVEL NUMBER IN DECLARE STATEMENT NUMBER xxx

Explanation: Compiler error found after structure level number in DECLARE statement.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

W IEM0547I THE IDENTIFIER yyyy DECLARED IN STATEMENT NUMBER xxx IS A NON-MAJOR STRUCTURE MEMBER AND HAS THE SAME NAME AS A FORMAL PARAMETER OR INTERNAL ENTRY POINT. ALL REFERENCES TO THE STRUCTURE MEMBER SHOULD BE QUALIFIED.

System Action: Same BCD treated as different identifiers.

T IEM0548I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN DECLARATION LIST.

Explanation: Compiler error found in list of declarations in DECLARE statement.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

E IEM0549I THE DECLARED LEVEL OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx SHOULD BE ONE. THIS HAS BEEN FORCED.

System Action: Illegal level number treated as 1.

S IEM0550I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH A TRUE LEVEL NUMBER GREATER THAN THE IMPLEMENTATION RESTRICTION OF 63. THE DECLARATION OF THE IDENTIFIER IS IGNORED.

E IEM0551I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH ZERO PRECISION. THE DEFAULT VALUE HAS BEEN ASSUMED.

T IEM0552I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN FACTORED ATTRIBUTE LIST IN DECLARE STATEMENT NUMBER xxx

Explanation: Compiler error found in factored attribute list.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM0553I THE IDENTIFIER yyyy HAS HAD A CONFLICTING ATTRIBUTE IGNORED IN DECLARE STATEMENT NUMBER xxx

Explanation: The two attributes may conflict as a

result of a feature that is not supported.

T IEM0554I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN PARAMETER LIST FOLLOWING 'GENERIC' ATTRIBUTE.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM0555I STORAGE CLASS ATTRIBUTES MAY NOT BE SPECIFIED FOR STRUCTURE MEMBER yyyy . ATTRIBUTE IGNORED.

User Response: Delete illegal storage class attribute for the structure member.

T IEM0556I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN PARAMETER LIST FOLLOWING AN 'ENTRY' ATTRIBUTE IN DECLARE STATEMENT NUMBER xxx

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM0557I THE MULTIPLE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

S IEM0558I IMPLEMENTATION RESTRICTION. NUMBER OF PARAMETER DESCRIPTIONS DECLARED FOR PROCEDURE OR ENTRY NAME yyyy IN STATEMENT NUMBER xxx TRUNCATED TO 64.

E IEM0559I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH CONFLICTING FACTORED LEVEL NUMBERS. THE ONE AT DEEPEST FACTORING LEVEL HAS BEEN CHOSEN.

E IEM0560I IN STATEMENT NUMBER xxx A CONFLICTING ATTRIBUTE HAS BEEN IGNORED IN THE DECLARATION OF THE RETURNED VALUE OF ENTRY POINT yyyy

S IEM0561I IN STATEMENT NUMBER xxx THE IDENTIFIER yyyy IS A MULTIPLE DECLARATION OF AN INTERNAL ENTRY LABEL. THIS DECLARATION IS IGNORED.

S IEM0562I THE IDENTIFIER yyyy IS DECLARED IN STATEMENT NUMBER xxx AS AN INTERNAL ENTRY POINT. THE NUMBER OF PARAMETERS DECLARED IS DIFFERENT FROM THE NUMBER GIVEN AT THE ENTRY POINT.

S IEM0563I THE IDENTIFIER yyyy DECLARED 'BUILTIN' IN STATEMENT NUMBER xxx IS NOT A BUILT-IN FUNCTION. DECLARATION IGNORED.

E IME0564I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH PRECISION GREATER THAN THE IMPLEMENTATION LIMITS. THE MAXIMUM VALUE HAS BEEN TAKEN.

E IEM0565I THE IDENTIFIER yyyy IS DECLARED IN STATMENT NUMBER xxx AS A MEMBER OF A GENERIC LIST, BUT ITS ATTRIBUTES DO NOT MAKE IT AN ENTRY POINT. THE DECLARATION OF THE IDENTIFIER HAS BEEN IGNORED.

E IEM0566I ONE OF THE PARAMETERS DECLARED FOR ENTRY POINT yyyy IN STATEMENT NUMBER xxx SHOULD BE AT LEVEL ONE. THIS HAS BEEN FORCED.

W IEM0567I IF FUNCTION yyyy IN STATEMENT NUMBER xxx IS INVOKED, THE DEFAULT ATTRIBUTES ASSUMED FOR THE VALUE RETURNED WILL CONFLICT WITH THE ATTRIBUTES IN THE PROCEDURE OR ENTRY STATEMENT FOR THAT VALUE.

Explanation: The data type to which a result will be converted at a RETURN (expression) will not be the same as that expected at an invocation of the entry label as a function.

System Action: None

User Response: Write an entry-point declaration (using the ENTRY or RETURNS attribute) in the containing block, giving the same data attributes as those on the PROCEDURE or ENTRY statement.

S IEM0568I THE IDENTIFIER zzzz IS CALLED BUT IS EITHER A BUILTIN FUNCTION OR IS NOT AN ENTRY POINT.

System Action: The erroneous statement is deleted.

T IEM0569I COMPILER ERROR NUMBER nnnn IN MODULE EP.

Explanation: Compiler error found in scan of chain of CALL statements.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

W IEM0570I THE ENTRY POINT yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx TO HAVE A RETURNED VALUE DIFFERENT FROM THAT GIVEN ON THE PROCEDURE OR ENTRY STATEMENT.

System Action: None

User Response: Change the declaration, or the PROCEDURE or ENTRY statement.

S IEM0571I IMPLEMENTATION RESTRICTION. IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS MORE THAN 32 DIMENSIONS. DIMENSION ATTRIBUTE IGNORED.

S IEM0572I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH THE ATTRIBUTE "NORMAL" OR "ABNORMAL". THE APPLICATION OF THIS ATTRIBUTE IS AN UNSUPPORTED FEATURE OF THE FOURTH VERSION, AND IT HAS BEEN IGNORED.

Explanation: An unsupported language feature has been used.

S IEM0573I THE SELECTION OF GENERIC FAMILY MEMBERS WHOSE PARAMETERS HAVE A STRUCTURE DESCRIPTION IS DEFERRED. ENTRY NAME yyyy, DECLARED IN STATEMENT NUMBER xxx, IS SUCH A MEMBER AND HAS BEEN DELETED.

Explanation: The usage referred to is not supported.

T IEM0574I THE MULTIPLE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

W IEM0575I THE REPEATED ATTRIBUTE IN THE DECLARATION OF FILE yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

E IEM0576I THE EXTERNAL FILE yyy DECLARED IN STATEMENT NUMBER xxx HAS THE SAME NAME AS THE EXTERNAL PROCEDURE. DECLARATION IGNORED.

E IEM0577I INCORRECT SPECIFICATION OF THE ARRAY BOUNDS, STRING LENGTH OR AREA SIZE OF THE NON-CONTROLLED PARAMETER yyyy IN DECLARE STATEMENT NUMBER xxx; THOSE OF THE CORRESPONDING ARGUMENT WILL BE ASSUMED.

Explanation: In the declaration of a non-controlled array, string or area parameter, the bounds, length or size must be given by a constant or by an asterisk.

System Action: Takes the bounds, length or size of the array, string or area passed as an argument.

E IEM0578I UNDIMENSIONED VARIABLE yyy DECLARED IN STATEMENT NUMBER xxx HAS INITIAL ATTRIBUTE WITH CONFLICTING SPECIFICATION FOR A DIMENSIONED VARIABLE. INITIAL ATTRIBUTE IGNORED.

W IEM0579I THE PARAMETER OF THE MAIN PROCEDURE SHOULD BE A FIXED LENGTH CHARACTER STRING OR HAVE THE ATTRIBUTES CHARACTER(100) VARYING.

S IEM0580I INVALID USE OF FILE yyy IN STATEMENT NUMBER xxx. IT HAS BEEN REPLACED BY A DUMMY REFERENCE.

S IEM0589I COMPILER ERROR. ITEM zzzz IN LIKE CHAIN IS NOT A STRUCTURE. ITEM IS IGNORED.

User Response: Save relevant data and inform system manager or administrator of the error.

S IEM0590I STRUCTURE ELEMENT zzzz WHICH HAS LIKE ATTRIBUTE ATTACHED TO IT, IS FOLLOWED BY AN ELEMENT WITH A NUMERICALLY GREATER STRUCTURE LEVEL NUMBER. LIKE ATTRIBUTE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0591I STRUCTURE ELEMENT zzzz IS LIKENED TO AN ITEM WHICH IS NOT A STRUCTURE VARIABLE. LIKE ATTRIBUTE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0592I STRUCTURE ELEMENT zzzz IS LIKENED TO A STRUCTURE WHICH

CONTAINS ELEMENTS WHICH HAVE ALSO BEEN DECLARED WITH THE LIKE ATTRIBUTE. LIKE ATTRIBUTE ON ORIGINAL STRUCTURE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0593I STRUCTURE NAME TO WHICH zzzz IS LIKENED IS NOT KNOWN. LIKE ATTRIBUTE IGNORED.

System Action: Self explanatory: may result in cascade errors.

E IEM0594I AMBIGUOUS QUALIFIED NAME yyyy USED AS A BASE IDENTIFIER. MOST RECENT DECLARATION USED.

E IEM0595I QUALIFIED NAME yyyy USED AS A BASE IDENTIFIER CONTAINS MORE THAN ONE IDENTIFIER AT THE SAME STRUCTURE LEVEL.

System Action: The erroneous statement is deleted.

S IEM0596I MAJOR STRUCTURE yyyy HAS BEEN LIKENED TO AN ITEM WHICH IS NOT A VALID STRUCTURE. DECLARATION OF STRUCTURE IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0597I IDENTIFIER zzzz WHICH IS NOT A FORMAL PARAMETER OR OF STORAGE CLASS CONTROLLED HAS BEEN LIKENED TO A STRUCTURE CONTAINING * DIMENSIONS OR LENGTH. * DIMENSIONS OR LENGTH HAVE BEEN IGNORED IN THE CONSTRUCTED STRUCTURE.

System Action: Self explanatory: may result in cascade errors from later phases.

S IEM0598I QUALIFIED NAME TO WHICH zzzz HAS BEEN LIKENED IS AN AMBIGUOUS REFERENCE. LIKE ATTRIBUTE HAS BEEN IGNORED.

S IEM0599I zzzz WHICH IS A PARAMETER OR A BASED VARIABLE, HAS BEEN DECLARED (USING THE LIKE ATTRIBUTE) AS A STRUCTURE WITH THE INITIAL ATTRIBUTE. THE INITIAL ATTRIBUTE IS INVALID AND HAS BEEN IGNORED.

User Response: Declare the parameter or based variable with the LIKE attribute specifying a structure without the INITIAL attribute.

S IEM0600I STATIC STRUCTURE zzzz HAS BEEN DECLARED BY MEANS OF THE LIKE ATTRIBUTE TO HAVE ADJUSTABLE EXTENTS. THE EXTENTS HAVE BEEN IGNORED.

Explanation: A STATIC variable cannot have adjustable extents.

System Action: All bounds on the offending variable are set to zero.

S IEM0601I OFFSET ATTRIBUTE ON PROCEDURE STATEMENT NUMBER xxx IS NOT BASED ON A BASED AREA. IT HAS BEEN CHANGED TO POINTER.

T IEM0602I IDENTIFIER IN BASED ATTRIBUTE ON zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT A NON-BASED POINTER.

System Action: Compilation is terminated.

T IEM0603I INVALID POINTER EXPRESSION IN BASED ATTRIBUTE ON zzzz IN STATEMENT NUMBER xxx

Explanation: The pointer associated with the based variable does not obey the implementation rules (e.g., it may be subscripted).

System Action: Compilation is terminated.

T IEM0604I LENGTH OR SIZE DECLARED FOR BASED STRING OR BASED AREA zzzz IN STATEMENT NUMBER xxx IS INVALID.

Explanation: The declaration violates the compiler implementation rules.

System Action: Compilation is terminated.

T IEM0605I BOUNDS DECLARED FOR BASED ARRAY zzzz IN STATEMENT NUMBER xxx ARE INVALID.

Explanation: The adjustable bounds declared are outside those permitted by this implementation.

System Action: Compilation is terminated.

S IEM0606I OFFSET VARIABLE zzzz HAS BEEN DECLARED IN STATEMENT NUMBER

xxx RELATIVE TO AN IDENTIFIER WHICH IS NOT A LEVEL 1 BASED AREA. IT HAS BEEN CHANGED TO A POINTER VARIABLE.

System Action: The offset is changed to a pointer to prevent the compiler from producing further error messages.

User Response: Consider assigning the ADDR of the required area to the pointer named in the declaration of a level 1 based area; this area can be validly named in the OFFSET attribute, and offset values for it will be correct for the other area.

W IEM0607I IF THE BASE OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx IS ALLOCATED WITH THE DECLARED BOUNDS THE DEFINING WILL BE IN ERROR.

Explanation: For correspondence defining not involving iSUB's, bounds of the defined array must be a subset of the bounds of the base. In this case bounds declared for the base do not satisfy this requirement. However, the base is of CONTROLLED storage class and if it is allocated with different bounds the defining may be legal.

System Action: Nothing further.

T IEM0608I ILLEGAL DEFINING IN STATEMENT NUMBER xxx. BASE IDENTIFIER zzzz IS A MEMBER OF A DIMENSIONED STRUCTURE.

Explanation: In the case of string class overlay defining where the base is an array, it is an error if it is a member of an array of structures.

System Action: Compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct error.

T IEM0609I DEFERRED FEATURE. DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx WITH A SUBSCRIPTED BASE.

Explanation: Overlay defining on a subscripted base is not supported.

System Action: Compilation is terminated.

User Response: Replace all references to the defined item by appropriate subscripted references to the base.

T IEM0610I DEFERRED FEATURE. DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ON A BASE OF CONTROLLED STORAGE CLASS.

Explanation: If the base is declared CONTROLLED, neither overlay defining nor correspondence defining is supported.

System Action: Compilation is terminated.

User Response: Replace all references to the defined item by appropriate references to the base.

T IEM0611I SCALAR zzzz DECLARED IN STATEMENT NUMBER xxx IS ILLEGALLY DEFINED WITH ISUBS.

Explanation: Only arrays may be correspondence defined using iSUB notation.

System Action: Compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct error.

E IEM0612I INITIAL ATTRIBUTE DECLARED FOR DEFINED ITEM zzzz IN STATEMENT NUMBER xxx WILL BE IGNORED.

Explanation: DEFINED items may not have the INITIAL attribute.

System Action: INITIAL attribute ignored.

S IEM0613I ELEMENT VARIABLE SPECIFIED IN REFER OPTION IN STATEMENT NUMBER xxx IS NOT AN INTEGER.

Explanation: Both I and N in (I REFER(N)) must be fixed binary integers; they must also be of the same precision.

System Action: Compilation continues. Any reference to the element variable may result in an execution error.

S IEM0614I ELEMENT VARIABLES SPECIFIED IN REFER OPTION IN STATEMENT NUMBER xxx DO NOT HAVE THE SAME PRECISION.

Explanation: Both I and N in (I REFER(N)) must be fixed binary integers of the same precision.

System Action: Compilation continues. Any reference to either element variable may result in an execution error.

T IEM0623I THE BASE SUBSCRIPT LIST USED WITH THE DEFINED VARIABLE zzzz IN STATEMENT NUMBER xxx ILLEGALLY REFERS TO OR IS DEPENDENT ON THE DEFINED VARIABLE.

Explanation: It is illegal for a base subscript list in the DEFINED attribute to refer directly, or via any further level of defining, to the defined item.

System Action: Compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct error.

T IEM0624I THE BASE IDENTIFIER FOR zzzz DECLARED IN STATEMENT NUMBER xxx IS DEFINED OR BASED.

Explanation: The base of DEFINED data may not itself be DEFINED.

System Action: Compilation is terminated.

User Response: Replace the specified base by an appropriate reference to its base.

T IEM0625I THE DEFINING BASE FOR zzzz DECLARED IN STATEMENT NUMBER xxx HAS THE WRONG NUMBER OF SUBSCRIPTS.

Explanation: If the base reference in a DEFINED attribute is subscripted, it must have the same number of subscript expressions as the dimensionality of the base array.

System Action: Compilation is terminated.

User Response: Correct the subscript list, or declaration of the base, whichever is appropriate.

T IEM0626I THE DEFINING BASE FOR zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT DATA.

Explanation: The only legal data types that may be used for defining bases are String, Arithmetic, Task, Event, and Label.

System Action: Compilation is terminated.

User Response: Check that the defining base is correctly written and declared.

T IEM0628I IMPLEMENTATION RESTRICTION. THE NESTING OF REFERENCES TO DATA DEFINED WITH A SUBSCRIPTED BASE IS TOO DEEP.

Explanation: The complexity of defining has resulted in a level of nesting which is too great for the compiler.

System Action: Compilation is terminated.

User Response: Reduce complexity of defining.

T IEM0629I ARRAY zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY HAS THE POS ATTRIBUTE WITH ISUB DEFINING.

Explanation: The POS attribute may not be specified for correspondence defining.

System Action: Compilation is terminated.

User Response: Delete POS attribute.

T IEM0630I THE DESCRIPTION OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx DOES NOT MATCH THAT OF THE DEFINING BASE.

Explanation: For correspondence defining, if either the base or the defined

item are arrays of structures, then both must be arrays of structures.

System Action: Compilation is terminated.

User Response: Correct the program. Note that POS (1) may be used to force overlay defining.

T IEM0631I IMPLEMENTATION RESTRICTION. THE CORRESPONDENCE DEFINING OF yyyy AN ARRAY OF STRUCTURES DECLARED IN STATEMENT NUMBER xxx

Explanation: Correspondence defining with arrays of structures is not supported by the compiler.

System Action: Compilation is terminated.

User Response: Declare the base arrays of the defined structure as correspondence defined on the matching base arrays of the base structure. Note that for this to be valid, the base arrays of the defined structure must have unique names and be declared at level 1. This alternative method precludes structure operations with the defined item, but achieves the desired mapping.

T IEM0632I THE BOUNDS OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx ARE NOT A SUBSET OF THE BASE.

Explanation: For correspondence defining not involving iSUB's, the bounds of the defined array must be a subset of the corresponding bounds of the base array.

System Action: Compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct program. Note that POS (1) may be used to force overlay defining.

S IEM0633I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx IS NOT AT LEVEL 1. THE STATEMENT HAS BEEN IGNORED.

Explanation: An identifier specified in an ALLOCATE statement must refer to a major structure or data not contained in a structure. A major structure identifier may optionally be followed by a full structure description.

System Action: The ALLOCATE statement is deleted.

User Response: Replace erroneous identifier by that of the containing major structure.

S IEM0634I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx HAS NOT BEEN DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CONTROLLED data may be allocated. Data may only obtain the attribute CONTROLLED from an explicit declaration.

System Action: The ALLOCATE statement is deleted.

User Response: Construct a DECLARE statement for the identifier.

S IEM0636I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx WAS NOT DECLARED CONTROLLED. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CONTROLLED data may be specified in ALLOCATE statements.

System Action: The ALLOCATE statement is deleted.

User Response: Declare the identifier CONTROLLED.

E IEM0637I A CONFLICTING ATTRIBUTE WAS GIVEN FOR zzzz IN STATEMENT NUMBER xxx. THE ATTRIBUTE HAS BEEN IGNORED.

Explanation: Attributes given for an identifier in an ALLOCATE statement may not conflict with those given explicitly or assumed by default from the declaration.

System Action: Ignores the attribute from the ALLOCATE.

S IEM0638I THE STRUCTURE DESCRIPTION GIVEN ON STATEMENT NUMBER xxx DIFFERS FROM THAT DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: If a description of a major structure is given on an ALLOCATE statement, the description must match that declared.

System Action: The ALLOCATE statement is deleted.

S IEM0640I AN INVALID ATTRIBUTE WAS GIVEN IN STATEMENT NUMBER xxx. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only String (CHAR or BIT), INITIAL, and Dimension attributes are permitted in ALLOCATE statements.

System Action: The ALLOCATE statement is deleted.

E IEM0641I CONFLICTING ATTRIBUTES HAVE BEEN GIVEN FOR zzzz IN STATEMENT NUMBER xxx. THE FIRST LEGAL ONE HAS BEEN USED.

Explanation: At most, one attribute in the following classes may be given for an identifier in an ALLOCATE statement: Dimension, String (CHAR or BIT), INITIAL.

System Action: All attributes after the first in a particular class are ignored.

S IEM0642I DIMENSIONALITY GIVEN IN STATEMENT NUMBER xxx DIFFERS FROM THAT DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: If a dimension attribute is given for an identifier in an ALLOCATE statement, the identifier must have been declared with the same dimensionality.

System Action: The ALLOCATE statement is deleted.

User Response: Correct declaration or ALLOCATE statement, whichever is appropriate.

W IEM0643I THE LEVEL NUMBER DECLARED FOR zzzz IS NOT THE SAME AS THAT GIVEN IN STATEMENT NUMBER xxx. THE FORMER HAS BEEN USED.

Explanation: If a structure description is given in an ALLOCATE statement, it must match the declaration. The indicated level number discrepancy may be an error.

System Action: Nothing further.

User Response: Check that ALLOCATE statement is as intended.

S IEM0644I STATEMENT NUMBER xxx CONTAINS AN ILLEGAL PARENTHESIZED LIST. THE STATEMENT HAS BEEN IGNORED.

Explanation: Factored attributes are not allowed on ALLOCATE statements.

System Action: Statement ignored.

User Response: Remove parentheses and any factored attributes.

S IEM0645I ATTRIBUTE GIVEN WITH BASED VARIABLE zzzz IN ALLOCATE STATEMENT NUMBER xxx HAS BEEN IGNORED.

Explanations: Based variable may not be specified with attributes.

User Response: Correct ALLOCATE statement.

S IEM0646I IDENTIFIER yyyy PRECEDING POINTER QUALIFIER IN STATEMENT NUMBER xxx IS NOT A NON-BASED POINTER VARIABLE

System Action: The identifier is replaced by a dummy dictionary reference; a later phase will delete the statement.

User Response: Correct the invalid statement

S IEM0647I POINTER-QUALIFIED IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A BASED VARIABLE

System Action: Identifier is replaced by a dummy dictionary reference; a later phase will delete the statement.

User Response: Correct the invalid statement.

T IEM0653I COMPILER ERROR. ILLEGAL ENTRY IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scan of statement.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM0655I QUALIFIED NAME BEGINNING yyyy USED IN STATEMENT NUMBER xxx BUT NO PREVIOUS STRUCTURE DECLARATION GIVEN. DUMMY REFERENCE INSERTED.

System Action: Reference to the illegal variable or the whole statement will be deleted by later phases.

User Response: Correct program by inserting DECLARE statement.

E IEM0656I MOST RECENT DECLARATION USED OF AMBIGUOUS QUALIFIED NAME OR STRUCTURE MEMBER BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0657I QUALIFIED NAME BEGINNING yyyy IN STATEMENT NUMBER xxx CONTAINS MORE THAN ONE IDENTIFIER AT THE SAME STRUCTURE LEVEL.

System Action: The statement is deleted.

S IEM0658I QUALIFIED NAME BEGINNING yyyy IN STATEMENT NUMBER xxx IS AN AMBIGUOUS REFERENCE. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phase.

S IEM0659I UNSUPPORTED FEATURE. STRING PSEUDO-VARIABLE APPEARS IN REPLY, KEYTO OR STRING OPTION IN STATEMENT NUMBER xxx. STATEMENT WILL BE DELETED BY A LATER PHASE.

S IEM0660I PSEUDO-VARIABLE yyyy IN STATEMENT NUMBER xxx IS INVALID BECAUSE IT IS NESTED IN ANOTHER PSEUDO-VARIABLE.

Explanation: Language restriction. Pseudo-variables cannot be nested.

System Action: Statement will be deleted by a later phase.

S IEM0661I USE OF 'PRIORITY' PSEUDO-VARIABLE IN STATEMENT NUMBER xxx IMPLIES MULTITASKING, WHICH IS NOT SUPPORTED IN TSS.

System Action: The statement is deleted.

Programmer Response: Correct statement and recompile.

E IEM0662I IN STATEMENT NUMBER xxx CHECK LIST CONTAINS DEFINED ITEM yyy. IT HAS BEEN REPLACED BY ITS BASE IDENTIFIER.

Explanation: Defined items are not permitted in CHECK lists.

System Action: The base item is assumed to replace the reference to the defined item in the CHECK list. References in the text to the defined item will not be checked.

S IEM0673I INVALID USE OF FUNCTION NAME ON LEFT HAND SIDE OF EQUAL SYMBOL, OR IN REPLY KEYTO OR STRING OPTION, IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases.

S IEM0674I STATEMENT NUMBER xxx CONTAINS ILLEGAL USE OF FUNCTION yyyy

System Action: Reference to function or whole statement will be deleted by later phases.

S IEM0675I IN STATEMENT NUMBER xxx IDENTIFIER yyyy AFTER GO TO IS NOT A LABEL OR LABEL VARIABLE KNOWN IN THE BLOCK CONTAINING THE GO TO.

System Action: Statement will be deleted by later phases.

S IEM0676I DEFERRED FEATURE. IDENTIFIER yyyy NOT ALLOWED AS A BUILD-IN FUNCTION OR PSEUDO-VARIABLE. DUMMY REFERENCE INSERTED IN STATEMENT NUMBER xxx

Explanation: A language feature has been used that is not supported.

System Action: Statement will be deleted by later phases.

User Response: Correct statement by removing reference to function in error.

S IEM0677I ILLEGAL PARENTHESIZED LIST IN STATEMENT NUMBER xxx FOLLOWS AN IDENTIFIER WHICH IS NOT A FUNCTION OR ARRAY. LIST DELETED.

S IEM0682I IN STATEMENT NUMBER xxx GO TO TRANSFERS CONTROL ILLEGALLY TO A FORMAT STATEMENT.

S IEM0683I zzzz WAS FOUND WHERE A FILE NAME IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY DICTIONARY REFERENCE REPLACES ILLEGAL ITEM.

System Action: Statement will be deleted by later phases.

W IEM0684I USE OF LABEL VARIABLE zzzz MAY RESULT IN AN ILLEGAL BRANCH IN STATEMENT NUMBER xxx

Explanation: It is possible that the label variable may contain a value which would cause control to branch illegally into a block.

System Action: None

User Response: Check validity of possible branches.

S IEM0685I zzzz IS NOT A STATEMENT LABEL ON AN EXECUTABLE STATEMENT. DUMMY REFERENCE INSERTED AFTER GO TO IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases.

S IEM0686I zzz APPEARS IN A FREE OR ALLOCATE STATEMENT BUT HAS NOT BEEN DECLARED CONTROLLED. DUMMY REFERENCE INSERTED IN STATEMENT NUMBER xxx

System Action: A dummy reference is inserted. The statement will be deleted by a later phase.

E IEM0687I IN STATEMENT NUMBER xxx GO TO zzzz TRANSFERS CONTROL ILLEGALLY TO ANOTHER BLOCK OR GROUP. EXECUTION ERRORS MAY OCCUR.

S IEM0688I COMPILER ERROR. TOO FEW LEFT PARENTHESES IN STATEMENT NUMBER xxx

Explanation: This is a compiler error.

System Action: None taken: cascade errors may result.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM0689I zzzz WAS FOUND WHERE A TASK IDENTIFIER IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases.

S IEM0690I zzzz WAS FOUND WHERE EVENT VARIABLE IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases.

S IEM0691I INVALID ITEM zzzz IN DATA LIST, OR 'FROM' OR 'INTO' OPTION, IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases.

S IEM0692I DATA DIRECTED I/O LIST OR FROM OR INTO OPTION IN STATEMENT NUMBER xxx CONTAINS A PARAMETER, DEFINED OR BASED ITEM zzzz.

System Action: Statement will be deleted by later phases.

S IEM0693I ILLEGAL USE OF FUNCTION zzzz IN INPUT LIST IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases.

S IEM0694I IN THE FORMAT LIST IN STATEMENT NUMBER xxx A REMOTE FORMAT ITEM REFERENCES zzzz, WHICH IS NOT A STATEMENT LABEL IN THE CURRENT BLOCK. DUMMY REFERENCE INSERTED.

System Action: Format item deleted by later phase.

S IEM0695I LABEL ARRAY zzzz IS NOT FOLLOWED BY A SUBSCRIPT LIST AFTER GO TO IN STATEMENT NUMBER xxx. DUMMY REFERENCE REPLACES REFERENCE TO ARRAY.

System Action: Statement will be deleted by later phases.

W IEM0696I IN STATEMENT NUMBER xxx IT IS AN ERROR IF THE PARAMETER zzzz IN A REMOTE FORMAT ITEM REFERS TO A FORMAT STATEMENT WHICH IS NOT INTERNAL TO THE SAME BLOCK AS THE REMOTE FORMAT ITEM.

Explanation: Remote formats become executable code, but not internal procedures.

Therefore, they must appear in the same block in which they are used.

System Action: Object-time error message is compiled.

User Response: Correct program and recompile.

S IEM0697I STATEMENT LABEL zzzz ATTACHED TO STATEMENT NUMBER xxx IS USED AS A REMOTE FORMAT ITEM IN THAT STATEMENT. A DUMMY REPLACES THE REMOTE FORMAT ITEM.

System Action: Statement will be deleted by a later phase.

S IEM0698I THE BASED VARIABLE zzzz IN LOCATE STATEMENT NUMBER xxx IS NOT AT LEVEL 1. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by a later phase.

S IEM0699I STRUCTURE ARGUMENT zzzz OF FROM OR INTO OPTION IN STATEMENT NUMBER xxx IS NOT A MAJOR STRUCTURE. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by a later phase.

S IEM0700I ILLEGAL USE OF FUNCTION, LABEL OR VARYING STRING zzzz AS ARGUMENT OF FROM OR INTO OPTION IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by a later phase.

S IEM0701I ARGUMENT zzzz OF SET OPTION IS NOT A POINTER VARIABLE. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by a later phase.

W IEM0702I LABEL, TASK OR EVENT VARIABLE zzzz USED IN FROM OR INTO OPTION IN STATEMENT NUMBER xxx MAY LOSE ITS VALIDITY IN TRANSMISSION

S IEM0703I INVALID IDENTIFIER zzzz FREED IN STATEMENT NUMBER xxx

Explanation: The identifier in the FREE statement is not:

1. A BASED or a CONTROLLED variable, or
2. A major structure with the BASED or CONTROLLED attribute.

System Action: Invalid identifier replaced by dummy.

S IEM0704I STATEMENT NUMBER xxx CONTAINS INVALID USE OF FUNCTION zzzz

System Action: Statement will be deleted by a later phase.

W IEM0705I IF THE LABEL VARIABLE IN GO TO STATEMENT NUMBER xxx ASSUMES THE VALUE OF ITS VALUE-LIST MEMBER zzzz, THE STATEMENT WILL CONSTITUTE AN INVALID BRANCH INTO AN ITERATIVE DO GROUP.

System Action: None

User Response: Check that branch will be valid at execution time.

S IEM0706I VARIABLE zzzz IN LOCATE STATEMENT IS NOT A BASED VARIABLE. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by a later phase.

User Response: Correct the invalid statement.

S IEM0707I ARGUMENT zzzz OF IN OPTION IS NOT AN AREA VARIABLE. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by a later phase.

User Response: Correct the invalid statement.

S IEM0715I TEXT yyyy ASSOCIATED WITH THE INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx IS ILLEGAL AND HAS BEEN IGNORED.

Explanation: The INITIAL attribute has been used incorrectly.

System Action: The INITIAL attribute is deleted.

S IEM0718I INVALID CHECK LIST IN STATEMENT NUMBER xxx. STATEMENT HAS BEEN CHANGED TO 'ON ERROR'.

S IEM0719I ELEMENT OF LABEL ARRAY zzzz WHICH IS DECLARED WITH INITIAL ATTRIBUTE USED AS STATEMENT LABEL ON STATEMENT NUMBER xxx

System Action: Label is deleted.

S IEM0720I SUBSCRIPTED IDENTIFIER zzzz USED AS LABEL ON STATEMENT NUMBER xxx IS NOT A LABEL ARRAY

System Action: Label is deleted.

S IEM0721I ELEMENT OF STATIC LABEL ARRAY zzzz USED AS LABEL ON STATEMENT NUMBER xxx

System Action: Label is deleted.

S IEM0722I ELEMENT OF LABEL ARRAY zzzz USED AS LABEL ON STATEMENT NUMBER xxx IN BLOCK OTHER THAN THE ONE IN WHICH IT IS DECLARED.

System Action: An error statement is inserted in the text in place of the offending label.

T IEM0723I COMPILER ERROR IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scan of text.

System Action: Compilation is terminated

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM0724I FORMAL PARAMETER zzzz IN CHECK LIST. PARAMETER IS IGNORED.

Explanation: The identifier list of a CHECK prefix must not contain formal parameters.

S IEM0725I STATEMENT NUMBER xxx HAS BEEN DELETED DUE TO A SEVERE ERROR NOTED ELSEWHERE.

System Action: The whole statement is replaced by an error statement.

S IEM0726I IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A FILE NAME. THE STATEMENT IS DELETED.

Explanation: The identifier has been used previously in a different context and is therefore not recognized as a file name.

S IEM0727I IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A CONDITION NAME. THE STATEMENT IS DELETED.

Explanation: The identifier has been used previously in a different context and is therefore not recognized as a condition name.

T IEM0728I COMPILATION TERMINATED DUE TO A PREVIOUSLY DETECTED SEVERE ERROR IN STATEMENT NUMBER xxx

Explanation: A previous module has inserted a dummy dictionary reference into the second file. The compiler cannot recover.

S IEM0729I COMPILER ERROR IN SCALE FACTOR IN PICTURE BEGINNING yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; all references to picture deleted.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM0730I MORE THAN ONE SIGN CHARACTER PRESENT IN A SUBFIELD OF PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0731I PICTURE CHARACTER M APPEARS IN NON-STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0732I FIELD MISSING IN STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0733I ILLEGAL EDIT CHARACTERS AT START OF STERLING PICTURE yyyy.

THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0734I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN POUNDS FIELD OF STERLING PICTURE yyyy.THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0735I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN SHILLINGS FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0736I WRONG NUMBER OF DELIMITER CHARACTERS M IN STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0737I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN PENCE FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0739I STATIC PICTURE CHARACTER $ S + - NOT AT EXTREMITY OF SUBFIELD. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

E IEM0740I MULTIPLE USE OF E K OR V IN PICTURE. PICTURE TRUNCATED AT ILLEGAL CHARACTER. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture trunctated at point indicated indicated.

E IEM0741I CR OR DB INCORRECTLY POSITIONED IN SUBFIELD. PICTURE TRUNCATED AT THIS POINT. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated.

E IEM0742I CR OR DB GIVEN FOR A NON-REAL, NON-NUMERIC OR FLOATING FIELD. PICTURE TRUNCATED BEFORE CR OR DB IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated.

S IEM0745I ILLEGAL USE OF PICTURE CHARACTER Z OR * IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

S IEM0746I STERLING MARKER FOUND IN OTHER THAN FIRST POSITION IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

S IEM0747I STERLING PICTURE CHARACTERS FOUND IN NON-STERLING PICTURE. SCANNING OF PICTURE STOPPED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

E IEM0748I ILLEGAL USE OF SCALING FACTOR IN PICTURE. SCALING FACTOR ONWARDS DELETED. PICTURE IN

ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated.

S IEM0749I ILLEGAL USE OF SCALING FACTOR IN PICTURE. SCAN OF PICTURE TERMINATED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0750I ILLEGAL CHARACTER PRESENT IN CHARACTER STRING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

S IEM0751I NO MEANINGFUL CHARACTERS IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

S IEM0752I ILLEGAL USE OF, OR ILLEGAL CHARACTERS IN, STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

S IEM0754I ILLEGAL CHARACTER IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

S IEM0755I ILLEGAL USE OF DRIFTING EDITING SYMBOLS S $ + - IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

S IEM0756I IMPLEMENTATION RESTRICTION. PRECISION TOO LARGE OR PICTURE TOO LONG IN PICTURE BEGINNING yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored; picture ignored by later phases.

T IEM0758I COMPILER ERROR IN PHASE FT.

Explanation: Compiler error found in scan of dictionary.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM0759I IMPLEMENTATION RESTRICTION. STERLING CONSTANT EXCEEDS 4166666666666.13.3L. HIGH ORDER DIGITS LOST DURING CONVERSION TO DECIMAL.

System Action: High order digits lost somewhere in the following conversion process: shift pounds field left one digit, double by addition. Add shillings field. Add result, doubled by addition, to result shifted left one digit. Add pence field.

S IEM0760I IMPLEMENTATION RESTRICTION. EXPONENT FIELD TOO LARGE IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Reference to picture deleted.

S IEM0761I PICTURE CHARACTER E OR K APPEARS WITHOUT AN EXPONENT FIELD FOLLOWING IT. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Reference to picture deleted.

User Response: Correct picture

S IEM0762I PICTURE CHARACTER E OR K IS NOT PRECEDED BY A DIGIT POSITION CHARACTER. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Reference to picture deleted.

S IEM0763I INVALID CHARACTER nnnn IN EXPONENT FIELD OF PICTURE yyyy IN STATEMENT NUMBER xxx.

System Action: Scan of picture continued with item ignored; picture ignored by later phase.

W IEM0764I ONE OR MORE FIXED BINARY ITEMS OF PRECISION 15 OR LESS HAVE BEEN GIVEN HALFWORD STORAGE. THEY ARE FLAGGED '*********' IN THE XREF/ATR LIST.

S IEM0769I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx AS EXPANDED IS TOO LONG AND HAS BEEN DELETED.

User Response: Simplify by splitting into two or more statements and recompile.

T IEM0770I COMPILER ERROR IN INPUT TO PHASE GA IN STATEMENT NUMBER xxx

Explanation: The compiler has encountered meaningless input to phase GA.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM0771I IMPLEMENTATION RESTRICTION. NESTING OF FORMAT LISTS IN STATEMENT NUMBER xxx EXCEEDS 20. STATEMENT DELETED.

S IEM0778I AN INTERMEDIATE VARIABLE HAS BEEN CREATED IN READ INTO STATEMENT NUMBER xxx. THIS STATEMENT SPECIFIES FILE zzzz, WHICH HAS BEEN DECLARED WITH THE EVN(COBOL) ATTRIBUTE AND THE EVENT OPTION. THE EVENT OPTION HAS BEEN DELETED.

Explanation: The intermediate variable has been created because there is a difference between the PL/I mapping and the COBOL mapping of the READ INTO variable. The READ INTO statement has been expanded into:

```
READ INTO (Intermediate
  variable);
Variable = Intermediate
  variable;
```

The READ statement must have been completed before the assignment takes place. The EVENT option has been deleted to ensure that the READ statement is complete before processing continues.

System Action: Delete the EVENT option and continue.

User Response: Check the use of the EVENT option or of the COBOL file.

S IEM0779I AN INTERMEDIATE VARIABLE HAS BEEN CREATED IN WRITE/REWRITE FROM STATEMENT NUMBER xxx. THIS STATEMENT SPECIFIES FILE zzzz, WHICH HAS BEEN DECLARED WITH THE ENV(COBOL) ATTRIBUTE AND THE EVENT OPTION. THE EVENT OPTION HAS BEEN DELETED.

Explanation: The intermediate variable has been created because there is a difference between the PL/I mapping and the COBOL mapping of the WRITE/REWRITE FROM variable. The WRITE/REWRITE FROM statement has been expanded to:

```
Intermediate
variable = variable;
WRITE/REWRITE FROM
(Intermediate variable);
```

The WRITE/REWRITE statement must have been completed before the intermediate variable can be deleted. The EVENT option has been deleted to ensure that the WRITE/REWRITE statement is complete before processing continues.

System Action: Delete the EVENT option and continue.

User Response: Check the use of the EVENT option or of the COBOL file.

S IEM0780I THE USE OF COBOL FILE zzzz IN LOCATE STATEMENT NUMBER xxx MAY LEAD TO ERRORS WHEN THE RECORD IS PROCESSED.

Explanation: The COBOL structure-mapping is not necessarily the same as the PL/I structure-mapping.

System Action: The statement is deleted.

User Response: Either the LOCATE statement must be replaced by a WRITE FROM statement, or a non-COBOL file must be used.

S IEM0781I THE USE OF COBOL FILE zzzz IN READ SET STATEMENT NUMBER xxx MAY LEAD TO ERRORS WHEN THE RECORD IS PROCESSED.

Explanation: The COBOL structure-mapping is not necessarily the same as the PL/I structure-mapping.

System Action: The statement is deleted.

User Response: Either the READ SET statement must be replaced by a READ INTO statement, or a non-COBOL file must be used.

S IEM0782I THE ATTRIBUTES OF zzzz USED IN RECORD I/O STATEMENT NUMBER xxx ARE NOT PERMITTED WHEN A COBOL FILE IS USED.

Explanation: The attributes referred to do not exist in COBOL.

System Action: The statement is deleted.

S IEM0784I INVALID ARGUMENT LIST FOR ALLOCATION FUNCTION IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED OR DELETED.

Explanation: Only a single argument can be given in the ALLOCATION function, and it must be one of the following:

1. A major structure.
2. An unsubscripted array or scalar variable, not in a structure.

It must also be of nonbased CONTROLLED storage class.

System Action: If the argument list begins with a valid operand, that operand is used as the argument; otherwise, the argument list is deleted.

W IEM0786I NAME, NOT VALUE, OF FUNCTION zzzz PASSED AS ARGUMENT IN STATEMENT NUMBER xxx

E IEM0787I INCORRECT NUMBER OF ARGUMENTS FOR FUNCTION OR ROUTINE zzzz IN STATEMENTS yyyy.

Explanation: Number of arguments differs from the ENTRY declaration.

System Action: Arguments are matched as far as possible; zzzz is invoked using all the arguments.

User Response: Ignore this message if zzzz is a non-PL/I routine that can accept a variable number of arguments. Otherwise, correct the program.

W IEM0791I NUMBER OF ARGUMENTS FOR FUNCTION OR SUBROUTINE zzzz IN STATEMENTS yyyy IS INCONSISTENT WITH NUMBER USED ELSEWHERE.

Explanation: The number of arguments for zzzz has not been explicitly declared. 'ELSEWHERE' refers either to the PROCEDURE or ENTRY statement for zzzz, or to a previous invocation of the function.

System Action: zzzz is invoked using all the arguments.

User Response: Ignore this message if zzzz is a non-PL/I routine that can accept a variable number of arguments. Otherwise, correct the program.

E IEM0792I IN STATEMENT NUMBER xxx IT IS IMPOSSIBLE TO CONVERT FROM THE ATTRIBUTES OF ARGUMENT NUMBER nnnn TO THOSE OF THE CORRESPONDING PARAMETER IN ENTRY zzzz. THE PARAMETER DESCRIPTION IS IGNORED.

Explanation: Self explanatory. Examples of circumstances under which the error message is generated are label arguments to data item parameters, array arguments to scalar parameters.

System Action: The parameter description is ignored. If the parameter description is correct, this will give rise to totally incorrect execution.

User Response: Correct the parameter description or the argument so that at least conversion is possible.

W IEM0793I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A SCALAR AND THE CORRESPONDING PARAMETER IS A STRUCTURE.

System Action: A temporary structure of the same type as the parameter description is created and the argument is assigned to each base element, converting where necessary.

S IEM0794I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz CONTAINS A SUBSCRIPTED VARIABLE WITH THE WRONG NUMBER OF SUBSCRIPTS. THE STATEMENT HAS BEEN DELETED.

S IEM0795I DEFERRED FEATURE. IN STATEMENT NUMBER xxx ARGUMNET NUMBER nnnn OF ENTRY zzzz CONTAINS A CROSS-SECTION OF AN ARRAY OF STRUCTURES. STATEMENT DELETED.

Explanation: The usage referred to is not supported.

S IEM0796I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A GENERIC ENTRY NAME AND THERE IS NO CORRESPONDING ENTRY DESCRIPTION. STATEMENT DELETED.

S IEM0797I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A BUILT-IN FUNCTION WHICH MAY NOT BE PASSED AS AN ARGUMENT. STATEMENT DELETED.

S IEM0798I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS NOT PERMISSIBLE. THE STATEMENT HAS BEEN DELETED.

Explanation: The message is generated, for example, when scalar arguments are given for array build-in functions. For the ADDR function, the message could indicate that the function cannot return a valid result because the argument does not satisfy the requirements of contiguous storage. This could occur, for example, if the argument was a cross-section of an array.

E IEM0799I IN STATEMENT NUMBER xxx A DUMMY ARGUMENT HAS BEEN CREATED FOR ARGUMENT NUMBER nnnn OF ENTRY zzzz. THIS ARGUMENT APPEARS IN A SETS LIST.

System Action: The value assigned to the temporary argument during the execution of the procedure is lost on return from the procedure.

W IEM0800I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn IN ENTRY zzzz IS A SCALAR AND THE CORRESPONDING PARAMETER IS AN ARRAY.

System Action: A temporary array with the attributes of the entry description is created and the scalar is assigned to each element of the array, converting the type if necessary.

S IEM0801I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn IN ENTRY zzzz IS SCALAR CORRESPONDING TO AN ARRAY PARAMETER WITH * BOUNDS. THE STATEMENT HAS BEEN DELETED.

W IEM0802I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz DOES NOT MATCH THE PARAMETER. A DUMMY ARGUMENT HAS BEEN CREATED.

T IEM0803I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx CONTAINS TOO MANY NESTED FUNCTION REFERENCES. LIMIT EXCEEDED AT ARGUMENT NUMBER nnnn OF ENTRY zzzz

System Action: Compilation is terminated.

User Response: Reduce depth of function call nesting.

T IEM0804I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

System Action: Compilation is terminated.

User Response: Reduce statement size.

S IEM0805I DEFERRED FEATURE. IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS AN EVENT. STATEMENT DELETED.

Explanation: A language feature has been used that is not supported.

S IEM0806I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS NOT CONTROLLED BUT THE CORRESPONDING PARAMETER IS.

System Action: An execution error will occur on entry to the called procedure.

S IEM0807I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF BUILD-IN FUNCTION zzzz IS AN ENTRY NAME. THE STATEMENT HAS BEEN DELETED.

E IEM0808I IN STATEMENT NUMBER xxx ARGUMENT NUMBER yyy WILL CAUSE A DUMMY ARGUMENT TO BE PASSED TO ENTRY yyy IN ANOTHER TASK.

T IEM0816I COMPILER ERROR. INVALID END OF STATEMENT NUMBER xxx

Explanation: Compiler error in scan of input text.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM0817I COMPILER ERROR IN LABEL CHAIN FOR STATEMENT NUMBER xxx

Explanation: Compiler error in scanning labels of a statement.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error. Note that this error can be avoided by using only one label on the statement.

T IEM0818I COMPILER ERROR IN DICTIONARY ENTRY FOR STATEMENT NUMBER xxx

Explanation: Compiler error in scanning source text.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM0819I COMPILER ERROR IN CHECK/NOCHECK LIST ENTRY FOR STATEMENT NUMBER xxx

Explanation: Compiler error in CHECK or NOCHECK list dictionary entry.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM0820I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx TOO LONG.

Explanation: Statement length exceeds text-block size.

System Action: Compilation is terminated.

User Response: Subdivide statement and recompile.

E IEM0821I INVALID ITEM zzzz IGNORED IN CHECK LIST IN STATEMENT NUMBER xxx

Explanation: Valid items in CHECK lists are: statement labels, entry labels, and scalar, array, structure, or label variables. Subscripted variable names and data having the DEFINED attribute are not allowed.

W IEM0823I IMPLEMENTATION RESTRICTION. NO ROOM FOR zzzz IN CHECK TABLE STATEMENT NUMBER xxx

Explanation: The CHECK list table has overflowed.

System Action: The item mentioned is ignored.

User Response: Do not CHECK so many items.

T IEM0824I IMPLEMENTATION RESTRICTION. TOO MANY CHECKED ITEMS WITHIN STATEMENT NUMBER xxx

Explanation: A stack used to trace nested IF statements has overflowed.

System Action: Compilation is terminated.

User Response: Rephrase IF statements or do not CHECK so many items.

T IEM0825I COMPILER ERROR IN READ DATA STATEMENT NUMBER xxx

Explanation: Compiler error in processing GET or READ DATA statement.

System Action: Compilation is terminated.

User Response: Supply an explicit DATA list.

W IEM0826I IMPLEMENTATION RESTRICTION. CHECK WILL NOT BE RAISED FOR zzzz IN STATEMENT NUMBER xxx BECAUSE OF EVENT OPTION

Explanation: The compiler does not raise the CHECK condition for variables when they are changed in statements containing an EVENT option.

S IEM0832I IN THE EXPANSION OF BY NAME ASSIGNMENT STATEMENT NUMBER xxx A SET OF MATCHING ELEMENTS HAS BEEN FOUND BUT THEY ARE NOT ALL BASE ELEMENTS. THE STATEMENT HAS BEEN DELETED.

Explanation: For a valid component scalar assignment to result from a BY NAME structure assignment, it is necessary that all the scalar names derived from original structure name operands have identical qualification relative to the structure name originally specified--e.g: DCL 1S, 2T, 2U, 3V; DCL 1W, 2T, 2U; W=S; gives rise to the component assignments W.T=S.T; W.U=S.U; the second of which is invalid.

System Action: The BY NAME assignment statement is deleted.

User Response: Refer to the PL/I Reference Manual - rules for expansion of structure assignment BY NAME - and correct the error.

S IEM0833I THE EXPANSION OF BY NAME ASSIGNMENT STATEMENT NUMBER xxx HAS RESULTED IN NO COMPONENT ASSIGNMENTS.

System Action: The statement is treated as a null statement.

S IEM0834I THE ASSIGNED OPERAND IN BY NAME ASSIGNMENT STATEMENT NUMBER xxx IS NOT A STRUCTURE OR AN ARRAY OF STRUCTURES. STATEMENT DELETED.

Explanation: In BY NAME assignment, the operand to the left of the equals sign must be a structure or an array of structures.

S IEM0835I ALL OPERANDS LEFT OF EQUAL SYMBOL IN MULTIPLE STRUCTURE ASSIGNMENT STATEMENT NUMBER xxx ARE NOT STRUCTURES. STATEMENT DELETED

Explanation: In multiple structure assignment, all the operands being assigned to must be structures.

System Action: Replaces statement by a null statement and continues.

User Response: Break statement up into a series of separate statements.

S IEM0836I ILLEGAL ARRAY REFERENCE IN STRUCTURE ASSIGNMENT OR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: In PL/I, arrays of scalars are invalid operands in structure, or array of structure, expressions.

T IEM0837I COMPILER ERROR IN INPUT TO PHASE IEMHF.

Explanation: Meaningless input. This message is also produced if an error is found in a DECLARE statement. In that case, a second message, of severity levels is issued giving details of the error.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM0838I EXPRESSION RIGHT OF EQUAL SYMBOL IN BY NAME ASSIGNMENT STATEMENT NUMBER xxx CONTAINS NO STRUCTURES. BY NAME OPTION DELETED.

Explanation: In an assignment statement having the BY NAME option, the expression to the right of the equal symbol = should contain at least one structure or array of structures.

S IEM0848I IMPLEMENTATION RESTRICTION. THE EXPANSION OF STRUCTURRE EXPRESSIONS IN STATEMENT NUMBER xxx HAS CAUSED A TABLE INTERNAL TO THE COMPILER TO OVERFLOW. STATEMENT DELETED.

Explanation: The nesting of structure expressions in argument lists is too deep.

User Response: Decrease the nesting.

S IEM0849I AN EXPRESSION OR ASSIGNMENT IN STATEMENT NUMBER xxx EITHER CONTAINS SEPARATE STRUCTURES WITH DIFFERENT STRUCTURING, OR CONTAINS BOTH A STRUCTURE AND AN ARRAY OF STRUCTURES. THE STATEMENT HAS BEEN DELETED.

Explanation: PL/I does not allow separate structures with different structuring within the same expression or assignment. This compiler does not support reference to both a structure and an array of structures within the same expression or assignment.

S IEM0850I THE BOUNDS OF THE BASE ARRAYS OF THE STRUCTURE OPERANDS OF THE STRUCTURE EXPRESSION OR ASSIGNMENT IN STATEMENT NUMBER xxx ARE NOT THE SAME. THE STATEMENT HAS BEEN DELETED.

Explanation: In a structure assignment or expression, all structure operands must have the same number of contained elements at the next level. Corresponding sets of contained elements must all be arrays of structures, structures, arrays, or scalars. The arrays must have the same dimensionality and bounds.

User Response: Refer to the PL/I Reference Manual - the expansion of array and structure expressions and assignments - and correct the program.

S IEM0851I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

Explanation: The expansion of structure expressions or assignments has given rise to a statement which exceeds one text block in length.

User Response: Decrease statement size.

S IEM0852I A SUBSCRIPTED REFERENCE TO AN ARRAY OF STRUCTURES IN STATEMENT NUMBER xxx HAS THE WRONG NUMBER OF SUBSCRIPTS. THE STATEMENT HAS BEEN DELETED.

Explanation: Subscripted references to arrays must have subscript expressions equal in number to the dimensionality of the array.

User Response: Correct subscripted reference.

S IEM0853I DEFERRED FEATURE. A STRUCTURE ASSIGNMENT OR EXPRESSION IN STATEMENT NUMBER xxx INVOLVES CROSS SECTIONS OF ARRAYS. STATEMENT DELETED.

Explanation: This compiler does not support reference to cross sections of arrays of structures.

User Response: Expand the statement in DO loops replacing '*'s in subscripts by the appropriate DO control variable.

S IEM0864I IMPLEMENTATION RESTRICTION. NESTING OF ARRAY ASSIGNMENTS OR I/O LISTS TOO DEEP. STATEMENT NUMBER xxx DELETED.

Explanation: Nesting of functions, combined with size of arrays involved, is too great.

User Response: Simplify by splitting into two or more statements.

S IEM0865I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

Explanation: Nesting of functions with array arguments involves a large expansion of text.

User Response: Simplify by splitting into two or more statements.

S IEM0866I NEITHER MULTIPLE ASSIGNMENT COMMA NOR ASSIGNMENT MARKER FOUND IN CORRECT POSITION. STATEMENT NUMBER xxx DELETED.

Explanation: An expression occurring on the left-hand side of an assignment must be contained within parentheses.

S IEM0867I NUMBER OF * SUBSCRIPTS SPECIFIED FOR zzzz IS NOT THE DIMENSIONALITY OF THE LEFTMOST ARRAY. STATEMENT NUMBER xxx DELETED

Explanation: Array references in expressions must have the same dimensionality.

S IEM0868I BOUNDS OF ARRAY zzzz ARE NOT SAME AS FOR LEFTMOST ARRAY IN ASSIGNMENT OR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Array references in expressions must have the same bounds.

S IEM0869I DIMENSIONS OF ARRAY zzzz ARE NOT THE SAME AS FOR LEFTMOST ARRAY IN ASSIGNMENT OR I/O EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Array references in expressions must have the same dimensionality.

S IEM0870I ARGUMENT OF PSEUDO-VARIABLE INVALID. STATEMENT NUMBER xxx IS DELETED.

Explanation: Arguments of pseudo-variables must be variables which are not expressions.

S IEM0871I SCALAR zzzz ON LEFT OF EQUAL SYMBOL IN ARRAY ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

S IEM0872I NUMBER OF SUBSCRIPTS SPECIFIED FOR LEFTMOST OPERAND zzzz IS NOT SAME AS DIMENSIONALITY. STATEMENT NUMBER xxx DELETED.

Explanation: The number of subscripts specified must be the same as the number of dimensions of the array.

S IEM0873I ARGUMENTS OF PSEUDO-VARIABLE COMPLEX INCORRECT. STATEMENT NUMBER xxx DELETED.

Explanation: Only one argument given for COMPLEX, or expression illegally used as argument for pseudo-variable.

S IEM0874I SECOND ARGUMENT OF PSEUDO-VARIABLE COMPLEX OR FIRST ARGUMENT OF REAL, IMAG, OR UNSPEC EITHER IS NOT FOLLOWED BY A RIGHT PARENTHESIS OR IS AN EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Too many arguments given for this pseudo-variable, or expression used as argument.

S IEM0875I ONSOURCE OR ONCHAR APPEARS IN A DIMENSIONED ARRAY ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Explanation: ONSOURCE and ONCHAR may only appear in scalar assignments.

S IEM0876I ARGUMENTS OF PSEUDO-VARIABLE SUBSTR INCORRECT. STATEMENT NUMBER xxx DELETED.

Explanation: Only one argument given for SUBSTR, or expression illegally used as argument for pseudo-variable.

S IEM0877I UNSUBSCRIPTED ARRAY zzzz IN SCALAR ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Explanation: Only scalars can be assigned to scalars.

S IEM0878I STRUCTURE zzzz FOUND IN ARRAY OR SCALAR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Structures may only be assigned to structures.

S IEM0879I PSEUDO-VARIABLE COMPLEX, REAL, IMAG, UNSPEC, COMPLETION, OR SUBSTR LACKS ARGUMENTS. STATEMENT NUMBER xxx DELETED.

Explanation: Pseudo-variables COMPLEX, REAL, IMAG, UNSPEC, COMPLETION, and SUBSTR require arguments.

User Response: Check whether the variable was intended as a pseudo-variable or whether it should have been declared otherwise.

S IEM0880I NUMBER OF SUBSCRIPTS SPECIFIED FOR zzzz IS NOT SAME AS DIMENSIONALITY. STATEMENT NUMBER xxx DELETED.

Explanation: The number of subscripts specified must be the same as the number of dimensions of the array.

S IME0881I NUMBER OF DIMENSIONS IN REFERENCE TO zzzz IS NOT SAME AS THAT OF EXPRESSION OR ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

T IEM0882I COMPILER ERROR. INVALID INPUT TO PHASE HK AT STATEMENT NUMBER xxx

Explanation: Illegal text has been encountered.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM0883I DEFERRED FEATURE. STRUCTURE zzzz PASSED AS ARGUMENT TO THE TRANSLATE OR VERIFY FUNCTION. STATEMENT NUMBER xxx DELETED.

User Response: Remove structure from statement.

T IEM0896I IMPLEMENTATION RESTRICTION. TOO MANY LEVELS OF ISUB NESTING IN STATEMENT NUMBER xxx

Explanation: Stack has overflowed scratch core. The maximum number of levels of nesting possible depends on the dimensionality of the arrays involved.

System Action: Compilation is terminated.

User Response: Reduce the number of levels of nesting in the statement.

S IEM0897I ISUB DEFINED OPERAND zzzz HAS NOT BEEN DECLARED AS AN ARRAY. ISUBS IN STATEMENT NUMBER xxx DELETED.

User Response: Declare defined item with dimension attribute.

T IEM0898I NO SUBSCRIPTS AFTER ISUB-DEFINED ITEM zzzz IN STATEMENT NUMBER xxx

Explanation: This is a compiler error.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM0899I MULTIPLIER IN ISUB DEFINING LIST FOR zzzz IN STATEMENT NUMBER xxx IS NOT A SCALAR EXPRESSION.

Explanation: A comma has been found within the ISUB multiplier expression.

System Action: Remainder of expression, after comma, ignored.

User Response: Rewrite expression.

E IEM0900I * USED AS SUBSCRIPT FOR ISUB DEFINED ITEM zzzz IN STATEMENT NUMBER xxx. ZERO SUBSTITUTED.

User Response: Rewrite without *.

E IEM0901I ISUB NUMBER IN DEFINING LIST FOR zzzz IN STATEMENT NUMBER xxx IS TOO GREAT. MAXIMUM NUMBER USED.

System Action: The ISUB number is replaced by the number of dimensions of the defined array.

User Response: Rewrite defining DECLARE statement.

E IEM0902I WRONG NUMBER OF SUBSCRIPTS FOR ISUB DEFINED ITEM zzzz IN STATEMENT NUMBER xxx. SUBSCRIPTS IGNORED OR ZERO SUPPLIED.

User Response: Rewrite with correct number of subscripts. The error may be in the reference to the defined item or in the defining DECLARE statement.

T IEM0903I COMPILER ERROR. ERROR DETECTED IN DEFINING ISUB LIST FOR zzzz IN STATEMENT NUMBER xxx

Explanation: Compiler error. Either (a) SUB not found where expected, or (b) SUB0 found without a multiplier expression.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

S IEM0906I STATEMENT DELIMITER FOUND WITHIN SUBSCRIPT LIST FOR zzzz IN STATEMENT NUMBER xxx

Explanation: The subscript scan routine has found a statement marker.

System Action: The present statement is dropped and the new one processed. Compilation will not be completed.

User Response: Check text, but this is probably a compiler error, in which case save relevant data and inform the system manager or adminstrator of the error.

S IEM0907I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Explanation: Statement length exceeds text block size.

System Action: Statement is truncated. Compilation will not be completed.

User Response: Simplify statement.

S IEM1024I ILLEGAL USE OF zzzz IN STATEMENT NUMBER xxx. A FIXED BINARY ZERO CONSTANT IS SUBSTITUTED.

Explanation: A non-scalar identifier has been specified in a context that requires a scalar identifier.

System Action: Replaces illegal identifier with arithmetic constant zero.

S IEM1025I IDENTIFIER zzzz ILLEGALLY USED AS SUBSCRIPT IN STATEMENT NUMBER xxx

Explanation: A subscript has been used which is not a scalar, a scalar expression, or a constant.

System Action: Replaces illegal subscript with arithmetic constant zero.

W IEM1026I STATEMENT NUMBER xxx IS AN UNLABELED FORMAT STATEMENT

Explanation: A FORMAT statement should have a label.

T IEM1027I THE SUBSCRIPTED STRUCTURE ITEM zzzz IS ILLEGALLY USED IN STATEMENT NUMBER xxx

Explanation: The indicated structure item is used in a statement other than an assignment statement or an I/O data list.

System Action: Compilation is terminated.

T IEM1028I COMPILER ERROR IN STATEMENT NUMBER xxx. ILLEGAL INPUT TEXT FOR PHASE IA.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1029I THE APPEARANCE OF THE ARRAY CROSS-SECTION IN STATEMENT NUMBER xxx IS NOT SUPPORTED BY THIS VERSION OF THE COMPILER.

Explanation: A feature has been used that is not supported.

System Action: Compilation is terminated.

T IEM1030I IMPLEMENTATION RESTRICTION. TOO MANY DUMMY ARGUMENTS ARE BEING PASSED IN STATEMENT NUMBER xxx. A MAXIMUM OF 64 DUMMY ARGUMENTS MAY BE PASSED IN EACH INVOCATION.

T IEM1040I DEFERRED FEATURE. STRUCTURE ARGUMENT IS BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx

Explanation: This compiler does not permit structures to be passed as arguments to built-in functions.

System Action: Compilation is terminated.

User Response: Rewrite program, avoiding unsupported feature.

T IEM1051I DEFERRED FEATURE. STRUCTURE ARGUMENT IS BEING PASSED TO PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx

Explanation: This compiler does not permit structures to

be passed as arguments to pseudo-variables.

System Action: Compilation is terminated.

User Response: Rewrite program, avoiding unsupported feature.

T IEM1056I INVALID ARGUMENT IS BEING PASSED TO ENTRY NAME zzzz IN STATEMENT NUMBER xxx

System Action: Compilation is terminated.

T IEM1057I DECIMAL INTEGER CONSTANT IS NOT BEING PASSED, AS REQUIRED, TO FUNCTION zzzz IN STATEMENT NUMBER xxx

Explanation: Argument to build-in function is not a decimal integer as expected.

System Action: Compilation is terminated.

T IEM1058I ARRAY OR STRUCTURE ARGUMENT IS NOT BEING PASSED, AS REQUIRED, TO FUNCTION zzzz IN STATEMENT NUMBER xxx

Explanation: Argument to built-in function is not an array or a structure, as expected.

System Action: Compilation is terminated.

T IEM1059I FIRST ARGUMENT BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx SHOULD BE AN ARRAY.

Explanation: Argument to built-in function is not an array as expected.

System Action: Compilation is terminated.

T IEM1060I TOO MANY ARGUMENTS ARE BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx

Explanation: Too many arguments are being passed to a built-in function.

System Action: Compilation is terminated.

T IEM1061I TOO FEW ARGUMENTS ARE BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Too few arguments are being passed to a built-in function.

System Action: Compilation is terminated.

T IEM1062I COMPILER ERROR. CORRECT GENERIC SELECTION FOR FUNCTION zzzz IN STATEMENT NUMBER xxx HAS NOT BEEN ACHIEVED.

Explanation: Compiler, although being given a legal argument to a generic built-in function, is unable to make the selection.

System Action: Function result is set to zero and compilation is terminated.

User Response: Save relevant data and inform the system nanager or administrator of the error.

T IEM1063I COMPILER ERROR. UNEXPECTED SITUATION HAS ARISEN IN THE SCANNING OF THE ARGUMENTS PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx

Explanation: Compiler is unable to correctly scan an argument list.

System Action: Function result is set to zero.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1064I COMPILER ERROR. THE GENERIC FAMILIES ASSOCIATED WITH ENTRY NAME zzzz HAVE BEEN INCORRECTLY FORMED IN THE DICTIONARY

Explanation: The dictionary entry for one or more of the generic families is not a recognizable entry type.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or adminstrator of the error.

T IEM1065I NO GENERIC SELECTION POSSIBLE FOR THE ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect use of the GENERIC attribute resulting in no selection being possible.

System Action: Compilation is terminated.

T IEM1066I MORE THAN ONE GENERIC SELECTION IS POSSIBLE FOR THE ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect use of the GENERIC attribute resulting in more than one selection being possible.

System Action: Compilation is terminated.

T IEM1067I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH AN ILLEGAL ARGUMENT.

Explanation: Argument to pseudo-variable cannot be converted to a legal type; or, structure argument being used with pseudo-variable.

System Action: Compilation is terminated.

T IEM1068I AN ARRAY IS BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx. THIS PRODUCES AN ARRAY EXPRESSION WHICH IS INVALID IN THIS CONTEXT.

System Action: Compilation is terminated.

S IEM1070I IMPLEMENTATION RESTRICTION. AN ARGUMENT OF A BUILT-IN FUNCTION USED IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED TO 32,767.

T IEM1071I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH TOO MANY ARGUMENTS.

System Action: Compilation is terminated.

T IEM1072I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH TOO FEW ARGUMENTS.

System Action: Compilation is terminated.

T IEM1073I COMPILER ERROR. CORRECT GENERIC SELECTION FOR PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxxx HAS NOT BEEN ACHIEVED.

Explanation: Compiler error. Although being given a legal argument to a generic pseudo-variable, compiler is unable to make the selection.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1074I COMPILER ERROR. UNEXPECTED SITUATION HAS ARISEN IN THE SCANNING OF THE ARGUMENTS PASSED TO PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx

Explanation: Unable to correctly scan an argument list of a pseudo-variable.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

W IEM1075I THE ARGUMENT zzzz OF THE STRING PSEUDO-VARIABLE IN STATEMENT NUMBER xxx CONTAINS A PICTURED ELEMENT. THIS IS NOT CHECKED FOR VALIDITY ON ASSIGNMENT.

Explanation: Invalid data in pictured element may cause subsequent errors.

T IEM1076I COMPILER ERROR IN PHASE JD

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1078I IMPLEMENTATION RESTRICTION. THE NUMBER OF FAMILY MEMBERS AND ARGUMENTS ASSOCIATED WITH THE GENERIC ENTRY NAME yyyy EXCEEDS THE LIMITATION IMPOSED.

Explanation: There is an implementation restriction on the number of family members and arguments associated with GENERIC entry names.

System Action: Compilation is terminated.

User Response: Divide the generic family into two or more generic families.

S IEM1082I STATEMENT NUMBER xxx CONTAINS AN INVALID USE OF AREA OR POINTER DATA. PART OR ALL OF THE STATEMENT HAS BEEN DELETED.

Explanation: The statement contains an operation that:

1. Is not permitted for AREA or POINTER data, or
2. Can only be used with AREA or POINTER data but such data is not the data specified for the operation.

System Action: Deletes the statement or clause responsible for the error.

T IEM1088I THE SIZE OF AGGREGATE zzzz IS GREATER THAN 8,388,607 BYTES. STORAGE ALLOCATION WILL BE UNSUCCESSFUL.

Explanation: The message is generated when an array or structure size exceeds $2^{23}-1$.

System Action: Array or structure mapping for the item is terminated, but the compilation continues. Execution of object decks containing references to the item will give incorrect results.

User Response: Check source code.

S IEM1089I THE RELATIVE VIRTUAL ORIGIN OF AGGREGATE zzzz IS LESS THAN -8,388,608 BYTES. STORAGE HAS NOT BEEN ALLOCATED.

Explanation: The low bounds of the arrays in the aggregate are too high.

System Action: Compilation terminates.

User Response: Reduce the size of the aggregate, or reduce the value of the low bounds in the aggregate.

S IEM1090I THE STRUCTURE zzzz DECLARED IN STATEMENT NUMBER xxx CONTAINS VARYING STRINGS AND MAY APPEAR IN A RECORD I/O STATEMENT

Explanation: VARYING strings in structures are not permitted in RECORD I/O statements.

System Action: The RECORD I/O statement is processed but the record will contain erroneous information.

User Response: Correct the source code.

W IEM1092I THE TASKS, EVENTS OR LABELS CONTAINED IN STRUCTURE zzzz DECLARED IN STATEMENT NUMBER xxx MAY LOSE THEIR VALIDITY IF USED IN A RECORD I/O STATEMENT.

Explanation: The TASK, EVENT, or LABEL variable may lose its validity in transmission.

User Response: Correct the source code if necessary.

E IEM1104I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx INVOLVES DATA NOT ALLOWED FOR STRING CLASS OVERLAY DEFINING.

Explanation: The programmer's use of the DEFINED attribute contravenes the language rules concerned with the permitted data types with dimensionality of base and defined item.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1105I THE DATA CHARACTERISTICS OF zzzz DECLARED IN STATEMENT NUMBER xxx DO NOT MATCH THOSE OF THE DEFINING BASE.

Explanation: For valid use of the DEFINED attribute, both the defined item and the base must be of the same defining class.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1106I THE DIMENSIONALITY OF zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT THE SAME AS THAT OF THE DEFINING BASE.

Explanation: With the exception of the case of string class defining, if either the base or the defined item are arrays, then both the base and the defined item must be arrays with the same dimensionality.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1107I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES VARYING STRINGS.

Explanation: In use of the DEFINED attribute, neither the base nor the defined item may involve strings declared VARYING.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1108I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES DATA AGGREGATES THAT ARE NOT UNALIGNED.

Explanation: In the case of string class overlay defining where either or both the base and the defined item are aggregates, then the aggregates must have the PACKED attribute.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1110I THE DEFINING BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS SHORTER THAN THE DEFINED ITEM.

Explanation: In the case of string class overlay defining, the defined item must occupy a subset of the base storage.

In the case of correspondence defining, the length of each defined element must not be greater than the length of each base element.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1111I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx INVOLVES A STRUCTURE HAVING ELEMENTS NOT ALL OF THE SAME DEFINING CLASS.

Explanation: In the case of string class overlay defining where the defined item or the base is a structure, then all the elements of the structure must be data of the same string defining class.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1112I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES THE POS ATTRIBUTE.

Explanation: The POSITION attribute may only be declared for data of the string class which is overlay defined.

System Action: Compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1113I THE STRUCTURE DESCRIPTION OF zzzz DECLARED IN STATEMENT

NUMBER xxx DOES NOT MATCH THAT OF THE DEFINING BASE.

Explanation: Where a structure or an array of structures is defined on a structure or an array of structures, and it is not string class overlay defining, then the two structure descriptions must be identical.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

W IEM1114I IF THE BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS ALLOCATED WITH THE DECLARED EXTENTS, THE DEFINING WILL BE IN ERROR.

Explanation: In the case of string class overlay defining, the defined item must occupy a subset of the base storage. If the base is of CONTROLLED storage class, its extents are not finally resolved until execution time.

System Action: No further action.

User Response: Check that when the base is allocated it is of adequate size to accommodate the defined item.

E IEM1115I THE DEFINING BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS AN ARRAY FORMAL PARAMETER. IF THE MATCHING ARGUMENT IS AN ELEMENT OF AN ARRAY OF STRUCTURES OR A CROSS SECTION OF AN ARRAY, THE DEFINING WILL BE IN ERROR.

Explanation: The base for string class overlay defining must occupy contiguous storage.

System Action: Comments and continues.

User Response: Check validity of arguments.

S IEM1120I COMPILER ERROR. INVALID SIGN FOUND IN INITIAL VALUE LIST FOR zzzz IN STATEMENT NUMBER xxx. TREATED AS PLUS.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM1121I COMPILER ERROR. INVALID MARKER FOUND IN INITIAL VALUE LIST FOR zzzz IN STATEMENT NUMBER xxx. INITIAL VALUE LIST TRUNCATED.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM1122I UNSUPPORTED FEATURE. AN EXPRESSION HAS BEEN USED TO INITIALIZE STATIC STRING zzzz IN STATEMENT NUMBER xxx. STRING INITIALIZED TO NULL.

Explanation: A complex expression has been used to initialize a STATIC string. This feature is not supported.

System Action: The string is initialized to null.

User Response: Amend source code. The restriction can be overcome by using an assignment statement instead of the INITIAL attribute.

S IEM1123I INITIAL VALUE FOR STATIC DATA ITEM zzzz IN nnnn IS NOT A CONSTANT. INITIALIZATION TERMINATED.

User Response: Use a constant in the INITIAL string.

S IEM1125I ITERATION FACTOR USED IN INITIALIZATION OF STATIC ARRAY zzzz IN STATEMENT NUMBER xxx IS TOO LARGE. REPLACED BY ZERO.

Explanation: Iteration factors are converted by the compiler to REAL FIXED BINARY with a default precision of 15,0. The iteration factor referred to in the message has a value greater than 2^{15}, and therefore exceeds the default precision.

System Action: The iteration factor is replaced by zero.

User Response: Amend source code so that iteration factor does not exceed 2^{15}.

T IEM1200I COMPILER ERROR. ILLEGAL TRIPLE IN TEXT. CURRENT STATEMENT NUMBER xxx

Explanation: Phase LA is out of step in scanning text.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1210I COMPILER ERROR NUMBER nnnn IN PHASE KE.

Explanation: Compiler error in dictionary or text scan.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error. Meanwhile, recompile with OPT=1 to bypass DO-loop optimization phases.

T IEM1211I COMPILER ERROR IN PHASE KE.

Explanation: Compiler error found in scan of dictionary.

System Action: Compilation terminated.

User Response: Save relevent data and inform your system manager or administrator.

T IEM1220I COMPILER ERROR NUMBER nnnn IN PHASE KU IN STATEMENT NUMBER xxx.

Explanation: A compiler error has occurred in the DO loop control optimization phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error. Meanwhile, recompile with OPT=1 to bypass DO-loop optimization phases.

T IEM1223I COMPILER ERROR. INVALID INPUT TYPE nnnn TO OPTIMIZING PHASE KO.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error. Meanwhile, recompile with OPT=1 to bypass DO-loop and subscript optimization phases.

T IEM1224I COMPILER ERROR NUMBER nnnn IN PHASE KA.

Explanation: Invalid request encountered by table-handling routines in Phase KA.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1226I COMPILER ERROR IN PHASE KG IN OR NEAR STATEMENT NUMBER xxx

Explanation: Error occurred while scanning text or tables.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error. Meanwhile, recompile with OPT=0 or 1.

T IEM1569I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: The number of symbolic register names generated by the code generation section of the compiler has exceeded the maximum allowed.

System Action: Compilation is terminated.

User Response: Break down the compilation into smaller modules.

T IEM1570I COMPILER ERROR. INVALID TRIPLE FOLLOWING WHILE PRIME TRIPLE.

Explanation: Input to phase LG of compiler is erroneous. A WHILE' triple is not followed by CV' or compiler label.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1571I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: No more main storage is available for the stack of nested DO statements (both in source language and those generated internally for array assignments).

System Action: Compilation is terminated.

User Response: Simplify nesting so as to reduce number of levels.

S IEM1572I ILLEGAL USE OF ARRAY OR STRUCTURE VARIABLE IN DO STATEMENT NUMBER xxx

Explanation: A non-scalar variable has been used as (1) the control variable, (2) a control variable subscript, or (3) a loop limit or increment value.

System Action: Generates an error stop at execution time.

S IEM1574I INVALID LOOP CONTROL EXPRESSION OR CONTROL VARIABLE SUBSCRIPT IN STATEMENT NUMBER xxx REPLACED BY FIXED BINARY TEMPORARY.

Explanation: Either something other than an arithmetic or string datum has been used as a subscript in the control variable, or something other than an arithmetic or string datum, label variable, or label constant has been used in an initial value, TO, or BY clause.

System Action: Ignores the erroneous expression and uses a fixed binary temporary.

S IEM1575I DO LOOP CONTROL PSEUDO-VARIABLE IN STATEMENT NUMBER xxx HAS AN INVALID ARGUMENT. BINARY INTEGER TEMPORARY ASSUMED.

Explanation: An invalid argument, such as an expression or function, has been used in a pseudo-variable.

System Action: Assigns invalid argument to binary temporary, and uses latter as argument.

W IEM1588I VARYING STRING HAS BEEN USED AS AN ARGUMENT TO ADDR FUNCTION IN STATEMENT NUMBER xxx

Explanation: The result of the ADDR function can only be assigned to a pointer qualifying a based variable. If the argument to the ADDR function is a VARYING string, the length of the data in the based variable may not be the length required in the program.

System Action: None.

User Response: Check this use of the ADDR function.

E IEM1599I A STATEMENT LABEL CONSTANT IS BEING PASSED AS AN ARGUMENT TO THE ADDR BUILT-IN FUNCTION IN STATEMENT NUMBER xxx

Explanation: The argument to the ADDR built-in function must be a variable.

T IEM1600I COMPILER ERROR. ILLEGAL ABSOLUTE REGISTER NUMBER. STATEMENT NUMBER xxx

Explanation: Compiler error. Fixed binary arithmetic uses an unassigned general register number greater than 15, or floating point arithmetic uses a floating point register greater than 6.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1601I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx REQUIRES MORE THAN 200 INTERMEDIATE RESULT DESCRIPTIONS.

Explanation: Compiler limitation. The temporary result stack, which holds 200 items, is full.

System Action: Compilation is terminated.

User Response: This error should only occur in very large statements. Divide the statement into two or more smaller statements.

T IEM1602I COMPILER ERROR. INSUFFICIENT NUMBER OF TEMPORARY RESULT

DESCRIPTIONS. STATEMENT NUMBER xxx

Explanation: Compiler error. A temporary result is required but the temporary result stack is empty. This can happen if triples are out of order or if extra triples were inserted.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1603I COMPILER ERROR. COUNT OF FREE FLOATING REGISTERS IS WRONG. STATEMENT NUMBER xxx

Explanation: Compiler error in expression evaluation phase. Error in control blocks for floating point registers.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1604I COMPILER ERROR. SECOND OPERAND FOR RS OR SS INSTRUCTION IS IN A REGISTER. STATEMENT NUMBER xxx

Explanation: Compiler error in expression evaluation phase. Attempt to generate an RS or SS type pseudo-code instruction using a register as the second operand.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM1605I IN STATEMENT NUMBER xxx FIXED DECIMAL VARIABLE CANNOT BE CORRECTLY CONVERTED TO BINARY DUE TO SIZE OF SCALE FACTOR.

Explanation: Error in source program. When a fixed decimal variable is corrected to fixed binary, the magnitude of its scale factor is multiplied by 3.31. If the original scale factor is >38 or <-38, then the fixed binary scale factor would be outside the range +127 to -128.

System Action: The fixed binary scale factor is set to +127 or -128. Processing continues.

User Response: The data in the expression must be redeclared with more suitable scale factors.

T IEM1606I COMPILER ERROR. FUNCTION NOT FOLLOWED BY RESULT DESCRIPTION. STATEMENT NUMBER xxx

Explanation: Compiler error. A function is not followed by TMPD or LEFT triples giving the result type.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or system administrator of the error.

S IEM1607I LABEL, EVENT, FILE, OR TASK ITEM zzzz IN STATEMENT NUMBER xxx IS USED IN AN EXPRESSION WHICH IS ILLEGAL.

Explanation: Error in source program. A label, event, file, or task datum cannot be used in an expression. Alternatively, this can be a compiler error when an unrecognizable dictionary entry is used in an expression.

System Action: Substitutes a fixed binary (31,0) data item (if the illegal item occurs in an arithmetic expression) or a null bit string (if it occurs in a string expression). Processing is continued.

User Response: If error in source program, correct it.

E IEM1608I LT, LE, GE, OR GT COMPARISON OPERATOR ILLEGALLY USED IN STATEMENT NUMBER xxx WITH COMPLEX OPERANDS. REPLACED WITH EQUALS OPERATOR.

Explanation: Error in source program. The only legal comparison between complex operands is '='.

System Action: The operator is replaced with '=' and processing continues.

User Response: Correct source program using either the ABS function or possibly the REAL and IMAG functions.

T IEM1609I COMPILER ERROR. ILLEGAL DICTIONARY REFERENCE X'00..'. STATEMENT NUMBER xxx

Explanation: Compiler error. The symbolic dictionary reference is less than 256.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or system administrator of the error.

T IEM1610I COMPILER ERROR IN PHASE LW AT STATEMENT NUMBER xxx. INSUFFICIENT NUMBER OF TEMPORARY RESULT DESCRIPTIONS.

Explanation: Compiler error. A temporary result is required but the temporary result stack is empty. This can happen if the triples are out of order or if extra triples have been inserted.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or system administrator of the error.

E IEM1611I IMPLEMENTATION RESTRICTION. A STRING RESULT LONGER THAN 32767 IS PRODUCED BY CONCATENATE IN STATEMENT NUMBER xxx. STRING TRUNCATED TO LENGTH 32767.

Explanation: Maximum string length for this implementation is 32767. This may be exceeded during concatenation, because the length of the intermediate result is the sum of the operand lengths.

System Action: Compilation continues with string result length truncated to 32767.

User Response: Shorter strings must be used.

W IEM1612I IMPLEMENTATION RESTRICTION IN STATEMENT NUMBER xxx. INTERMEDIATE WORK SPACE IS OBTAINED MORE THAN 50 TIMES IN A STRING EXPRESSION. SOME WORK SPACE WILL NOT BE RELEASED UNTIL THE END OF THE BLOCK.

Explanation: The intermediate work space is required each time a function returns a string result or each time a library module is called.

System Action: The first 50 areas of work space are released. The remainder may not be released until the end of the block. Compilation continues and execution is valid.

User Response: Divide the string expression into several sub-expressions.

S IEM1613I ILLEGAL USE OF ARRAY OR STRUCTURE VARIABLE IN STATEMENT NUMBER xxx

Explanation: Illegal source program.

System Action: Severe error message and object program branch. Compilation continues, assuming scalar of same type if array, or fixed binary (31,0) type if structure.

User Response: Insert DO blocks for array, or break down structure into its components.

W IEM1614I IMPLEMENTATION RESTRICTION. A VARYING STRING RESULT LONGER THAN 32767 MAY BE PRODUCED BY CONCATENATE IN STATEMENT NUMBER xxx. STRING TRUNCATED TO LENGTH 32767.

Explanation: The sum of the maximum lengths of two strings in a concatenation operation exceeds the implementation restriction of 32767. Since one or both of the operands is a VARYING string, it is not known at compile-time whether the restriction will be exceeded at execution time.

System Action: Compilation continues with string result maximum length truncated to 32767.

User Response: Shorter strings must be used if the sum of the execution-time current lengths will ever exceed 32767.

E IEM1615I SECOND ARGUMENT IN THE SUBSTR FUNCTION IN STATEMENT NUMBER xxx IS ZERO, WHICH IS INVALID. ZERO HAS BEEN REPLACED BY ONE.

E IEM1616I SECOND ARGUMENT IN THE SUBSTR PSEUDO-VARIABLE IN STATEMENT NUMBER xxx IS ZERO, WHICH IS INVALID. ZERO HAS BEEN REPLACED BY ONE.

T IEM1617I COMPILER ERROR. ILLEGAL RETURN FROM SCAN ROUTINE. STATEMENT NUMBER xxx

Explanation: An illegal return of control has been made by the SCAN routine which supports the code generation phases.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM1618I PSEUDO-VARIABLE IN STATEMENT NUMBER xxx INCORRECTLY SPECIFIED. REPLACED BY FIXED BINARY TEMPORARY.

Explanation: A pseudo-variable in the given source statement has been incorrectly specified, for example, has an incorrect number of arguments.

System Action: Ignores the pseudo-variable and uses a fixed binary temporary instead.

S IEM1619I RIGHT HAND SIDE OF STATEMENT NUMBER xxx CANNOT BE ASSIGNED TO A PSEUDO-VARIABLE. ASSIGNMENT IGNORED.

Explanation: The expression on the right-hand side of the specified statement cannot be assigned to a pseudo-variable, that is, it is not an arithmetic or string datum.

System Action: The assignment is deleted from the text.

S IEM1620I 'IMAG' IN STATEMENT NUMBER xxx HAS REAL ARGUMENT. REPLACED BY ASSIGNMENT TO TEMPORARY FIXED BINARY INTEGER.

Explanation: The pseudo-variable 'IMAG' is meaningful only if its argument is of type complex.

System Action: A fixed binary temporary target is provided for the assignment or input data list item and the pseudo-variable is ignored.

S IEM1621I ILLEGAL PSEUDO-VARIABLE ARGUMENT IN STATEMENT NUMBER xxx REPLACED BY BINARY TEMPORARY.

Explanation: A pseudo-variable in the specified statement has an illegal argument, that is, one whose data type is not permissible in that context.

System Action: A temporary whose type is legal in the context is used to replace the erroneous argument and the latter is removed from the text.

S IEM1622I FIRST ARGUMENT OF PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS NOT A STRING VARIABLE. ARGUMENT HAS BEEN CONVERTED TO STRING TEMPORARY AND THE ASSIGNMENT MADE THERETO.

Explanation: SUBSTR pseudo-variable cannot have a first argument which is not a string variable.

System Action: Code is compiled to assign to a string temporary. The original argument remains unchanged.

W IEM1625I PSEUDO-VARIABLE REAL IN STATEMENT NUMBER xxx DOES NOT HAVE COMPLEX ARGUMENT. ARGUMENT HAS BEEN TREATED AS HAVING ZERO IMAGINARY PART.

System Action: Code is generated to perform assignment to the specified REAL argument.

S IEM1626I ILLEGAL NEGATIVE SECOND ARGUMENT IS BEING PASSED TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1627I ILLEGAL NEGATIVE THIRD ARGUMENT IS BEING PASSED TO THE FUNCTION

SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1628I THE SUBSTRING SPECIFIED BY THE SECOND AND THIRD ARGUMENTS TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx DOES NOT LIE WITHIN THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

S IEM1629I THE SECOND ARGUMENT TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

T IEM1630I COMPILER ERROR IN CEIL/FLOOR/TRUNC IN-LINE FUNCTION IN STATEMENT NUMBER xxx

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1631I COMPILER ERROR IN MOD IN-LINE FUNCTION IN STATEMENT NUMBER xxx

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

W IEM1632I THE INVOCATION OF THE ROUND FUNCTION IN STATEMENT NUMBER xxx WILL ALWAYS GIVE A ZERO RESULT.

Explanation: (p - q + r) is zero or negative, where p = precision, q = scale factor, and r = rounding position.

System Action: Result is set to zero.

User Response: Check scale and precision of the first argument in ROUND function.

S IEM1633I ILLEGAL NEGATIVE SECOND ARGUMENT IS BEING PASSED TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1634I ILLEGAL NEGATIVE THIRD ARGUMENT IS BEING PASSED TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1635I THE SUBSTRING SPECIFIED BY THE SECOND AND THIRD ARGUMENTS TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx DOES NOT LIE WITHIN THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

S IEM1636I THE SECOND ARGUMENT TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

S IEM1637I THE THIRD ARGUMENT TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

S IEM1638I THE THIRD ARGUMENT TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

T IEM1639I COMPILER ERROR. INCORRECT INPUT TO SUBROUTINE 6 IN MODULE IEMMF IN STATEMENT NUMBER xxx.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1640I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz DOES NOT MATCH ANY OF THE FAMILY MEMBERS.

System Action: Compilation is terminated.

User Response: Provide correct parameter description.

W IEM1641I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz DESCRIBES THE ENTRY NAME'S RESULT TYPE RATHER THAN ARGUMENT TYPE. IF POSSIBLE, GENERIC SELECTION WILL BE MADE ON THE BASIS OF THIS RESULT TYPE.

User Response: Provide fuller parameter description.

T IEM1642I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz IS NOT SUFFICIENT FOR THE PURPOSES OF GENERIC SELECTION.

System Action: Compilation is terminated.

User Response: Provide fuller parameter description.

T IEM1643I COMPILER ERROR. THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz IS INCORRECTLY FORMED IN THE DICTIONARY.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1644I COMPILER ERROR. THE GENERIC FAMILIES ASSOCIATED WITH ENTRY NAME zzzz HAVE BEEN INCORRECTLY FORMED IN THE DICTIONARY.

Explanation: The dictionary entry for one or more of the generic families is not a recognizable entry type.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1645I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz RESULTS IN MORE THAN ONE POSSIBLE FAMILY MEMBER SELECTION.

System Action: Compilation is terminated.

User Response: Provide fuller parameter description.

T IEM1648I COMPILER ERROR. FUNCTION REFERENCE MISSING FROM TEXT IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1649I COMPILER ERROR. INCORRECT FORMATION OF ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1650I COMPILER ERROR. INCORRECT HANDLING OF ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1651I COMPILER ERROR. ARGUMENT REFERENCE MISSING FROM ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1652I IMPLEMENTATION RESTRICTION. INVOCATIONS ARE NESTED BEYOND THE MAXIMUM PERMITTED LEVEL IN STATEMENT NUMBER xxx

Explanation: Nesting level exceeds implementation limit.

System Action: Compilation is terminated.

User Response: Reduce nesting level.

T IEM1653I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE. NUMBER OF SYMBOLIC REGISTERS EXCEEDS LIMIT.

Explanation: Too many symbolic registers required.

System Action: Compilation is terminated.

User Response: Subdivide the program.

T IEM1654I THE GENERIC PROCEDURE zzzz IS BEING INVOKED WITHOUT AN ARGUMENT LIST IN STATEMENT NUMBER xxx

System Action: Compilation is terminated.

User Response: Supply argument list.

T IEM1655I IMPLEMENTATION RESTRICTION. TOO MUCH WORKSPACE REQUIRED FOR TEMPORARY RESULTS IN STATEMENT NUMBER xxx

System Action: Compilation is terminated.

User Response: Subdivide the statement in question into two or more separate statements.

T IEM1656I COMPILER ERROR. INCORRECT INPUT TO PHASE MF FOR COMPLETION BUILT-IN FUNCTION IN STATEMENT NUMBER xxx.

Explanation: The compiler has encountered incorrect input to phase MF.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

E IEM1657I THE FILE zzzz, WHICH HAS BEEN DECLARED WITH THE COBOL OPTION, IS BEING PASSED AS AN ARGUMENT IN STATEMENT NUMBER xxx.

Explanation: In this compiler, files with the COBOL option may not be passed as arguments.

System Action: Comments and continues.

User Response: Correct source program if necessary.

S IEM1658I IN STATEMENT NUMBER xxx, zzzz IS NOT A PERMISSIBLE ARGUMENT. AN EXECUTION ERROR WILL RESULT IF THE CORRESPONDING PARAMETER IS REFERENCED.

Explanation: A condition name appears as an argument in a CALL statement or function reference. This is illegal.

System Action: Attempts to pass the argument.

T IEM1670I STATEMENT NUMBER xxx HAS CAUSED A TABLE INTERNAL TO THE COMPILER TO OVERFLOW.

Explanation: Either the nesting of procedure arguments requiring dummies is too deep, or too many temporary results are required between the assignment of an argument expression to a dummy and the procedure call.

System Action: Compilation is terminated.

User Response: Reduce complexity of argument expressions.

T IEM1671I COMPILER ERROR NUMBER MP nnnn IN STATEMENT NUMBER xxx

Explanation: This is a compiler error.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1680I COMPILER ERROR. TRIPLE OPERATOR NOT RECOGNIZED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM1687I COMPILER ERROR. OPTIMIZED SUBSCRIPT INCORRECTLY FORMED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system

manager or administrator of the error.

T IEM1688I COMPILER ERROR. ARRAY NAME zzzz INCORRECTLY DESCRIBED AS DEFINED IN STATEMENT NUMBER xxx

Explanation: Array incorrectly described by a previous phase as having the DEFINED attribute.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1689I COMPILER ERROR. ARRAY zzzz IS INCORRECTLY SUBSCRIPTED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM1692I IMPLEMENTATION RESTRICTION. SUBSCRIPT NESTED TO DEPTH GREATER THAN 50 LEVELS IN STATEMENT NUMBER xxx

Explanation: Subscript nesting exceeds fifty levels.

System Action: Compilation is terminated.

User Response: Reduce amount of nesting and recompile.

T IEM1693I NUMBER OF SUBSCRIPTS ASSOCIATED WITH ARRAY zzzz IN STATEMENT NUMBER xxx IS INCORRECT.

Explanation: The number of subscripts given does not agree with the declared dimensionality of the array.

System Action: Compilation is terminated.

User Response: Add or delete subscripts as appropriate.

W IEM1695I TRANSLATE FUNCTION IN STATEMENT NUMBER xxx HAS A CHARACTER OR BIT DUPLICATED IN ITS THIRD ARGUMENT.

Explanation: This may be a source program error.

User Response: Check that the character or bit was intentionally duplicated.

W IEM1696I VERIFY FUNCTION IN STATEMENT NUMBER xxx HAS A CHARACTER OR BIT DUPLICATED IN ITS SECOND ARGUMENT.

Explanation: This may be a source program error.

User Response: Check that the character or bit was intentionally duplicated.

S IEM1750I zzzz IS AN ILLEGAL OPERAND IN AN IF STATEMENT OR WHILE CLAUSE IN STATEMENT NUMBER xxx. IT HAS BEEN REPLACED BY A ZERO BIT STRING.

S IEM1751I THE IDENTIFIER zzzz IS AN ILLEGAL ARGUMENT OF THE RETURN STATEMENT NUMBER xxx AND HAS BEEN DELETED.

Explanation: Illegal arguments include arrays and structures.

W IEM1752I THE ATTRIBUTES OF THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx CONFLICT WITH THE ATTRIBUTES OF SOME OR ALL OF THE ENTRY POINTS OF THE CONTAINING PROCEDURE. AN EXECUTION FAILURE MAY OCCUR AT THIS STATEMENT.

Explanation: After a call to a procedure through an entry point with POINTER, AREA, or data attributes, any RETURN statement encountered must return a value of type POINTER or AREA or of a data type compatible with the data attributes of the entry point.

System Action: The ERROR condition is raised.

E IEM1753I THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx AND THE ATTRIBUTES OF THE CONTAINING PROCEDURE ARE IMCOMPATIBLE. EXECUTION OF THIS STATEMENT WILL RESULT IN A FAILURE.

Explanation: After a call to a procedure through an entry point with POINTER, AREA, or data attributes, any RETURN

statement encountered must return a value of type POINTER or AREA or of a data type compatible with the data attributes of the entry point.

System Action: The ERROR condition is raised.

E IEM1754I THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx IS INVALID

Explanation: The only permitted arguments are data types STRING, POINTER, and AREA.

System Action: Raise ERROR condition on execution of the statement.

E IEM1755I OPTION SPECIFICATION CONTAINS INVALID ARGUMENT, DEFAULT USED FOR SORMGIN.

Explanation: This message is written directly on SYSPRINT, or SYSOUT. The compiler found that an argument to the SORMGIN option was either zero or greater than 100.

System Action: The default interpretation for SORMGIN, as set at system generation, is used.

W IEM1790I DATA CONVERSIONS WILL BE DONE BY SUBROUTINE CALL IN THE FOLLOWING STATEMENTS yyyy

User Response: Check to see if the conversion can be avoided or performed in line.

S IEM1793I ILLEGAL ASSIGNMENT OR CONVERSION IN STATEMENT NUMBER xxx. EXECUTION WILL RAISE THE ERROR CONDITION.

Explanation: Illegal assignment or conversion in source statement, e.g., label to arithmetic.

System Action: An instruction is compiled which will cause termination if the statement is executed.

T IEM1794I COMPILER ERROR IN STATEMENT NUMBER xxx PHASE OE.

Explanation: Compiler error caused by input text in bad format.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM1795I INVALID ITEM IN FREE STATEMENT NUMBER xxx

Explanation: Variable in FREE statement is either not CONTROLLED or not at level 1.

System Action: Error condition and message given at object time.

E IEM1796I ASSIGNMENT OF AN ILLEGAL LABEL CONSTANT IN STATEMENT NUMBER xxx.

Explanation: The label constant does not appear in the value list in the DECLARE statement for the label variable.

System Action: Accepts label constant as if in value list and continues compilation.

W IEM1797I CONVERSION OF NULL VALUES IN POINTER/OFFSET ASSIGNMENTS IS INVALID. NULLO HAS BEEN REPLACED BY NULL, OR NULL BY NULLO, IN STATEMENT NUMBER xxx

Explanation: A NULLO offset type constant has been assigned to a pointer, or a NULL pointer type constant to an offset. Conversion of null values is not allowed. The constant type has been corrected.

System Action: The assignment is unaffected.

S IEM1800I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FLOATING-POINT. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

System Action: Truncates result.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1801I AN ERROR HAS OCCURRED IN THE CONVERSION TO FLOATING-POINT OF THE STERLING CONSTANT WHICH HAS

DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1802I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FIXED BINARY. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1803I AN ERROR HAS OCCURRED IN THE CONVERSION TO FIXED BINARY OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1804I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FIXED DECIMAL. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

System Action: Truncates result.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1805I AN ERROR HAS OCCURRED IN THE CONVERSION TO FIXED DECIMAL OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USE OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1806I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO DECIMAL NUMERIC FIELD. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1807I AN ERROR HAS OCCURRED IN THE CONVERSION TO DECIMAL NUMERIC FIELD OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1808I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO STERLING NUMERIC FIELD. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1809I AN ERROR HAS OCCURRED IN THE CONVERSION TO STERLING NUMERIC FIELD OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1810I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO BIT STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1811I AN ERROR HAS OCCURRED IN THE CONVERSION TO BIT STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1812I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO CHARACTER STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1813I AN ERROR HAS OCCURRED IN THE CONVERSION TO CHARACTER STRING OF THE STERLING CONSTANT WHICH HAS DECIMAI PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1814I AN ERROR HAS OCCURRED IN THE CONVERSION OF THE CONSTANT yyyy TO PICTURED CHARACTER STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1815I AN ERROR HAS OCCURRED IN THE CONVERSION TO PICTURED CHARACTER STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1816I zzzz USED IN FILE OPTION IN STATEMENT NUMBER xxx IS NOT A FILE. OPTION HAS BEEN IGNORED. EXECUTION ERROR WILL RESULT

Explanation: Dictionary reference of file triple was not file constant or file parameter code.

System Action: Ignores option, but continues to scan statement.

E IEM1817I INVALID KEYTO OPTION zzzz IGNORED IN STATEMENT NUMBER xxx

Explanation: KEYTO option must be scalar character string variable.

S IEM1818I zzzz USED IN KEY/KEYFROM OPTION IN STATEMENT NUMBER xxx IS NOT A SCALAR. OPTION IGNORED.

System Action: Ignores option but continues scan of statement.

S IEM1819I zzzz USED IN THE IGNORE OPTION IN STATEMENT NUMBER xxx IS NOT A SCALAR. OPTION IGNORED.

System Action: Ignores option but continues scan of statement.

User Response: Correct IGNORE variable.

T IEM1823I COMPILER ERROR DETECTED IN PHASE NJ/NK.

Explanation: NJ/NK found some unexpected input. Register 9 in dump will indicate cause of error.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM1824I OPTIONS IN OPEN STATEMENT NUMBER xxx ARE IN CONFLICT WITH PAGESIZE AND/OR LINESIZE.

E IEM1825I INVALID REPLY OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1826I INVALID MESSAGE IN DISPLAY STATEMENT NUMBER xxx. STATEMENT IGNORED.

S IEM1827I INVALID ARGUMENT TO DELAY STATEMENT NUMBER xxx. STATEMENT IGNORED.

T IEM1828I COMPILER ERROR. INCORRECT NUMBER OF TMPDS FOLLOWING ZERO OPERAND IN STATEMENT NUMBER xxx

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

S IEM1829I INVALID SCALAR EXPRESSION OPTION IN WAIT STATEMENT NUMBER xxx. MAXIMUM EVENT COUNT GIVEN.

Explanation: The optional scalar expression in the WAIT statement cannot be converted to an integer.

System Action: The number of event names in the list is assumed as the event count.

T IEM1830I COMPILER ERROR. INCORRECT INPUT TO PHASE NG IN WAIT STATEMENT NUMBER xxx.

Explanation: The compiler has encountered incorrect input to phase NG and cannot continue.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

E IEM1831I INVALID KEYTO OPTION IN STATEMENT NUMBER xxx.

Explanation: KEYTO option must be scalar character-string variable.

E IEM1832I INVALID PAGE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1833I INVALID LINE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1834I MULTIPLE COPY OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Only the first option is used.

E IEM1835I INVALID FILE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1836I INVALID STRING OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1837I NO FILE OR STRING SPECIFIED IN STATEMENT NUMBER xxx. STATEMENT IGNORED.

Explanation: No FILE or STRING given in GET/PUT statement.

E IEM1838I INVALID TITLE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1839I INVALID IDENT OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1840I INVALID LINESIZE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1841I INVALID PAGESIZE OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1843I NO FILE SPECIFIED IN OPEN/CLOSE STATEMENT NUMBER xxx. ANY OPTIONS ARE IGNORED.

T IEM1844I COMPILER ERROR. INCORRECT NUMBER OF TMPDS FOLLOWING ZERO OPERAND IN STATEMENT NUMBER xxx

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

E IEM1845I MULTIPLE DATA SPECIFICATIONS IGNORED IN STATEMENT NUMBER xxx

E IEM1846I INVALID SKIP OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1847I NO DATA SPECIFICATIONS GIVEN FOR GET STATEMENT NUMBER xxx. STATEMENT DELETED.

S IEM1848I NO DATA SPECIFICATIONS OR PRINT OPTIONS GIVEN FOR PUT STATEMENT NUMBER xxx. STATEMENT DELETED.

W IEM1849I THE USE OF THE BUILT-IN FUNCTION NULL IN STATEMENT NUMBER xxx IS INVALID; NULLO HAS BEEN SUBSTITUTED. CHECK ALL SIMILAR USES OF NULL.

System Action: Substitutes NULLO.

W IEM1850I THE USE OF THE BUILT-IN FUNCTION NULLO IN STATEMENT NUMBER xxx IS INVALID; NULL HAS BEEN SUBSTITUTED. CHECK ALL SIMILAR USES OF NULLO.

System Action: Substitutes NULL.

S IEM1860I THE ILLEGAL ITEM zzzz HAS BEEN DELETED FROM THE I/O DATA LIST IN STATEMENT NUMBER xxx

S IEM1861I AN ILLEGAL TEMPORARY RESULT OR SUBSCRIPTED ELEMENT HAS BEEN DELETED FROM THE I/O DATA LIST IN STATEMENT NUMBER xxx

S IEM1862I AN EXPRESSION OR FUNCTION INVOCATION IS AN ILLEGAL DATA ITEM AND HAS BEEN DELETED FROM

THE DATA-DIRECTED I/O STATEMENT NUMBER xxx.

E IEM1870I THE FORMAT LIST IN STATEMENT NUMBER xxx CONTAINS NO DATA FORMAT ITEMS AND WILL BE EXECUTED ONCE IF THE STATEMENT IS INVOKED.

System Action: At execution time, on finding no data format items, control passes out of the statement at the end of the format list.

S IEM1871I IN STATEMENT NUMBER xxx THE FORMAT LIST CONTAINS AN E OR F FORMAT ITEM WITH AN ILLEGAL SPECIFICATION. THE FORMAT ITEM HAS BEEN DELETED.

W IEM1872I IN STATEMENT NUMBER xxx AN E FORMAT ITEM HAS A FIELD WIDTH WHICH WOULD NOT PERMIT PRINTING OF A MINUS SIGN.

S IEM1873I IMPLEMENTATION RESTRICTION. IN STATEMENT NUMBER xxx AN A, B OR CONTROL FORMAT ITEM SPECIFIES AN EXCESSIVE LENGTH WHICH HAS BEEN REPLACED BY THE MAXIMUM OF 32,767.

S IEM1874I IN STATEMENT NUMBER xxx AN INPUT STATEMENT CONTAINS A FORMAT ITEM WHICH MAY BE USED ONLY IN OUTPUT STATEMENTS.

Explanation: PAGE, SKIP, LINE, COLUMN, and format items A and B with no width specification may be used only for output.

System Action: Invalid format item deleted.

W IEM1875I IN STATEMENT NUMBER xxx AN E FORMAT ITEM HAS AN ILLEGAL SPECIFICATION IF USED FOR AN OUTPUT DATA ITEM.

Explanation: The specification violates the restriction that the field width w must be greater than s+n+2.

System Action: There will be an error at execution time.

User Response: Correct specification.

T IEM2304I COMPILER ERROR. DICTIONARY ENTRY zzzz UNRECOGNIZED IN STATIC CHAIN.

Explanation: Due to a compiler error, a dictionary entry with an unrecognized code byte has been found in the static chain.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM2305I COMPILER ERROR. DOPE VECTOR REQUESTED BY NON-STRING, NON-STRUCTURE MEMBER zzzz

Explanation: Due to a compiler error, the allocation of a dope vector has been requested for an item which should never require one.

System Action: Compilation is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM2352I THE AUTOMATIC VARIABLES IN THE BLOCK HEADED BY STATEMENT NUMBER xxx ARE MUTUALLY DEPENDENT. STORAGE CANNOT BE ALLOCATED.

Explanation: A number of automatic variables are mutually dependent, which makes it impossible to allocate storage in order of dependency.

System Action: Compilation is terminated.

User Response: Rewrite statement, eliminating mutual dependency.

T IEM2650I IMPLEMENTATION RESTRICTION. OPTIMIZATION TABLE OVERFLOWED WHILE PROCESSING STATEMENT NUMBER xxx.

System Action: Compilation is terminated.

User Response: Recompile with one of the following changes:

1. Use a larger partition or region.
2. Specify OPT=0 or OPT=1.
3. Reduce the number of subscripts in the DO loop that contains the statement indicated.

T IEM2660I COMPILER ERROR IN INPUT TO PHASE RD. IN STATEMENT NUMBER xxx A PREVIOUS PHASE HAS GENERATED A LABEL NUMBER GREATER THAN THE MAXIMUM SHOWN.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2661I COMPILER ERROR. INTERNAL TABLE ENTRY IN PHASE RD IS INCORRECT.

Explanation: An entry in the internal table of compiler-generated labels does not point to a label in text.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error. The program may be recompiled successfully by omitting OPT=2 option.

T IEM2700I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. SPECIAL ASSIGNED REGISTER IN FORMAT/DATA LIST CODE CANNOT BE FOUND.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2701I COMPILER ERROR. PHASE IEMRF, STATEMENT NUMBER xxx. PSTOR GREATER THAN 32K.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2702I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. BCT WITHOUT DICTIONARY REFERENCE AS DESTINATION.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2703I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. LINK REGISTER IN BALR IS NOT ASSIGNED.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2704I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. 'USNG' ITEM DOES NOT HAVE ASSIGNED REGISTER.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2705I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. DROPPED REGISTER NOT ACTIVE.

Explanation: Register number in field in DROP item is not in register table nor in storage.

System Action: Continues compilation, ignoring DROP. Execution is inhibited.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2706I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. NOT ALL REGISTERS IN 'DRPL' ITEM CAN BE FOUND.

System Action: Ignores DRPL item and continues.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2707I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. NOT ALL SYMBOLIC REGISTERS DROPPED AT END OF PROCEDURE OR BEGIN BLOCK.

Explanation: One or more symbolic registers have been used in the PROCEDURE or BEGIN block, but no corresponding DROP has occurred.

System Action: Inserts in listing at end of block: the register number, the offset from register 9 at which the register is stored, and the words 'ERROR STOP'.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2708I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. ASSIGNED REGISTER USED IN SOURCE FIELD IS NOT INITIALIZED.

Explanation: The assigned register should have a previous value (e.g., X in AR X,Y, or L Y,10(X)), but none can be found.

System Action: Register 13 is used instead of the correct number, and compilation is continued.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2709I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. SYMBOLIC REGISTER SHOULD HAVE PREVIOUS VALUE, BUT HAS NOT.

Explanation: Register X in an instruction, such as AR X,Y, or L Y,10(X), has not been set up previously.

System Action: Inserts register 12 and continues compilation.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2710I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. MORE THAN ONE REGISTER PAIR REQUIRED IN AN INSTRUCTION.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2711I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. ASSIGNED REGISTER IS STILL IN USE AT THE START OF A PROCEDURE.

Explanation: Assigned register status should be zero at the start of each procedure.

System Action: Drops the assigned register and continues compilation.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2712I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. IPRM/IPRM' OR EPRM/EPRM' PAIRS ARE NOT MATCHED IN PREVIOUS STATEMENT.

System Action: Compilation is continued.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2816I ILLEGAL ENVIRONMENT OPTION IN STATEMENT NUMBER xxx.

System Action: Remainder of environment attributes ignored.

S IEM2817I COMPILER ERROR. INVALID ATTRIBUTE CODE IN STATEMENT NUMBER xxx.

Explanation: An invalid attribute marker has been found in the dictionary entry corresponding to the file attributes in the statement specified.

System Action: Ignores the rest of the entry.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2818I CONFLICTING ATTRIBUTE IN STATEMENT NUMBER xxx IGNORED.

Explanation: An attribute other than 'ENVIRONMENT' clashes with previously declared attributes in the specified statement.

System Action: Ignores this attribute.

S IEM2819I ERRONEOUS USE OF PARENTHESIS IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx.

Explanation: Misplaced parenthesis in ENVIRONMENT attribute.

System Action: Remainder of ENVIRONMENT attribute ignored.

S IEM2820I ERRONEOUS USE OF COMMA IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx.

Explanation: Misplaced comma in ENVIRONMENT attribute.

System Action: Remainder of ENVIRONMENT attribute ignored.

S IEM2821I ILLEGAL CHARACTER IN KEYWORD IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx.

Explanation: Invalid keyword in ENVIRONMENT attribute.

System Action: Remainder of ENVIRONMENT attribute ignored.

S IEM2822I FIELD TOO LARGE IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx.

Explanation: Field in item in ENVIRONMENT attribute too large.

System Action: Remainder of ENVIRONMENT attribute ignored.

S IEM2823I ERROR IN FORMAT OF ENVIRONMENT ATTRIBUTE IN STATEMENT NUMBER xxx.

Explanation: Format of item in ENVIRONMENT attribute incorrect.

System Action: Remainder of ENVIRONMENT attribute ignored.

S IEM2824I CONFLICT BETWEEN ENVIRONMENT ATTRIBUTE AND OTHER ATTRIBUTES IN STATEMENT NUMBER xxx.

Explanation: An option in the ENVIRONMENT attribute clashes with either another ENVIRONMENT option or with a declared attribute.

System Action: Remainder of ENVIRONMENT attribute ignored.

S IEM2825I CONFLICTING OPTIONS IN ENVIRONMENT ATTRIBUTE IN STATEMENT NUMBER xxx. REST OF ENVIRONMENT IGNORED.

System Action: DECLARE control block is constructed from attributes which have already been processed. The rest are ignored.

User Response: Correct ENVIRONMENT option.

S IEM2826I IMPLEMENTATION RESTRICTION. DIRECT FILE zzzz DECLARED IN STATEMENT NUMBER xxx MUST HAVE AN ORGANIZATION SUBFIELD IN THE ENVIRONMENT ATTRIBUTE.

System Action: No compile-time action, but execution will fail.

User Response: Provide ENVIRONMENT attribute.

W IEM2827I A D COMPILER OPTION HAS BEEN DECLARED IN THE ENVIRONMENT LIST IN STATEMENT NUMBER xxx. IT HAS BEEN IGNORED.

W IEM2828I ENVIRONMENT OPTIONS CLTASA AND CLT360 HAVE BOTH BEEN DECLARED IN STATEMENT NUMBER xxx. THE SECOND ONE LISTED WILL BE IGNORED.

W IEM2829I IN STATEMENT NUMBER xxx THE PARAMETER SPECIFIED IN THE INDEXAREA OPTION IS GREATER THAN 32767 AND HAS BEEN IGNORED.

Explanation: If the parameter is not specified or is outside the permitted range, data management uses as much main storage as is required for the master index.

T IEM2830I FILE DECLARED TRANSIENT IN STATEMENT NUMBER xxx DOES NOT HAVE MANDATORY ENVIRONMENT OPTION G OR R.

System Action: Compilation is terminated.

User Response: Correct file declaration.

E IEM2831I THE NCP VALUE IN STATEMENT NUMBER xxx EITHER IS NOT AN INTEGER CONSTANT OR LIES OUTSIDE THE PERMITTED RANGE OF 1 TO 99. A VALUE OF 1 HAS BEEN ASSUMED.

User Response: Correct the NCP value and recompile.

S IEM2833I COMPILER ERROR. OPERAND OF CL OR SL NOT LABEL.

Explanation: The dictionary entry referenced after a compiler label or statement label marker in the text is not in fact a label.

System Action: The label definition is ignored.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2834I COMPILER ERROR. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2835I COMPILER ERROR. SUBSCRIPTED LABEL CHAIN ERROR.

Explanation: Subscripted labels in the source program result in the creation of chains of dictionary entries. An error in the chaining has occurred.

System Action: The label definition is ignored.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2836I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: Not enough storage is available for the generated label number table created by this phase. The condition arises when a large number of such labels have been used, and this in turn is related to the size of the program.

System Action: Compilation is terminated.

User Response: Break the program into smaller modules.

T IEM2837I COMPILER ERROR. MULTIPLY DEFINED LABEL OR INVALID LABEL NUMBER.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2838I THE TELEPROCESSING FORMAT OPTION IN STATEMENT NUMBER xxx CONTAINS NO CONSTANT. A VALUE OF ZERO HAS BEEN ASSUMED.

S IEM2840I NUMERIC FIELD IN ENVIRONMENT OPTION FOR FILE zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT IN PARENTHESES AND HAS BEEN DELETED.

Explanation: If an environment option requires a numeric field, then that field must be enclosed in parentheses.

S IEM2852I COMPILER ERROR. NON-ZERO OFFSET IN PSEUDO-REGISTER REFERENCE.

Explanation: Use of a pseudo-register accompanied by literal offset has been called for by compiled code. This cannot be assembled owing to the manner in which pseudo-register relocation is performed.

System Action: The literal offset is ignored.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2853I COMPILER ERROR. REFERENCE TO INVALID DICTIONARY ENTRY.

Explanation: A dictionary reference in the input text does not correspond to a legal dictionary entry.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2854I COMPILER ERROR. INVALID DICTIONARY REFERENCE OFFSET.

Explanation: A dictionary reference in the input text corresponds to a valid dictionary entry, but the dictionary reference offset is not valid.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2855I COMPILER ERROR. REQUESTED OFFSET NOT ASSIGNED.

Explanation: Although implied by a dictionary reference in the text, storage has not been allocated.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2865I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS AND/OR CONTROLLED VARIABLES.

Explanation: The compiler allocates a pseudo-register entry for each block and CONTROLLED variable in the source program. The maximum number of such entries is 1,024.

System Action: No pseudo-registers are allocated for items after the limit has been reached.

User Response: Reduce number of blocks, or CONTROLLED variables, in program to less than 1025.

W IEM2866I THIS PL/I COMPILATION HAS GENERATED EXTERNAL NAMES IN WHICH THE FIRST LEADING CHARACTER OF THE EXTERNAL PROCEDURE NAME HAS BEEN REPLACED BY A SPECIAL CHARACTER.

Explanation: The external procedure name, with its first character changed, is being used as a base for generating names for External Symbol Dictionary entries. If the same thing happens in another compilation, and the two are then joined by the Linkage Editor, two External Symbol Dictionary entries may have the same name.

System Action: None

E IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME zzzz HAS BEEN TRUNCATED TO 7 CHARACTERS.

Explanation: External identifiers are restricted to seven characters.

System Action: Name of External Symbol Dictionary entry truncated by taking first four and last three characters; phase then carries on normally.

User Response: Shorten the name.

W IEM2868I THIS PL/I COMPILATION HAS GENERATED EXTERNAL NAMES IN WHICH THE SECOND LEADING CHARACTER OF THE EXTERNAL PROCEDURE NAME HAS BEEN REPLACED BY A SPECIAL CHARACTER.

Explanation: The external procedure, with its second character changed, is being used as a base for generating names for External Symbol Dictionary entries. If the same thing happens in another compilation, and the two are then joined by the Linkage Editor, two External Symbol Dictionary entries may have the same name.

System Action: None

T IEM2881I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2882I COMPILER ERROR IN STATEMENT NUMBER xxx. OPERAND OF DC CODE INVALID.

Explanation: The operand of a DCA4 pseudo-code item is not valid--the operand should always be relocatable.

System Action: An offset of zero is assembled into the text.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2883I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID REQUEST FOR RELOCATABLE TEXT.

Explanation: The operand of a branch instruction has been found to require relocation.

System Action: An offset of zero is assembled into the text.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2884I COMPILER ERROR IN STATEMENT NUMBER xxx. NON-ZERO OFFSET IN PSEUDO-REGISTER REFERENCE.

Explanation: Use of a pseudo-register accompanied by a literal offset has been called for by compiled code. This cannot be assembled owing to the manner in which pseudo-register relocation is performed.

System Action: The literal offset is ignored.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2885I COMPILER ERROR IN STATEMENT NUMBER xxx. REFERENCE TO INVALID DICTIONARY ENTRY.

Explanation: A dictionary reference in the input text does not correspond to a legal dictionary entry.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2886I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID DICTIONARY REFERENCE OFFSET.

Explanation: A dictionary reference in the text corresponds to a valid dictionary entry, but the dictionary reference offset is not valid.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2887I COMPILER ERROR IN STATEMENT NUMBER xxx. REQUESTED OFFSET NOT ASSIGNED.

Explanation: Although implied by a dictionary reference in the input text, storage has not been allocated.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM2888I COMPILER ERROR IN STATEMENT NUMBER xxx. UNDEFINED LABEL.

Explanation: No offset has been assigned to a label generated by the compiler.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM2897I IMPLEMENTATION RESTRICTION. QUALIFIED NAME zzzz LONGER THAN 256 CHARACTERS.

Explanation: The fully qualified name of the variable indicated will not fit into its Symbol Table entry.

System Action: Leaves Symbol Table entry incomplete and carries on with the initialization of the Static Internal control section.

User Response: Shorten the qualified name.

S IEM2898I DATA-DIRECTED 'GET/PUT' STATEMENT WITH NO LIST IN A PROCEDURE OR BEGIN BLOCK WHICH HAS NO DATA VARIABLES.

System Action: Zeros are inserted in the argument list for the call to the library routine to 'GET/PUT DATA', and compilation continues.

User Response: Correct GET/PUT statement.

W IEM2899I INITIALIZATION SPECIFIED FOR TOO FEW ELEMENTS IN STATIC ARRAY zzzz

System Action: Initialization terminated when end of initial string is found.

W IEM2900I INITIALIZATION SPECIFIED FOR TOO MANY ELEMENTS IN STATIC ARRAY zzzz

System Action: Initialization is terminated when every element has been initialized.

T IEM2913I COMPILER ERROR. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM3088I -3213I THE CONFLICTING ATTRIBUTE aaaa HAS BEEN IGNORED IN THE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx

Explanation: The attribute given in the message conflicts with another attribute declared for the same identifier, or is invalid for that identifier.

System Action: The attribute given in the message is ignored.

T IEM3216I COMPILER ERROR. UNABLE TO INITIATE ENTRY FOR PHASE zzz IN PHASE DIRECTORY. INITIALIZATION TERMINATED.

W IEM3218I CRITICAL ERROR FOUND DURING SYNTAX CHECK. CONTINUE? Y OR N.

W IEM3219I CRITICAL ERROR FOUND DURING SYNTAX CHECK IN CONVERSATIONAL TASK. USER REQUESTED COMPILATION TO CONTINUE.

Explanation: This message will appear only in the LISTDS or LISTOUT diagnostics. It will not appear at the terminal.

T IEM3220I CRITICAL ERROR FOUND DURING SYNTAX CHECK IN CONVERSATIONAL TASK. USER REQUESTED COMPILATION TO BE TERMINATED.

Explanation: This message will appear only in the LISTDS or LISTOUT diagnostics. It will not appear at the terminal.

T IEM3221I CRITICAL ERROR FOUND DURING SYNTAX CHECK. COMPILATION TERMINATED.

W IEM3222I CONFLICTING COMPILER OPTIONS NOLOAD AND DECK HAVE BEEN SPECIFIED. THE DECK OPTION HAS BEEN DELETED.

E IEM3584I AN UNBALANCED NUMBER OF PARENTHESES HAS BEEN DETECTED WITHIN A STATEMENT AT OR NEAR STATEMENT NUMBER xxx

Explanation: An occurrence of a comma immediately followed by a period at or near the given statement has been taken as a statement delimiter. The statement contains an unbalanced number of parentheses.

T IEM3839I COMPILER ERROR. INVALID ERROR MESSAGE CHAINS.

Explanation: Compiler is unable to print error diagnostics.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3842I COMPILER ERROR AT STATEMENT NUMBER xxx. ALL TEXT BLOCKS IN CORE ARE BUSY. BLOCK REFERENCED BY PHASE yyy CANNOT BE BROUGHT INTO CORE.

Explanation: All blocks in main storage have become busy. Compiler cannot continue since an external block cannot be read in.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3843I COMPILER ERROR AT STATEMENT NUMBER xxx. ATTEMPTED USE BY PHASE yyy OF ZDABRF WITH A BLOCK WHICH IS NOT IN CORE.

Explanation: Referenced block is not in main storage.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3844I IMPLEMENTATION RESTRICTION DICTIONARY ENTRY FOR AN ITEM IN STATEMENT NUMBER xxx IS TOO LONG FOR THIS ENVIRONMENT.

Explanation: The dictionary block size is too small to contain a required entry. The entry is probably a string constant, picture, or dope vector skeleton.

System Action: Compilation is terminated.

User Response: Increase the SIZE option, or reprogram to break down the excessively long constant, picture, structure, or array of varying strings.

T IEM3845I COMPILER ERROR AT STATEMENT NUMBER xxx. TEXT BLOCK REFERENCED BY PHASE yyy IS NOT IN CORE.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3846I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE BY STATEMENT NUMBER xxx. ALL TEXT BLOCKS FULL WHEN PHASE yyy IS BEING EXECUTED.

Explanation: There is no more space for text in this environment.

System Action: Compilation is terminated.

User Response:

1. Subdivide the program and recompile.
2. If OPT=2 has been specified for this compilation, recompile specifying OPT=1 or OPT=0.

Both possibilities may be tried together.

T IEM3847I COMPILER ERROR AT STATEMENT NUMBER xxx. PHASE yyy HAS REQUESTED MORE THAN 4K OF SCRATCH CORE.

Explanation: Request for main storage exceeds 4096 bytes.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3848I COMPILER ERROR AT STATEMENT NUMBER xxx. PHASE yyy HAS REQUESTED A RELEASE OF UNALLOCATED CORE.

Explanation: Attempt to release unallocated main storage.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3849I COMPILER ERROR. PHASE yy IN RELEASE LIST IS NOT IN PHASE DIRECTORY.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3850I COMPILER ERROR. PHASE yy IN LOAD LIST IS NOT IN PHASE DIRECTORY.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3851I COMPILER ERROR. PHASE yy NOT MARKED. IT IS LOADED.

Explanation: An unmarked phase is loaded.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3852I COMPILER ERROR AT STATEMENT NUMBER xxx. BLOCK REFERENCED BY PHASE yyy IS NOT IN USE. COMPILER CANNOT CONTINUE.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3853I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE. DICTIONARY IS FULL.

Explanation: This message is written directly on SYSPRINT, or SYSOUT.

System Action: Compilation is terminated.

User Response: Subdivide into more than one program and recompile.

T IEM3855I ERROR IN PHASE yy.

Explanation: This message is written directly on SYSPRINT, or SYSOUT. A compiler error has been discovered during the printing of compile-time diagnostic messages (if phase quoted in message is BM) or of source-program diagnostic messages (if phase is XA).

System Action: Compilation is terminated. Note that only the diagnostic message output is incomplete. All other output files have been generated satisfactorily.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3856I COMPILER ERROR. PROGRAM CHECK TYPE nnnn HAS OCCURRED IN PHASE yy AT OR NEAR STATEMENT NUMBER xxx.

Explanation: A program check has occurred during compilation This is due to a compiler failure which may have been exposed by an error in the source code. The check type is given as the code for a program interrupt as follows:

Code	Cause of Program Interrupt
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data

System Action: Compilation is terminated.

User Response: Check source code carefully. If an error is found, correcting it may enable compilation to be completed successfully. Whether or not an error is found, save

relevant data and inform the system manager or administrator of the error.

T IEM3857I COMPILER ERROR. ATTEMPT TO PASS CONTROL TO AN UNNAMED PHASE. AN UNMARKED PHASE HAS BEEN ENCOUNTERED.

Explanation: An unmarked phase has been encountered. Compiler cannot continue.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3858I COMPILER ERROR. REQUESTED OR UNWANTED PHASE NOT IN PHASE DIRECTORY.

Explanation: Request to mark a phase which is not in phase directory. Compiler cannot continue.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

T IEM3859I INSUFFICIENT CORE IS AVAILABLE TO CONTINUE THIS COMPILATION.

Explanation: An attempt is being made to expand the number of text blocks in main storage. The GETMAIN routine has failed to get the storage. This will only occur where less than 45,056 bytes are available to the compiler. This message is written directly onto SYSPRINT.

System Action: Compilation is terminated. Message IEM3865I may follow.

User Response: Check that the required main storage is available in the system on which the compilation is being run.

E IEM3860I I/O ERROR ON SYSIN. RECORD ACCEPTED AS INPUT.

Explanation: The error may be a machine error, or, if SYSIN is a card reader, there may be a hole pattern which does not represent a valid System/360 character (validity check).

System Action: The error message number is printed in the source listing before the record in error. The record is accepted as input.

User Response: If SYSIN is a card reader, check that every column of the indicated card contains a valid code. If the I/O error persists, have the computing system checked.

T IEM3861I I/O ERROR IN LOAD DATA SET. GENERATION OF LOAD DATA SET IS TERMINATED.

User Response: Attempt to recompile. If error persists, save relevant data and inform the system manager or administrator.

T IEM3865I ERROR IN COMPILER ABORT

Explanation: This message is written directly on SYSPRINT (SYSOUT). The compiler has tried twice to abort and cannot do so. Compilation will therefore terminate without the production of any further diagnostic messages.

System Action: Compilation is terminated.

User Response: If this message follows directly after message IEM3859I, correct the error which gave rise to that message; otherwise, save relevant data and inform the system manager or administrator of the error.

T IEM3872I I/O ERROR ON SYSUT3

Explanation: This message is written directly on SYSPRINT, or SYSOUT. There is an I/O error on SYSUT3 (PLIMAC). The compiler cannot continue.

System Action: Compilation is terminated.

User Response: Recompile, and if I/O error persists, inform the system manager or administrator.

E IEM3873I I/O ERROR ON PLIMAC. RECORD ACCEPTED AS INPUT.

System Action: The error message number is printed in the source listing before the record in error. The record is accepted as input, and compilation continues.

User Response: Recompile, and if I/O error persists, inform the system manager or administrator.

T IEM3874I UNABLE TO OPEN SYSIN

Explanation: This message is written directly on SYSPRINT (SYSOUT). Unable to open SYSIN (PLIINPUT). The compiler cannot continue.

System Action: Compilation is terminated.

User Response: Recompile, and if error persists, inform the system manager or administrator.

T IEM3878I INSUFFICIENT VIRTUAL MEMORY. COMPILATION TERMINATED

S IEM3889I LOAD DATA SET BLOCKSIZE NOT A MULTIPLE OF 80. THE LOAD OPTION HAS BEEN DELETED.

Explanation: On opening the load data set, the blocksize definition, either in the DDEF statement or in the data set label, was not a multiple of 80.

System Action: The LOAD option is deleted.

User Response: Correct the blocksize definition and recompile.

W IEM3890I NO RECFM GIVEN FOR SYSIN. U TYPE RECORDS ARE ASSUMED.

Explanation: No RECFM definition has been found in the DCB parameter of the SYSIN DDEF statement or the data set label.

System Action: Compilation proceeds assuming U type records. U format will be specified in the data set label when SYSIN is closed.

User Response: Check RECFM definition in SYSIN DDEF statement and rerun if necessary.

W IEM3898I COMPILER CORE REQUIREMENT EXCEEDED SIZE GIVEN. AUXILIARY STORAGE USED.

System Action: SPILL file opened.

W IEM3899I A BLOCK FOR OVERFLOW DICTIONARY ENTRY OFFSETS WAS CREATED DURING COMPILER PHASE yy

Explanation: This message occurs only in compilations run with the extended dictionary option. An entry offset table in a dictionary block became full before the entries filled the block.

Sytem Action: The block is created to hold the entry offsets overflowing from any entry offset tables during this compilation.

E IEM3900I ERROR IN PROCESS STATEMENT.

Explanation: This message is written directly on SYSPRINT (SYSOUT). The syntax of the PROCESS statement is incorrect.

System Action: An attempt is made to interpret the statement correctly. Actual results will depend on the nature of the syntax error.

User Response: Check that the options required have been correctly applied. If not, and recompilation is necessary, correct the syntax of the PROCESS statement.

E IEM3901I ERROR IN PROCESS STATEMENT. DEFAULT OPTIONS ASSUMED.

Explanation: This message is written directly on SYSPRINT (SYSOUT). Invalid syntax in the PROCESS statement has rendered the options unrecognizable.

System Action: The installation defaults are assumed for all options.

User Response: If the use of installation default options is unsatisfactory, correct the syntax of the PROCESS statement and recompile.

E IEM3902I OBJNM FIELD TOO LARGE. FIRST EIGHT CHARACTERS OF NAME HAVE BEEN USED.

Explanation: The name specified in the OBJNM option may not have more than eight characters.

System Action: First eight characters of name used.

User Response: Either amend object module name as required, or alter other references to object module to correspond with truncated name.

W IEM3903I CARRIAGE CONTROL POSITION LIES WITHIN THE SOURCE MARGIN. IT HAS BEEN IGNORED.

User Response: Recompile with carriage control position outside source margin.

E IEM3904I THE FOLLOWING STRING NOT IDENTIFIED AS A KEYWORD - yyyy

Explanation: This message is written directly on SYSPRINT (SYSOUT). The compiler was processing the option list passed to it as an invocation parameter, when it found a character string that it could not identify as a keyword.

System Action: The offending character string is ignored.

User Response: Correct the erroneous parameter and recompile.

E IEM3905I THE FOLLOWING KEYWORD DELETED, DEFAULT USED FOR - yyyy

Explanation: This message is written directly on SYSPRINT (SYSOUT). The compiler was processing the option list passed to it as an invocation parameter, when it found an option keyword that had been deleted at system generation.

System Action: The keyword passed at invocation time is ignored. The default interpretation for the option, as set at system generation, is used.

User Response: None, unless it is required to reinstate the deleted keyword, in which case it is necessary to generate the required version of the compiler with a system generation run.

E IEM3906I OPTION SPECIFICATION CONTAINS INVALID WYNTAX, DEFAULT USED FOR - yyyy

Explanation: This message is written directly on SYSPRINT (SYSOUT). The compiler was processing the option list passed to it as an invocation parameter, when it found that a sub-parameter, associated with the keyword given in the diagnostic message, was incorrectly specified.

System Action: The keyword passed at invocation time is ignored. The default interpretation for the option, as set at system generation, is used.

User Response: Correct the erroneous parameter and recompile.

E IEM3907I THE FOLLOWING NAME IGNORED AS IT DOES NOT APPEAR IN THE PHASE DIRECTORY - yy

Explanation: This message is written directly on SYSPRINT (SYSOUT). The two characters given in the message were used as parameters to the DUMP option. This usage is incorrect since the characters do not represent the name of a compiler phase.

System Action: The processing of the DUMP option continues, unless the two characters were used to indicate the first phase of an inclusive phase dump, in which case the scan of the DUMP option is terminated.

User Response: Correct the erroneous parameter and recompile.

S IEM3908I SYNTAX ERROR IN DUMP OPTION SPECIFICATION

Explanation: This message is written directly on SYSPRINT (SYSOUT). Incorrect use of delimiters in the specification of the DUMP option parameters.

System Action: Processing of DUMP option is terminated.

User Response: Correct the erroneous specification and recompile.

T IEM3909I EXTENDED DICTIONARY CAPACITY EXCEEDED. COMPILATION TERMINATED.

Explanation: This message occurs only in compilations run with the extended dictionary option. The block created to hold overflow dictionary entry offsets is full.

System Action: Compilation is terminated.

User Response: Subdivide program and recompile.

T IEM3910I SYSPRINT BLOCKSIZE IS TOO LARGE.

Explanation: The buffer area allowed is smaller than that required by the specified blocksize.

System Action: Compilation is terminated.

User Response: Use smaller blocksize.

T IEM3912I SYSIN BLOCKSIZE IS TOO LARGE.

Explanation: The buffer area allowed is smaller than that required by the buffers for SYSIN, or for SYSIN and SYSPRINT (SYSOUT) together.

System Action: Compilation is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

S IEM3914I LOAD DATA SET BLOCKSIZE IS TOO LARGE. THE LOAD OPTION HAS BEEN DELETED.

Explanation: The buffer area allowed is smaller than that required by the specified load data set blocksize.

System Action: The LOAD option is deleted.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM3915I THE CONFLICTING COMPILER OPTIONS MACDCK AND NOMACRO HAVE BEEN SPECIFIED. THE MACDCK OPTION HAS BEEN DELETED.

E IEM3916I THE CONFLICTING COMPILER OPTIONS DECK AND NOCOMP HAVE BEEN SPECIFIED. THE DECK OPTION HAS BEEN DELETED.

E IEM3917I THE CONFLICTING COMPILER OPTIONS LOAD AND NOCOMP HAVE BEEN SPECIFIED. THE LOAD OPTION HAS BEEN DELETED.

Compile-Time Diagnostic Messages

Note: The record number in the messages refers to the record in which the error was found. The incorrect statement may have commenced in an earlier record.

S IEM4106I UNEXPECTED END-OF-FILE IN STRING AT OR BEYOND RECORD NUMBER xxx. A STRING DELIMITER HAS BEEN INSERTED.

Explanation: End-of-file encountered while scanning for closing quote of a string constant.

System Action: Closing quote inserted before end-of-file.

T IEM4109I REPLACEMENT VALUE IN RECORD NUMBER xxx CONTAINS UNDELIMITED STRING. PROCESSING TERMINATED.

Explanation: End-of-string delimiter cannot be found in a replacement value.

E IEM4112I ILLEGAL CHARACTER IN APPARENT BIT STRING IN RECORD NUMBER xxx. STRING TREATED AS A CHARACTER STRING.

S IEM4115I UNEXPECTED END-OF-FILE IN COMMENT AT OR BEYOND RECORD NUMBER xxx. A COMMENT DELIMITER HAS BEEN INSERTED.

Explanation: End-of-file encountered while scanning for end-of-comment delimiter.

T IEM4118I REPLACEMENT VALUE IN RECORD NUMBER xxx CONTAINS UNDELIMITED COMMENT. PROCESSING TERMINATED.

Explanation: End-of-comment delimiter cannot be found in a replacement value.

E IEM4121I INVALID CHARACTER HAS BEEN REPLACED BY BLANK IN OR FOLLOWING RECORD NUMBER xxx

Explanation: Invalid character found in source text.

T IEM4124I COMPILER ERROR. PUSH DOWN STACK OUT OF PHASE

System Action: Processing is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

T IEM4130I UNDECLARED IDENTIFIER zzzz REFERENCED AT RECORD NUMBER xxx. PROCESSING TERMINATED.

Explanation: An attempt is made to execute a statement which references an identifier for which a DECLARE statement has not been executed.

E IEM4133I % ENCOUNTERED IN LABELLIST OF STATEMENT IN RECORD NUMBER xxx. IT HAS BEEN IGNORED.

User Response: Remove % from label list.

E IEM4134I UNEXPECTED COLON WITHOUT PRECEDING LABEL IN RECORD NUMBER xxx. COLON HAS BEEN IGNORED.

S IEM4136I STATEMENT TYPE NOT RECOGNIZABLE IN RECORD NUMBER xxx. STATEMENT DELETED.

E IEM4139I PREVIOUS USAGE OF IDENTIFIER zzzz CONFLICTS WITH USE AS LABEL IN RECORD NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless an attempt is made to execute a statement which references the ill-defined identifier.

E IEM4142I LABEL zzzz IN RECORD NUMBER xxx MULTIPLY DEFINED. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement which references the multiply defined label is executed.

W IEM4143I LABELS BEFORE DECLARE STATEMENT IN RECORD NUMBER xxx ARE IGNORED.

E IEM4148I IDENTIFIER zzzz IN RECORD NUMBER xxx USED WITH CONFLICTING ATTRIBUTES. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: Usage of identifier conflicts with a previous usage or declaration. If the record number refers to a procedure END statement, the error occurred within the procedure.

System Action: No action unless a statement is executed which references the identifier in error.

E IEM4150I FORMAL PARAMETER zzzz WAS NOT DECLARED IN PROCEDURE ENDING IN RECORD NUMBER xxx. TYPE CHARACTER HAS BEEN FORCED.

E IEM4151I LABEL zzzz IS NOT DEFINED. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the label.

E IEM4152I END OF FILE OCCURS BEFORE END FOR CURRENT PROCEDURE OR DO. END HAS BEEN INSERTED AT RECORD NUMBER xxx.

E IEM4153I LABEL zzzz IS UNDEFINED IN THE PROCEDURE ENDING IN RECORD NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: Label may have been defined outside of procedure, but transfers out of procedures are not allowed.

System Action: Any reference to the label in the procedure will terminate processing.

E IEM4154I SEMICOLON TERMINATES IF EXPRESSION IN RECORD NUMBER xxx. SEMICOLON HAS BEEN IGNORED.

S IEM4157I NEITHER % NOR THEN FOLLOWS IF EXPRESSION IN RECORD NUMBER xxx. IF STATEMENT DELETED.

E IEM4160I % MISSING BEFORE THEN OF IF STATEMENT IN RECORD NUMBER xxx. % HAS BEEN INSERTED.

E IEM4163I THEN MISSING FOLLOWING % IN IF STATEMENT IN RECORD NUMBER xxx. A THEN HAS BEEN INSERTED.

E IEM4166I COMPILE TIME STATEMENT MUST FOLLOW THEN OR ELSE IN RECORD NUMBER xxx. A % HAS BEEN INSERTED IN FRONT OF STATEMENT.

Explanation: % does not precede the first sstatement in the THEN or ELSE clause of an IF statement.

User Response: If the statement in question is meant to be a non-compile time statement, it should be put inside of a "% DO" group.

E IEM4169I THEN MISSING FORM IF STATEMENT AT RECORD NUMBER xxx IN A COMPILE TIME PROCEDURE. A THEN HAS BEEN INSERTED.

E IEM4172I THE % IN RECORD NUMBER xxx IS NOT ALLOWED IN COMPILE TIME PROCEDURES. IT HAS BEEN IGNORED.

W IEM4175I LABELS BEFORE ELSE IN RECORD NUMBER xxx HAVE BEEN IGNORED.

Explanation: Labels found preceding an ELSE statement.

S IEM4176I NO STATEMENT FOLLOWS THEN OR ELSE IN RECORD NUMBER xxx. A NULL STATEMENT HAS BEEN INSERTED.

E IEM4178I ELSE WITHOUT PRECEDING IF IN RECORD NUMBER xxx HAS BEEN IGNORED.

S IEM4184I ASSIGNMENT STATEMENT IN RECORD NUMBER xxx MUST END WITH SEMICOLON. TEXT DELETED TILL SEMICOLON IS FOUND.

E IEM4187I LABEL MISSING FROM PROCEDURE STATEMENT IN RECORD NUMBER xxx. A DUMMY LABEL HAS BEEN INSERTED.

T IEM4188I IMPLEMENTATION RESTRICTION. NO MORE THAN 254 COMPILE-TIME PROCEDURES MAY BE DEFINED IN A COMPILATION. PROCESSING TERMINATED.

User Response: Delete excess procedures.

E IEM4190I LABEL zzzz ON PROCEDURE IN RECORD NUMBER xxx IS PREVIOUSLY DEFINED. ANY REFERENCE TO IT WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the multiply defined label.

E IEM4193I ILLEGAL USE OF FUNCTION NAME zzzz ON LEFT HAND SIDE OF EQUALS SYMBOL. ANY REFERENCE WILL TERMINATE PROCESSING.

E IEM4196I PREVIOUS USE OF IDENTIFIER zzzz CONFLICTS WITH USE AS ENTRY NAME IN RECORD NUMBER xxx.. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the erroneous identifier.

S IEM4199I FORMAL PARAMETER zzzz IS REPEATED IN PARAMETER LIST IN RECORD NUMBER xxx. THE SECOND OCCURRENCE HAS BEEN REPLACED BY A DUMMY PARAMETER.

S IEM4202I IMPLEMENTATION RESTRICTION. MORE THAN 15 PARAMETERS OCCUR IN RECORD NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: Processing is terminated if an attempt is made to execute a statement which references the procedure that has more than 15 parameters.

E IEM4205I FORMAL PARAMETER MISSING IN RECORD NUMBER xxx. A DUMMY HAS BEEN INSERTED.

E IEM4208I UNRECOGNIZABLE PARAMETER yyyy IN RECORD NUMBER xxx. IT HAS BEEN REPLACED BY A DUMMY PARAMETER.

S IEM4211I PARAMETER IN RECORD NUMBER xxx NOT FOLLOWED BY COMMA OR PARENTHESIS. TEXT DELETED TO NEXT COMMA OR END OF STATEMENT.

S IEM4212I UNEXPECTED END OF PROCEDURE STATEMENT IN RECORD NUMBER xxx. RIGHT PARENTHESIS INSERTED.

Explanation: A semicolon was encountered during scan of an apparent parameter list.

System Action: A right parenthesis in inserted before the semicolon and processing continues.

E IEM4214I ILLEGAL FORM OF RETURNS OPTION IN RECORD NUMBER xxx. RETURNS (CHAR) HAS BEEN ASSUMED.

Explanation: RETURNS option should be of the form RETURNS(CHAR|FIXED).

E IEM4215I DATA ATTRIBUTE IN PROCEDURE STATEMENT IN RECORD NUMBER xxx IS NOT PARENTHESIZED AND IS NOT PRECEDED BY RETURNS. RETURNS AND PARENTHESES HAVE BEEN ASSUMED.

E IEM4216I THERE IS NO RIGHT PARENTHESIS FOLLOWING THE DATA ATTRIBUTE OF THE RETURNED VALUE IN RECORD NUMBER xxx. ONE HAS BEEN ASSUMED.

E IEM4217I NO ATTRIBUTE FOR RETURNED VALUE IN RECORD NUMBER xxx. CHARACTER ATTRIBUTE IS USED.

System Action: CHARACTER attribute is assigned.

S IEM4220I SEMICOLON NOT FOUND WHERE EXPECTED IN PROCEDURE STATEMENT IN RECORD NUMBER xxx. TEXT DELETED UP TO NEXT SEMICOLON.

E IEM4223I ENTRY ATTRIBUTE AND PROCEDURE STATEMENT FOR ENTRY zzzz DISAGREE ON THE NUMBER OF PARAMETERS. THE LATTER IS USED.

System Action: The number of parameters specified in the PROCEDURE statement is used.

E IEM4226I RETURNS ATTRIBUTE AND PROCEDURE STATEMENT FOR ENTRY zzzz DISAGREE ON ATTRIBUTE OF RETURNED VALUE.

System Action: The returned value will first be converted to the type on the procedure statement and will then be converted to the type given in the RETURNS attribute. A third conversion can occur if the type given in the returns attribute does not agree with the type required where the result is used.

S IEM4229I PROCEDURE STATEMENT AT RECORD NUMBER xxx MAY NOT BE USED WITHIN A PROCEDURE. PROCEDURE HAS BEEN DELETED.

Explanation: Compile-time precedures may not be nested.

System Action: Text is deleted up to and including the first % END following the erroneous PROCEDURE statement.

S IEM4232I PROCEDURE STATEMENT AT RECORD NUMBER xxx MAY NOT FOLLOW THEN OR ELSE. PROCEDURE HAS BEEN REPLACED BY A NULL STATEMENT.

Explanation: A PROCEDURE statement may appear in a THEN or ELSE clause only if it is inside a compile-time DO group.

S IEM4235I RETURN STATEMENT IN RECORD NUMBER xxx IS NOT ALLOWED OUTSIDE OF COMPILE-TIME PROCEDURE. STATEMENT DELETED.

E IEM4238I RETURNED VALUE MUST BE PARENTHESIZED IN RECORD NUMBER xxx. PARENTHESIS INSERTED.

E IEM4241I RETURNS EXPRESSION IN RECORD NUMBER xxx DOES NOT END RETURN STATEMENT. REMAINDER OF STATEMENT HAS BEEN IGNORED.

S IEM4244I GOTO IN RECORD NUMBER xxx IS NOT FOLLOWED BY LABEL. STATEMENT DELETED.

E IEM4247I PREVIOUS USE OF IDENTIFIER zzzz CONFLICTS WITH USE AS OBJECT OF GOTO IN RECORD NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the erroneous identifier.

S IEM4248I SEMICOLON NOT FOUND WHERE EXPECTED IN GOTO STATEMENT IN RECORD NUMBER xxx. TEXT DELETED UP TO NEXT SEMICOLON.

T IEM4250I GOTO zzzz IN RECORD NUMBER xxx TRANSFERS CONTROL INTO INTERATIVE DO OR ENCLOSED INCLUDED TEXT. PROCESSING TERMINATED.

S IEM4253I ACTIVATE OR DEACTIVATE IN RECORD NUMBER xxx NOT ALLOWED IN A COMPILE-TIME PROCEDURE. STATEMENT DELETED.

E IEM4254I EMPTY ACTIVATE OR DEACTIVATE STATEMENT IN RECORD NUMBER xxx. STATEMENT DELETED.

E IEM4256I SURPLUS COMMA IN ACTIVATE OR DEACTIVATE IN RECORD NUMBER xxx. THE COMMA HAS BEEN DELETED.

S IEM4259I UNRECOGNIZABLE FIELD IN ACTIVATE OR DEACTIVATE STATEMENT IN RECORD NUMBER xxx. THE FIELD HAS BEEN DELETED.

S IEM4262I ONLY PROCEDURES OR VARIABLES MAY HAVE ACTIVITY CHANGED. IDENTIFIER zzzz IN RECORD NUMBER xxx HAS BEEN DELETED FROM STATEMENT.

S IEM4265I COMMA MUST SEPARATE FIELDS OF ACTIVATE AND DEACTIVATE STATEMENTS. IN RECORD NUMBER xxx TEXT AFTER IDENTIFIER yyyy HAS BEEN DELETED UP TO NEXT COMMA.

S IEM4271I INVALID SYNTAX IN DO STATEMENT IN RECORD NUMBER xxx. IT HAS BEEN CONVERTED TO A GROUPING DO.

W IEM4277I NO MAXIMUM VALUE WAS SPECIFIED IN ITERATIVE DO IN RECORD NUMBER xxx. PROGRAM WILL LOOP UNLESS ALTERNATE EXIT IS PROVIDED.

E IEM4280I UNEXPECTED % IN RECORD NUMBER xxx TREATED AS HAVING BEEN PRECEDED BY SEMICOLON.

E IEM4283I MULTIPLE TO'S HAVE OCCURRED IN DO STATEMENT IN RECORD NUMBER xxx. SECOND 'TO' HAS BEEN CHANGED TO 'BY'.

E IEM4286I MULTIPLE BY'S HAVE OCCURRED IN DO STATEMENT IN RECORD NUMBER xxx. SECOND 'BY' HAS BEEN CHANGED TO 'TO'.

E IEM4289I DO STATEMENT IN RECORD NUMBER xxx SHOULD END WITH SEMICOLON. TEXT TO SEMICOLON DELETED.

E IEM4292I END STATEMENT AT RECORD NUMBER xxx MAY NOT FOLLOW THEN OR ELSE. A NULL STATEMENT HAS BEEN INSERTED BEFORE THE END STATEMENT.

E IEM4295I SEMICOLON NOT FOUND WHERE EXPECTED IN END STATEMENT IN RECORD NUMBER xxx. TEXT DELETED UP TO SEMICOLON.

E IEM4296I END STATEMENT IN RECORD NUMBER xxx NOT PRECEDED BY DO OR PROCEDURE STATEMENT. END HAS BEEN DELETED.

Explanation: An END statement has been encountered which is not preceded by a DO or PROCEDURE statement that has not already been terminated.

E IEM4298I LABEL REFERENCED ON END STATEMENT IN RECORD NUMBER xxx CANNOT BE FOUND. END TREATED AS HAVING NO OPERAND.

Explanation: The label cannot be found on a DO or PROCEDURE statement that has not already been terminated.

E IEM4299I END STATEMENT ENDING PROCEDURE IN RECORD NUMBER xxx DID NOT HAVE A PRECEDING PERCENT. A PERCENT IS INSERTED.

Explanation: The END statement referred to in this message is logical end of the procedure.

E IEM4301I IDENTIFIER zzzz ON END STATEMENT IN RECORD NUMBER xxx IS NOT A LABEL. END TREATED AS HAVING NO OPERAND.

E IEM4304I PROCEDURE zzzz DID NOT INCLUDE A RETURN STATEMENT.

Explanation: Language syntax requires use of RETURN statement in a procedure.

System Action: A null value is returned if the procedure is invoked.

S IEM4307I INCLUDE STATEMENT AT RECORD NUMBER xxx IS NOT ALLOWED IN COMPILE-TIME PROCEDURES. STATEMENT DELETED.

E IEM4310I IMPLEMENTATION RESTRICTION. DDNAME IN RECORD NUMBER xxx HAS BEEN TRUNCATED TO 8 CHARACTERS.

Explanation: The first of a pair of data set identifiers in an INCLUDE statement is a DDNAME and as such is limited to a maximum of 8 characters.

S IEM4313I UNRECOGNIZABLE FIELD IN INCLUDE STATEMENT AT RECORD NUMBER xxx. FIELD HAS BEEN DELETED.

System Action: Text is deleted up to next comma or semicolon.

S IEM4319I EMPTY INCLUDE STATEMENT IN RECORD NUMBER xxx. STATEMENT DELETED.

Explanation: At least one identifier must appear in an INCLUDE statement, that is, the data set member name.

E IEM4322I IMPLEMENTATION RESTRICTION. MEMBER NAME IN RECORD NUMBER xxx HAS BEEN TRUNCATED TO 8 CHARACTERS.

System Action: Uses first 8 characters of member name.

User Response: Correct data set member name in INCLUDE statement.

E IEM4325I RIGHT PARENTHESIS INSERTED AFTER MEMBER NAME IN RECORD NUMBER xxx.

E IEM4326I LEFT PARENTHESIS INSERTED BEFORE MEMBER NAME IN RECORD NUMBER xxx.

T IEM4328I COMPILER ERROR. DICTIONARY INFORMATION INCORRECT.

Explanation: A name containing an invalid character is found in the dictionary.

System Action: Processing is terminated.

User Response: Save relevant data and inform system manager or administrator of the error.

S IEM4331I DECLARE STATEMENT IN RECORD NUMBER xxx IS ILLEGAL AFTER THEN OR ELSE. STATEMENT DELETED.

User Response: Correct program. A DECLARE statement can appear in the THEN or ELSE clause of an IF statement if it is inside a DO group.

E IEM4332I EMPTY DECLARE STATEMENT IN RECORD NUMBER xxx. STATEMENT DELETED.

S IEM4334I IMPLEMENTATION RESTRICTION. FACTORING IN DECLARE STATEMENT IN RECORD NUMBER xxx EXCEEDS 3 LEVELS. REMAINDER OF STATEMENT DELETED.

User Response: Reduce level of factoring in DECLARE statement.

E IEM4337I SURPLUS COMMA HAS BEEN FOUND IN DECLARE STATEMENT IN RECORD NUMBER xxx. THIS COMMA HAS BEEN DELETED.

E IEM4340I IDENTIFIER MISSING WHERE EXPECTED IN RECORD NUMBER xxx. A DUMMY IDENTIFIER HAS BEEN INSERTED.

E IEM4343I IDENTIFIER zzzz IN RECORD NUMBER xxx HAS MULTIPLE DECLARATIONS. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: An identifier may be declared only once.

System Action: No action unless a statement is executed which references the multiply declared identifier.

S IEM4346I UNRECOGNIZABLE SYNTAX IN DECLARE STATEMENT IN RECORD NUMBER xxx. STATEMENT DELETED.

E IEM4349I LABEL zzzz CANNOT BE DECLARED IN RECORD NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: An attempt has been made to declare an identifier which has already been used as a label.

System Action: No action unless a statement is executed which references the declared label.

E IEM4352I EXTRA PARENTHESIS DELETED IN RECORD NUMBER xxx.

E IEM4355I ILLEGAL ATTRIBUTE yyyy IN RECORD NUMBER xxx. ATTRIBUTE HAS BEEN DELETED.

Explanation: Legal attributes are FIXED, CHARACTER, ENTRY, and RETURNS.

System Action: The illegal attribute is deleted.

E IEM4358I CLOSING RIGHT PARENTHESIS INSERTED IN RECORD NUMBER xxx.

E IEM4361I RETURNS ATTRIBUTE OCCURRED WITHOUT ENTRY ATTRIBUTE FOR PROCEDURE zzzz IN DECLARE STATEMENT AT OR BEFORE RECORD NUMBER xxx.

Explanation: Both ENTRY and RETURNS attributes must be declared for a compile-time procedure name.

System Action: The identifier is treated as an ENTRY name. If it is referenced, the arguments will be converted to the types declared for the procedure parameters.

E IEM4364I NO ATTRIBUTES WERE DECLARED FOR IDENTIFIER zzzz IN DECLARE STATEMENT AT OR BEFORE RECORD NUMBER xxx. CHARACTER HAS BEEN ASSIGNED.

E IEM4367I RETURNS ATTRIBUTE NOT GIVEN FOR ENTRY NAME zzzz IN DECLARE

STATEMENT AT OR BEFORE RECORD NUMBER xxx.

Explanation: Both ENTRY and RETURNS attributes must be declared for a compile-time procedure name.

System Action: The attribute of the returned value is determined by the relevant PROCEDURE statement.

E IEM4370I ENTRY ATTRIBUTE DISAGREES WITH DECLARATION FOR FORMAL PARAMETER zzzz. THE LATTER HAS BEEN USED.

Explanation: An ENTRY attribute in a DECLARE statement does not agree with the parameter attributes declared in the procedure.

System Action: If the relevant procedure is referenced, the argument will be converted to the type declared for the formal parameter.

E IEM4373I RETURNS ATTRIBUTE IN RECORD NUMBER xxx MUST BE PARENTHESIZED. PARENTHESIS INSERTED.

E IEM4376I ONLY FIXED OR CHARACTER ARE ALLOWED IN RETURNS ATTRIBUTE IN RECORD NUMBER xxx. ATTRIBUTE IGNORED.

Explanation: An illegal attribute was found.

System Action: The attribute of the returned value is determined by the relevant PROCEDURE statement.

E IEM4379I ATTRIBUTE yyyy IS ILLEGAL IN ENTRY ATTRIBUTE IN RECORD NUMBER xxx. NO CONVERSION WILL BE DONE.

Explanation: An invalid attribute was found.

System Action: No conversion to an ENTRY attribute will be carried out. However, if the relevant procedure is referenced, arguments will be converted to the types declared for the procedure parameters.

E IEM4382I ATTRIBUTE CONFLICTS WITH PREVIOUS ATTRIBUTE FOR IDENTIFIER zzzz IN RECORD NUMBER xxx. ATTRIBUTE IGNORED.

E IEM4383I PREVIOUS USAGE OF IDENTIFIER zzzz CONFLICTS WITH ATTRIBUTE IN RECORD NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

E IEM4391I OPERAND MISSING IN RECORD NUMBER xxx. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

S IEM4394I ILLEGAL OPERATOR yyyy IN RECORD NUMBER xxx. IT HAS BEEN REPLACED BY A PLUS.

W IEM4397I A LETTER IMMEDIATELY FOLLOWS CONSTANT yyyy IN RECORD NUMBER xxx. AN INTERVENING BLANK HAS BEEN ASSUMED.

E IEM4400I OPERATOR .NOT. IN RECORD NUMBER xxx USED AS AN INFIX OPERATOR. IT HAS BEEN REPLACED BY .NE.

T IEM4403I COMPILER ERROR. EXPRESSION SCAN OUT OF PHASE.

System Action: Processing is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM4406I PREVIOUS USAGE OF IDENTIFIER zzzz CONFLICTS WITH USE IN EXPRESSION IN RECORD NUMBER xxx.

System Action: Processing is terminated if an attempt is made to execute a statement which references the identifier in question.

S IEM4407I UNDECIPHERABLE KEYWORD. nnn IDENTIFIERS HAVE BEEN DELETED BEFORE yyyy IN RECORD NUMBER xxx.

Explanation: The processor has found a mismatch while scanning a keyword consisting of more than one identifier.

System Action: The identifiers preceding the non-matching identifier are deleted.

S IEM4409I OPERATOR MISSING IN RECORD NUMBER xxx. A PLUS HAS BEEN INSERTED.

S IEM4412I NO EXPRESSION WHERE ONE IS EXPECTED IN RECORD NUMBER xxx. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

S IEM4415I ILLEGAL OPERAND yyyy IN RECORD NUMBER xxx HAS BEEN REPLACED BY A FIXED DECIMAL ZERO.

E IEM4421I MISSING LEFT PARENTHESIS INSERTED AT BEGINNING OF EXPRESSION IN RECORD NUMBER xxx.

T IEM4433I REFERENCE IN RECORD NUMBER xxx TO STATEMENT OR IDENTIFIER WHICH IS IN ERROR. PROCESSING TERMINATED.

S IEM4436I EXCESS ARGUMENTS TO FUNCTION zzzz IN RECORD NUMBER xxx. EXTRA ARGUMENTS HAVE BEEN DELETED.

Explanation: Too many arguments appear in a procedure reference.

W IEM4439I TOO FEW ARGUMENTS TO FUNCTION zzzz IN RECORD NUMBER xxx. MISSING ARGUMENTS HAVE BEEN REPLACED BY NULL STRINGS OR FIXED DECIMAL ZEROS.

Explanation: Too few arguments appear in a procedure reference.

E IEM4448I NO ENTRY DECLARATION FOR PROCEDURE zzzz REFERENCED IN RECORD NUMBER xxx. ATTRIBUTES TAKEN FROM PROCEDURE.

Explanation: All procedure names must be declared with ENTRY and RETURNS attributes before the procedure is referenced.

T IEM4451I PROCEDURE zzzz REFERENCED IN RECORD NUMBER xxx CANNOT BE FOUND. PROCESSING TERMINATED.

T IEM4452I RECURSIVE USE OF PROCEDURE zzzz IN RECORD NUMBER xxx IS DISALLOWED. PROCESSING TERMINATED.

E IEM4454I TOO FEW ARGUMENTS HAVE BEEN SPECIFIED FOR THE BUILTIN FUNCTION SUBSTR IN RECORD NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

E IEM4457I TOO MANY ARGUMENTS HAVE BEEN SPECIFIED FOR THE BUILTIN FUNCTION SUBSTR IN RECORD NUMBER xxx. EXTRA ARGUMENTS HAVE BEEN IGNORED.

E IEM4460I FIXED OVERFLOW HAS OCCURRED IN RECORD NUMBER xxx. RESULT TRUNCATED.

System Action: Truncation occurs on left to five decimal digits.

E IEM4463I ZERO DIVIDE HAS OCCURRED AT RECORD NUMBER xxx. RESULT SET TO ONE.

S IEM4469I END-OF-FILE FOUND IMBEDDED IN STATEMENT IN RECORD NUMBER xxx. EXECUTION OF STATEMENT WILL CAUSE TERMINATION.

E IEM4472I IDENTIFIER BEGINNING zzzz IN STATEMENT AT RECORD NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Explanation: Identifiers may not exceed 31 characters in length.

System Action: The identifier is truncated to the first 31 characters.

S IEM4473I CONSTANT yyyy IN RECORD NUMBER xxx HAS PRECISION GREATER THAN 5. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

Explanation: Implementation restriction. Precision of fixed decimal numbers is limited to five digits.

System Action: A value of zero is assigned.

E IEM4475I QUESTION MARK IN RECORD NUMBER xxx HAS NO SIGNIFICANCE. IT HAS BEEN IGNORED.

Explanation: Question mark, although a recognizable character in PL/I, has nc syntactical meaning.

T IEM4478I STRING IN RECORD NUMBER xxx CONVERTS TO A FIXED DECIMAL NUMBER WITH PRECISION GREATER THAN 5. PROCESSING TERMINATED.

Explanation: Implementation restriction. Precision of fixed decimal numbers is limited tc five digits.

System Action: Processing is terminated.

T IEM4481I CHARACTER STRING IN RECORD NUMBER xxx CONTAINS CHARACTER OTHER THAN 1 OR 0 AND CANNOT BE CONVERTED TO A BIT STRING. PROCESSING TERMINATED.

T IEM4484I STRING IN RECORD NUMBER xxx OR IN PROCEDURE REFERENCED IN SAID RECORD NUMBER CANNOT BE CONVERTED TO A FIXED DECIMAL CONSTANT. PROCESSING TERMINATED.

T IEM4499I A % STATEMENT IS FOUND IN A REPLACEMENT VALUE IN RECORD NUMBER xxx. PROCESSING TERMINATED.

Explanation: A replacement value may not contain a compile-time statement.

T IEM4502I AN IDENTIFIER zzzz WITH CONFLICTING USAGE OR MULTIPLE DEFINITIONS IS REFERENCED IN RECORD NUMBER xxx. PROCESSING TERMINATED.

Explanation: An attempt is made to execute a statement which references an identifier that was not properly defined.

S IEM4504I VARIABLE zzzz IS USED IN RECORD NUMBER xxx BEFORE IT IS INITIALIZED. IT HAS BEEN GIVEN NULL STRING OR ZERO VALUE.

Explanation: A value must be assigned to variables before they are referenced after being declared.

T IEM4505I DD STATEMENT FOR INCLUDE zzzz MISSING IN RECORD NUMBER xxx. PROCESSING TERMINATED.

Explanation: A DDEF statement must be present in the command stream for the compilation with a name in the name field that corresponds to the ddname identifier in the INCLUDE statement. If no ddname is specified in the INCLUDE statement, a SYSLIB DDEF statement is required.

User Response: Insert appropriate DDEF statement and recompile.

T IEM4508I UNRECOVERABLE I/O ERROR WHILE SEARCHING FOR MEMBER OF INCLUDE zzzz IN RECORD NUMBER xxx. PROCESSING TERMINATED.

User Response: Check DDEF statement and reattempt compilation. If error persists, check computing system.

T IEM4510I INVALID DSORG FOUND FOR INCLUDE MEMBER IN RECORD NUMBER xxx. PROCESSING TERMINATED.

T IEM4511I ILLEGAL RECORD FORMAT SPECIFIED FOR INCLUDE zzzz IN RECORD NUMBER xxx. PROCESSING TERMINATED.

Explanation: Included records must be a fixed length of not more than 100 characters with a maximum blocking factor of five. Blocksize must be a multiple of the record length.

T IEM4514I MEMBER OF INCLUDE zzzz IN RECORD NUMBER xxx NOT FOUND ON DATA SET. PROCESSING TERMINATED.

User Response: Check INCLUDE statement, DDEF statement, and data file.

W IEM4517I RECORD LENGTH NOT SPECIFIED FOR INCLUDE zzzz IN RECORD NUMBER xxx. RECORD LENGTH EQUAL TO BLOCKSIZE HAS BEEN ASSUMED.

User Response: Correct record length specification in DDEF statement if necessary.

W IEM4520I BLOCKSIZE NOT SPECIFIED FOR INCLUDE zzzz IN RECORD NUMBER xxx. BLOCKSIZE EQUAL TO RECORD LENGTH HAS BEEN ASSUMED.

User Response: Correct blocksize specification in DDEF statement if necessary.

W IEM4523I RECORD LENGTH AND BLOCKSIZE NOT SPECIFIED FOR INCLUDE zzzz IN RECORD NUMBER xxx. RECORD LENGTH OF 80 AND BLOCKSIZE OF 400 HAVE BEEN ASSUMED.

User Response: Correct record length and block size specifications in DDEF statement if necessary.

T IEM4526I I/O ERROR WHILE READING TEXT INCLUDED FROM zzzz AT RECORD NUMBER xxx. PROCESSING TERMINATED.

User Response: Check DDEF statement and reattempt compilation. If error persists, check computing system.

T IEM4529I IMPLEMENTATION RESTRICTION. EXCESSIVE LEVEL OF NESTING OR

REPLACEMENT AT RECORD NUMBER xxx. PROCESSING TERMINATED.

Explanation: Level of nesting in this case is calculated by summing the number of current unbalanced left parentheses, the number of current nested DO's, the number of current nested IF's, and the number of current nested replacements. A level of 50 is always acceptable.

T IEM4532I INPUT RECORD AT RECORD NUMBER xxx IS TOO LONG. FOLLOWING TEXT DELETED -- (up to 10 characters)

Explanation: Input record contains more than 100 characters.

System Action: The indicated text is deleted and compilation continues.

User Response: If you want, press the attention key and edit the data set.

T IEM4535I INPUT RECORD CONTAINS FEWER CHARACTERS THAN SORMGIN REQUIRES. PROCESSING TERMINATED.

Explanation: The length of the input record is less than the left margin of the SORMGIN specification.

User Response: Check SORMGIN option in PLI command.

T IEM4547I COMPILER ERROR. INSUFFICIENT SPACE FOR TABLES.

System Action: Processing is terminated.

User Response: Save relevant data and inform the system manager or administrator of the error.

E IEM4550I RIGHT PARENTHESIS INSERTED IN RECORD NUMBER xxx TO END ARGUMENT LIST FOR PROCEDURE zzzz.

Explanation: The argument list referred to is in a source program reference to a compile-time procedure.

T IEM4553I IN RECORD NUMBER xxx ARGUMENT LIST FOR PROCEDURE zzzz CONTAINS COMPILE TIME CODE. PROCESSING TERMINATED.

Explanation: Compile-time code may not be embedded in argument list of compile-time procedure reference.

E IEM4559I LEFT PARENTHESIS BEGINNING ARGUMENT LIST OF PROCEDURE zzzz WAS NOT FOUND. PROCEDURE WAS INVOKED AT RECORD NUMBER xxx WITHOUT ARGUMENTS.

Explanation: The argument list referred to is in a source program reference to a compile-time procedure.

E IEM4562I IDENTIFIER IN RECORD NUMBER xxx EXCEEDS 31 CHARACTERS. REPLACEMENT WAS DONE ON TRUNCATED FORM zzzz.

Explanation: A non-compile-time source text identifier consists of more than 31 characters.

E IEM4570I THE THIRD ARGUMENT OF BUILT-IN FUNCTION SUBSTR IS NEGATIVE, IN RECORD NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

E IEM4572I THE THIRD ARGUMENT OF BUILT-IN FUNCTION SUBSTR EXCEEDS THE STRING LENGTH, IN RECORD NUMBER xxx. THE SUBSTRING HAS BEEN TRUNCATED AT THE END OF THE ORIGINAL STRING.

E IEM4574I THE COMBINED SECOND AND THIRD ARGUMENTS OF BUILT-IN FUNCTION SUBSTR EXCEED THE STRING LENGTH, IN RECORD NUMBER xxx. THE SUBSTRING HAS BEEN TRUNCATED AT THE END OF THE ORIGINAL STRING.

E IEM4576I THE SECOND ARGUMENT OF BUILT-IN FUNCTION SUBSTR IS LESS THAN ONE, IN RECORD NUMBER xxx. ITS VALUE HAS BEEN RESET TO ONE.

E IEM4578I THE SECOND ARGUMENT OF BUILT-IN FUNCTION SUBSTR EXCEEDS THE STRING LENGTH, IN RECORD NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

S IEM4580I AN UNINITIALIZED VARIABLE HAS BEEN FOUND IN A BUILT-IN FUNCTION ARGUMENT LIST, IN STATEMENT NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

User Response: Initialize the variable before invoking the built-in function.

Object-Time Diagnostic Message Forms

An object-time diagnostic message takes one of the following forms:

1. IHEnnnI FILE name - text AT location message
2. IHEnnnI rtname - text AT location message
3. IHEnnnI text AT location message

where "name" is the name of the file associated with the error (given only in I/O diagnostic messages); "rtname" is the name of the library routine in which the error occurred (given only for computational subroutines); and "location message" is either

OFFSET ± hhhhh FROM ENTRY POINT E1

or

OFFSET ± hhhhh FROM ENTRY POINT OF cccc ON-UNIT

Note: Since the phrase "AT location message" is common to each object-time diagnostic message, it will not be shown as part of the message in the following list. It will, however, appear, and have the meaning given above, with each issued object-time diagnostic message.

If the statement number compiler option has been specified, each message will also contain "IN STATEMENT nnnnn" prior to AT location message; nnnnn gives the number of the statement in which the condition occurred.

Diagnostic messages are printed at execution time for two main reasons:

1. An error occurs for which no specific ON-condition exists in PL/I. A diagnostic message is printed, and the ERROR ON-condition is raised.
2. An ON-condition is raised, by compiled code or by the library, and the action required is system action, for which the language specifies COMMENT as part of the necessary action.

The object-time diagnostic messages are grouped as follows:

Message Sequence	Condition
IHE003I-IHE011I	General execution errors.
IHE018I-IHE039I	I/O errors.
IHE100I-IHE161I	I/O ON-conditions: all these conditions may be raised by the SIGNAL statement.
IHE200I-IHE213I	Computational errors.
IHE300I-IHE362I	Computational ON-conditions: all these conditions may be raised by the SIGNAL statement.
IHE380I-IHE382I	Structure and array errors.
IHE500I-IHE501I	Condition-type ON-conditions.
IHE550I-IHE571I	Errors associated with tasking: associated with the WAIT statement or with the COMPLETION psuedo-variable.
IHE600I-IHE609I	Conversion ON-conditions: conversion errors occur most often on input, owing to an error either in the input data or in a format list. For example, in edit-directed input, if the field width of one of the items in the data list is incorrectly specified in the format list, the input stream will get out of step with the format list and a conversion error is likely to occur.
IHE700I-IHE799I	Conversion errors, non-ON-type.
IHE800I-IHE806I	Noncomputational program interrupt errors: certain program interrupts may occur in a PL/I program because the source program has an error which is severe but which cannot be detected until execution time. An example is a call to an unknown procedure, which will result in an illegal operation program interrupt. Other program interrupts, such as addressing, specification, protection, and data interrupts, may arise if PL/I control blocks have been destroyed. This can occur if an assignment is

made to an array element whose subscript is out of range, since, if SUBSCRIPTRANGE has not been enabled, the compiler does not check array subscripts; a program interrupt may occur at the time of the assignment or at a later stage in the program. Similarly, an attempt to use the value of an array element whose subscript is out of range may cause an interrupt. Care must be taken when parameters are passed to a procedure. If the data attributes of the arguments of the calling statement do not agree with those of the invoked entry point, or if an argument is not passed at all, a program interrupt may occur. The use of the value of a variable that has not been initialized, or has had no assignment made to it, or the use of CONTROLLED variables that have not been allocated, may also cause one of these interrupts.

IHE850I-IHE857I TSS/360 execution messages.

IHE900I-IHE902I Storage management errors: associated with the handling of storage and transfer of control out of blocks. In some cases, these errors are the result of a program error, but it is possible that the messages may be issued because the save area chain, allocation chain, or psuedo-register vector have been overwritten.

Object-Time Diagnostic Messages

IHE003I SOURCE PROGRAM ERROR IN STATEMENT nnnnn

This message will always contain a statement number whether or not the compiler option is specified.

IHE004I INTERRUPT IN ERROR HANDLER - PROGRAM TERMINATED

Explanation: When an unexpected program interrupt occurs during the handling of another program interrupt, it indicates that the program has a disastrous error in it, such as overwritten instructions or such. The program is abnormally terminated, and the above message is printed out at the console.

IHE006I PROCEDURE INCORRECTLY INVOKED. PROCEDURE TERMINATED.

Explanation: This message is caused by one of the following:

1. A PL/I subroutine called from PL/I code has been called by module name; it should have been called by procedure name.
2. A PL/I program called from command mode has no external procedure with the option MAIN.

IHE009I ABEND-INTERRUPT IN PL/I DUMP.

Explanation: An unrecoverable error has occurred during execution of the IHEDUM dump routine, e.g., incorrect chaining of save areas caused by a previous error.

IHE011I KEY ERROR WHEN CLOSING FILE AT END OF TASK

Explanation: An unresolved key error exists for which no condition can now be raised.

IHE018I FILE name - FILE TYPE NOT SUPPORTED

IHE020I FILE name - ATTEMPT TO READ OUTPUT FILE

IHE021I FILE name - ATTEMPT TO WRITE INPUT FILE

IHE022I GET/PUT STRING EXCEEDS STRING SIZE

Explanation: For input: programmer requested more than exists on the input string. For output: programmer is trying to write more than his output string will hold.

IHE023I FILE name - OUTPUT TRANSMIT ERROR NOT ACCEPTABLE

Explanation: The ERROR is raised (1) upon return from a TRANSMIT ON-unit, if the device in error is

other than a printer, or (2) if access to a file by RECORD I/O has been attempted after the TRANSMIT condition has been raised for output.

IHE024I FILE name - PRINT OPTION/FORMAT ITEM FOR NON-PRINT FILE

Explanation: Attempt to use PAGE, LINE, or SKIP ≤ 0 for a non-print file.

IHE025I DISPLAY - MESSAGE OR REPLY AREA LENGTH ZERO

Explanation: This message appears only if the REPLY option is exercised.

IHE026I FILE name - DATA DIRECTED INPUT - INVALID ARRAY DATUM

Explanation: Number of subscripts on external medium does not correspond to number of declared subscripts.

IHE027I GET STRING - UNRECOGNIZABLE DATA NAME

Explanation:

1. GET DATA - name of data item found in string is not known at the time of the GET statement, or

2. GET DATA data list - name of data item found in string is not specified in the list.

IHE029I FILE name - UNSUPPORTED FILE OPERATION

Explanation: Programmer has executed an I/O statement with an option or verb not applicable to the specified file. For example:

I/O Option or Verb	File Attribute
READ SET| LOCATE	DIRECT| (SEQUENTIAL UNBUFFERED)
REWRITE (without FROM)| EXCLUSIVE| UNLOCK| (READ NOLOCK)	(SEQUENTIAL INPUT|OUTPUT| UPDATE)| (DIRECT INPUT| OUTPUT)
KEYTO	REGIONAL DIRECT
LINESIZE| PAGESIZE	STREAM (INPUT| UPDATE)

IHE030I FILE name - REWRITE/DELETE NOT IMMEDIATELY PRECEDED BY READ

IHE031I FILE name - INEXPLICABLE I/O ERROR

Explanation: TSS/360 data management has detected some error in the current input/output operation. The message could be caused by one of the following:

1. INDEXED data set with F-format records: a previously created data set was reopened for sequential output and the key of the record to be added was not higher in the collating sequence than that of the last key on the data set.

2. An input/output error occurred for which no information was supplied by data management.

IHE033I FILE name - NO COMPLETED READ EXISTS (INCORRECT NCP VALUE)

Explanation: This message may be issued because the correct NCP value has not been specified or it may be due to incorrect source code. For example, the source code attempts to rewrite a record that was read with event but for which a WAIT has not been issued.

IHE034I FILE name - TOO MANY INCOMPLETE I/O OPERATIONS

Explanation: The number of incomplete I/O operations equals the NCP value.

IHE035I FILE name - EVENT VARIABLE ALREADY IN USE

IHE036I FILE name - IMPLICIT OPEN FAILURE, CANNOT PROCEED

Explanation: There has been a failure in an implicit OPEN operation.

IHE037I FILE name - ATTEMPT TO REWRITE OUT OF SEQUENCE

Explanation: An intervening I/O statement occurs between a READ statement and a REWRITE statement referring to the same record.

IHE038I FILE name - ENDFILE FOUND UNEXPECTEDLY IN MIDDLE OF DATA ITEM.

Explanation: The ERROR condition is raised when end-of-file is encountered before the delimiter

when scanning list-directed or data-directed input, or if the field width in the format list of edit-directed input would take the scan beyond the end-of-file.

IHE039I FILE name - ATTEMPT TO CLOSE FILE NOT OPENED IN CURRENT TASK

IHE100I FILE name - UNRECOGNIZABLE DATA NAME

Explanation: Initiating ON-condition: NAME.

1. GET DATA - name of data item found on external medium is not known at the time of the GET statement, or
2. GET DATA data list - name of data item found on external medium is not specified in the list.

IHE110I FILE name - RECORD CONDITION SIGNALED

IHE111I FILE name - RECORD VARIABLE SMALLER THAN RECORD SIZE

Explanation: The variable specified in the READ statement INTO option allows fewer characters than exist in the record.

F format records:
a WRITE statement attempts to put a record smaller than the record size.

All formats:
a REWRITE attempts to replace a record with one of smaller size. (Note: This condition cannot be detected for U-format records read for UNBUFFERED or DIRECT files.)

IHE112I FILE name - RECORD VARIABLE LARGER THAN RECORD SIZE

Explanation: A WRITE statement attempts to put out a record greater than the available record size; or a REWRITE statement attempts to replace a record with one of greater size.

IHE113I ATTEMPT TO WRITE/LOCATE ZERO LENGTH RECORD

Explanation: A WRITE or REWRITE statement attempts to put out a record of zero length, or a LOCATE statement attempts to get buffer space for a record of zero length.

IHE120I FILE name - PERMANENT INPUT ERROR

Explanation: Initiating ON-CONDITION: TRANSMIT.

IHE121I FILE name - PERMANENT OUTPUT ERROR

Explanation: Initiating ON-condition: TRANSMIT.

IHE122I FILE name - TRANSMIT CONDITION SIGNALED

IHE130I FILE name - KEY CONDITION SIGNALED

IHE131I FILE name - KEYED RECORD NOT FOUND

Explanation: READ, REWRITE, or DELETE statement specified record key which does not match with records of data set. If the access is sequential, keys may be out of sequence or keys that should not be equal are.

IHE132I FILE name - ATTEMPT TO ADD DUPLICATE KEY

Explanation: WRITE statement specified a key value which already exists within data set. For INDEXED data sets, the condition is detected for both SEQUENTIAL and DIRECT access.

IHE133I FILE name - KEY SEQUENCE ERROR

Explanation: WRITE statement specified, during creation of data set (OUTPUT SEQUENTIAL), a key which for INDEXED data sets is lower in binary collating sequence than prior key.

IHE135I FILE name - KEY SPECIFICATION ERROR

Explanation: INDEXED: the KEYFROM or KEY expression may be the NULL string. Alternatively, RKP does not equal zero and the embedded key is not identical with that specified by the KEYFROM option (or the KEY option in the case of a rewrite statement). A third possibility is that an attempt has been made during SEQUENTIAL UPDATE to replace a record by one whose embedded key does not match that of the original record.

IHE137I FILE name - NO SPACE AVAILABLE TO ADD KEYED RECORD

Explanation: WRITE statement attempted to add record, but either the maximum number of

overflow pages or the maximum size of a shared data set has been exceeded.

IHE140I FILE name - END OF FILE ENCOUNTERED

Explanation: Initiating ON-condition: ENDFILE.

IHE150I FILE name - CANNOT BE OPENED, NO DDEF

Explanation: Initiating ON-condition: UNDEFINEDFILE.

IHE151I FILE name - CONFLICTING DECLARE AND OPEN ATTRIBUTES

Explanation: Initiating ON-condition: UNDEFINEDFILE.

There is a conflict between the declared PL/I file attributes. For example:

Attribute	Conflicting Attributes
PRINT	INPUT, UPDATE, RECORD, DIRECT, SEQUENTIAL, TRANSIENT, BACKWARDS, BUFFERED, UNBUFFERED, EXCLUSIVE, KEYED
STREAM	UPDATE, RECORD, DIRECT, TRANSIENT, SEQUENTIAL, BACKWARDS, BUFFERED, UNBUFFERED, EXCLUSIVE, KEYED
EXCLUSIVE	INPUT, OUTPUT, SEQUENTIAL, TRANSIENT, BACKWARDS, BUFFERED, UNBUFFERED
DIRECT	SEQUENTIAL, TRANSIENT, BACKWARDS, BUFFERED, UNBUFFERED
UPDATE	INPUT, OUTPUT, BACKWARDS, TRANSIENT
OUTPUT	INPUT, BACKWARDS
BUFFERED	UNBUFFERED

Some attributes may have been supplied when a file is opened implicitly. Example of attributes implied by I/O statements are:

I/O Statement	Implied Attributes
DELETE	RECORD, DIRECT, UPDATE
GET	INPUT
LOCATE	RECORD, OUTPUT, SEQUENTIAL, BUFFERED
PUT	OUTPUT
READ	RECORD, INPUT
REWRITE	RECORD, UPDATE
UNLOCK	RECORD, DIRECT, UPDATE, EXCLUSIVE
WRITE	RECORD, OUTPUT

In turn, certain attributes may imply other attributes:

Attribute	Implied Attributes
BACKWARDS	RECORD, SEQUENTIAL, INPUT
BUFFERED	RECORD, SEQUENTIAL
DIRECT	RECORD, KEYED
EXCLUSIVE	RECORD, KEYED, DIRECT, UPDATE
KEYED	RECORD
PRINT	OUTPUT, STREAM
SEQUENTIAL	RECORD
UNBUFFERED	RECORD, SEQUENTIAL
UPDATE	RECORD

Finally, a group of alternate attributes has one of the group as a default. The default is implied if none of the group is specified explicitly or is implied by other attributes or by the opening I/O statement. The groups of alternates are:

Group	Default
STREAM|RECORD	STREAM
INPUT|OUTPUT|UPDATE	INPUT
SEQUENTIAL|DIRECT (RECORD files)	SEQUENTIAL
BUFFERED|UNBUFFERED (SEQUENTIAL files)	BUFFERED

IHE152I FILE name - FILE TYPE NOT SUPPORTED

Explanation: Initiating
ON-condition: UNDEFINEDFILE.

The user has attempted to open or use a regional or transient file.

IHE153I FILE name - BLOCKSIZE NOT SPECIFIED

Explanation: Initiating
ON-condition: UNDEFINEDFILE

Block size not specified on DDEF statement nor on environment. However, will never occur for PRINT file, because default block size is assumed.

IHE154I FILE name - UNDEFINEDFILE CONDITION SIGNALED

IHE156I FILE name - CONFLICTING ATTRIBUTE AND ENVIRONMENT PARAMETERS

Explanation: Initiating
ON-condition: UNDEFINEDFILE.

Examples of conflicting parameters are:

ENVIRONMENT Parameter	File Attribute
No file organization parameter	KEYED
INDEXED	STREAM
CONSECUTIVE	DIRECT| EXCLUSIVE
INDEXED	OUTPUT without KEYED
Blocked records	UNBUFFERED

IHE157I FILE name - CONFLICTING ENVIRONMENT AND/OR DDEF PARAMETERS

Explanation: Initiating
ON-condition: UNDEFINEDFILE.

IHE158I FILE name - KEYLENGTH NOT SPECIFIED

Explanation: Initiating
ON-condition: UNDEFINEDFILE A keylength has not been specified for an INDEXED, file that is being opened for OUTPUT.

IHE159I FILE name - INCORRECT BLOCKSIZE AND/OR LOGICAL RECORD SIZE IN STATEMENT NUMBER xxx

Explanation: Initiating
ON-condition: UNDEFINEDFILE.

One of the following situations exists:

1. F-format records:
   (a) the specified block size is less than the logical record length, or
   (b) the specified block size is not a multiple of the logical record length.
2. V-format records:
   (a) the specified block size is less than the logical record length + 4, or
   (b) the logical record length is less than 14 for a RECORD file or 15 for a STREAM file.

IHE160I FILE name - LINESIZE GREATER THAN IMPLEMENTATION DEFINED MAXIMUM LENGTH

Explanation: Initiating
ON-condition: UNDEFINEDFILE. The implementation-defined maximum linesize is: F-format records 32759, V-format records 32751

IHE161I FILE name - CONFLICTING ATTRIBUTE AND DDEF PARAMETERS

Explanation: Initiating
ON-condition: UNDEFINEDFILE. The user has attempted to associate a file with the BACKWARDS attribute with a non-magnetic-tape device.

IHE200I rtname - X LE 0 IN SQRT(X)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE202I rtname - X LE 0 IN LOG(X) OR LOG2(X) OR LOG10(X)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE203I rtname - ABS(X) GE (2**50)*K IN SIN(X) OR COS(X) (K=PI) OR SIND(X) OR COSD(X) (K=180)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE204I rtname - ABS(X) GE (2**50)*K IN TAN(X) (K=PI) OR TAND(X) (K=180)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE206I rtname - X=Y=0 IN ATAN(Y,X) AND ATAND(Y,X)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE208I rtname - ABS(X) GT 1 IN ATANH(X)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE209I rtname - X=0, Y LE 0 IN X**Y

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE210I rtname - X=0, Y NOT POSITIVE REAL IN X**Y

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE211I rtname - Z=+I OR -I IN ATAN(Z) OR Z=+1 OR -1 IN ATANH(Z)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE212I rtname - ABS(X) GE (2**18)*K IN SIN(X) OR COS(X) (K=PI) OR SIND(X) OR COSD(X) (K=180)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

IHE213I rtname - ABS(X) GE (2**18)*K IN TAN(X) (K=PI) OR TAND(X) (K=180)

Explanation: A computational error has been detected by the routine identified as "rtname". A list of routine names and their identifiers follows message IHE213I.

Note: Messages IHE200I-IHE213I are issued when a computational error is detected. The identifier of the detecting routine, which immediately follows the message identifier, is associated with the routine name indicated below.

Identifier	Routine Name
IHESQS	Short float square root
IHELNS	Short float logarithm
IHETNS	Short float tangent
IHEATS	Short float arctan
IHESNS	Short float sine and cosine
IHEHTS	Short float hyperbolic arctan
IHESQL	Long float square root
IHELNL	Long float logarithm
IHETNL	Long float tangent
IHEATL	Long float arctan
IHESNL	Long float sine and cosine
IHEHTL	Long float hyperbolic arctan
IHEXIS	Short float integer exponentiation
IHEXIL	Long float integer exponentiation
IHEXXS	Short float general exponentiation
IHEXXL	Long float general exponentiation
IHEXIW	Short float complex integer exponentiation
IHEXIZ	Long float complex integer exponentiation
IHEXXW	Short float complex general exponentiation
IHEXXZ	Long float complex general exponentiation
IHEATW	Short float complex arctan and hyperbolic arctan

IHEATZ Long float complex arctan and hyperbolic arctan

IHE300I OVERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the exponent of a floating-point number exceeds the permitted maximum, as defined by implementation.

IHE310I [FILE name] SIZE

Explanation: This condition is raised, by Library routines or by compiled code, when assignment is attempted where the number to be assigned will not fit into the target field. This condition can be raised by allowing the fixed overflow interrupt to occur on account of SIZE. If associated with I/O, then "FILE name" will be inserted between the message number and the text.

IHE320I FIXEDOVERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the result of a fixed-point binary or decimal operation exceeds the maximum field width as defined by implementation.

IHE330I ZERODIVIDE

Explanation: This condition is raised, by Library routines or by compiled code, when an attempt is made to divide by zero, or when the quotient exceeds the precision allocated for the result of a division. The condition can be raised by hardware interrupt or by special coding.

IHE340I UNDERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the exponent of a floating-point number is smaller than the implementation-defined mimimum. The condition does not occur when equal floating-point numbers are subtracted.

IHE350I STRINGRANGE

Explanation: This condition is raised by Library routines when an invalid reference by the SUBSTR built-in function or pseudo-variable has been detected.

IHE360I AREA CONDITION RAISED IN ALLOCATE STATEMENT

Explanation: There is not enough room in the area in which to allocate the based variable.

IHE361I AREA CONDITION RAISED IN ASSIGNMENT STATEMENT

Explanation: There is not enough room in the area to which the based variable is being assigned.

IHE362I AREA SIGNALED

IHE380I IHESTR - STRUCTURE OR ARRAY LENGTH GE 16**6 BYTES

Explanation: During the mapping of a structure or array, the length of the structure or array has been found to be greater than or equal to 16**6 bytes.

IHE381I IHESTR - VIRTUAL ORIGIN OF ARRAY GE 16**6 OR LE -16**6

Explanation: During the mapping of a structure, the address of the element with zero subscripts in an array, whether it exists or not, has been computed to be outside the range (-16**6 to +16**6).

IHE382I IHESTR - UPPER BOUND LESS THAN LOWER BOUND

Explanation: During the mapping of an array or structure, an upper bound of a dimension has been found to be less than the corresponding lower bound. If only an upper bound was declared then it may currently be less than one, the implied lower bound.

IHE500I SUBSCRIPTRANGE

Explanation: This condition is raised, by Library routines or by compiled code, when a subscript is evaluated and found to lie outside its specified bounds, or by the SIGNAL statement.

IHE501I CONDITION

Explanation: This condition is raised by execution of a SIGNAL (identifier) statement, referencing a programmer-specified EXTERNAL identifier.

IHE550I ATTEMPT TO WAIT ON AN INACTIVE AND INCOMPLETE EVENT

IHE553I WAIT ON MORE THAN 255 INCOMPLETE EVENTS

IHE571I TASK (name) TERMINATED. COMPLETION CODE= hhh.

Explanation: hhh is a hexadecimal number.

IHE600I CONVERSION CONDITION SIGNALED

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE601I CONVERSION ERROR IN F-FORMAT INPUT

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE602I CONVERSION ERROR IN E-FORMAT INPUT

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE603I CONVERSION ERROR IN B-FORMAT INPUT

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE604I ERROR IN CONVERSION FROM CHARACTER STRING TO ARITHMETIC

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE605I ERROR IN CONVERSION FROM CHARACTER STRING TO BIT STRING

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE606I ERROR IN CONVERSION FROM CHARACTER STRING TO PICTURED CHARACTER STRING

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE607I CONVERSION ERROR IN P-FORMAT INPUT (DECIMAL)

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE608I CONVERSION ERROR IN P-FORMAT INPUT (CHARACTER)

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

IHE609I CONVERSION ERROR IN P-FORMAT INPUT (STERLING)

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and text.

Conversion Errors, Non-ON-Type

IHE700I INCORRECT E(W,D,S) SPECIFICATION

Explanation: An EDIT operation was incorrectly specified.

IHE701I F FORMAT W SPECIFICATION TOO SMALL

Explanation: An EDIT operation was incorrectly specified.

IHE702I A FORMAT W UNSPECIFIED AND LIST ITEM NOT TYPE STRING

Explanation: An EDIT operation was incorrectly specified.

IHE703I B FORMAT W UNSPECIFIED AND LIST ITEM NOT TYPE STRING

Explanation: An EDIT operation was incorrectly specified.

IHE704I A FORMAT W UNSPECIFIED ON INPUT

Explanation: An EDIT operation was incorrectly specified.

IHE705I B FORMAT W UNSPECIFIED ON INPUT

Explanation: An EDIT operation was incorrectly specified.

IHE706I UNABLE TO ASSIGN TO PICTURED CHARACTER STRING

Explanation: A source datum which is not a character string cannot be assigned to a pictured character string because of a mismatch with the PIC description of the target.

IHE798I ONSOURCE OR ONCHAR PSEUDOVARIABLE USED OUT OF CONTEXT

Explanation: This message is printed and the ERROR condition raised if an ONSOURCE or ONCHAR pseudo-variable is used outside an ON-unit, or in an ON-unit other than either a CONVERSION ON-unit or an ERROR or FINISH ON-unit following from system action for CONVERSION.

IHE799I RETURN ATTEMPTED FROM CONVERSION ON-UNIT BUT SOURCE FIELD NOT MODIFIED

Explanation: A CONVERSION ON-unit has been entered as a result of an invalid conversion, and an attempt has been made to return, and hence reattempt the conversion, without using one or other of the pseudo-variables ONSOURCE or ONCHAR to change the invalid character.

IHE800I INVALID OPERATION

IHE801I PRIVILEGED OPERATION

IHE802I EXECUTE INSTRUCTION EXECUTED

IHE803I PROTECTION VIOLATION

IHE804I ADDRESSING INTERRUPT

IHE805I SPECIFICATION INTERRUPT

IHE806I DATA INTERRUPT

Explanation: This condition can be caused by an attempt to use the value of a FIXED DECIMAL variable when no prior assignment to, or initialization of, the variable has been performed.

IHE850I MULTITASKING NOT SUPPORTED IN TSS-EXECUTION TERMINATED.

IHE851I CHECKPOINT NOT SUPPORTED IN TSS-EXECUTION CONTINUES.

IHE852I CHECKPOINT CANCEL NOT SUPPORTED IN TSS-EXECUTION CONTINUES.

IHE853I SORT-MERGE NOT SUPPORTED IN TSS-EXECUTION TERMINATED.

IHE854I PROCEDURE ENTERED DIRECTLY FROM COMMAND MODE. USE MODULE NAME INSTEAD OF PROCEDURE NAME.

IHE856I TASK NOT IN 24 BIT MODE. EXECUTION TERMINATED.

IHE857I RESTART NOT SUPPORTED IN TSS-EXECUTION TERMINATED.

IHE900I TOO MANY ACTIVE ON-UNITS AND ENTRY PARAMETER PROCEDURES

Explanation: There is an implementation limit to the number of ON-units and/or entry parameter procedures which can be active at any time. An entry parameter procedure is one that passes an entry name as parameter to a procedure it calls. The total permissible number of these ON-units/entry parameter procedures is 127.

IHE902I GOTO STATEMENT REFERENCES A LABEL IN AN INACTIVE BLOCK

Explanation: The label referred to cannot be found in any of the blocks currently active in the current task; blocks are not freed. The statement number and offset indicate the GO TO statement causing the error.

Where more than one page reference is given, the major reference is first.

PL/I Programmer's Guide Printed in U.S.A. GC28-2049-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)