IBM

Application Program

H20-0626-0

Problem Language ANalyzer (PLAN)

Users' Introduction

Program Nos. 1130-CX-25X, 360A-CX-26X and 360A-CX-27X

This manual presents the basic ideas and facilities of the PLAN system, with references to more advanced manuals. It is organized into four chapters. The first chapter is a general introduction. Each of the others is an independent unit addressed to one group of users of the PLAN system (application users, application designers, or application programmers).

First Edition (June 1969)

Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

This edition applies to Version 1, Modification Level 0 of Problem Language ANalyzer (PLAN) (1130-CX-25X, 360A-CX-26X, and 360A-CX-27X) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the specifications herein. Therefore, before using this publication, consult the latest 1130 and System/360 SRL Newsletters (N20-1130, N20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to: IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601.

CONTENTS

CHAPTER 1: INTRODUCTION

This manual introduces the PLAN system to those who will actively use it. Some fundamentals of computer problem-solving are discussed, and the basic capabilities of the PLAN system are described. References to other PLAN documentation are also given for further study. The object of this manual is to convey just enough information and knowledge to the reader so that he can begin to operate in a PLAN environment.

The material in this manual is intermediate in detail between the PLAN Application Description Manual (H20-0490) and the PLAN Program Description Manual (H20-0594). After reading Chapter 1, the reader should go directly to one of the other three chapters. Each of these is a self-contained unit. Each contains a description of PLAN, a discussion of its features, and necessary references and examples from the point of view of one group of PLAN users.

The user of PLAN may take one or more of three roles: the PLAN application user, the PLAN application designer, or the PLAN application programmer.

1. The PLAN application user works as a problem-solver. He views the computer as a resource. When he uses it, he needs to concentrate on his own problem rather than on the details of computer use. This user does not expect to write computer programs. Instead, he wants a convenient input language to describe each particular problem. (The language input indirectly specifies the previously programmed routines that are needed to solve this particular problem. The language input also furnishes the data needed for each solution.)

2. The PLAN application designer is generally a full-time computer systems analyst. He designs application languages and supporting modular programs that allow the PLAN user to solve problems. Application designers work with the users to determine their input language needs. The application designer also determines the need for new supporting subprograms and submits programming specifications for them. The application designer is not required to perform as a programmer, but he must understand the resources available to programmers under PLAN. The PLAN system provides a PLAN language for language definition to help designers avoid their usual programming involvement.

3. The PLAN programmer uses standard programming languages (mainly FORTRAN) to produce the modular subprograms that are specified by the PLAN application designer. The programmer's product is essentially identical to a group of conventional subroutines. He does not prepare connecting mainline programs. Using the new facilities of PLAN with good judgment, programmers can produce generalized routines that will be useful in many different applications (for example, a high-performance I/O package with variable data conversions and formats).

BASIC PLAN FEATURES

Machine and Operating System

The PLAN system has been produced to a uniform external specification for the IBM 1130 and for the IBM System/360, using DOS or any of the three OS/360 options (PCP, MFT, or MVT). Programs written in basic FORTRAN IV (supplemented by the PLAN subroutine set) may be compiled and executed in all three environments without source

change. Advanced or unique features of each system are also supported (for example, FORTRAN H), but their use may limit conversion to some degree.

Language Processor

The PLAN system provides a generalized input language processor. The statement formats of completely unrelated PLAN-based languages can be cataloged and processed by a single program. In this way each PLAN installation can choose its own mixture of PLAN application languages while using a single IBM-written processor.

Program Execution Facilities

The PLAN system is built around a special program loader that brings programs into core storage in whatever sequence is required for a particular problem. The overall sequence is determined by the user's choice of input language statements. The program sequence for any one input language statement is usually specified in the cataloged definition of the statement. Use of the PLAN loader allows a module (or a statement) to be used in many applications without rigid linkages to other programs (or other statements). The importance of this variable-linkage concept in PLAN cannot be emphasized too strongly.

USE OF THIS MANUAL

Chapter 2 is directed to the PLAN application user. It describes the basic ideas of computer-aided problem-solving and the creation and usefulness of problem-oriented languages.

Chapter 3 is directed to the PLAN application designer. It contains an elementary discussion of PLAN operation and introduces the new techniques and features that will be most useful in developing new applications.

Chapter 4 is directed to the PLAN application programmer. It also contains a discussion of PLAN operation and features. A description is also given of PLAN subroutines for program linkage, secondary storage management, unit record input and output, and error recovery.

Chapters 3 and 4 both contain simple examples and references to the PLAN Program Description Manual (H20-0594). This manual should be available when Chapters 3 and 4 are studied.

The reader should now proceed directly to Chapter 2, 3, or 4, depending on the role that he expects to fill in using PLAN.

CHAPTER 2: USE OF PLAN FOR PROBLEM-SOLVING

People who use the PLAN system operate at one or more of three levels: as PLAN application users, PLAN application designers, or PLAN programmers. This section is addressed mainly to the PLAN application user. No special preparation is presumed.

The PLAN application user is a problem-solver. He must personally understand and control the problem solution process, but he has neither the time nor the desire to become a computer programmer. To work at his most effective level he needs to continue using the terminology and procedures that are accepted and understood in his business and by his associates.

A computer can perform any step in a solution that can be predefined, but rare is the problem for which all the solution steps can be both predefined and presequenced. A problem-solver must specify which of these steps is to be performed for a particular problem. His choices depend on personal experience and judgment. PLAN provides basic mechanisms to allow this kind of user-guided problem-solving.

An application that has been produced under PLAN allows its user to engage in a form of man-computer dialogue. As each user's statement in a problem description is supplied to the computer, the PLAN processor uses a language catalog to recognize the statement and to determine which previously programmed routines in the library should be performed. It immediately executes these routines and then prepares to accept another input statement. At each step in the problem processing, the user has the opportunity to consider partial results and to choose how the problem solution should continue.

A successful PLAN application depends on good communication between its proposed users and the PLAN application designer. To make this communication easier, some basic ideas and techniques of computer use and computer-aided problem-solving should be understood by the user.

BASIC TECHNIQUES

Input

Some provision must be made for entering problem data into a computer. If it is quite brief, this data may be entered during the problem solution by the problem-solver, using a console device such as a keyboard. As a rule, most data is entered via punched cards or other machine-readable documents that have been prepared ahead of time.

Most significant problems also require the use of stored information such as tables or historical records. This information is generally kept on computer storage devices that allow a rapid transfer of data between storage and the computer processing unit. Magnetic tapes, disks, and drums are generally used for this kind of high-density, fast-access data storage.

Output

As the input data for a problem is being processed, results become available within the computer. Some of these results must be returned to the problem-solver in a form that he can use. If the problem results (intermediate or final) are brief, they may be typed on a keyboard device at a speed of perhaps 15 characters per second. In the more usual case, when a problem step produces many values, a line printer is required. Speeds from

100 to 1000 lines per minute are normal. Results that are too complex to be easily understood in printed form may also be displayed graphically by using a plotting device or a cathode ray tube (a device similar to the television screen).

Intermediate and final results that may be useful for another problem are often placed on punched cards, magnetic tape, or disks. In any of these forms the data will be machine-readable, and the cost and effort of transcribing can be saved.

Program Components

The logical capability of a computer is so elementary that problems must finally be broken down into strings of very simple operations (for example, add A to B, store B in memory location C, etc.). A computer program may perform millions of these simple operations to solve a modest-size problem.

When viewed at the program level, a computer application is almost incomprehensible. As an example of this level of detail, imagine an automobile trip from New York to Los Angeles described in terms of all the left and right turns that were made. The description of the route may be correct, but an overview of the trip is not provided. For clear understanding, this trip should be outlined on a map, all the intermediate destinations selected, and the routes between them specified by highway number or name. Thinking of the trip in terms of a number of smaller trips between intermediate destinations gives a fairly clear idea of the whole.

The solution of a problem by computer can also be broken up into larger units, each having a single purpose. These are called subprograms (sometimes called modules or components). For example, one subprogram might determine the natural frequency of vibration of a crankshaft, given its dimensions. A financial analysis application might contain a subprogram to search sales records for all stocks with sales over x shares since the day's opening. The solutions of whole problems can be understood most easily as combinations of these unit operations. Groups of applications, like trips within a region of the country, may have many components in common. A large saving of programming costs and training costs results when subprogram functions are carefully built to be used for more than one application.

Program Communication

Moving correctly from subprogram to subprogram within a problem solution requires information, just as choosing the correct route at a highway junction requires knowledge. It has been usual to provide a program component called the "mainline", which controls the sequence of execution of subprograms. The logic of the mainline program provides a series of paths that are just as precise as the pattern of tracks and switches of a railroad. To take an unexpected calculation path, however, requires the mainline to be rewritten, just as track has to be laid for new service on a railroad.

To obtain really flexible control of subprogram execution, the problem-solver must be able to take side trips without becoming a programmer. A problem-description language under PLAN can give him this capability. The degree of success and flexibility of his input language depends mainly on the way the designer has built the data transfer mechanisms. This depends in turn on the application user's initial and continued communication with the designer.

Error Detection

Every problem-solving system needs checks to detect obvious or probable errors. The designers of an application under PLAN can automatically apply rules of reasonableness to the data for any statement if these rules are suggested by the user group. When an error is detected, either new data can be entered to take a predetermined correction step, or a warning can be written automatically. The number of diagnostics and warnings that the user obtains from a system may make him feel too secure. Always remember that the overall validity of a series of problem solution steps cannot be checked automatically. The user must always apply judgment and monitor his results for the problem as a whole.

PROBLEM-ORIENTED LANGUAGE

People who are engaged in technical work usually converse with each other in the jargon of their field. A layman may have difficulty understanding it, but such a "language" has great value in terms of speed and convenience for those who use it. To illustrate this, a tennis player and a mathematician, each using the word "set", have in mind something that is carefully defined but quite different. A person who is not acquainted with either field will need a lot of explanation to understand either meaning completely.

The computer application that is of greatest value to a problem-solver allows him to describe his problem (submit input) in his own jargon. This allows him to feel at ease with the application and inclines him to use it. A computer input language designed to suit the needs of the specific problem area is referred to as a problem-oriented language, or POL. The problem-solver writes POL statements to describe and solve each problem; the computer reads these statements and translates them into computer operations that yield a solution (or part of a solution).

This kind of language is different from the procedural languages that may already be familiar to the reader. FORTRAN is an example of a procedural language that programmers use to instruct a computer how to process the data for a particular class of problems. In the earlier analogy to an automobile trip, the series of left and right turns was a procedural description. A programmer describes the process for a problem solution, while the user of a POL describes a particular problem and its data. In fact, the user of a POL does not have to know anything about computers. He need know only the rules (grammar, statement format, etc.) for his particular POL.

To be most effective a problem-oriented language must contain enough power to solve most of the problems within its intended application area. This goal is never fully achieved; therefore, it is extremely important that a POL allow new problem-solving capabilities to be added with a minimum incremental effort.

Finally, a problem-oriented language must be easy to learn and use. The problem-solver is not interested in becoming a language specialist; he wants rapid solutions to specific problems. The statement formats, grammar, and syntax of a problem-oriented language should be easy to remember and easy to write.

The following description of a simple stock accounting job illustrates the kind of problem-oriented language statements that can be made available to application users.

```
CALCULATION STOCK SPLIT, COMPANY 'XYZ CORPORATION',
   RATIO = 5/4, SHARES 3, HEADING 'J. JONES,
   5/4 SPLIT', DATE '5-5-64', USER 12927;

SHARE, CERTIFICATE 'K188167', NUMBER 1, PURCHASE
   586.57, DATE '6-1-62';

SHARE, CERTIFICATE 'K311022', PURCHASE 386.88,
   DATE '10-1-62';

SHARE, CERTIFICATE 'K446110', PURCHASE 346.88,
   DATE '3-18-63';
```

Notice that the first of these statements provides information; the second, third, and fourth indicate the tasks that the user wants the computer system to perform.

A full description of the steps that were taken to make this capability available is contained in Chapter 3.

The preceding discussion indicates the value of a problem-oriented language. How does a user get his own problem-oriented language? The IBM Problem Language ANalyzer (PLAN) was designed especially to lower the cost of defining, implementing, and using problem-oriented languages.

PLAN is a set of programs that operates on three levels:

1. A support level to help the PLAN programmer produce new modules

2. A language design level to give the PLAN application designer standard commands for cataloging the semantics of a new POL

3. An interpretive level to accept the user's input statements and to cause the execution of the program modules that produce his results

The third level of PLAN service supports the PLAN application user or problem-solver. At this level, no technical knowledge of the PLAN system is required. The user does, however, need an up-to-date catalog and explanation of the POL statements that have been defined for his use. PLAN application designers must prepare this documentation. Here, too, the value of communication cannot be stressed enough.

Most of this discussion has ignored the potential that exists for PLAN application users to design new problem-oriented languages or to change existing languages. In fact, individual users can readily suggest input statements and responses that will be more useful or profitable. A significant improvement can often be made by nothing more complex than a new statement definition. Every conversation between the PLAN application user and the designer will improve the communication that is essential for application growth. Users who are already familiar with FORTRAN programming may, in fact, do the best design job of all.

CHAPTER 3: PLAN SYSTEM SUPPORT FOR APPLICATION DESIGNERS

The key to a successful application is the designer. The degree of user satisfaction depends directly on his understanding of the requirements and the way he uses the resources he has.

Input from user groups may lack the precision and the degree of completeness that are necessary for a successful design. The designer must use his knowledge of systems requirements to get added input and to explain programming requirements to people who may have no background in the application area.

PLAN helps the application designer in several ways:

1. He can design an input language and test it without writing all the implied supporting programs.

2. He can create an application without being forced into its programming.

3. He can develop his application one feature at a time, rather than all at once.

Each of these improvements helps to significantly reduce the development cycle, development risk, and development cost.

Application designers who use PLAN must be (or become) qualified in the usual skills of a systems analyst.

1. He must have enough application knowledge to understand a statement of user needs, even if he is not an expert in this field.

2. He must be familiar with existing program library capabilities, first, to avoid unnecessary work, and second, to suggest improvements or additions to proposed systems on the basis of existing work.

3. He must have the necessary experience to profitably correlate human skill, machine and programming support, and machine record structures.

In addition to these, the PLAN analyst must become thoroughly familiar with the PLAN system in order to use its new facilities to best advantage.

In this chapter, frequent references are made to sections in the PLAN Program Description Manual (H20-0594). These should be studied before actually using the PLAN feature that is involved.

FEATURES OF PLAN AS A POL PROCESSOR

PLAN offers many features the analyst would like to have in a POL processor. Four major attributes are discussed, with their meaning to the analyst.

Machine Independence

High costs are usually associated with machine upgrading, operating system changes, and extending or modifying an application or its data files.

PLAN extends the level of practical machine independence. File definitions, user input language definition, program structure, and overlay processing are efficiently supported to eliminate source code changes that are usually required between 1130, DOS/360, and OS/360 (if the application follows standard PLAN conventions). Using high-performance or machine-optimized options is also supported (for example, full FORTRAN IV, 16-, and 48-bit words on 1130 systems). These options may cause some loss in compatibility.

Few Grammatical Rules

The following are the absolute rules for PLAN input languages:

1. The unit of input is a statement 1 to 450 characters long ended by a semicolon.

2. The first part of each valid statement contains or implies a command comprising phrases that have cataloged definitions.

3. If data follows, the command is ended by a comma.

4. Words that form phrases and name data may contain only alphabetic characters.

5. The name of a data element precedes its value if a name is used (for example, TIME=4.00, not 8 BELLS). Data names can be omitted in many cases (for example, LIST 1, 3, 5, 7.5, 14, 29).

Working within these rules and using the facilities of PLAN, a designer can create a syntax, vocabulary, and discipline that suits almost any purpose.

It is important to note that PLAN does not preempt the input data stream. Blocks of formatted non-PLAN input may be interspersed in a PLAN job stream as input to program modules that are invoked within a problem solution. It will be quite common to have a PLAN statement (for example, REPRODUCE CARDS) followed by a block of data that does not conform to PLAN syntax (for example, a FORTRAN source program).

Unlimited Vocabulary

The primary attraction of a POL is user control and the use of familiar and natural terms in problem description. However, most POL users find that the language vocabulary and syntax fixed by national or industry agreement is not as suitable to their individual problems as they might wish. Each user would prefer to use a vocabulary tailored to his problem area, and to have a voice in its determination. PLAN permits complete flexibility in this area. Using PLAN, each installation can specify its own user's languages (vocabulary and syntax). Many POL's can be processed by the same PLAN system. Besides easy and variable language definition, PLAN also allows the extension and modification of its POL's without requiring the base language definition to change.

Convenient Debugging

The PLAN system provides the PLAN analyst with a very convenient debugging capability. After PLAN POL statements have been written for a given application, but before supporting program modules have been written, the statements may be processed by the system and subjected to all the desired input tests, checks, and conversions.

The designer can use the facilities of the dynamic loader to insert core dump modules, trace modules, and file dumps during execution. This mode of operation makes debugging of new program modules easier.

Most important of all, the PLAN system allows the designer to test language statements and program modules individually without writing a special test program for each case.

BASIC PLAN SYSTEM CONCEPTS

The following definitions are needed when discussions of the PLAN system are being studied.

Word. The smallest element in a PLAN language. It is composed of one or more alphabetic characters. A word may be of any length, but the PLAN system recognizes only the first three characters as significant. Except for the character E, which cannot be used alone, the choice of words in a PLAN language is completely open to the designer.

Phrase. A fixed sequence of one to five words separated by any number of blanks.

Command. A sequence of one or more phrases that define the tasks to be performed by a statement.

Statement. A command optionally followed by data. The maximum length of a statement is 450 characters. It is terminated by a semicolon.

Data. The symbols and/or values following the command portion of a statement.

LANGUAGE DEFINITION

Add Phrase

A PLAN problem-oriented language (POL) is initially cataloged in the PLAN system to help build the user's own collection of POL phrases. The system can be directed to catalog a new phrase by using the IBM-defined command ADD PHRASE. Each new entry in the catalog is generated by a PLAN statement of the form:

ADD PHRASE: Phrase Text, Phrase Data;

Delete Phrase

Any phrase cataloged in the PLAN dictionary (including the IBM-defined phrases ADD PHRASE and DELETE PHRASE) may be deleted. The format for this system command is:

DELETE PHRASE: Phrase Name;

Alter Phrase

The ALTER PHRASE system command is simply a combination of DELETE PHRASE followed by ADD PHRASE. The format for this system command is:

ALTER PHRASE: Phrase Text, Phrase Data;

SAMPLE PROBLEM

To correlate some of the information already presented within this section, the language definition for a sample problem follows. It uses only a few basic facilities of PLAN, but it should indicate to the reader the framework within which a PLAN application operates. System-independent material is covered first, followed by the variations that are normal in a particular system environment (that is, 1130, DOS, or OS).

Statement of Problem

A broker has the portfolios of his customers on direct access storage. As a service, he believes it would be profitable to give them their cost per share, on inquiry. A discussion with his system designer suggests the following requirements:

1. For each inquiry about a customer's stockholdings, a report shall be produced. The necessary input will be the customer's number. The output will list all his stocks in a columnar format with the following headings:

Number of Shares	Certificate Number	Cost per Share	Name of Company

2. In the event of a stock split, new fractional shares will be added to the portfolio. When a split has occurred, the split ratio must be given, with the new certificate number.

System Specification

The analyst first examines known capabilities. Customer portfolios are stored in a permanent file named CLIENTS. The existence of a permanent file suggests that insert, retrieve, and print programs may already exist, so the designer searches the program library and verifies the existence of routines named INSERT and PRINT that can be used for the CLIENTS file.

Each user request supplies the input parameters to determine the actions to be taken. A simple inquiry for a customer's holdings will retrieve and print that portion of the CLIENTS file.

A stock split requires computing a new cost per share, updating that customer's file, and printing the updated file. The computation routine, to be named SPLIT, will use the stock split ratio as input. SPLIT will also invoke a new routine called UPDATE to revise the permanent file. A new certificate number is required by INSERT to add to the customer's stock list.

Specifications to be communicated to the programmer must first define the computations to be performed by SPLIT. SPLIT will be followed by UPDATE to record the new acitivity; therefore, the programmer must be told how to transfer information to UPDATE. UPDATE will invoke INSERT and PRINT to revise the customer's holding and permit verification (INSERT and PRINT are existing routines).

The following phrase definitions are for phrases illustrated earlier in this manual. Following the definitions are the control cards that will be required to enclose the language definition cards in order to have them processed into the language definition file by PLAN.

Phrase Definitions

```
ADD PHRASE:  CALCULATION STOCK SPLIT, LEVEL 1,
(125) COMPANY 'COMPANY NOT DEFINED', (1) RATIO 1,
SHARES 0, USER 0, (30) HEADING 'STANDARD HEADING',
(50) DATE '1-31-69', PROGRAM 'SPLIT';

ADD PHRASE:  SHARE, (3) USER 0, NUMBER 1, PURCHASE
-*RA 'PRICE NOT DEFINED', (60) DATE '10-1-67',
(100) CERTIFICATE 0, PROGRAM 'UPDATE, INSERT, PRINT';
```

1130 CONTROL

```
// JOB
// XEQ PLAN
   •
   •
   •
/* RETURN TO MONITOR
```

DOS CONTROL

```
// JOB
// EXEC DFJPLAN
   •
   •
   •
```

OS CONTROL

```
//XXXX         JOB
//              EXEC PGM=DFJPLAN
//PLANLIB      DD DSNAME=PLANPDS, DISP=OLD
//PLSYSTAB     DD DSNAME=PFILE, DISP=OLD
//PLOUT100     DD SYSOUT=A
//PLINP001     DD *
     •
     •
     •
```

After studying this sample, check the following points:

1. Phrase definitions are processed only once, for entry into a catalog.

2. The FORTRAN program SPLIT must take its input from COMMON. It closely resembles a subroutine, but it is compiled as a mainline so that it can be placed in the system library of executable code.

3. Using the inquiry language does not require the use or even any knowledge of PLAN language definition or programming languages. Stored definitions and object code are used by the PLAN processor to respond to user inputs.

4. The programs and language definitions are alike for all three machine environments, but control cards for compilation and job execution follow JCL conventions for the respective systems.

GENERAL CONCEPTS

The example presented above should be clear in concept, even though some details that are shown have not been explained. Four areas are discussed below to give additional understanding of the example.

Program Lists

The PLAN loading mechanism (that is, the PLAN loader) brings modules into core only if their names appear within the "pop-up list". Program names may be added to this list in two ways: from the definition of a phrase previously stored by the PLAN application designer, or via loader CALLs invoked by the programmer. The latter method is discussed in Chapter 4.

Specifying a program list in a phrase definition is accomplished by using the keyword PROGRAMS in the data section of an ADD PHRASE statement. A list of program module names follows the keyword. This list will be passed to the loader whenever the phrase that is being defined is later used in a PLAN input statement. The sequence of names is the sequence of executions intended. In the following example a catalog entry is being made, informing the PLAN system that the programs ABC, BCD, and CDE are to be executed in that order whenever the phrase called NAME is found in an input statement:

```
ADD PHRASE: NAME, PROGRAMS 'ABC, BCD, CDE';
```

Note that each of these modules will be brought into core individually. That is, they will overlay one another. The designer can also indicate that the programs are to share core at execution time; to do so, he encloses the desired programs' names in parentheses. Both options are possible within a single program name list.

Communication Array Position

A significant feature of the PLAN system is an improved method of communication between one or more program modules, using an unlabeled COMMON area.

This improvement in COMMON use avoids the OS/360 requirement for linkage editing all modules that share COMMON into a single program structure. PLAN also contains provisions to allow the location of data and the length of COMMON to change during execution without reprogramming or recompilation. These features lower the cost and improve the flexibility of an application design.

Each program module that is loaded by PLAN must have a BLANK COMMON control section. Part of this space is used by the PLAN loader, and part of it is the "communication array" (words 641 to the end of COMMON). Many language definition options, such as default data values, require the designer to specify various locations in the communication array. These are referred to as communication array positions, or CAPs. Since the location of a CAP is its displacement relative to the beginning of the communication array, the term "subscript" is sometimes used interchangeably with CAP. Application designers can designate a CAP by using either constants, variables, or computed values.

For further details, the reader is directed to the PLAN Program Description Manual (H20-0594), sections 4.3.6 through 4.3.26.

Switch Words

To maintain communication between independently designed program modules, a small part of COMMON is allocated to 15 "switch words". These are located between the PLAN loader and the communication array. The reader is directed to section 4.3.21 in the PDM for a full description of the use of these switch words.

The most important function of the switch words in application design is to contain pointers to data strings. Instead of placing his input and output data strings at predetermined locations, the designer or programmer may establish and manipulate them as variables, communicating the starting addresses of the strings in the four switch words provided for that purpose. For further details the reader is directed to section 4.3.22 in the PDM.

Operation of PLAN Input Processor

Figure 1 should be used for reference in the following discussion. It illustrates the closed-cycle submonitor operation of PLAN when a user's problem is being solved. The discussion follows one complete cycle; this cycle is repeated for each input statement in the problem description (except the PLAN initiation step).

Initiating the PLAN JOB, the monitor or operating system readies the interpreter (1), the loader (2), and the input buffer (3) for the first cycle. The interpreter analyzes the PLAN statement in the input buffer. First, its command phrase is used to access the dictionary (4), where a definition of each acceptable phrase has been stored. Information from this dictionary entry permits the interpreter to scan the remainder of the input statement. At the end of statement scanning, data from the user's statement and default data from the definition have both been stored in COMMON in the communication array (5). The names of the program modules that are to be executed have also been taken from the dictionary and placed in the pop-up list (6—part of the loader). Data conversion, input checking, and formula evaluation are also performed, as specified in the dictionary entry. At this point the need for the interpreter is temporarily ended. Control passes to the loader, which loads the first module named in the pop-up list from the library (7) and removes the loaded module's name from its list. (The interpreter is overlaid.)

When module execution is complete, each module must return control to the loader, which loads the module whose name is at the top of the list. Loading and execution continue serially until the pop-up list is empty. Then the interpreter is reloaded to access and interpret a new statement. Output is produced by the application program modules that are executed, following their own logic. (They could also have taken non-PLAN input from SYSIN.)

The efficiency of this procedure may increase when more than one module is in main storage at a time. This is particularly true for modules that are used in a looping fashion.

One additional block in Figure 1 should be mentioned. The error processor (8) is loaded instead of the expected application module when an error is found in an input statement. This error-processing module can also be invoked by any other module during execution.

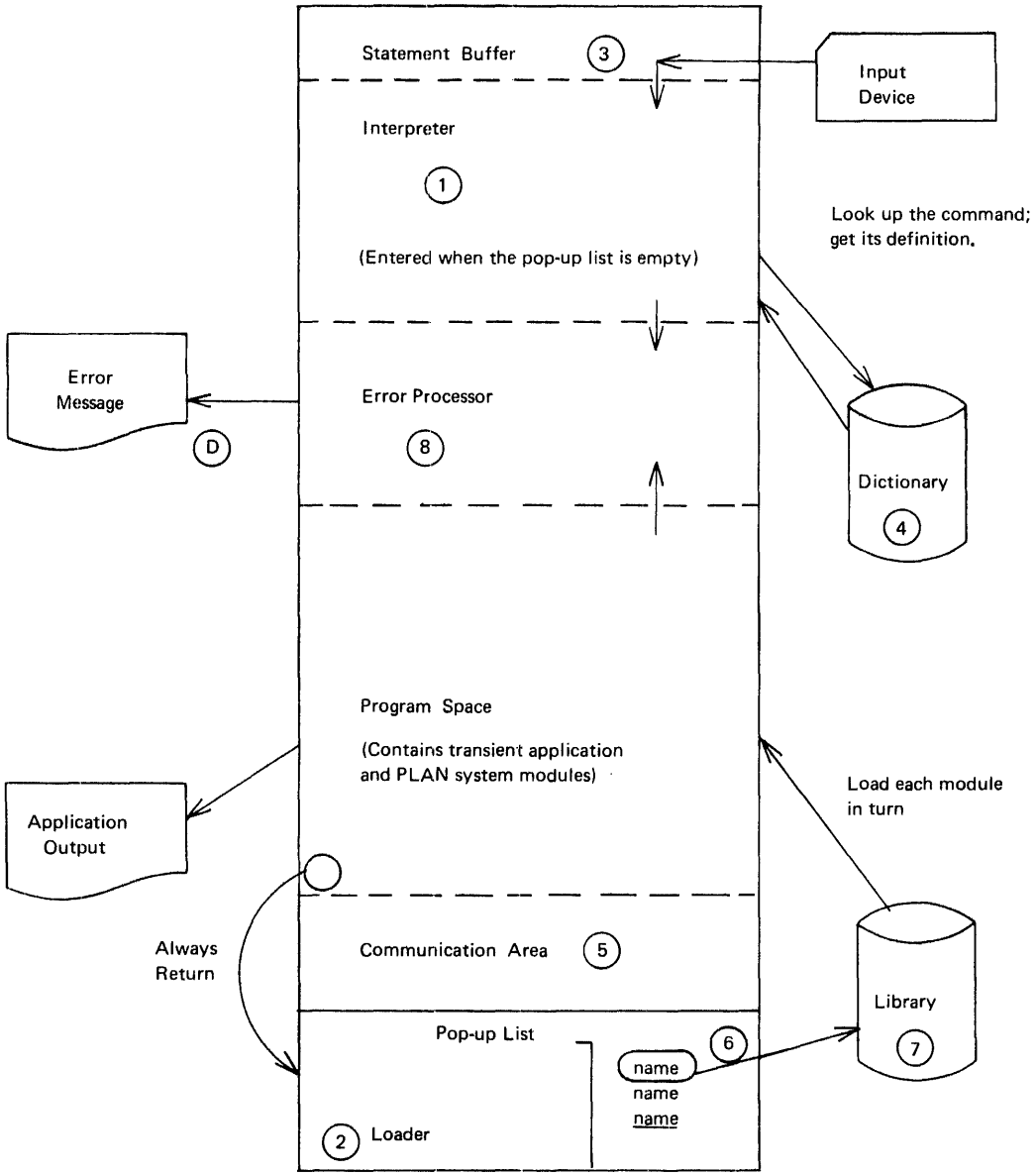

Figure 1. PLAN operating concepts

Note that management of the pop-up list is not confined to the interpreter. Any program module (through PLAN subroutines) may examine, extend, or alter the pop-up list contents.

CAPABILITIES THAT SUPPORT LANGUAGE DESIGN

Now that the general concepts of the system have been explained and the operation of the system has been outlined, the capabilities that are most important to the PLAN designer can be discussed. After a first reading, the specified reference to the PDM should be studied.

Phrase Levels

The effect of level structure in PLAN language processing is like that of an outline in a book or paper. Suppose phrases and levels are defined as indicated below:

```
ANIMAL    (1)
MINERAL   (1)
REPTILE   (2)
SNAKE     (3)
CORAL     (4)
```

The way the statement sequence ANIMAL, CORAL, REPTILE, SNAKE, CORAL, MINERAL, CORAL, will be treated by PLAN can be indicated either by an indented outline or by concatenation.

Indented Outline	Concatenation
ANIMAL	ANIMAL
CORAL	ANIMAL. CORAL
REPTILE	ANIMAL. REPTILE
SNAKE	ANIMAL. REPTILE. SNAKE
CORAL	ANIMAL. REPTILE. SNAKE. CORAL
MINERAL	MINERAL.
CORAL	MINERAL. CORAL

PLAN automatically concatenates phrase symbol tables and specified input data and calculation results in the way that is implied above. Thus, three important objectives can be achieved:

1. A powerful language with a minimum of repetitious input. Once furnished, a higher level statement remains in effect, as though repeated before each subordinate statement. It is overridden by any equal or higher level statement, which then remains in effect, and so on.

2. A language that allows the same word to have different meanings in various contexts (for example, CORAL in the sequence above).

3. A language that can be processed correctly after an input error is found. Suppose the level 3 command SNAKE had been input as SHAKE, which is undefined. The level structure allows PLAN to reject the following level 4 statement, which is dependent, but PLAN can still accept and process the independent statement MINERAL and its successors.

Operation of this feature is explained in detail in section 4.3.3 of the PDM. Note that only the data results in one part of the communication array (the "managed area") are affected by PLAN level management.

DATA SPECIFICATIONS

The largest part of PLAN language definition is the specification of data symbols, values, conversions, and editing that are to be available. The following terms are used frequently in the PLAN literature.

Data Name

A data name is a word (for example, VALUE) that symbolically represents the contents of a particular CAP (storage location). These symbols form a large part of the vocabulary of a problem-oriented language. Data names may be changed at any time without changing any of the supporting program modules. (Programs refer to storage with their own symbols, which are unknown to the application designer.) For further details refer to sections 4.1.7 and 4.3.11 in the PDM.

Subscripts

All definitions of data names assume that the name corresponds to the first position of an array. As previously indicated, any CAP may be referenced by using the appropriate subscript. VALUE (5), for example, is the value stored at CAP 104 if VALUE was defined at CAP 100. Refer to section 4.2.5 of the PDM.

Data Values

Besides a name, each CAP has some value. This may be an input value or a default value. Value specification can take many forms in PLAN input.

Constants

A constant value may be given as a signed or unsigned fixed-point (integer) or floating-point (real decimal) number. Constants may be in normal or exponential notations:

```
VALUE 3.1417
```

Refer to section 4.1.8 of the PDM.

Literals

A CAP can be given a literal value, which will probably fill adjacent higher numbered CAPs as well. Literals are identified by surrounding delimiters (', @, or "). Using ' or @ (the normal choice) causes the literal character count to be saved in the named CAP with the literal following.

Example:

```
VALUE 'JOE JONES AND COMPANY'
```

Refer to section 4.1.9 of the PDM.

Arithmetic Expressions

The value assigned to a data name may also be an expression that uses the common arithmetic operators: plus +, minus -, asterisk *, slash /, and parentheses (). An arithmetic expression is introduced with an equal sign =. Its operands are usually data names (words) and constants. Expressions are evaluated in floating-point mode. The result is stored in real or integer format, as specified for this data name.

Example:

VALUE = VALUE +1,

Refer to sections 4.1.11 and 4.3.16 of the PDM.

Logical Expressions

A data name can also be given a logical value (TRUE or FALSE). The ability to perform logical operations within PLAN language statements is provided by the relational operators "greater than" (>) "less than" (<) and "equal to" (=), preceded by a data name and followed by an arithmetic operand. A logical expression may also contain logical operands and operators. Operands either have the value FALSE or are treated as TRUE. Operators are AND (&), OR (|), and NOT (¬). Parentheses may be used to indicate the sequence of evaluation and to subscript a data name. The whole logical expression begins with a colon. For example,

TEST: (A > 5) & (A < 10),

is a logical expression that will cause the contents of TEST to be set to TRUE if A contains a number greater than 5 and less than 10.

Default Values

Multiple default values may be given in the definition of a phrase. When the phrase is used in a command, its defaults are placed in the appropriate CAPs. The values may be literals, logical constants, or numeric constants. This technique is used to save effort and simplify input. Refer to section 4.3.12 of the PDM.

Scaling

The designer of a language can arrange for the user's input data to be scaled by a specified power of 10 in the range of plus/minus 7. A plus sign indicates movement of the decimal point to the right; a minus sign indicates movement to the left. For example, prices in cents can be scaled to dollars if the implied processing programs expect dollars. Refer to section 4.3.13 of the PDM.

Mode

The normal mode of operation for PLAN is real (floating-point). The designer may also specify integer (fixed-point) storage of a value. The usual I-N naming convention is not required, and input may or may not contain decimal points.

Example: TABLE 3, 4, 5, 7, 8.2 will cause the integer values 3, 4, 5, 7, and 8 to be stored at TABLE (1) through TABLE (5), if TABLE is specified as integer in the ADD PHRASE entry.

CHAPTER 4: PROGRAMMING SUPPORT IN PLAN

This section of the manual is directed to the PLAN programmer. He is responsible for developing program modules that allow the user to conveniently and efficiently solve his problem.

The PLAN programmer is expected to be proficient in FORTRAN. He must learn to understand the value of modular programming, and he should want to make his programs as efficient as possible for the widest range of usage. To do so requires considerable planning and careful use of the available programming tools. The PLAN system furnishes some new tools (features) to help achieve these objectives with a minimum of effort.

The intent of this chapter is to present to the PLAN programmer a number, but not all, of the PLAN facilities available to him. Continual references to the PLAN Program Description Manual (PDM) will be made so that the reader can obtain more details. By understanding these fundamental capabilities he will be able to write a variety of programs for the PLAN environment. After gaining experience with PLAN, the PLAN programmer will undoubtedly want to read the entire PDM to better understand how his work fits with that of the user and the application designer.

PLAN SYSTEM CAPABILITIES AVAILABLE TO PLAN PROGRAMMER

Before discussing the specific programming features available in PLAN, a working understanding of the system is required. The following topics deal with the main PLAN features that both the PLAN programmer and the PLAN application designer must understand.

Program Lists

The PLAN loading mechanism (that is, the PLAN loader) brings into core for execution just those modules whose names appear within the "pop-up list". Program names may be added to this list from a program list supplied by the PLAN analyst or by the programmer using loader subroutines. This section of the manual is mainly concerned with the programmer's options, but the program list is usually controlled by the analyst.

A PLAN application is conceived by its designer as the execution of a series of "phrases", which he has defined. Within each phrase definition the designer may include a list of program module names. Later, when the phrase is used in an input statement, these programs are brought into core for execution in the sequence of the list. The list may specify that the programs are to be brought into core individually, overlaying one another, or that they are to coexist in core at execution time. To the extent that the programmer chooses to issue loader subroutine calls, he may find himself in conflict with the application designer. This conflict can or should be avoided by communication and documentation.

Communication Array Position

A significant feature of the PLAN system is an improved method of communication between one or more program modules, using an unlabeled COMMON area.

This improvement in COMMON use avoids the OS/360 requirement for linkage-editing all modules that share COMMON into a single program structure. PLAN also contains provisions to allow the location of data and the length of COMMON to change during

execution without reprogramming or recompilation. These features lower the cost and improve the flexibility of an application design.

Each program module that is loaded by PLAN must have a BLANK COMMON control section. Part of this space is used by the PLAN loader, and part of it is the "communication array" (words 641 to the end of COMMON). Many language definition options, such as default data values, require the designer to specify various locations in the communication array. These are referred to as communication array positions, or CAPs. Since the location of a CAP is its displacement relative to the beginning of the communication array, the term "subscript" is sometimes used interchangeably with CAP. Application designers can designate a CAP by using either constants, variables, or computed values.

For further details, the reader is directed to the PDM, sections 4.3.6 through 4.3.26.

SWITCH WORDS

To maintain communication between independently designed program modules, a small part of COMMON is allocated to 15 "switch words". These are located between the PLAN loader and the communication array. The reader is directed to section 4.3.21 in the PDM for a full description of the use of these switch words.

The most important function of the switch words in application design is to contain pointers to data strings. Instead of placing his input and output data strings at predetermined locations, the designer or programmer may establish and manipulate them as variables, communicating the starting addresses of the strings in the four switch words provided for that purpose. For further details the reader is directed to section 4.3.22 in the PDM.

OPERATION OF PLAN INPUT PROCESSOR

Figure 1 (in Chapter 3) should be used for reference in the following discussion. It illustrates the closed-cycle submonitor operation of PLAN when a user's problem is being solved. The discussion follows one complete cycle; this cycle is repeated for each input statement in the problem description (except the PLAN initiation step).

Initiating the PLAN JOB, the monitor or operating system readies the interpreter (1), the loader (2), and the input buffer (3) for the first cycle. The interpreter analyzes the PLAN statement in the input buffer. First, its command phrase is used to access the dictionary (4), where a definition of each acceptable phrase has been stored. Information from this dictionary entry permits the interpreter to scan the remainder of the input statement. At the end of statement scanning, data from the user's statement and default data from the definition have both been stored in COMMON in the communication array (5). The names of the program modules that are to be executed have also been taken from the dictionary and placed in the pop-up list (6—part of the loader). Data conversion, input checking, and formula evaluation are also performed, as specified in the dictionary entry. At this point the need for the interpreter is temporarily ended. Control passes to the loader, which loads the first module named in the pop-up list from the library (7) and removes the loaded module's name from its list. (The interpreter is overlaid.)

When module execution is complete, each module must return control to the loader, which loads the module whose name is at the top of the list. Loading and execution continue serially until the pop-up list is empty. Then the interpreter is reloaded to

H20-0626-0

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Arithmetic Expressions

The value assigned to a data name may also be an expression that uses the common arithmetic operators: plus +, minus -, asterisk *, slash /, and parentheses (). An arithmetic expression is introduced with an equal sign =. Its operands are usually data names (words) and constants. Expressions are evaluated in floating-point mode. The result is stored in real or integer format, as specified for this data name.

Example:

VALUE = VALUE +1,

Refer to sections 4.1.11 and 4.3.16 of the PDM.

Logical Expressions

A data name can also be given a logical value (TRUE or FALSE). The ability to perform logical operations within PLAN language statements is provided by the relational operators "greater than" (>) "less than" (<) and "equal to" (=), preceded by a data name and followed by an arithmetic operand. A logical expression may also contain logical operands and operators. Operands either have the value FALSE or are treated as TRUE. Operators are AND (&), OR (|), and NOT (¬). Parentheses may be used to indicate the sequence of evaluation and to subscript a data name. The whole logical expression begins with a colon. For example,

TEST: (A > 5) & (A < 10),

is a logical expression that will cause the contents of TEST to be set to TRUE if A contains a number greater than 5 and less than 10.

Default Values

Multiple default values may be given in the definition of a phrase. When the phrase is used in a command, its defaults are placed in the appropriate CAPs. The values may be literals, logical constants, or numeric constants. This technique is used to save effort and simplify input. Refer to section 4.3.12 of the PDM.

Scaling

The designer of a language can arrange for the user's input data to be scaled by a specified power of 10 in the range of plus/minus 7. A plus sign indicates movement of the decimal point to the right; a minus sign indicates movement to the left. For example, prices in cents can be scaled to dollars if the implied processing programs expect dollars. Refer to section 4.3.13 of the PDM.

Mode

The normal mode of operation for PLAN is real (floating-point). The designer may also specify integer (fixed-point) storage of a value. The usual I-N naming convention is not required, and input may or may not contain decimal points.

Example: TABLE 3, 4, 5, 7, 8.2 will cause the integer values 3, 4, 5, 7, and 8 to be stored at TABLE (1) through TABLE (5), if TABLE is specified as integer in the ADD PHRASE entry.

Refer to section 4.3.14 of the PDM.

CHECKING FACILITIES

Values within the communication array may be checked during the input processing of PLAN language statements before the execution of any problem program module is allowed. The reader should be aware of this capability from the beginning because he can exercise great control by using checking. However, the use of this feature may easily be delayed until some initial experience with PLAN has been obtained. Refer to sections 4.3.15 and 4.3.25 of the PDM, in that order.

USER-EXIT PROGRAMS

During input language processing, exits may be activated to user-written programs for special conversions and other uses. As many as three different user-exit routines may be specified for any given PLAN phrase. The user-exit programs are effectively subroutines of the PLAN processor that can be loaded dynamically. Use of this option should not be attempted until the designer has had an opportunity to become quite familiar with the PLAN system. For details, refer to section 4.3.18 of the PDM.

STATEMENT SAVE

Any PLAN statement or group of statements may be retained temporarily or permanently within the system for subsequent execution. This gives the designer or even the application user an ability to store and use "super-macros" (cataloged procedures). For further details refer to section 4.3.23 of the PDM.

CONCLUSION

By this time the reader is expected to understand the basic operations of the PLAN system and the fundamental features of the system that are available to him. Many features intentionally were not discussed. After making use of the system, the PLAN designer should carefully read the entire PDM in order to understand and use the full capabilities of the PLAN system. He will find it advantageous to mark the location of the following information within the PDM.

1. To solidify his understanding of language definition under PLAN, a question and answer section has been provided. Refer to section 4.4.0.

2. For reference purposes all major subjects, concepts, and routines are listed in an index at the back of the manual.

3. After understanding the general requirements of the PLAN system, the PLAN designer will find individual appendices that are directed to specific environments. Refer to Appendix A, section 8.0.0 for 1130 PLAN specifications; Appendix B, section 9.0.0 for System/360 DOS PLAN specifications; and Appendix C, section 10.0.0 for System/360 OS PLAN specifications.

4. A rather complete set of diagnostic messages is provided by the system, at both phrase interpretation and execution time. These diagnostics are found starting at section 13.6.0.

5. While writing a PLAN application, a convenient list of all system limitations may prove desirable. Such a list is contained in Appendix H, section 15.0.0.

At this point the reader should also study Chapter 4 of this manual to be sure he understands the programming facilities that he can call on. Some of these, particularly the dynamic file and data transfer routines, are especially important for good design work.

CHAPTER 4: PROGRAMMING SUPPORT IN PLAN

This section of the manual is directed to the PLAN programmer. He is responsible for developing program modules that allow the user to conveniently and efficiently solve his problem.

The PLAN programmer is expected to be proficient in FORTRAN. He must learn to understand the value of modular programming, and he should want to make his programs as efficient as possible for the widest range of usage. To do so requires considerable planning and careful use of the available programming tools. The PLAN system furnishes some new tools (features) to help achieve these objectives with a minimum of effort.

The intent of this chapter is to present to the PLAN programmer a number, but not all, of the PLAN facilities available to him. Continual references to the PLAN Program Description Manual (PDM) will be made so that the reader can obtain more details. By understanding these fundamental capabilities he will be able to write a variety of programs for the PLAN environment. After gaining experience with PLAN, the PLAN programmer will undoubtedly want to read the entire PDM to better understand how his work fits with that of the user and the application designer.

PLAN SYSTEM CAPABILITIES AVAILABLE TO PLAN PROGRAMMER

Before discussing the specific programming features available in PLAN, a working understanding of the system is required. The following topics deal with the main PLAN features that both the PLAN programmer and the PLAN application designer must understand.

Program Lists

The PLAN loading mechanism (that is, the PLAN loader) brings into core for execution just those modules whose names appear within the "pop-up list". Program names may be added to this list from a program list supplied by the PLAN analyst or by the programmer using loader subroutines. This section of the manual is mainly concerned with the programmer's options, but the program list is usually controlled by the analyst.

A PLAN application is conceived by its designer as the execution of a series of "phrases", which he has defined. Within each phrase definition the designer may include a list of program module names. Later, when the phrase is used in an input statement, these programs are brought into core for execution in the sequence of the list. The list may specify that the programs are to be brought into core individually, overlaying one another, or that they are to coexist in core at execution time. To the extent that the programmer chooses to issue loader subroutine calls, he may find himself in conflict with the application designer. This conflict can or should be avoided by communication and documentation.

Communication Array Position

A significant feature of the PLAN system is an improved method of communication between one or more program modules, using an unlabeled COMMON area.

This improvement in COMMON use avoids the OS/360 requirement for linkage-editing all modules that share COMMON into a single program structure. PLAN also contains provisions to allow the location of data and the length of COMMON to change during

execution without reprogramming or recompilation. These features lower the cost and improve the flexibility of an application design.

Each program module that is loaded by PLAN must have a BLANK COMMON control section. Part of this space is used by the PLAN loader, and part of it is the "communication array" (words 641 to the end of COMMON). Many language definition options, such as default data values, require the designer to specify various locations in the communication array. These are referred to as communication array positions, or CAPs. Since the location of a CAP is its displacement relative to the beginning of the communication array, the term "subscript" is sometimes used interchangeably with CAP. Application designers can designate a CAP by using either constants, variables, or computed values.

For further details, the reader is directed to the PDM, sections 4.3.6 through 4.3.26.

SWITCH WORDS

To maintain communication between independently designed program modules, a small part of COMMON is allocated to 15 "switch words". These are located between the PLAN loader and the communication array. The reader is directed to section 4.3.21 in the PDM for a full description of the use of these switch words.

The most important function of the switch words in application design is to contain pointers to data strings. Instead of placing his input and output data strings at predetermined locations, the designer or programmer may establish and manipulate them as variables, communicating the starting addresses of the strings in the four switch words provided for that purpose. For further details the reader is directed to section 4.3.22 in the PDM.

OPERATION OF PLAN INPUT PROCESSOR

Figure 1 (in Chapter 3) should be used for reference in the following discussion. It illustrates the closed-cycle submonitor operation of PLAN when a user's problem is being solved. The discussion follows one complete cycle; this cycle is repeated for each input statement in the problem description (except the PLAN initiation step).

Initiating the PLAN JOB, the monitor or operating system readies the interpreter (1), the loader (2), and the input buffer (3) for the first cycle. The interpreter analyzes the PLAN statement in the input buffer. First, its command phrase is used to access the dictionary (4), where a definition of each acceptable phrase has been stored. Information from this dictionary entry permits the interpreter to scan the remainder of the input statement. At the end of statement scanning, data from the user's statement and default data from the definition have both been stored in COMMON in the communication array (5). The names of the program modules that are to be executed have also been taken from the dictionary and placed in the pop-up list (6—part of the loader). Data conversion, input checking, and formula evaluation are also performed, as specified in the dictionary entry. At this point the need for the interpreter is temporarily ended. Control passes to the loader, which loads the first module named in the pop-up list from the library (7) and removes the loaded module's name from its list. (The interpreter is overlaid.)

When module execution is complete, each module must return control to the loader, which loads the module whose name is at the top of the list. Loading and execution continue serially until the pop-up list is empty. Then the interpreter is reloaded to

access and interpret a new statement. Output is produced by the application program modules that are executed, following their own logic. (They could also have taken non-PLAN input from SYSIN.)

The efficiency of this procedure may increase when more than one module is in main storage at a time. This is particularly true for modules that are used in a looping fashion.

One additional block in Figure 1 should be mentioned. The error processor (8) is loaded instead of the expected application module when an error is found in an input statement. This error-processing module can also be invoked by any other module during execution.

Note that management of the pop-up list is not confined to the interpreter. Any program module (through PLAN subroutines) may examine, extend, or alter the pop-up list contents.

PLAN SUBROUTINES

Linkage Facilities

PLAN module loading is controlled by a pop-up list. The PLAN programmer may specify and manipulate the contents of this list, but this is not generally required of him. Normally, the sequence of modules is controlled by the application designer and user; the programmer simply ends each module with a CALL to the loader (CALL LRET), which is equivalent to the RETURN in a subroutine.

On occasion, it may be desirable to preprogram the execution of one or more modules together with or in addition to the one that is being written (to handle an exception, for example). The simplest PLAN routine is an unconditional transfer (similar to FORTRAN CALL LINK). This loader subroutine (CALL LEX) uses variable arguments, however, and can specify movement of any number of program names to the list from any array of literals. No strictly comparable FORTRAN linkage exists.

The programmer may also specify from within execution the names of the modules to be executed next, after this module finishes execution and exits to the loader (CALL LIST). Finally, the programmer can roll out his own module, execute one or more modules by placing their names in the stack, and then resume execution at the next statement in his rolled-out program (CALL LCHEX). No comparable FORTRAN capability exists.

Local

PLAN execution time facilities allow for dynamic loading of either one module or a group of modules for execution. Each new load overlays the last one. However, a special kind of module loading may also be requested through the use of CALL LOCAL. When this subroutine is executed, the programs loaded next do not overlay the program modules currently in core. That is, dynamic loading capabilities are available that do not destroy active modules in core. In the 1130 and DOS environments, CALL LOCAL requires significant preplanning, but dynamic loading always simplifies the construction of overlays and may reduce the total core requirement of an application. In OS/360, CALL LOCAL is automatic. Note that CALL LOCAL and the 1130 MONITOR LOCAL facility are quite different and should not be equated.

In the section index at the back of the PDM the reader will find many references to these loading routines. Read section 5.11.1 first.

Those readers primarily concerned with an OS/360 environment should read section 10.8.0 carefully.

DISK I/O PROCESSING FACILITIES

Perhaps the most important new capability that PLAN provides for the programmer is in I/O processing. A new temporary storage capability is provided via "dynamic file support".

The use of PLAN I/O facilities can both simplify and generalize a program. Furthermore, these facilities are the same in an 1130 or S/360 PLAN environment, and so their use further enhances the opportunity for complete compatibility.

Dynamic File Support

Conventionally, programmers are expected to name and describe all the data files that may be used by an application. For DASD files, definitions are required for name, extent, record length, and format. The descriptors are constant and cannot change unless the source code is changed. A general purpose file I/O program cannot be written.

PLAN dynamic files do not have to be predefined and are not limited to a particular physical record structure. The use of dynamic files gives programmers word addressable secondary storage, which is dynamically allocated on the basis of actual use. The reading and writing of the actual data sets is done by PLAN subroutines that automatically allocate, deallocate, block, unblock, span, and chain physical records. The appearance of the data to programmers is that of a variable number of variable-length one-dimensional arrays, which he can READ or WRITE as needed, in variable-length blocks, beginning on any word boundary. All parameters of the dynamic file are execution time variables. Efficiency is high for these file routines, and they are a key factor in writing routines that are flexible enough to serve small and large problems without recoding. They do allow generalized file I/O programs to be written.

The subroutines provided by PLAN for this support are FIND (which operates as an OPEN), READ, WRITE, and RELES. The information contained in section 5.11.2 will show the reader how to use these routines. The reader should devote a considerable amount of time to understanding this facility.

Permanent File Support

These PLAN subroutines are used by programmers exactly like dynamic files, but the data space is not dynamically allocated or deallocated. Instead, each permanent file can be only as long as the disk extent that was provided for this JOB.

Permanent files for S/360 must be legible to regular FORTRAN I/O. In fact, they are provided mainly for compatibility with current FORTRAN applications.

See section 5.11.3 for a discussion of GDATA (open), RDATA (read), and WDATA (write) usage. These are the PLAN permanent file routines.

SEQUENTIAL I/O SUPPORT ROUTINES

A number of unit record I/O, buffering, and conversion routines are provided for the FORTRAN programmer. They perform device code conversion and also provide selective conversions of fixed-point to integer, floating-point to decimal, etc. These routines furnish a powerful variable reread and editing capacity, which is badly needed for data processing with FORTRAN.

Subroutines are also provided to test for end of file and perform other device control operations (stacker select, skip carriage, etc.).

See section 5.11.9 of the PDM for coding examples and explanations.

SORT/MERGE CAPABILITY

Subroutines are provided by PLAN for sorting and merging dynamic files on disk. Sort/merge keys may be located at random within uniform-length logical records. Binary, alphameric, and numeric keys are allowed and can be mixed. The sorted file replaces the original file on disk, while the merged file creates a new file replacing previously sorted dynamic files. Note that this is an inline CALL to a Disk Sort/Merge, not a utility. It cannot be used for any but PLAN files and must run under PLAN.

Details of the use of this capability are found in section 5.11.7.

Note: After the PLAN programmer has made use of the features described above, he will find descriptions of other PLAN I/O support available to him within the PDM. Appendix E (section 12.0.0) may be of particular interest.

ARRAY AND DATA MANIPULATION

A significant number of PLAN subroutines have been provided to the programmer for obtaining, manipulating, and outputting lists of data. The most important are those for argument list movement to and from COMMON (CALL PARGO, CALL PARGI).

For details concerning these subroutines and a general description of each, refer to sections 5.10.0 and 5.11.10.

ERROR INTERFACE SUBROUTINES

PLAN provides subroutines for the programmer to generate diagnostic messages with his own choice of codes and literal messages without the effort or overhead of inline coding. The selection of routines and the format of the output messages are described in section 5.11.6. After reading this section, the programmer should also note the contents of section 13.3.0. Four levels of message severity are offered, with varied exit and recovery options.

CONCLUSION

By this time the reader as a prospective PLAN programmer is familiar with the operation of the PLAN system and the fundamental features designed for his use. Many parts of the system intentionally were not discussed. After using and becoming familiar with the elementary uses of the system, the programmer should carefully read the entire

PDM to understand all the capabilities of the PLAN system, from the application designer's and user's viewpoints as well as his own.

The PLAN programmer will find it convenient to mark the location of the following information within the PDM.

1. For reference purposes all major subjects, concepts, and routines are listed in an index at the back of the manual.

2. After understanding the programming tools provided by the PLAN system, the PLAN programmer should make use of the individual appendices that are directed to specific environments. Refer to Appendix A, section 8.0.0 for 1130 PLAN specifications; Appendix B, section 9.0.0 for System/360 DOS PLAN specifications; and Appendix C, section 10.0.0 for System/360 OS PLAN specifications.

3. A rather complete set of diagnostic messages is provided by the system. These are found starting at section 13.6.0.

4. While writing a PLAN program, a convenient list of all system limitations may prove desirable. Such a list is contained in Appendix H, section 15.0.0.

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)