

AD732322

Final Technical Report

INFORMATION SYSTEMS DESIGN

January 1971

Under Contract F44620-70-C-0014

between

Air Force Systems Command
United States Air Force
Directorate of Information Sciences
1400 Wilson Boulevard
Arlington, Virginia 22209

and

University of Pennsylvania
The Moore School of Electrical Engineering
Philadelphia, Pennsylvania 19104

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
Springfield, Va. 22151

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DDC
RECEIVED
NOV 9 1971
RECEIVED
C

1260

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing notation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

University of Pennsylvania
The Moore School of Electrical Engineering
Philadelphia, Pennsylvania 19104

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

INFORMATION SYSTEMS DESIGN

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific Final

5. AUTHOR(S) (First name, middle initial, last name)

M. Rubinoff

6. REPORT DATE

January 1971

7a. TOTAL NO. OF PAGES

260

7b. NO. OF REFS

14

8a. CONTRACT OR GRANT NO.

F44620-70-C-0014

b. PROJECT NO.

9769

c.

61102F

d.

681304

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

AFOSR - TR - 71 - 2739

10. DISTRIBUTION STATEMENT

Approved for public release;
distribution unlimited.

11. SUPPLEMENTARY NOTES

TECH, OTHER

12. SPONSORING MILITARY ACTIVITY

Air Force Office of Scientific Research (OSR)
1400 Wilson Boulevard
Arlington, Virginia 22209

13. ABSTRACT

This report describes the results of a study conducted under Contract F44620-70-C-0014 to develop principles and implement techniques for computerized information handling. Concepts associated with the design of a physical property information system are described. Methods for cathode ray tube man-machine interaction in design are discussed, with emphasis upon chemical process and electrical network flow sheets and structural design.

TABLE OF CONTENTS

	Page
Summary	iii
Introduction	1
Numerical Data Base	2
Network Computer Graphics	5
Computer Graphics for Structural Design and Transportation Network Design	7
In Conclusion	9
References	10

SUMMARY

This report describes the results of a study conducted under Contract F44620-70-C-0014 to develop principles and implement techniques for computerized information handling. Concepts associated with the design of a physical property information system are described. Methods for cathode ray tube man-machine interaction in design are discussed, with emphasis upon chemical process and electrical network flow sheets and structural design.

INFORMATION SYSTEM DESIGN

I. INTRODUCTION

This report describes the research completed in a Moore School Information Systems Laboratory study conducted under Contract F44620-70-C-0014. The study has been concerned with developing principles and implementing techniques for computerized information handling, with special emphasis on mechanized information storage and retrieval. These techniques are important to systems for computer-aided design because they automatically provide conventional analysis, simulation and design programs with the information services required. As we developed new information handling techniques, we interfaced them to several previously existing analysis, simulation and design programs at the University of Pennsylvania. These include U.P.PACER and REMUS for steady-state and transient chemical process computations, ECAP for electrical network simulation, and a nonlinear structural analysis program.

Three new information handling techniques were studied in this work:

1. Development of an interactive numerical data base. A prototype physical property information system was implemented to store and retrieve property information for use in chemical process calculations.
2. Development of graphics tools for interactive communication with network analysis programs. Methods for interfacing these tools to the U.P.PACER, REMUS and ECAP programs have been evolving over the past four years. This project enabled us to develop the network graphics tools and make substantial progress toward developing generalized interfacing methods.
3. Development of graphics tools for communication with structural design programs. These tools were applied to the design of civil engineering structures. Similarly, this project enabled us to develop 3-D graphics tools and make substantial progress toward developing generalized interfacing methods.

The following sections summarize the results of each study. A list of publications is provided and copies of the most recent papers are presented in the Appendix. Other papers have previously been transmitted to the AFOSR office.

II. NUMERICAL DATA BASE

Information systems that handle textural data have been developed by the Moore School Information Systems Laboratory over the past seven years. The need to carry out computer-aided design calculations provided sufficient incentive to extend these methods for numerical data. This project enabled us to design and implement a prototype numerical information system. Rather than attempt to handle generalized numerical information initially, it was decided to concentrate on physical property information for chemical process calculations. In retrospect, this was an excellent decision and it has now led to a strong basis from which to consider the more general problem; namely, to store and retrieve numerical information in a data base custom-built by the computer for each design application. In the following paragraphs, some of the history in the development of our prototype physical property information system is presented. Related information has been presented in References 4 and 7.

The first objective of our physical property information system was to allow the engineer to prepare application programs that require no modifications when the chemical components in a mixture are altered and when new and different property estimation techniques are required.

This was accomplished by separating the "variant" and "invariant" information required to request a property value. Only invariant information such as the identity of the physical property (for example, heat capacity), the names of program variables that contain stored values for the independent variables (for example, T and P representing temperature and pressure), and the name of the program variable that contains mole fraction values for each mixture component (for example, X), are incorporated into the application program's request for property values. Variant information such as the identity of components in the mixture, and the identity of property estimation methods are specified separately from the application program. Invariant information is supplied during program preparation, while variant information is supplied just prior to each execution of the application program.

Another objective of our physical property information system has been to allow easy storage and retrieval of physical property data. As compared with other information systems, data retrieval is complicated by the need to also retrieve and apply estimation procedures for estimating the requested property values. Our numerical data base is designed to store three basic prerequisite types of data: constants, correlation coefficients, and tables. The estimation program library contains programs that combine correlation coefficient or tabular data with independent variable and mole fraction specifications to compute "property values". The former programs allow for estimation by correlation, the latter by interpolation. Other estimation programs combine a sequence of estimated property values to produce a single value. For example,

when estimating a mixture enthalpy, one procedure would combine zero pressure enthalpy and enthalpy pressure-correction estimates.

In order to achieve this objective, property data was stored in fixed format data records which contained several key words and the essential data:

- 1) Property code,
- 2) Contributor code,
- 3) Validity ranges of two independent variables,
- 4) Maximum expected error,
- 5) Estimation routine number,
- 6) Data type,
- 7) Component code(s), and
- 8) Property data.

Four illustrative data records are illustrated in Figure 1. The first three data records contain heat capacity data for H₂O over different temperature ranges and $0.5 < P < 2.0$. The fourth data record contains liquid density data for the mixture, methane, ethane, propane, n-butane, and n-pentane, for $400 < T < 490^{\circ}\text{R}$ and $0.5 < P < 2.0$ atm.

Keys	Data Record 1	Data Record 2	Data Record 3		Data Record 4	
Property	liq. heat capacity	vap. heat capacity	vap. heat capacity		liquid density	
Contributor	142	142	153		178	
Validity range (1)	492-612 ^o R	672-3240 ^o R	3240-6840 ^o R		400-490 ^o R	
(2)	0.5-2.0atm	0.5-2.0atm	0.5-2.0atm		15-25atm	
Maximum error	0.5%	1.0%	2.0%		2.0%	
Estimation routine	-	15	14		14	
Data type	constant	coefficient	tabular		tabular	
Component(s)	water	water	water		2,3,4,6,8	
Data	1.0	0.426 1.42x10 ⁻⁵ 3.88x10 ⁻⁸ -7.35x10 ⁻¹²	T	C	T	ρ
			3240	0.656	400	0.1778
			4000	0.701	420	0.2060
			5000	0.743	440	0.2355
			6000	0.771	460	0.2660
			6840	0.782	490	0.3136

Sample Data Records

Figure 1

A master data base containing property data is maintained by the property system librarian, and provides a common information pool available to all users. A private data base is provided for engineers who obtain property data from the literature or by experiment. The engineer enters this data into his private data base. Only the system librarian has authority to enter new data into the master data base, whereas the engineer is free to add new data, update existing data, or delete data only as regards his private data base using the PPIS storage routine.

The PPIS prototype achieved our first two objectives, to provide for mixture independent application programs and to allow storage and retrieval of property data. Many general purpose programs have been prepared for use with systems such as U.P.PACER for total process material and energy balancing. General purpose absorber, stripper, and flash programs have been run as U.P.PACER subroutines when material and energy balancing a natural gas process.

However, an important shortcoming of the PPIS prototype was its inaccessibility to the engineer who wishes preliminary estimates of property values; the system compelled him to write a FORTRAN program for the purpose. This placed an excessive burden upon engineers interested in obtaining first estimates of property values to be used in specific application programs.

This shortcoming, coupled with trends in information system design toward interactive inquiry-response systems, led us to add capability for communication between PPIS and a typewriter terminal. The PPIS prototype proved its utility especially after conversion to an interactive system. Soon afterward PPIS assumed a new role as a sophisticated handbook. Students began to use the inquiry-response features to examine property values conveniently, often in preparation for use of PPIS in application programs.

The PPIS prototype is comprised of FORTRAN IV programs that run on an IBM 360/75. It was converted to an interactive program for the RCA Spectra 70/46 in one man-month. The interactive PPIS is executed within the RCA Time-Sharing Operating System (TSOS). This operating system provides for telecommunication with teletypewriter terminals, IBM selectric typewriter terminals, and the RCA video data terminal (alphanumeric cathode ray tube). Examples of its usage are presented in Reference 7.

The inquiry-response feature markedly increased the engineer's accessibility to PPIS and thereby demonstrated the utility of computer-based information systems for engineering purposes. The most obvious benefits follow. The inquiry-response PPIS provided a tool for estimation of mixture property values when handbook values do not exist or are not easily accessible. It provided a vehicle for property estimation using iterative algorithms that are difficult to implement manually. And, in so doing, it often provided more

accurate property values in shorter time. The inquiry-response PPIS enabled the engineer to scan graphs and tables of property values over independent variable ranges under study in application programs. This is proving to be a powerful tool whereby engineers may confirm the accuracy of bench-mark values and gain confidence in an estimation procedure's ability to reproduce physically known variations with the independent variable values.

The interactive property information system has been modified to enable the user to communicate with the computer via a RCA video display terminal. This terminal enables rapid display of physical property information in the form of tables and graphs. For example, an engineer commonly wishes to know the density of a liquid mixture at various temperatures and pressures. Heretofore, the teletype-writer terminal presented tabular and graphic information slowly at 10-15 characters per second. The video display terminal allows a graph to be displayed in 2-3 seconds, an approximately ten-fold improvement in the speed of preparation.

The property information system has been reprogrammed to enable random access of property information from disk. Previously all property information was transferred from disk into the main memory of the computer prior to execution of a program that required physical property information. The new version of the property information system retrieves property data from disk only when it has not previously been transferred to main memory. Transfer of data occurs only when needed and only when the data is not already present in main memory.

III. NETWORK COMPUTER GRAPHICS

In the mid-1960's several problem-oriented programs were developed for the analysis of networks comprised of interconnected modules; for example, U.P.PACER and REMUS for analysis of chemical plants and ECAP for analysis of electrical circuits. Because these programs often carry out expensive iterative computations, there has been great incentive to minimize costs by allowing the engineer to monitor the calculations very closely. We decided to explore the feasibility of using an interactive cathode ray tube display (DEC-338) to improve man-machine interaction in network analysis. Initially we constructed a prototype that enabled a process designer to "draw" a process flow sheet using the light pen. Next, we built an interface between the DEC-338 and the RCA Spectra 70/46 to allow for direct communication with the U.P.PACER and REMUS programs. A progress report summarizing the capability was prepared by W. D. Seider, J. Ball, and M. Zaborowski (see Reference 2).

Concurrently, P. Delaney developed a more advanced software package for preparation of electrical networks. The principal

improvement was in the data structure which allows for very efficient storage of network information and easy modification of the same. Some of the features of this graphics system are described below, with emphasis upon the data structure. Further information is provided in Reference 8.

A key feature is the display file monitor which handles all display file modifications. To do this effectively, a display file structure has been designed that resembles a tree structure. In this graphics system, the instructions to position the beam are organized in a tree structure that contains the topology of the network. When an electrical device, or a wire connecting two devices, is added or deleted, the display file is modified by the display file monitor. Efficiency is achieved in that changes in topology cause the display instructions and connectivity information to be modified simultaneously. Consequently, the DEC-338 responds very rapidly to the designer's commands when drawing an electrical circuit diagram using the light pen and push buttons.

Still, the electrical network package was specific to electrical networks. It was prepared using DEC-338 assembly instructions and required extensive modifications to handle chemical process networks, for example. This was particularly discouraging in that most of the features of chemical and electrical network packages are similar, and some are identical.

Consequently, we began work to define a general purpose graphics system. First, Dr. C. West and Mr. J. Kulick identified three types of graphics users:

1. The "pictorial" user. This type of user is distinguished by the need to work with pictures. For example, the user of an architectural design system. This user requires facilities for picture generation and picture storage and retrieval. These facilities must be controlled both by the user at the graphics terminal, and by computer programs operating on the data.
2. The "non-pictorial" user that uses the graphics system as a data display device. Typically, he requires facilities for preparation of graphs that summarize data. He is not interested in the display of abstract line drawings. This type of user may be exemplified by the user of a simulation system, where input is in the form of card images, and output is in the form of two-dimensional graphs. This user needs simple picture definition facilities for output, and an alphanumeric keyboard facility for input. There may also be a need for the terminal user to select among a list of options using light pen and function keys.

3. The "meta-pictorial" user. This type of user is concerned with properties of the picture. Typically, these properties are not stored in the picture explicitly, but must be computed from the picture. For example, topological properties such as connectivity of a graph. One possible way to do this is to post-process the picture and assign these features via pattern recognition techniques. Another possibility is to monitor the actions of the user and his programs at the terminal, and to immediately interpret each action. As each action is interpreted, cues are generated to aid in later interpretation of the picture. This approach requires that the user programs be able to sequence actions at the terminal, and generate appropriate cues.

During this project, West and Kulick designed the languages for a general purpose graphics system that would service all three types of users. Their languages were designed to enable easy interfacing of application programs to the graphics terminal. One language was designed to schedule the DEC-338 terminal and the other to send and receive messages between the terminal and an application program operating in the RCA Spectra 70/46. The languages were designed to study the feasibility of implementing and operating such a graphics system using a small terminal connected to a large computer with a low-speed telephone line.

During the past year, J. Kulick has been implementing the system design and is preparing a complete description in his Ph.D. dissertation. The graphics system when complete should enable rapid interfacing of the graphics terminal to application programs such as U.P.PACER, REMUS and ECAP. We expect that the interfacing time will be reduced by an order of magnitude (to approximately 50 hours) because there will no longer be a requirement to program in assembly language.

IV. COMPUTER GRAPHICS FOR STRUCTURAL DESIGN AND TRANSPORTATION NETWORK DESIGN

During the past two years, we concentrated upon the development of a unified computer graphics program for structural design and transportation network design.

The following summarizes the results of the studies in interactive computer graphics in Structural Design. Developments in computing software and hardware make it possible to consign more of the design labor to the computer than has been possible, and permit the designer to exercise more fully his role as policy maker. Formula manipulation computer languages which can be used to produce exact differentials of very large nonlinear expressions, and interactive graphic terminals which make it possible to modify on-line

the nonlinear programming problem representing a design situation, permit structural designs to be generated by the computer which are complete in all respects, where all design parameters necessary to define the configurations can be used, and where building code provisions are used unchanged as the set of constraints. The architecture of a man-machine interactive system has been studied which allows the designer to effectively mold the design with a light pen as it is being evolved by the computer with the nonlinear programming algorithm and which appears in effect as an animation on the scope.

A case study of an optimally proportioned beam is described in Reference 5. A conventional nonlinear optimization program examined various I-Beam dimensions in the course of solving a constrained nonlinear optimization problem. These dimensions were saved at frequent intervals throughout the optimization procedure and were used to prepare an animated movie that demonstrates the variations in design during the optimization procedure.

Ideally, the engineer could interactively monitor the optimization process to accelerate the design process even further. A significant advantage of being able to operate in this manner results from the fact that execution time is unnecessarily inflated if a large number of constraints never become active in any one of the cycles. These constraints still have to be carried along as computational overhead in the matrix operations that are involved in solving the nonlinear programming problem if they are initially a part of the problem. There might be considerable economic merit in being able to operate within a bare framework of constraints that are likely to be active in any given situation, and to be able to insert dynamically any additional necessary constraints into the structure of the problem because of violations that might be detected in any of the iteration cycles. For example, beams over certain lengths are hardly ever governed by shear, and it would be a waste of computational time to carry all possible code restrictions on shear alone in the framework of the nonlinear programming problem for a general beam design routine. If conditions become such that shear does happen to govern for a certain situation, the violation is easily detectable and the appropriate constraints can then be inserted without otherwise interrupting the progress of the design.

Douty and Shore developed the principles and procedures in a batch environment because there was no interactive facility yet available. Their results were demonstrated with an animated movie, proving that such a system is effective and should be designed and implemented. J. Kulick's Ph.D. dissertation involves the design and implementation of a general purpose graphics facility that would indeed serve this role (see Section III of this report).

V. IN CONCLUSION

AFOSR funds enabled us to develop several computerized information handling tools. These tools have been and continue to be incorporated in computer-aided design systems. Several such programs are currently in use by students in the engineering schools for their design work. Among other things, this research enabled us to launch a major effort in the development of systems for teaching the fundamentals of process design using the computer. Extensions to this work are currently being funded by the Esso Education Foundation.

In addition, the work carried out on this project was a contributing factor to the election of Dr. Warren D. Seider as Chairman of the National Academy of Engineering's CACHE (Computer Aids for Chemical Engineering Education) Committee. The CACHE Committee is comprised of 17 educators from 16 colleges and universities. Its goal is to cooperatively further the development of computing systems for use in chemical engineering education.

REFERENCES

1. Cautin, H.; "Real English: A Translator to Enable Natural Language Man-Machine Conversation", Ph.D. dissertation presented to the Moore School of Electrical Engineering, University of Pennsylvania, May 1969.
2. Seider, W. D., Ball, J. and Zaborowski, M.; "Chemical Process Flow Sheet Computer Graphic System", April 1970.
3. Rubinoff, M. and West, C. H.; "Report and Recommendations on Computer Graphics Facilities in The Moore School", January 1970.
4. Poznanovic, D. S.; "Information Storage and Retrieval System for Physical Properties of Chemicals", Master's Thesis presented to the Moore School of Electrical Engineering, University of Pennsylvania, December 1969.
5. Shore, S. and Douty, R.; "Technique for Interactive Computer Graphics in Design", Journal of the Structural Division, ASCE, Vol. 97, No. ST1, Proc. Paper 7837, January 1971, pp. 273-288.
- *6. Niyogi, P.; "Application of Computer Graphic Techniques to Civil Engineering Design Problems; Transportation Engineering", September 1969.
7. Poznanovic, D. S. and Seider, W. D.; "A Physical Property Information System for Undergraduate Education", presented at Chicago meeting of AIChE, November 1970.
8. Delaney, P.; "An Interactive Flow Chart Graphics System", November 1970.

* Report prepared for internal distribution.

Journal of the
STRUCTURAL DIVISION
Proceedings of the American Society of Civil Engineers

TECHNIQUE FOR INTERACTIVE COMPUTER GRAPHICS IN DESIGN^a

By Richard Douty,¹ M. ASCE and Sidney Shore,² F. ASCE

INTRODUCTION

Events of the past few years relating to the direct design of engineering systems by appropriate mathematical optimization techniques indicate that the role of the engineer designing in such an environment could conceivably and eventually consist merely of having to construct an acceptable criterion function, array the set of constraints that govern the design situation, and supply the initial trial configuration that is necessary for setting mathematical optimization processes in motion, and it is beginning to appear that most of these tasks can be automated as well.

This paper is an attempt to demonstrate that when the designer is relieved thusly of having to expend energy, even in a computer-aided environment, in order to obtain an economically sound configuration, then an opportunity exists for him to interact dynamically with the machine, permitting him to treat the design on a system scale not previously possible. Further, the interaction is most effective if the designer's role is primarily that of the key decision maker, able to observe the progress of the system as it is being mathematically generated, but retaining the option of being able to alter the direction of the system's synthesis when unforeseen or undesirable trends are observed.

A nonpassive role such as this, of course, is only feasible if the evolution of the system is presented pictorially in all its shades and nuances, a capability which is possible with the availability of interactive graphic display

Note.—Discussion open until June 1, 1971. Separate discussions should be submitted for the individual papers in this symposium. To extend the closing date one month, a written request must be filed with the Executive Director, ASCE. This paper is part of the copyrighted Journal of the Structural Division, Proceedings of the American Society of Civil Engineers, Vol. 97, No. ST1, January, 1971. Manuscript was submitted for review for possible publication on March 1, 1970.

^aPresented at the August 31–September 2, 1970, ASCE Fifth Conference on Electronic Computation, held at Purdue Univ., Lafayette, Ind.

¹Prof. of Civ. Engrg., Univ. of Missouri–Columbia, Columbia, Mo.; on leave 1969–70 as Sr. Fellow, Towne School of Civ. and Mech. Engrg., Univ. of Pennsylvania, Philadelphia, Pa.

²Prof. and Chmn., Grad. Div. of Civ. Engrg., Towne School of Civ. and Mech. Engrg., Univ. of Pennsylvania, Philadelphia, Pa.

terminals having the vector capability for line drawing. A graphic display augmented design system employing this concept is, in fact, significantly different from many existing graphic design systems, both operational and under development, where the designer still must piece by piece assemble and modify the system as the prime mover in the procedure, attempting to manually produce an optimal configuration while trying to keep it from straying beyond the bounds of the design constraints, but never really knowing how efficiently the balance is maintained. The dissimilarity in the two approaches is due to reliance in the former on mathematical optimization to carry the greater share of the design task.

DESIGN BY OPTIMIZATION VERSUS TRADITIONAL PRACTICE

Mathematically generated design employing optimization methods might well be viewed as an approach that is complementary to the design process as it has been practiced traditionally. In the traditional sense, the response of a trial configuration is critically examined (i.e., analyzed) for suitability of behavior with respect to a given imposed loading. If the response is not suitable, the configuration is altered in an upward direction until suitability is reached. On the other hand, if the response is overly suitable, the configuration may be altered downwardly in an attempt at economy, although in truth nonsuitability is deemed much more critical than oversuitability; whereas the former condition requires alteration, the latter does not. In fact, the cost of the engineering labor that may be expended towards achieving economies through shaving may easily obliterate any savings gained thereby.

The ability to move a design subjectively in an economic direction while maintaining a precise balance between all cost factors in the entire procedure is not acquired easily, and the art of doing so has been endowed with certain mystical traits largely attributed to conditioned intuition reinforced by years of experience.

The traditional approach is marked by the characteristic that it is the analysis of a completely specified system that is the primary computational component of the entire procedure. Unfortunately, analysis as an end product does not contain any rational clues as to which is the best direction in which to proceed next. If design improvement is the object, and if a good design is eventually to be achieved in this manner, it must be accomplished heuristically. Computer-aided design techniques which are based completely on this traditional approach are somewhat encumbered because no matter how sophisticated the equipment, the subjective nature of the designer permeates the process whether he has accomplished the programming of member selection routines or whether he merely utilizes standard analysis routines (STRUDEL, FRAN, etc.) as a basis for carrying out design improvement of more involved systems. The overhanging threat of computational expense dampens the search for the most imaginative and economical avenues of development.

On the other hand, when optimization is employed as an approach to the design situation from an entirely different direction, the boundary condition for the problem, rather than being a trial configuration, is imposed on the response that the system is required to exhibit. Computational effort (the burden of the machine) then, is primarily expended towards determining that

best configuration that meets these constraints. As there are some interesting developing computational techniques that can seek a good design which satisfies a given set of constraints, it appears that much of the mysticism that has surrounded the craft of obtaining a good design is dissipating; in fact, the lower level of design, such as proportioning of girders, etc., now appears to be much more of a rational process than once was thought. As a rational process it is very likely to be consigned to the computer, as has almost all of the processes of analysis. Just how far the concept can be extended into the broader levels of the design process is at this time problematical, but it may well turn out to be further advanced than most of us now imagine.

The problem that is encountered in a design situation is to produce a system configuration that gives a good value to some criterion, which, in the case of structures, might be weight or cost, while at the same time staying within the confines of a set of limitations, such as those imposed on stresses or deflections. This, however, is the form of the so-called mathematical programming problem, the solution of which is obtainable by rational techniques, although admittedly the facility of obtaining the solution depends greatly on the form that the functions exhibit. If the criterion and all design constraints are linear, then the problem is that subset of mathematical programming called linear programming, the solution to which is easily obtained by any of a variety of standard routines available on call on most computing equipment. Thus, finding the solution to linear programming problems has actually become a trivial matter. If any of the functions contain nonlinearities, then finding a solution is decidedly more difficult. Algorithms, however, are being continually developed and improved and the situation is becoming more favorable. Unfortunately, we exist in a nonlinear world, and design situations that can be formulated as linear programming problems, or even simplified to that happy state are exceedingly rare.

In any event, the end product in solving the mathematical programming problem representing a design situation is the design itself, rather than an analysis of a trial design. Any computer-aided design system which is not based on optimization methods will eventually be regarded as fairly sterile, and no doubt will suffer when placed in competition with systems that are.

There are several reasons, however, why mathematically-based design systems have not gained ascendancy over design systems based on more traditional methods. The difficulties can be traced to certain characteristics associated with working engineering systems that have not been easy to overcome by existing mathematical techniques. For example, the yield stress of the grades of steels that are commercially available for the design of a structure are integer (36, 42, 46, etc.), but they belong to only a small set of integers having the regrettable property of being nonadjacent, so that if yield stress happens to be a design variable in a given problem, integer programming (another subset of mathematical programming where the solution is normally selected from the set of all integers lying within the confines of the feasible design space) is not easy to employ. A more difficult situation than this is encountered in the section of plates used in weldments. In this case, the standard thicknesses that are commercially available are not even integers. So-called discrete programming techniques devised to handle such situations, while under active development, still have a long way to go before they can be applied to significant structures. Of course, the solution can always be rounded off, as long as care is taken that this action does not cause constraint

violation. This is probably the best expedient at this time even though the more puristic approach would undoubtedly yield better optima.

A much more serious difficulty that is not so easily disposed of is shown in the case of structural systems where the set of constraints that define a structural design, being for the most part drawn directly from building codes, contains varieties of cascading discontinuities. As one example, any of a variety of buckling formulas may govern the compression stresses in a certain structural component (14), depending on the most critical slenderness ratio in that component that will prevail in the eventual design. Even within these several formulas, however, that slenderness ratio may take any of several forms, depending on the placement of lateral bracing in the several directions that buckling can occur.

Such characteristics, together with the extremely large and cumbersome nonlinearities that must be dealt with and which produce a huge number of local minima, give rise to iterative oscillations and other aberrational computational behaviorisms that also are not easy to handle. Progress is being made towards developing fairly decent algorithms, however, and with this expectation it is probably time to consider and prepare for the emerging role of the engineer designing in an environment where mathematical programming will accomplish automatically most of that which, at this time, is still laboriously evolved by hand.

MAN-MACHINE DESIGN TEAM WITH DUAL DECISION-MAKING LEVELS

Although at first there seems to be an ominous ring in this for the engineer with respect to his acquired skills and intellectual capital, with further thought it will be realized that there has been and probably always will be two distinct levels of decision-making in the design process; a secondary level that can be completely assigned to the computer, and a primary level that has to be shouldered by the designer because perfection in the state of the art can never be achieved. At present, the secondary level for the most part in existing systems involves the chore of grinding out analyses. There may be a few legitimate optimization-based programs devised for special situations, but as yet there is no broadly based generalized scheme which will admit the computer into the picture as a fully versatile design partner. Most design programs that are being used, in fact, employ an exhaustive search among a predefined set of possibilities, but this is practical only for problems of fairly limited scope.

As the use of mathematical programming improves in reliability and generality, the secondary level will begin to involve more and more the occupation of design in the general sense. Even though the rate of progress in this direction is somewhat obscured by a lack of information of what is technically possible in the way of computing hardware and software, the beginning of the team effort can be seen with a bit more clarity. In the area of structures, for example, the secondary level initially will undoubtedly be charged with generating a meritorious configuration that does not violate the confines of building specifications. The primary level will include such tasks as defining range limits on design parameter values, choosing the applicable building specifications, deciding the form that the structure is to take (frame, shell, truss), and the materials that are to be used.

As the state of the art inevitably advances, the demarcation zone between the secondary and primary levels of design decision making will move inexorably upward, and it is fair to assume that as time passes the computer will be able to take on more and more of the tasks that previously had been the exclusive domain of the designer. For example, it is possible to store on secondary storage the results of each design as it is accomplished and in this way accumulate over a sufficient period of time a wealth of design experience. On the basis of the data stored, it might be possible, using statistical regression techniques, to have reasonable range limits for a given type of structure of a certain type and loading generated by the computer, rather than established by the designer.

New tools, both in computer software and hardware, have been or are becoming operational which begins to indicate clearly the structure that an interactive design system which is based primarily on optimization techniques may take, and the evolutionary development that will probably occur thereafter. Notable among these tools on the hardware side are on-line graphic display terminals with vector capability, and on the software side, powerful procedure-oriented languages, such as PL/I (6), which are not only efficient in carrying out numerical calculations, but are also adept both at character-string manipulation and the storage and retrieval of both kinds of data on secondary storage. Important also are special purpose formula manipulation languages (15) that can generate the exact differentials of large nonlinear expressions. And of course, an effective meshing of all these elements into an interactive design system is possible only because of the versatile and efficient computer operating systems that are now available for controlling and scheduling the events that have to take place in the machine.

A comprehensive man-machine interactive design system based on a dual-mode assignment of the decision-making responsibility might be viewed as the following set of capabilities: (1) Synthesis of design constraints; (2) design criteria; (3) optimization; (4) analysis; (5) on-line access; (6) data input; and (7) data output. A large measure of generality is indicated as all of these elements would be brought to bear on any design situation, whether in the area of structures, transportation, or water resources, the three primary system areas that make up the field of civil engineering (11). Some elements in the list, in fact, would be identical for all three areas, such as the module to assemble design constraints, the functionally distinctive data transfer capabilities of: (1) On-line access via graphic display; (2) system initialization and modification by cards or tape to establish data bases and extend system capabilities; and (3) production of hardcopy output for permanent records.

On the other hand, the form of some of the elements are dependent on the particular area to be serviced. For example, while matrix analysis routines have been reasonably well standardized for articulated structures and networks under steady state conditions (3,9), the same programs would be inadequate for the analysis of transportation and water resource systems if transient loadings and discrete behaviorisms are vital design considerations. In fact, it may be that simulation would have to serve most often as the analysis element of the system for both of these latter two areas if they were to be handled as nonlinear programming problems.

Similarly, the form of the optimization technique which should be employed depends on the nature of the system which is to be optimized. In the complete absence of constraints, any of a variety of simple functional hill climbing

methods ought to suffice. This situation is quite rare, though, and the design situation normally has to be formulated as a nonlinearly constrained problem, which means that a nonlinear programming problem has to be confronted. This appears to be the case with structures.

Occasionally, if the system possesses certain desirable topological properties that permit little or no feedback of loading into the determination of the design, the method of dynamic programming can be used profitably (8,13). These characteristics are often found in transportation and water resource systems and the potential for its employment is probably highest in these areas. The most telling argument is that in dynamic programming a decomposition of the system is effected and suboptimization carried out independently on each of the parts, leading, however, to the optimization of the total system as a unit. Because any one part is likely to be easily analyzed by existing formulas, etc., an analysis of the total system as a unit is not required, obviating the need for the relatively cumbersome necessity to simulate. One has to be prepared, however, to sacrifice the significantly large amounts of computing time and storage that dynamic programming requires. If there is considerable amount of feedback in the system, as is the case with indeterminate structures (the configuration that results from a set of internal forces may produce a different set of forces, which in turn requires a configuration change, etc.), then dynamic programming tends to become too cumbersome to use.

Even though the matter of design criteria has been fairly well established for structures, it has not been for the other two areas. With structures, weight or erected cost should serve adequately as the function to be minimized; the former can be formulated quite precisely in terms of the design parameters, but the latter is not so clear cut. It may be that even primitive formulations of cost functions might be adequate, because there appears to be some evidence that the design of a structure is not as sensitive to the specifics of a cost function as had been thought. For example, flanges of WF beams tend to be as far from the neutral axis of the beam as constraints permit under almost any reasonable objective function.

Design criteria for transportation and water resource systems are not so easily arrived at, or universally accepted, because of the complex interplay of socio-politico-economic factors that are involved in the design of such systems. Cost means little in the face of the social disruption that an unwise urban highway system can cause; however, if the formulation of some sort of happiness function is attempted as an alternative, it soon is realized that one man's happiness is bound to be another man's despair.

Assuming however that, even in the face of these difficulties, the elements previously noted are the primary modules that have to be considered in an interactive design system, a clear picture of the flow of data that would have to occur emerges. Such a dual-mode graphics-augmented design system is outlined in Fig. 1.

The cardinal feature of a dual-mode decision-making system is that there be provision for instant communication between the primary level (man) and the secondary level (machine) as the latter is in the process of going about its assigned task. The engineer should not be limited to monitoring decisions made by the machine only by inspecting what is too often literally reams of neat columns of numbers. By the time an undesirable secondary decision-making tendency is perceived, it might well be too late to do anything about

it. (In fact, in most operations such output is available only after the job has terminated.) Rather, the monitoring should be visual, meaning that a display device has to be employed that is able to keep the engineer informed instantly as to what the machine is doing. Further, the display should not be that type which displays only text, because of the same lag period that is experienced with trying to interpret tables of numbers: Only a relatively few lines of numbers can be displayed on such devices anyway. The only device that would really provide adequate visual communication when engineering systems are involved is a cathode ray scope with vector capability where a pictorial representation of the system can be displayed as it is evolving. Further, an interrupt capability is necessary in order to be able to inject into the process modifications originating from the primary level. Thus, the scope should have a light pen attachment and function keyboard as each device has its own particular merits depending on the action that is desired.

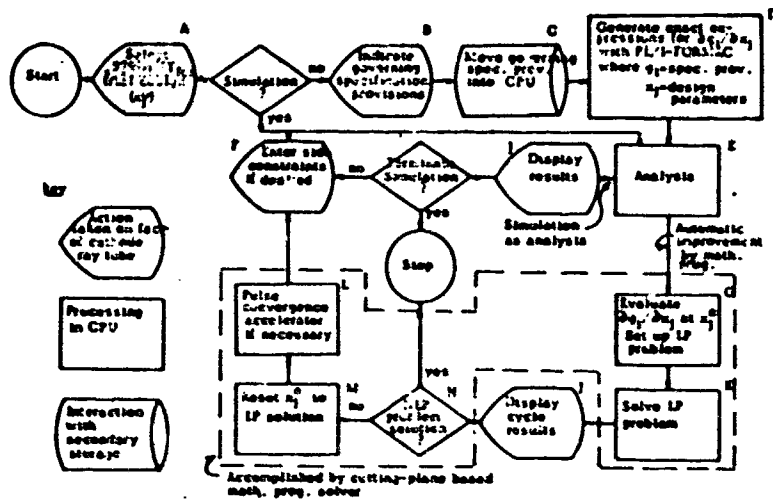


FIG. 1.—DUAL-MODE OPTIMAL DESIGN SYSTEM

It would be adequate as a beginning to have a set of preprogrammed displays stored and available on call with either the keyboard, or to be selected with the light pen from a displayed menu of possibilities. In a structural design system the menu would include a variety of trusses, frames, girders, etc. (A). Similarly, the particular specification or building code that is to govern the design could be selected in either of the same three modes (B). If particular structural modules have been preprogrammed into the system, there is no difficulty in determining which provisions of the selected codes govern at critical points in the structure, as this could just as well be included in the program. Eventually, however, as designers become comfortable and fairly adept at using such a system, there is certain to be some frustration and dissatisfaction experienced with being limited to certain preprogrammed modules. Creativity, which is certain to blossom as designers are released from the shackles of having to involve themselves in the secondary

decision making, will generate a demand for system generality where any system configuration can be sketched and manipulated. It is not difficult to imagine how the sketching might be accomplished and topology determined; this reduces to a matter of the manhours necessary to evolve the program. In fact, scattered efforts in this direction have been underway for some time.

Generality does cause some concern as to how the constraints might be selected during the sketching operation, though it might be as simple as touching the light pen to critical parts of the system in order to cause the appropriate constraints to be generated for that point. In this way, again in the case of a structure, all building code provisions for shear could be arrayed in the constraint set where the shear force and design parameters in those provisions were those at the point where the light pen touched the structure. The designer might do this by depressing a function key labeled SHEAR and with the light pen touch as many points on the structure as he felt should be investigated for that condition. In a similar manner, he could cause multiple constraints for MOMENT and DEFLECTION, etc. to join the set of constraints.

Obviously, every tenth point of each component in the assemblage cannot be investigated for all possible limitations; the result is likely to produce a mathematical programming problem much too immense. The fact that the designer would have to exercise his skill in applying selectivity is merely another indication that designer and machine will be most productive when interacting dynamically as a team.

Of course, the operation of manually indicating where critical points are to be considered does not actually formulate the mathematical programming problem which must be constructed to generate an optimal design. The situation is much different than if preprogrammed modules are employed where the constraints can be easily arrayed as part of the program. If generality is to be permitted, an additional system capability would be needed which would move from mass storage only those building code provisions which would be indicated by the above light pen-function key operation, and then proceed to formulate the mathematical programming problem needed to generate the design. This capability is shown in Fig. 1 as blocks C and D.

Fortunately, newly developed languages adept at character string manipulation (e.g., PL/I) make it possible to store in their natural alphabetic form all provisions of as many building codes as desired or available on mass secondary storage and to move into the central processing unit only those that are applicable for a given situation. Further, there are editing capabilities in these same languages that permit the parsing of code formulas, extracting both the design parameters that are an important element of the mathematical programming problem, and the data elements that must be supplied externally. Even more spectacular, in view of the requirement of most nonlinear programming algorithms that differentials of all constraints be taken with respect to each variable ($\partial g_i / \partial x_j$ in Fig. 1), some of these same languages (15) can generate exact expressions for the differentials of gigantic expressions; the kind of expressions that result, say, if the AISC interaction formula for a beam-column in a frame is expressed in terms of the 10 design parameters that are necessary to define the four cross-sectional dimensions (wide flange) and strengths of both components (the stiffnesses of the connecting members determine the effective length of a beam column).

The magnitude of this task cannot be overstated. As an example, if there

are, say, 100 mostly nonlinear constraints that define a system of, say, 40 variables, and if the nonzero density of the differential matrix $\partial u_i / \partial x_j$ is 0.1, then 400 differentiations would have to be carried out, any one of which is likely to be a formidable task if done by hand. Further, the differentials have to be exact in all respects if the nonlinear programming algorithm is to converge to a solution. The alternative is to go about the job with numerical differentiation, but there may be serious questions as to the effectiveness of a purely numerical approach for the extremely large, nonlinear, and discontinuous problems that are typical of practical systems. It is always more effective, of course, to employ exact differentials in a differential process when they can be obtained, and the new software described makes this possible. (It is a fascinating turn of events that the computer, which is respected for its number smashing capabilities and which, because of this, has greatly accelerated the science of numerical analysis, is acquiring equal respect from the scientific community for its character-string manipulative abilities, a capability previously thought to have applications almost exclusively in commercial data processing, but which appears also to have great potential in scientific processing.)

As the set of constraints on the design will contain internal forces resulting from the loading that is imposed on the system, an analysis is needed to determine those forces before the math programming program is solved (E). This might be accomplished by a generalized matrix method in the case of structures, or as pointed out, a simulation routine in the other areas, assuming that the design parameters in a network of transportation or water resource components must be simultaneously balanced as the only rational approach to the design of these systems, and that because of transients and discrete behaviors (queueing situations, traffic lights, etc.), such systems virtually defy mathematical analysis. It might be speculated that some sort of gradient coefficients might be obtainable with multiple simulations, however the process is likely to be much too time consuming. Functional relationships for such systems are probably the only practical approach if automatic design is the object, but so much is still to be done in this area. For this reason, it is probable that the first broad range of practical applications for the entire system as outlined in Fig. 1 will come in the area of structures, where the matter of analysis is much more well defined and mathematically tidy.

In traditional methods of design there is a pause at this point in the process in order to afford the designer the opportunity of pouring over the results of the analysis to ascertain if, say, stress and stiffness requirements are satisfied. If this is not the case, the system components are strengthened and the analysis is perhaps reexecuted in order to check the effect of the changes. It is worthy to note that if a computer analysis costs several thousand dollars, serious thought would no doubt be given as to the necessity of making such a check. This brings up a point relevant to direct design where an analysis is required for each design improvement cycle that is conducted by mathematical programming. It is obvious that automatic optimization of, say, a large roof constructed of a grid of trusses, which may require thousands of dollars of computer time for each analysis, is not exactly around the corner. Smaller systems than that will have to be attacked first, where analysis is not such a costly factor, such as is the case with continuous girders and gable frames.

Parenthetically, note that a simply supported girder is a true structural system, as the term system has been used vis a vis optimization, because a change in any of the design parameters, such as stiffener spacing, affects the choice of other parameters, such as web depth and thickness when a balance must be sought to maximize the merit of the entire assemblage. It is this interconnected behavior, or feedback, that makes it so difficult to achieve such a balance by human endeavor. By this definition a simply supported rolled beam of 36-ksi steel is not a true structural system because only one parameter, the section modulus, needs to be chosen in order to obtain the optimal design, and that can be achieved simply by a one dimensional search through a table of available shapes. If, however, a variety of steel strengths are available for selection as well as the section modulus, system behavior is present because a balance between the two design variables must be brought about for an optimal situation. This effort may still be within the feasibility of optimization by direct search, similar to the section modulus table search, as there aren't an overwhelming number of combinations to choose from. However, as a further extension one might try to compute the number of possible combinations of a simply supported hybrid beam where the strengths of the flange and web can be different, where the plate widths might be in increments of, say, sixteenths of an inch, and where the thicknesses would be the set of standard thicknesses. It is at this point that mathematical programming begins to demonstrate its utility in being able to home in directly on an optimal combination of parameters without the need for carrying out an exhaustive search among all possibilities.

Structures, then, such as hybrid beams, hybrid girders, trusses and frames will be treated most effectively first by direct optimal systems, such as that shown in Fig. 1, and larger systems such as roof truss grids, multi-story buildings, etc. will no doubt have to be deferred until major breakthroughs occur in reducing the times for analysis. This may come about with significant increases in computing speed, new and novel techniques, or perhaps simply because of studies that may show approximate techniques of analysis are adequate for large systems until the last few cycles of the math programming problem solution. This is not to say, however, that portions of those large systems could not be treated as independent assemblages for the purposes of design, much as they have been in the past.

After an analysis is obtained, the math programming problem which has been formulated, either in preprogrammed modules or by a generalized processor, must be solved. And because of the mathematically impure characteristics (discontinuities, many local optimal, etc.) exhibited by practical systems, the algorithm which accomplishes this task has to be a real workhorse, able to demonstrate computational efficiency and reliability for almost any situation that it is likely to be handed. Work is quite actively in progress at a variety of research centers to produce such algorithms and the eventual availability of a general processor should not be discounted.

In the particular system being described herein, the algorithm being used is a modification (3) of the well-known cutting plane approach (2,7,10,12) to the solution of nonlinear programming problems, a modification which has been used quite successfully to design in detail several types of structural systems, including gable frames, with built-up beam columns as members, both prismatic and nonprismatic, and where either tapered or curved haunches were employed, complete details of both also designed as part of the same

problem (1). Briefly, the modification (L) involves primarily a dynamic constriction of the feasible space at strategic intervals and a unique provision for handling cyclic infeasibilities which may be caused thereby (4). It has handled in an efficient computational fashion nonlinear programming problems having over 150 mostly nonlinear constraints and over 50 design variables, and when used to design structures invariably produces complete designs which are quite competitive with designs evolved in the traditional manner. Though it still has yet to stand the test of time, there seems to be some promise of it being a candidate for the workhorse algorithm which would be required for a generalized system.

To review briefly the operation of the cutting plane algorithm, a Taylor series expansion is employed to reduce the nonlinear functions in the objective and constraint set to a linear programming problem in which a trial design is used as the series expansion point for the first cycle, and the solution to any given cycle is used as the expansion point for the subsequent cycle (M). When this sequence of linear programming problems produces unchanging solutions (H), convergence to the solution of the nonlinear programming problem has been obtained.

If this is all there were to design, no participation of the designer subsequent to initiating the process would be necessary (block B in a generalized system). However, the nature of the course of events that may follow produces interesting opportunities for creative interaction by the designer if, as previously described: (1) He is able to monitor the progress of the design as it is being algorithmically generated (J); and (2) if he can, on the basis of what has been perceived, modify the direction that the design seems to be taking, and do so before a conclusion is reached (F).

Suppose, for example, that the math programming problem that was formulated for a fairly substantial structure is in the process of being solved, and that its evolution towards an optimal configuration is being observed (J), in fact almost as an animation. The designer might suddenly realize that the web depths, in the mathematical struggle towards optimality, are tending to be rather deep and that an implicit desire for a certain amount of headroom, whether from a practical standpoint or the view of esthetics, is in danger of being violated because the web depth maxima were somehow omitted in the formulation of the problem. (Such would not be an unusual situation if a generalized system were being employed.) If, under the usual batch mode operating environment, a termination and restart were necessary in order to correct the situation, not only time but a significant amount of computing money would have been lost for this one small point that was overlooked. On the other hand if the additional web depth constraint could suddenly be made an active part of the constraint set by use of an interrupt capability such as that afforded by a light pen or function key, then the situation would be instantly corrected and an acceptable design still would be obtained in the same program execution. The capability for entering (or deleting) constraints online as the computations are in progress is a remarkable tool for enhancing the creative instincts of the designer. In effect, he would be molding the structure almost as a piece of sculpture, pulling here, pushing there, in order to fit spatial requirements and esthetic inclinations. This would be accomplished with the freedom from worry that is afforded by knowing that when he is finished with the kneading and shaping and the designed has converged to a final configuration, all the stress and deflection limitations will be satisfied. In

fact, the most critical ones will be satisfied exactly, for this is the highly desirable result that mathematical programming accomplishes.

Interaction need not only strengthen creativity, it can improve the efficiency of the program execution as well. The rate of convergence of a mathematical programming problem in general varies inversely with some higher order power of the number of variables involved, although the figures cannot be determined because so much depends on the characteristics of the particular problem which is being solved. (Occasionally, even a large problem can snap into a final solution with amazing speed.) If, however, as is possible when MP solution progress is being visually monitored, the designer notices that one or several of the parameters are not changing much from cycle to cycle, he easily could freeze the current value by using one of the interrupt devices, thus effectively switching the status of the parameter from that of a problem variable to that of a constant. The mechanics of doing this mathematically are actually quite simple, involving nothing more than setting the elements in column j of the left-hand side coefficient matrix of the associated linear programming problem (K) equal to zero for each variable j which is to be so altered, as well as the coefficient of the same variable which appears in the linearized objective function (14). Removing even just a few variables from the math programming problem in this manner quite often has a dramatic effect on the rate of convergence.

Existing hardware and software, or at least aspects of these that are in advanced states of development, hint at even some more alluring capabilities which could be made a part of a generalized system. For example, there are, in almost every graphics system being used, capabilities for rotating about any axis the configuration which is being displayed, so that the designer could visually inspect every portion of the generated system, and for magnifying portions thereof in order to clarify details. Hardware is even now commercially available which will respond vocally according to programmed instructions, so that communication between the secondary level decision maker (machine) and the primary decision maker (man) need not be entirely visual.

Although several, in fact the most critical, of the components shown in Fig. 1 are operating quite smoothly, the total system as described is not yet an operating reality; too many of the additional capabilities that would make it so have just arrived on the scene. But sufficient progress is being experienced with each of these so that a drawing together of all of them into the mosaic necessary for total system capability appears to be merely a matter of man hours.

ILLUSTRATIVE EXAMPLES

Several working applications have been developed in order to aid in developing the system. Even though they were not made particularly sophisticated because of this role, they do serve to illustrate the nature of the design system and the potential that it presents, thus meriting some discussion.

In a first example (Fig. 2), a minimum weight simple beam is designed for a given shear and moment, to be welded from plates of 36-ksi steel. The system as presently constructed accepts range limits from the designer, or, in the absence of these, defaults to the limits of commercially available values if there exists such a set, such as for plate thicknesses. In the absence of a

standard set, such as for plate widths, the designer must supply range limits as problem data. The trial vector is either also given by the designer, or, as in this case, defaulted to the lower range limits, an approach which somehow seems to work quite well for a large variety of systems and which will probably be retained as a permanent feature.

Several prominent events are worthy of note in the ensuing iteration which was carried out algorithmically. For 20 cycles a wild oscillation in two of the parameters is observed, and is typical of the behavior that can be expected in even the smallest of nonlinear systems, such as the beam of this example. Two of the parameters remained tight against the lower range limit

Initial Conditions: Shear = 12 k, Moment = 720 in-k



Trial Design	Lower Range Limits	Upper Range Limits
b	2.0	10.0
t	.25	1.0
d	6.0	20.0
w	.25	1.0

Convergence Behavior

Cycle	b	t	d	w
1	2.0	.25	9.884	.25
2	2.0	.25	14.830	.25
3	2.0	1.00	7.125	.25
4	2.0	.25	14.775	.25
5	2.0	1.00	7.095	.25
6	2.0	.25	14.736	.25
7	2.0	1.00	7.073	.25
8	2.0	.25	14.708	.25
9	2.0	1.00	7.058	.25
10	2.0	.25	14.688	.25
11	2.0	1.00	7.047	.25
12	2.0	.25	14.673	.25
13	2.0	1.00	7.035	.25
14	2.0	.25	14.662	.25
15	2.0	1.00	7.033	.25
16	2.0	.25	14.654	.25
17	2.0	1.00	7.028	.25
18	2.0	.25	14.649	.25
19	2.0	1.00	7.025	.25
20	2.0	.25	14.644	.25
21	2.0	.583	11.123	.25
22	2.0	.933	11.387	.25
23	2.0	.992	12.855	.25
24	2.0	.999	13.159	.25
25	2.0	.992	13.236	.25
26	2.0	.992	13.236	.25
27	2.0	1.000	13.163	.25
28	2.0	1.000	13.164	.25
29	2.0	1.000	13.164	.25

FIG. 2.—GENERATION OF OPTIMAL BUILTUP BEAM

throughout the run. At the event marked A, the accelerator (4) mentioned previously was employed as an impulse to dampen the oscillation (Block L in Fig. 1), and it can be seen that the operation was quite effective in that the design then quickly converged to the solution shown in cycle 26. The flange thickness, however, at convergence was not commercially admissible, so an additional operation, denoted as event B, was invoked which rounded that thickness to the nearest admissible value, and recycled the process to further optimize on those parameters which were not so restricted. The final design

is admissible, the bending stress is almost exactly the specified limit of 22 ksi, and the weight is about five % less than that of the lightest rolled section that would suffice, although caution should be exercised in making such a comparison because of the welding costs that would have been incurred in building up a section. If the lower range limit on the web plate thickness had been lower, a lower weight than indicated would undoubtedly have been obtained.

This example was allowed to run to a conclusion without manual intervention. In a dual-mode decision-making environment, however, the designer would have noticed earlier by inspection, at least by the fifth or sixth cycle, that the iteration had entered a pathologic oscillation, and could manually have triggered the acceleration impulse long before the twentieth cycle, at which time it is normally programmed to occur (every 10 cycles thereafter). Or he might have noticed quite early in the iteration that the flange width was likely to be less than would be required for a certain attachment he had in mind and which he had thought of after the iteration had begun. In this case he could have raised that particular lower limit while the iteration was in progress, perhaps by placing the light pen on the outer boundary of the flange width and dragging it outward. Being able to carry out such measures as these make a lot of sense when one is dealing with a much larger system where it is difficult to anticipate undesirable practical or esthetic features that might result because of the design that is finally generated and be able to build appropriate constraints into the constraint set beforehand.

A significant advantage of being able to operate in this manner results from the fact that execution time is unnecessarily inflated if a large number of constraints never become active in any one of the cycles. These constraints still have to be carried along as computational overhead in the matrix operations that are involved in solving the nonlinear programming problem if they are initially a part of the problem. There might be considerable economic merit in being able to operate within a bare framework of constraints that are likely to be active in any given situation, and to be able to insert dynamically any additional constraints that are necessary into the structure of the problem because of violations that might be detected in any of the iteration cycles. For example, beams over certain lengths are hardly ever governed by shear, and it would be a waste of computational time to carry all possible code restrictions on shear alone in the framework of the nonlinear programming problem for a general beam design routine. If conditions become such that shear does happen to govern for a certain situation, the violation is easily detectable and the appropriate constraints can then be inserted without otherwise interrupting the progress of the design.

A second example which was formulated also to test the graphics monitoring capability (Block J in Fig. 1) demonstrates a simulator, albeit primitive, for a 10-mile stretch of open highway containing both an intersecting turnoff and a tunnel constriction. Vehicles are injected into the three end points of the system at various rates, and are seen to move along the simulated highway system as dots of light. In a practical situation the designer might notice visually that queues are forming at critical junctions in the system and take immediate action, as the simulation is proceeding, to change one or more design parameters in an attempt to correct the situation, permitting the simulation to continue almost without interruption. While systems using simulation in order to determine system response are not yet being

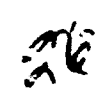
improved algorithmically by mathematical programming, the broad framework of the system of Fig. 1 can still be employed, particularly the graphics and data transfer capabilities.

CONCLUSIONS

As nonlinear programming algorithms improve with respect to scope and reliability, an increasing amount of the design task will be consigned to the computer and the designer can perform more strongly the role of policy maker in the design process. Developments contributing to this changeover recently include developed formula manipulation languages which can produce exact differential expressions of large nonlinear functions and cathode ray tube terminals which permit on-line modification of the mathematical programming problem as is going about the task of producing the design. The computer is then seen to assume a more active role as a partner in the man-machine design team, freeing the designer to exercise his creative inclinations on a higher level as it, the computer rather than the designer, guides the design in optimal directions that do not violate imposed constraints.

APPENDIX.—REFERENCES

1. Chai, J. W., "Detailed Design of Structural Frames by Nonlinear Programming," thesis presented to the University of Missouri-Columbia, College of Engineering, at Columbia Mo., June, 1970, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.
2. Douty, R. T., "Optimization of a Two-Span Cover-Plated Steel Beam," *Computers in Engineering Design Education-Civil Engineering*, Vol. III University of Michigan College of Engineering, April, 1966.
3. Douty, R. T. and Chai, J. W., "A Discipline for Generating Practical Optimal Structural Designs," *Proceedings of the Sixth Annual Design Automation Workshop*, sponsored by SHARE, Association for Computing Machinery Institute of Electrical and Electronics Engineers at Hotel Carillon, Miami Beach, June 8-12, 1969.
4. Douty, R. T., "An Algorithm for Large, Nonconvex, Discontinuous Nonlinear Programming Problems (abstract)," *Digest Record of the 1969 Joint Conference on Mathematical and Computer Aids to Design*, Sponsored by Association for Computing Machinery, Society for Industrial and Applied Mechanics, Institute of Electrical and Electronics Engineers at the Anaheim, Calif. Convention Center, October 1969.
5. ECAP/360-Electronic Circuit Analysis Program, 360D-164.001, (Local IBM Branch Offices).
6. IBM System/360 Operating System PL/I(F) Programmer's Guide, Form C28-6594, (Local IBM Branch Offices), 1966.
7. Kelley, J. E., "The Cutting Plane Method for Solving Convex Problems," *Society for Industrial and Applied Mathematics Journal*, Vol. 8, No. 4, December 1960.
8. Lewis, A. D. M., "Optimal Design of Structural Steel Framing for Tier-Type Buildings," *Computers in Engineering Design Education-Civil Engineering*, Vol. III University of Michigan College of Engineering, April 1966.
9. Logcher, R. D., et al., *ICES STRUDL-I Engineering User's Manual*, Dept. of Civil Engineering, MIT, Cambridge, Mass. September 1967.
10. Moses, E., "Optimum Structural Design Using Linear Programming," *Journal of the Structural Division, ASCE*, Vol. 90, No. ST6, Proc. Paper 4163 December, 1964, pp. 89-104.
11. Niyogi, P., Application of Computer Graphics Techniques to Civil Engineering Design Prob-

- lems; Transportation Engineering. *The Towne School of Civil and Mechanical Engineering Report*, University of Pennsylvania, Philadelphia, Pennsylvania, October 1967.
12. Reinschmidt, K. R., Cornell, C. A. and Brothie, J. F., "Iterative Design and Structural Optimization," *Journal of the Structural Division, ASCE*, Vol. 92, No. ST6, Proc. Paper 5015 December 1966, pp. 231-318.
13. Schilling, C. H., "Optimal Design of a Timber Warehouse Floor," *Computers in Engineering Design Education-Civil Engineering*, Vol. III University of Michigan College of Engineering, April, 1966.
14. *Specification for the Design, Fabrication & Erection of Structural Steel for Buildings*, American Institute of Steel Construction, New York, N.Y., 1969.
15. Tobey, R. et al., *PL/I-FROMAC Interpreter, MOD 03 J.00M*, Contributed Program Library, IBM Program Information Department, Hawthorne, New York, October, 1967.
- 

7837 INTERACTIVE COMPUTER GRAPHICS IN DESIGN

KEY WORDS: automation; computers; design; drawings; graphic methods; nonlinear programming; structural engineering

ABSTRACT: Developments in computing software and hardware make it possible to consign more of the design labor to the computer than has been possible, and permit the designer to exercise more fully his role as policy maker. Formula manipulation computer languages which can be used to produce exact differentials of very large nonlinear expressions and interactive graphic terminals which make it possible to modify on-line the nonlinear programming problem representing a design situation permit structural designs to be generated by the computer which are complete in all respects, where all design parameters necessary to define the configurations can be used, and where building code provisions are used unchanged as the set of constraints. The architecture of a man-machine interactive system is presented which allows the designer to effectively mold the design with a light pen as it is being evolved by the computer with the NLP algorithm and which appears in effect as an animation on the scope. A case study of an optimally proportioned beam is given in tabular form.

REFERENCE: Daulty, Richard, and Shore, Sidney, "Technique for Interactive Computer Graphics in Design," *Journal of the Structural Division, ASCE*, Vol. 97, No. ST1, Proc. Paper 7837, January, 1971, pp. 213-228.

A PHYSICAL PROPERTY INFORMATION SYSTEM
FOR UNDERGRADUATE EDUCATION

by

Daniel S. Poznanovic* and Warren D. Seider

The School of Chemical Engineering

and

The Moore School of Electrical Engineering

University of Pennsylvania
Philadelphia, Pennsylvania 19104

November, 1970

ABSTRACT

A physical property information system, including data base and estimation routines for providing property values directly to a FORTRAN program, has been implemented. The system allows for the development of chemical engineering programs that need not be altered with chemical mix and/or property estimation procedures. The system includes an inquiry-response interface for communication from a typewriter terminal. Property values can be graphed and tabulated interactively at the request of the engineer. In addition, property data can be stored interactively in correlation coefficient and tabular form for subsequent estimation of property values using correlation algorithms or interpolation procedures.

The role of the physical property information system in education is discussed with emphasis on the material and energy balance course and the preparation of more general process analysis, simulation, and design programs on the senior and graduate levels.

* Moore School of Electrical Engineering only.

INTRODUCTION

Physical property information systems are related to computer-aided process design, as stoichiometry and thermodynamics are related to chemical engineering education. Whereupon, it follows that as computer-aided methods are adapted by chemical engineering educators for analysis and design purposes, physical property (among other) information systems will play an increasingly significant role in courses such as stoichiometry and thermodynamics.

First, however, it is for us to improve upon computer-aided process analysis, simulation and design methods, and to discover courses and problem types wherein these methods improve the quality of education. Several approaches including curve-fitting process data, steady-state and dynamic process unit simulation, material and energy balancing chemical processes with recycle, and optimization of process unit designs have been examined.^(1,2,3) And, to these ends, the role of new man-machine interfaces, time-sharing and video display terminals, and of information systems are being considered.^(4,5,6) For to improve computer-aided process design methods, interactive information systems are an important ingredient.

The purpose of this paper is to describe a physical property information system (PPIS) for computer-aided process design computations in undergraduate education.⁽⁷⁾ The evolution of PPIS will be discussed with emphasis placed upon the reactions of student users at various stages of development. Naturally, only the salient features of PPIS are described; the PPIS User's Manual should be consulted for details.⁽⁸⁾

PHYSICAL PROPERTY VALUES FOR CHEMICAL ENGINEERING APPLICATION PROGRAMS

Computer programs for chemical engineering applications often require thermophysical property values for pure substances and chemical mixtures. A common approach to supplying property values is to prepare property data cards that are read by the program. Often the property values are assumed to be constant for the temperature and pressure ranges under consideration. When this assumption cannot be made, that is, when independent variable dependence is significant, property estimation procedures are required. Often the estimation procedures are coded into the application program with correlation coefficients, tables and constants being read from cards by the application program.

A more general approach for providing property values is to separate estimation algorithms from the application program in the form of subprograms that have access to data necessary for property value estimation. The application program calls upon the property estimation subprograms, rather than reading data from cards and estimating property values. This approach is common today, especially when property estimation is required for material and energy balance and design computations. (2,9,10,11)

These approaches to supplying property values restrict the applicability of the engineer's program. When a program requiring property values is prepared, call statements referencing estimation subprograms by name are coded into the program, directly linking the program with a specific set of estimation subprograms.

Often, changes in mixture components and/or temperature and pressure ranges require different estimation routines. When application programs must be modified frequently, general process material and energy balance, simulation and design program libraries are difficult to establish and maintain. High operating and maintenance costs help to explain why few truly general purpose program libraries are widely used today.

It has been our experience that general purpose programs for material and energy balancing, simulation, and design of process units, among other applications, are important to the chemical engineering educator whose time is not well spent writing and debugging or modifying programs. Hence, an important design goal for our PPIS has been to reduce modifications, due to variations in physical property estimation methodology, in an otherwise general purpose application program library.

For the sake of completeness, we note that no mention of APPES, the AIChE Physical Property Estimation System,^(12,13) and other such physical property information systems has been made because these systems are not conveniently interfaced with application programs, if at all. However, we have incorporated several APPES property estimation procedures in our PPIS library.

PHYSICAL PROPERTY INFORMATION SYSTEM PROTOTYPE

A principal objective of our physical property information system has been to allow the engineer to prepare application programs

requiring no modifications when the components in a mixture are altered and when new and different property estimation techniques are required.

One method of eliminating program modification is to separate the "variant" and "invariant" information required to request a property value. PPIS separates variant and invariant information as follows. Only invariant information such as the identity of the physical property (for example, heat capacity), the names of program variables that contain stored values for the independent variables (for example, T and P representing temperature and pressure), and the name of the program variable that contains mole fraction values for each mixture component (for example, X), are incorporated into the application program's request for property values. Variant information such as the identity of components in the mixture, and the identity of property estimation methods are specified separately from the application program. Invariant information is supplied during program preparation, while variant information is supplied just prior to each execution of the application program.

Another objective of our physical property information system has been to allow easy storage and retrieval of physical property data. As compared with other information systems, data retrieval is complicated by the need to also retrieve and apply estimation procedures for estimating the requested property values. Our "data base" is designed to store data in the form of constants, correlation coefficients, and tables. The estimation program library contains programs that combine correlation coefficient or tabular data with independent variable and

mole fraction specifications to compute "property values". The former programs allow for estimation by correlation, the latter by interpolation. Other estimation programs combine a sequence of estimated property values to produce a single value. For example, when estimating a mixture enthalpy, one procedure would combine zero pressure enthalpy and enthalpy pressure-correction estimates.

Retrieval of Property Values

PPIS has been designed to provide property values for pure chemicals and mixtures when requested during execution of FORTRAN programs through calls upon retrieval routines. The retrieval routines are FORTRAN functions and subroutines that retrieve property data from the data base, and call upon property estimation routines to compute requested property values.

Three retrieval routines are available for use in an application program. The function PPCP is used to request pure substance property values. Two retrieval routines are used to request mixture property values. The routine PPCF is a function subprogram that is used to request a single property value for a mixture (for example, molal average heat capacity). The routine PPCS is a subroutine that is used to request a property value for each component of a mixture (for example, mixture equilibrium coefficients - K values).

The standard requests for property values using the PPCP, PPCF and PPCS retrieval routines are:

PROP = PPCP(MP,V1,V2,INDEX,IC) - Pure substance
property values

PROP = PPCF(MP,V1,V2,X,IC) - "Average" mixture
property values

CALL PPCS(MP,V1,V2,X,RES,IC) - Mixture component
property values

The arguments are:

MP - property code (for example, 401 = vapor enthalpy
given temperature, pressure, and composition, or 305 =
equilibrium coefficients, given temperature, pressure,
and composition)

V1 - value of the first independent variable, a real number
or real variable (for example, temperature T)

V2 - value of the second independent variable, a real
number or real variable (for example, pressure P)

INDEX - chemical index number, an integer or integer variable
(see component identification table discussion below)

X - mole fraction values stored in a singly subscripted
variable for the chemicals indexed in the component
identification table

RES - estimated property values (results) stored in a singly
subscripted variable. Each member of the array contains
a property value for each component listed in the
component identification table.

IC - completion code variable, an integer variable. Retains
zero value when unusual circumstances do not occur during
data retrieval and estimation; otherwise, is set unequal
to zero.

Figure 1 contains a list of physical property codes currently available in PPIS. Associated with each property code are two independent variables. For example, 401 is the code for super-heated vapor enthalpy given temperature (the first independent variable) and pressure (the second independent variable) as well as compositions.

Figure 2 contains a list of chemicals for which some property data is currently stored in our PPIS. Note that a data base component number has been assigned to each chemical. Just prior to executing an application program, the engineer furnishes PPIS with a component identification table (see Figure 3). This table contains variant information that identifies for PPIS the chemicals in a mixture, or in a list for which pure chemical physical properties are needed. Each chemical in the table is assigned an index number so as to order the chemicals in the mixture or list of pure chemicals.

Figure 4 illustrates use of the PPCF retrieval routine in an application program. The PPCF routine is used to obtain the enthalpy of the five component mixture specified in the component identification table. The property code is 400 for enthalpy given temperature, T, and pressure, P. The mole fractions for the five component mixture are stored in the array X.

Physical property data and estimation procedures often differ with phase. For this reason, we have indexed the property codes such that those ending with the digit 1, 2, or 3 refer to vapor, liquid, or solid, respectively. In addition, property codes ending in 0 refer

to a phase unspecified property.

When the property code supplied to a retrieval routine is phase unspecified, a phase determination is performed. Figure 5 summarizes the phase determination capabilities of the retrieval routines. At present, phase determination is carried out only for properties whose independent variables are temperature and pressure. Mixture liquid and solid phases are not distinguished, and all liquids are assumed miscible. Further, it is the responsibility of the engineer to avoid requesting single phase property values, such as density and compressibility factor, for a two phase mixture.

Several property estimation procedures are currently available in the PPIS library. These are summarized in Figure 6. New estimation procedures are added to the library by the system librarian as required in our course work. It should be recognized that the scope of a library for educational purposes need not be as comprehensive as a library for industrial design purposes.^(9,10,11) In part, this is due to limited university resources for programming and testing estimation programs, or converting estimation procedures prepared by others into PPIS terminology. Currently, it is not possible for the engineer to easily enter his own estimation procedures without knowledge of PPIS internal details (Chapter 6, User's Manual⁽⁸⁾). Steps are being taken to simplify the installation procedure for estimation programs.

The estimation procedures combine data in the data base to estimate property values as illustrated in Figure 7. Commonly,

estimation procedures interpolate tables, or compute property values using correlation formulac, or both. Some estimation procedures combine other property values, as well; for example, an estimation procedure for K-values combines a liquid fugacity value with vapor fugacity and activity coefficient values, and so on.

Data Base and Data Storage

Having considered the retrieval routines for use in a FORTRAN program, we turn next to storage of data in the data base. Property data is stored in the form of data records. Data records contain property data characterized by key words. A data record contains seven key words and the data itself:

- 1) Property code,
- 2) Contributor code,
- 3) Validity ranges of two independent variables,
- 4) Maximum expected error,
- 5) Estimation routine number,
- 6) Data type,
- 7) Component code(s), and
- 8) Property data.

Four sample data records are illustrated in Figure 8. The first three data records contain heat capacity data for H_2O over different temperature ranges and $0.5 \leq P \leq 2.0$. The fourth data record contains liquid density data for the mixture, methane, ethane, propane, n-butane, and n-pentane, for $400 \leq T \leq 490^\circ R$ and $0.5 \leq P \leq 2.0$ atm.

The data record key words deserve some explanation. The property code identifies the physical property and its associated independent variables. The contributor code identifies the individual responsible for storing the data record in the data base. The validity ranges delimit the high and low limits for independent variable values within which the data record applies. For example, the vapor heat capacity data in the second data record (Figure 8) is used only when temperature falls within $672-3240^{\circ}\text{R}$ and the pressure within $0.5-2.0$ atm; data in the third data record is used when $3240 \leq T \leq 6840^{\circ}\text{R}$ and $0.5 \leq P \leq 2.0$ atm. The maximum error is the upper-bound estimate of the error when using the data record (within the independent variable validity ranges). The estimation routine number identifies the estimation routine for computing a property value using the data contained in the data record. For example, in the second data record (Figure 8) routine number 15, a third degree polynomial correlation program, is specified to compute vapor heat capacities using the data stored in the data record. In the third and fourth data records routine number 14, a linear interpolation program, is specified to interpolate the tabular data stored in the data record. And, in the first data record no estimation program is specified, since a single constant value for liquid heat capacity holds over the ranges, $492 \leq T \leq 672^{\circ}\text{R}$ and $0.5 \leq P \leq 2.0$ atm. The data type identifies the nature of the data stored in a data record; either a single constant, several correlation coefficients, or tabular data. The component code(s) indicate the component(s) for which the property data is valid. Finally, the data itself appears.

The application program furnishes the retrieval routines with a property code, two independent variable values, and mole fractions. The component codes are supplied by the component identification table. When servicing a request for property values, the retrieval routines search the data base for acceptable data records. An acceptable data record:

- 1) Contains the specified property code,
- 2) Contains independent variable validity ranges that surround the supplied independent variable values, and
- 3) Contains data base component codes specified in the component identification table.

The retrieval routines select an acceptable data record and call the estimation routine identified in the data record. The estimation routine obtains property data from the data record and other property values produced by parallel requests to the retrieval routines. The retrieval routines return the requested property value(s) to the application program.

Data Retrieval Constraints

Often the retrieval routines can select from among several acceptable data records. These data records may differ in estimation routine, maximum percentage error, contributor code, and data. The retrieval routines select the first acceptable data record located in a search of the data base, unless directed otherwise.

FPIS has been designed to allow the engineer to specify one of several acceptable data records if he so desires. He records his

preferences just prior to execution of the application program in a "retrieval constraint table" (when there are preferences). The retrieval constraint table is another form of variant information that allows the engineer to specify, for any physical property, (1) the estimation routine, (2) the maximum allowable error, and (3) the data contributor. Figure 9 illustrates a retrieval constraint table. Observe that the table constrains PPIS to locate a vapor heat capacity (431) data record with maximum expected error less than 1.5%, using estimation routine 15, and contributor 142; when $672 \leq T \leq 3240^{\circ}\text{R}$ and $0.5 \leq P \leq 2.0 \text{ atm}$, the second data record in Figure 8 is located from among other possible alternatives. K-value and vapor density data records are to be located with maximum expected error less than 1%. In addition, vapor density data records are to use estimation routine 2. The data record for critical temperature is to have been furnished by contributor 4. Finally, data records for all other properties are to be located with maximum expected error less than 2%. When the constraint degree is not entered, upon failing to locate a data record that satisfies the constraints, PPIS will use any applicable data record and print a comment describing the action taken. When the constraint degree is absolute, *, PPIS prints a message describing the unavailability of the constrained data record and aborts.

A master data base containing approved property data is maintained by the property system librarian, and is available to all users. A private data base is provided for engineers that obtain property data from the literature or by experiment. The engineer enters this data

into his private data base. Only the system librarian may enter new data into the master data base. Whereas the engineer is free to add new data, update existing data, or delete data from his private data base using the PPIS storage routine. For security purposes, each user selects a password upon entering property data into his private data base for the first time. The password must be supplied to the system whenever modification of the private data base is attempted.

Material and Energy Balance Problem

PPIS has been used to provide enthalpy and phase equilibrium coefficient values, K-values, in a program prepared for flash separation. The program determines the flow rates, mole fractions, and temperature of vapor and liquid streams leaving a flash separator, illustrated in Figure 10. Data to the program includes feed flow rate, mole fractions, temperature and pressure, product pressure, and the rate of heat loss, Q.

The method of solution assumes that the liquid and vapor phases are separated perfectly without entrainment. Material and energy balance equations and phase equilibrium constraints are summarized below for an N_c component mixture:

Material Balance Equations -

$$Z_j F = Y_j V + X_j L \quad j = 1, 2, \dots, N_c$$

Mole Fraction Constraints -

$$\sum_{j=1}^{N_c} z_j = \sum_{j=1}^{N_c} y_j = \sum_{j=1}^{N_c} x_j = 1$$

K-Value Constitutive Equations -

$$K_j = K_j(T_v, P_v) \quad j = 1, 2, \dots, N_c$$

Phase Equilibrium Constraints -

$$y_j = K_j x_j \quad j = 1, 2, \dots, N_c$$

Enthalpy Constitutive Equations -

$$h_f = h_f \{ T_f, P_f, z_1, z_2, \dots, z_{N_c} \}$$

$$h_v = h_v \{ T_v, P_v, y_1, y_2, \dots, y_{N_c} \}$$

$$h_l = h_l \{ T_l, P_l, x_1, x_2, \dots, x_{N_c} \}$$

Energy Balance Equation -

$$h_f F = h_v V + h_l L + Q$$

Equipment Constraint Equations -

$$T_l = T_v$$

$$P_l = P_v$$

There are $3N_c + 9$ equations and $4N_c + 13$ variables; hence, $N_c + 4$ design variable values may be specified. The set of design variables selected is $\{ F, T_f, P_f, Z_1, Z_2, \dots, Z_{N_c-1}, P_v, Q \}$. The program determines all other variable values. Figure 11 illustrates the algorithm for solution of the equations. N_c and design variable values are read from cards, T_v^* is estimated, and K-values are computed. Next, the material balance equations are solved, stream enthalpies are computed, and the energy balance equation is used to estimate a new T_v . When the T_v convergence tolerance is satisfied, results are printed. Otherwise, a new guess value for T_v is prepared and the material and energy balance computations repeated.

Observe that the PPIS requests for mixture K-values (MP = 305) and enthalpies (MP = 401,402) do not change with chemical mix or estimation procedure. Hence, the program is a general flash program with respect to estimated physical property values.

Figure 12 illustrates a flash curve computed during execution of the algorithm in Figure 11. PPIS was constrained to use data records that provide for enthalpy estimation using routines ENTH, ENSV, ENSL, ENSHV, ENSHL and K-value estimation using routine KTABLE.

Experiences Using the PPIS Prototype

PPIS has been used to furnish property values in many process unit material and energy balancing programs, such as the flash unit program. It does achieve the design objectives to provide for mixture independent application programs and to allow storage and retrieval

of property data. Many general purpose programs have been prepared for use with systems such as U.P.PACER (University of Pennsylvania PACER) for total process material and energy balancing.⁽¹⁵⁾ General purpose absorber, stripper, and flash programs have been run as U.P.PACER subroutines when material and energy balancing a natural gas process, for example.

An important shortcoming of the PPIS prototype has been its inaccessibility to the engineer who wishes preliminary estimates of property values. In order to obtain property values, the engineer is compelled to write a FORTRAN program. This places an excessive burden upon engineers interested in obtaining first estimates of property values to be used in application programs, no matter how compact the FORTRAN program need be. In many cases, students prefer to use handbook values or roughly estimated values rather than to write and debug a "simple" FORTRAN program.

This shortcoming, coupled with trends in information system design toward interactive inquiry-response systems, led us to add capability for communication between PPIS and a typewriter terminal. These developments are described in the next section.

Other problems are encountered when PPIS uses long-executing estimation procedures; for example, the Chao-Seader method for estimating K-values (routine KVAL) and the Benedict-Webb-Rubin equation to estimate densities (routines BWRLD and BWRVD). Repeated execution of these estimation routines is expensive when short-cut methods are

unavailable. As a result, we are currently developing methods for curve-fitting data using spline polynomial methods. The spline coefficients will be stored in a data record for rapid interpolation during design calculations.

INQUIRY-RESPONSE (INTERACTIVE) PROPERTY INFORMATION SYSTEM

The PPIS prototype proved its utility especially after conversion to an interactive system that communicates with the engineer through a typewriter terminal. Soon afterward PPIS assumed the role of a sophisticated handbook. Students now use the inquiry-response features to examine property values conveniently, often in preparation for use of PPIS in application programs.

The PPIS prototype is comprised of FORTRAN IV programs that run on an IBM 360/75. It was converted to an interactive program for the RCA Spectra 70/46 in one man-month. The interactive PPIS is executed within the RCA Time-Sharing Operating System (TSOS). This operating system provides for telecommunication with teletypewriter terminals, IBM selectric typewriter terminals, and the RCA video data terminal (alphanumeric cathode ray tube).

Using the Inquiry-Response PPIS

After a telephone call to the RCA Spectra 70/46 computer, the engineer requests to use PPIS (during "log-on" sequence). PPIS initiates the conversation with the engineer by prompting the latter for information. PPIS types a message (in capital letters

on IBM selectric typewriters), skips to the next line, types an asterisk (*), and awaits the engineer's response (in lower case letters on IBM selectric typewriters). The first few lines of conversation are:

UNIVERSITY OF PENNSYLVANIA PHYSICAL PROPERTY INFORMATION SYSTEM

PLEASE IDENTIFY YOURSELF.

*j.b. omega

DO YOU NEED HELP? (YES OR NO)

*no

THE AVAILABLE COMMANDS ARE:

STORE..DEFINE..REVIEW..CONSTRAIN..CHANGE..RETRIEVE..HALT.

PLEASE TYPE A COMMAND.

PPIS requests that the engineer identify himself and questions whether assistance is required to use the system. When help is requested, PPIS types detailed instructions for using the system. Thereupon, PPIS lists the seven commands available (each will be described below) and requests the engineer to type a command.

The "define" command allows the engineer to define the components and compositions of a mixture for which property values are subsequently to be retrieved. After the engineer types the define command, the conversation proceeds as follows:

*define

HOW MANY COMPONENTS?

*5

INPUT THE COMPONENT IDS. (ONE PER LINE)

*methane

*ethane

*propane

*n-butane

*n-pentane

INPUT THE MOLE FRACTIONS.

* 0.10 0.17 0.25 0.38 0.10

PLEASE TYPE A COMMAND.

The components may be identified using either the component names or the data base component numbers listed in Figure 2. Mole fraction values are separated by spaces.

After the mixture has been defined, the "retrieve" command may be used to request property values. The four options available are illustrated in the conversations below, beginning with "simple" retrieval:

```
*retrieve
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE, GRAPH, NONE.
*simple
  INPUT THE PROPERTY ID.
*enthalpy
  INPUT THE TEMPERATURE DEG R AND PRESSURE ATM
* 450.0  5.0
  RESULT= -57523.50 BTU/LB-MOLE COMPLETION CODE= 0
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE, GRAPH, NONE.
```

By simple retrieval is meant retrieving a property value (or values) for a single pair of independent variable values. Note that a request for an enthalpy value (the property code = 400 may also be specified) requires that PPIS determine the phase (or phases) of the mixture at the independent variable values.

The simple retrieval option is used also to request a set of K-values for a mixture. Such estimates are especially useful when preparing to use the generalized flash program described earlier.

```
*retrieve
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE GRAPH, NONE.
*simple
  INPUT THE PROPERTY ID.
*equilibrium coefficients
  INPUT THE TEMPERATURE DEG R AND PRESSURE ATM
*450.0  5.0
  RESULT=  16.71063      2.287533      0.4604993      0.1016921
           0.2697054E-01
  UNITS:  NO UNITS      COMPLETION CODE= 0
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE, GRAPH, NONE.
```

The "table" retrieval option is used to request that a table of property values be prepared at various independent variable values.

For example,

```
*retrieve
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE, GRAPH, NONE.
*table
  INPUT THE PROPERTY ID.
*v.enthalpy
  ENTER TEMPERATURE DEG R RANGE AND NO. OF INTERVALS.
* 470.0 520.0 5
  ENTER PRESSURE ATM RANGE AND NO. OF INTERVALS.
* 5.0 5.0 0
  ANY OTHER PROPERTIES? (YES OR NO)
*no
  THE TABLE IS READY. DO YOU WANT IT PRINTED
  AT THE TERMINAL? (YES OR NO)
*yes
  PROPERTIES: 401
  PRESSURE: 5.00 ATM.

  TEMP. 401
  -----
  470.00 -57336.813
  480.00 -56949.844
  490.00 -56572.797
  500.00 -56203.918
  510.00 -55841.609
  520.00 -55484.414
  -----
  UNITS: BTU/LB-MOLE
  DO YOU WANT A GRAPH ALSO? (YES OR NO)
*no
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE, GRAPH, NONE.
```

Values for as many as five property types can be tabulated simultaneously as they vary with one of the two independent variables.

Or, values for a single property type can be tabulated as it varies with its two independent variables (in a two-dimensional table). The engineer is notified when the table is ready at which time he decides whether the table is to be printed at the terminal or on the line printer at the computing center. Finally, he is offered the option

to have a graph prepared of the tabulated property values. Note that no phase determination is performed by PPIS when the property type is specified as v.enthalpy (=401).

The "graph" retrieval option is used to request that a graph of property values be prepared at various independent variable values.

For example,

```
*retrieve
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE, GRAPH, NONE.
*graph
  INPUT THE PROPERTY ID.
*1.density
  ENTER TEMPERATURE DEG R RANGE AND NO. OF INTERVALS.
* 400.0 490.0 9
  ENTER PRESSURE ATM RANGE AND NO. OF INTERVALS.
* 20.0 20.0 0
  ANY OTHER PROPERTIES? (YES OR NO)
*no
  THE GRAPH IS READY. DO YOU WANT IT PRINTED.
  AT THE TERMINAL? (YES OR NO)
*yes
  PRESSURE= 20.00 ATM.
  PROPERTIES: 202
  SYMBOLS : *
  UNITS : LB-MOLE/FT3
```

	TEMP. DEG R	*
*	400.00	0.1778
*	410.00	0.1917
*	420.00	0.2060
*	430.00	0.2206
*	440.00	0.2355
*	450.00	0.2506
*	460.00	0.2660
*	470.00	0.2817
*	480.00	0.2976
*	490.00	0.3136

```
DO YOU WANT A TABLE ALSO? (YES OR NO)
*no
  CHOOSE A RETRIEVAL OPTION: SIMPLE, TABLE, GRAPH, NONE.
```

Values for as many as five property types can be plotted on a single graph as they vary with one of the two independent variables.

At this time, graphs as a function of one independent variable as abscissa and the other independent variable as a parameter cannot be plotted. The other options are similar to those for table retrieval.

The "constrain" command allows the engineer to direct PPIS to locate specific data records during subsequent retrieval requests. The following conversation enables the engineer to supply retrieval constraint table information interactively:

```
*constrain
  PLEASE SUPPLY THE FOLLOWING:
  PROPERTY ID.
*401
  CONTRIBUTOR:
*
  ROUTINE NO.:
*
  MAX. ALLOWED ERROR:
*5.0
  ANY MORE CONSTRAINTS? (YES OR NO)
*no
  PLEASE TYPE A COMMAND.
```

The above conversation instructs PPIS not to use data records for vapor enthalpy in which the maximum expected error is greater than 5.0%.

The "store" command allows the engineer to add a data record to his private data base; he may also modify or delete a data record. Using this command, the engineer can maintain a private data base including constant, correlation coefficient, and tabular property data. Upon receiving the store command, PPIS first requests key word information to identify a data record. PPIS searches the private data base for such a data record. When no such data record is located, PPIS

requests that the engineer type data values, as illustrated below.

Upon locating such a data record, PPIS offers the engineer the option of modifying or deleting any entry in the data record (not illustrated below).

```
*store
  PLEASE SUPPLY THE FOLLOWING:
  PROPERTY ID.:
*1.density
  CONTRIBUTOR:
*omega,j.b.
  ESTIMATION ROUTINE NO.:
*14
  VALIDITY RANGE FOR TEMPERATURE DEG R
*400.0 490.0
  VALIDITY RANGE FOR PRESSURE ATM
*15.0 25.0
  MAXIMUM EXPECTED ERROR (%):
* 4.0
  HOW MANY COMPONENTS?
*5
  TYPE IN THE COMPONENT IDS. (ONE PER LINE)
*methane
*ethane
*propane
*n-butane
*n-pentane
  DATA TYPE: (CONSTANT, COEFFICIENT, TABULAR)
*tabular
  HOW MANY VALUES OF TEMPERATURE DEG R?
*5
  HOW MANY VALUES OF PRESSURE ATM      ?
*0
  TYPE IN THE DATA.
*400.0 0.1778
*420.0 0.2060
*440.0 0.2355
*460.0 0.2660
*490.0 0.3136
  PLEASE TYPE A COMMAND.
```

The store command facilitates the storage of tabular values generated by the table or graph retrieve command for subsequent interpolation. In the illustration above, liquid density values estimated

using the Benedict-Webb-Rubin equation iteratively (see the graph retrieve command illustration), are stored for subsequent linear interpolation (estimation routine number 14). The speed of interpolation is especially important for iterative design computations at varying temperature values.

The other interactive PPIS commands, "review", "change", and "halt" are briefly described, but not illustrated. The review command causes PPIS to print (1) component names and mole fractions for the most recently defined mixture, and (2) a summary of the current retrieval constraints. The change command is used to modify selectively the components or mole fractions in a previously defined mixture. And, the halt command terminates an inquiry-response session.

Evaluation of the Inquiry-Response PPIS

The inquiry-response feature markedly increases the engineer's accessibility to PPIS and thereby demonstrates the utility of computer-based information systems for engineering purposes. The most obvious benefits follow. The inquiry-response PPIS provides students with a tool for estimation of mixture property values when handbook values do not exist or are not easily accessible. It provides a vehicle for property estimation using iterative algorithms that are difficult to implement manually. And, in so doing, it often provides more accurate property values in shorter time. The inquiry-response PPIS enables the engineer to scan graphs and tables of property values over independent variable ranges under study in application programs. Thereby,

the engineer may confirm the accuracy of bench-mark values and gain confidence in an estimation procedure's ability to reproduce physically known variations with the independent variable values.

Of course, the overhead cost associated with preparation of data for machine storage, day-to-day disk or tape rentals, and maintenance and execution of an estimation program library is considerable. One goal of our research, accompanying the goal to improve the utility of physical property information systems, is to reduce overhead costs. Some ideas are included in the following paragraphs.

The estimation of property values using iterative algorithms sometimes requires 0.1 CPU seconds on the RCA Spectra 70/46 computer. This time is attributed almost entirely to the iterative nature of the computations when compared with approximately 0.001 CPU seconds for table interpolation computations. For the occasional interactive query, the order of magnitude difference in estimation times, is not a critical factor. But for the designer, whose application programs require repeated estimates of property values, the CPU sec./estimate is significant, especially when the application program is a unit operation module for material and energy balancing using U.P.PACER. (15)

Consequently, it is often desirable to store in tabular form property values accurately estimated over a range of independent variable values, for subsequent high-speed interpolation. As an example, the store command was used earlier to store a table of liquid density values computed iteratively using the Benedict-Webb-Rubin

equation. To eliminate some of the data preparation (typing) overhead associated with the store command, work is in progress to automate the procedure for tabulating property values followed by direct storage in a data record (at the engineer's request upon satisfaction with the tabulated values). We are also installing spline polynomial curve-fitting algorithms that compute and store spline correlation coefficients for high-speed computation of property values.

A reliable and up-to-date public file of data records (master data base) and estimation procedures is important to the successful operation of PPIS. The PPIS programs must be maintained, new data records stored, old data records purged, and new estimation procedures installed and documented, among other routine information handling activities. For these purposes, a PPIS librarian is required; a person well-versed in property estimation methods and digital computation. Since research-oriented faculty and students are not inclined to assume librarian duties, we expect that engineering schools will soon turn to non-academic staff members to maintain the increasing number of information systems (the common practice in university libraries).

A recurring problem when estimating property values using alternate methods in overlapping independent variable ranges is "consistency". The consistency problem occurs when two estimation methods predict the same property value trends in an overlapping independent variable range, but property values differ over the range due to nearly constant errors in one or both of the methods. Consequently, a step change in

property values occurs when switching between methods, with the possibility of distortion in the application program results, especially when the application program model is based upon the rate of change of property values with independent variable values. The inquiry-response PPIS allows the engineer to scan graphs and tables of property value estimates near those to be used during subsequent execution of the application program. When inconsistency is observed, new data can be obtained and other estimation procedures invoked.

The consistency problem demonstrates the desirability of a report generation facility that lists all data records and estimation procedures used during execution of an application program. Such a report would summarize the frequency of usage of data records and estimation procedures, and their associated CPU times. It would identify unexpected use of data records or estimation programs due to unusual excursions in independent variable values or improperly specified constraints.

As mentioned previously, methods are provided for the engineer to enter new estimation procedures (see Chapter 6, User's Manual⁽⁸⁾). Currently these methods require knowledge of PPIS details not known to the average engineer. Hence, the engineer must use public library programs or enlist a PPIS expert to install a new estimation procedure. Methods for automatic translation of stand-alone property estimation procedures into PPIS terminology are under study. The approach used for automatic translation of FORTRAN unit balance programs into U.P.PACER programs is especially promising.⁽¹⁶⁾

For most application programs, not all data records in the data base and estimation procedures in the library are required; in fact, typically only a few are used. When this is the case, it is impractical to search all data records for the few that apply. Hence, we are implementing an extension to PPIS that enables the engineer to prepare a "customized" data base for each new application problem. For example, a student engaged in the design of an ammonia process will prepare a customized data base that includes property data for N_2 , H_2 , NH_3 , Ar, and CH_4 only. The customized data base is essential to the successful operation of PPIS on a small machine, where searching a large data base is impractical.

The inquiry-response PPIS has been used by several undergraduate students engaged in the preparation of general purpose material and energy balancing algorithms for absorption, stripping, heat exchange unit operations, among others. We plan to introduce the system to chemical engineering sophomores in thermodynamics and stoichiometry courses during the Spring semester. At that time, we should be better prepared to discuss the economics of using such an information system in the classroom.

SUMMARY AND CONCLUSIONS

The role of physical property information systems (PPIS) for use in chemical engineering application programs is discussed, with emphasis on the design and implementation of a PPIS for undergraduate course work. Two design objectives were specified for a prototype PPIS:

- 1) To allow the engineer to prepare application programs requiring no modifications when the components in a mixture are altered and when new property estimation techniques are required.
- 2) To allow the engineer to easily store physical property data for subsequent retrieval and estimation of property values using interpolation and correlation algorithms.

The paper discusses these objectives and presents the significant features of the prototype PPIS. A typical material and energy balancing algorithm for a flash separator is presented, with emphasis on the role of PPIS in providing for a more generalized unit operation program to be used in the sophomore stoichiometry course. Several shortcomings of the prototype PPIS are discussed; the most important being the requirement to write a FORTRAN program simply to obtain property values.

The inquiry-response interface to PPIS was designed and implemented to provide interactive communication through a typewriter terminal. In this way, the engineer may interactively study the performance of estimation procedures for computing property values over a wide range of independent variable values. And, the engineer may interactively store property data for subsequent interpolation or use with correlation programs. The paper discusses these objectives and presents the significant features of the inquiry-response PPIS. The role of the inquiry-response PPIS in the selection of property data and estimation procedures to be used with application programs is described. Finally,

the performance of the inquiry-response PPIS is reviewed, with emphasis on plans for reducing the cost of data preparation and management and for improving its utility in engineering computations.

We conclude that our physical property information system is a fore-runner of the numerous computer-based information systems that will play a significant role in computer-aided analysis and design computations. Several examples and discussions serve to demonstrate the role of PPIS in undergraduate education today and our expectations for PPIS in the near future.

ACKNOWLEDGEMENT

The assistance and services provided by the University of Pennsylvania Computer Center (IBM 360/75) and the Moore School Computing Center (RCA Spectra 70/46) are gratefully acknowledged. Partial financial support for Dan Poznanovic and many aspects of the work were provided by the Esso Education Foundation and the Air Force Office of Scientific Research, Arlington, Virginia. The authors appreciate the opportunity to conduct this research within the confines of the Moore School Information System Laboratory, under the direction of Professor Morris Rubinoff. The advice of ISL members and Chemical Engineering Calculation System Project members (too numerous to list) is greatly appreciated.

Physical Property Code		Independent Variables	
100	Density	T	P
101	- vapor	T	P
102	- liquid		
200	Fugacity Coefficient	T	P
201	- vapor		
202	- liquid		
216	Vapor Pressure	T	-
300	Activity Coefficient	T	P
302	- liquid		
305	Equilibrium Coefficients	T	P
400	Enthalpy	T	P
401	- vapor		
402	- liquid		
411	Enthalpy Pressure Correction - vapor	T	P
412	- liquid		
425	Zero Pressure Enthalpy	T	P
430	Heat Capacity	T	P
431	- vapor		
432	- liquid		
515	Bubble Pt. Temperature	P	-
516	Dew Pt. Temperature	P	-
517	Temperature	Enthalpy	P
518	Vapor Fraction	T	P
1001	Critical Temperature	-	-
1002	Critical Pressure	-	-
1003	Acentric Factor	-	-
1004	Solubility Parameter	-	-
1005	Molar Volume	-	-
1006	Molecular Wt.	-	-
1007	Melting Pt. Temperature	-	-
1008	Normal Boiling Pt. Temperature	-	-
1111	Critical Compressibility	-	-

Physical Property Codes

Figure 1

<u>Compound or Element</u>	<u>Data Base Component Number</u>	<u>Compound or Element</u>	<u>Data Base Component Number</u>
Hydrogen	1	1-Pentene	29
Methane	2	Cis-2-Pentene	30
Ethane	3	Trans-2-Pentene	31
Propane	4	2-Methyl-1-Butene	32
i-Butane	5	3-Methyl-1-Butene	33
n-Butane	6	2-Methyl-2-Butene	34
i-Pentane	7	1-Hexene	35
n-Pentane	8	Cyclopentane	36
neo-Pentane	9	Methylcyclopentane	37
n-Hexane	10	Cyclohexane	38
n-Heptane	11	Methylcyclohexane	39
n-Octane	12	Benzene	40
n-Nonane	13	Toluene	41
n-Decane	14	O-Xylene	42
n-Undecane	15	M-Xylene	43
n-Dodecane	16	P-Xylene	44
n-Tridecane	17	Ethylbenzene	45
n-Tetradecane	18	Ammonia	46
n-Pentadecane	19	H ₂ O	47
n-Hexadecane	20	Ethyl Alcohol	48
n-Heptadecane	21	Acetone	49
Ethylene	22	Nitrogen	50
Propylene	23	Oxygen	51
1-Butene	24	Carbon Monoxide	52
Cis-2-Butene	25	Carbon Dioxide	53
Trans-2-Butene	26	Air	54
i-Butene	27	Argon	55
1,3-Butadiene	28		

Data Base Component Numbers

Figure 2

<u>Chemical Index</u>	<u>Data Base Component Number</u>	<u>Component Name</u>
1	2	Methane
2	10	n-Hexane
3	5	i-Butane
4	4	Propane
5	20	n-Hexadecane

Component Identification Table

Figure 3

DIMENSION X(10) READ (5,10) NCOMP, T, P READ (5,10) (X(I), I = 1, NCOMP) . . . H = PPCF(400,T,P,X,IC) . . .			Segment of application program.
Chemical Index	Data Base Component Code	Component Name	Component Identification Table.
<u> </u>	<u> </u>	<u> </u>	
1	3	Ethane	
2	4	Propane	
3	6	n-Butane	
4	8	n-Pentane	
NCOMP=5	2	Methane	

Retrieval
Routine

Phase Determination Capabilities

PPCP

If temperature $\leq$ melting point, retrieve solid property value.

If pressure $\leq$ vapor pressure, retrieve vapor property.

If pressure $>$ vapor pressure, retrieve liquid property.

PPCF - PPCS

If vapor fraction = 0.0, retrieve liquid property.

If vapor fraction = 1.0, retrieve vapor property.

PPCF only

If $0.0 < \text{vapor fraction} < 1.0$, then two phases exist. Vapor and liquid mole fractions are computed, and vapor and liquid properties retrieved.

Result = $\text{VF} * \text{vapor property} + (1.0 - \text{VF}) * \text{liquid property}$,
where VF = vapor fraction.

Phase Determination

Figure 5

<u>Routine Name</u>	<u>Routine Number</u>	<u>Purpose</u>
<u>Thermodynamic Property Estimation</u>		
ZPK	3	Zero pressure enthalpy of a mixture
BUBTPL	4	Bubble point temperature of a mixture using Newton's method
DEWTPL	5	Dew point temperature of a mixture using Newton's method
BWRHPC	6	Enthalpy deviation due to pressure using Benedict-Webb-Rubin equation of state
ENSV	16	Enthalpy departure of superheated vapor mixture using a corresponding state correlation based upon the table in reference (14), p. 595
ENSL	17	Enthalpy departure of subcooled liquid mixture using a corresponding state correlation based upon the table in reference (14), p. 595
ENSHV	18	Enthalpy departure of saturated vapor mixture using a corresponding state correlation based upon the table in reference (14), p. 598
ENSHL	19	Enthalpy departure of saturated liquid mixture using a corresponding state correlation based on the table in reference (14), p. 598
VFRAC	7	Vapor fraction of a mixture by Chao-Seader method
VAPFRC	21	Vapor fraction of a mixture by material balance using K-values
FUGL1	8	Liquid fugacity coefficient for each component of a mixture using Chao-Seader-Grayson-Streed method
ACTL1	9	Liquid activity coefficient for each component of a mixture by Chao-Seader method

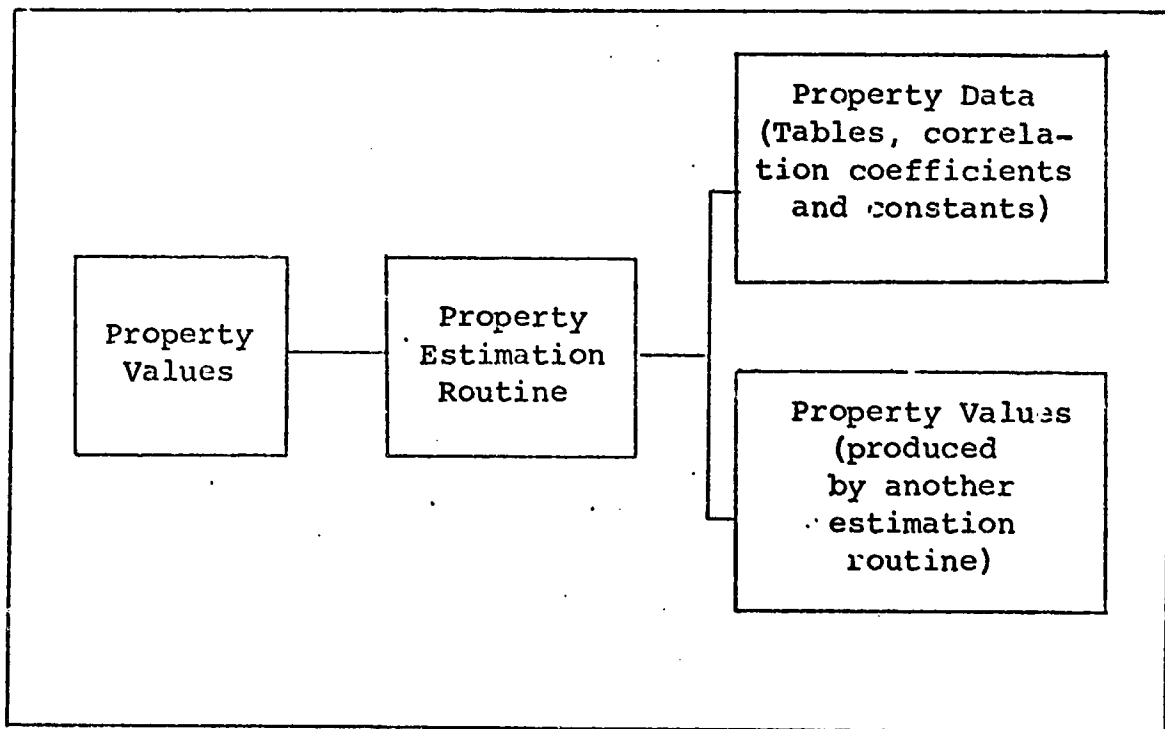
Property Estimation Procedures

Figure 6

<u>Routine Name</u>	<u>Routine Number</u>	<u>Purpose</u>
<u>Thermodynamic Property Estimation (cont.)</u>		
FUG	10	Vapor fugacity coefficient for each component of a mixture using Benedict-Webb-Rubin equation of state
KVAL	11	Vapor-liquid equilibrium coefficient for each component of a mixture by Chao-Seader method
KTABLE	20	Vapor-liquid equilibrium coefficient for each component of a mixture using a corresponding state correlation based on the table in reference (14), p. 441
ENTH	12	Enthalpy of a mixture by adding zero pressure enthalpy and enthalpy deviation due to pressure
TSUBH	13	Temperature of a mixture given the enthalpy by Newton's method
VPRESS	22	Vapor pressure of a component using the Antoine correlation
<u>Physical Property Estimation</u>		
BWRLD	1	Liquid density of a mixture using the Benedict-Webb-Rubin equation of state
BWRVD	2	Vapor density of a mixture using the Benedict-Webb-Rubin equation of state
<u>Interpolation Procedures</u>		
XINT	14	Linear interpolation of tabular data
<u>General Correlation Procedures</u>		
POLY3D	15	Third degree polynomial correlations

Property Estimation Procedures

Figure 6 (cont.)



The Role of Estimation Procedures

Figure 7

	Data Record 1	Data Record 2	Data Record 3	Data Record 4
Property	: liq. heat capacity	vap. heat capacity	vap. heat capacity	liquid density
Contributor	: 142	142	153	178
Validity range(1)	: 492-672°R	672-3240°R	3240-6840°R	400-490°R
(2)	: 0.5-2.0atm	0.5-2.0atm	0.5-2.0atm	15-25atm
Max. error	: 0.5%	1.0%	2.0%	2.0%
Estimation routine:	-	15	14	14
Data type	: constant	coefficient	tabular	tabular
Component(s)	: water	water	water	2, 3, 4, 6, 8
Data	: 1.0		T C	T ρ
		0.428	3240 0.656	400 0.1778
		1.42×10^{-5}	4000 0.701	420 0.2060
		3.88×10^{-8}	5000 0.743	440 0.2355
		-7.35×10^{-12}	6000 0.771	460 0.2660
			6840 0.782	490 0.3136

Sample Data Records

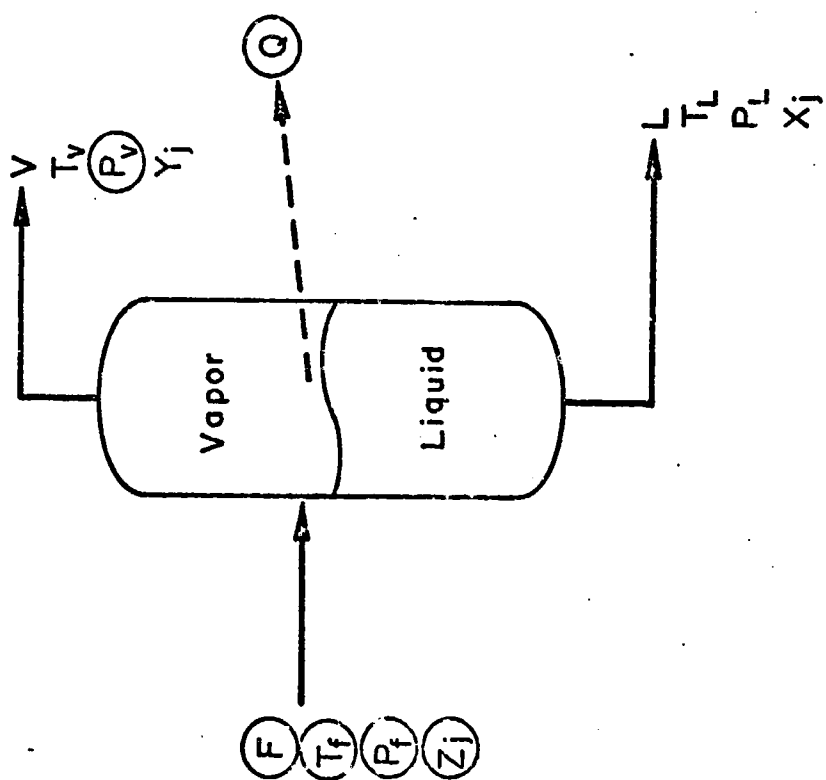
Figure 8

Property Code	Allowable Error	Estimation Routine No.	Contributor Code	Constraint Degree
Vap. heat cap.	1.5%	15	142	-
K-values	1%	-	-	-
Vap. density	1%	2	-	*
Crit. temp	-	-	4	-
-	2%	-	-	-

Retrieval Constraint Table

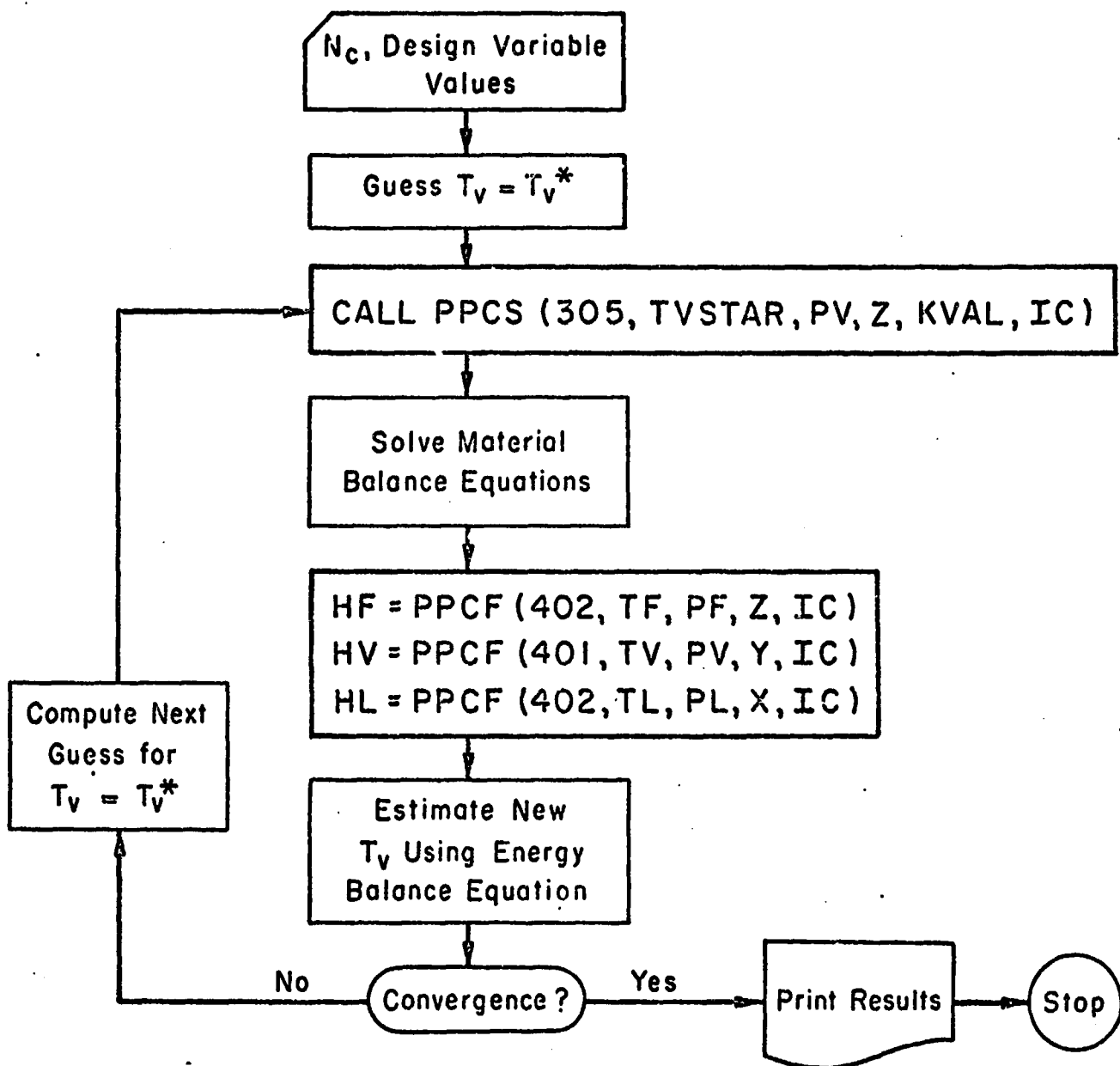
Figure 9

Note: Circled quantities
are specified



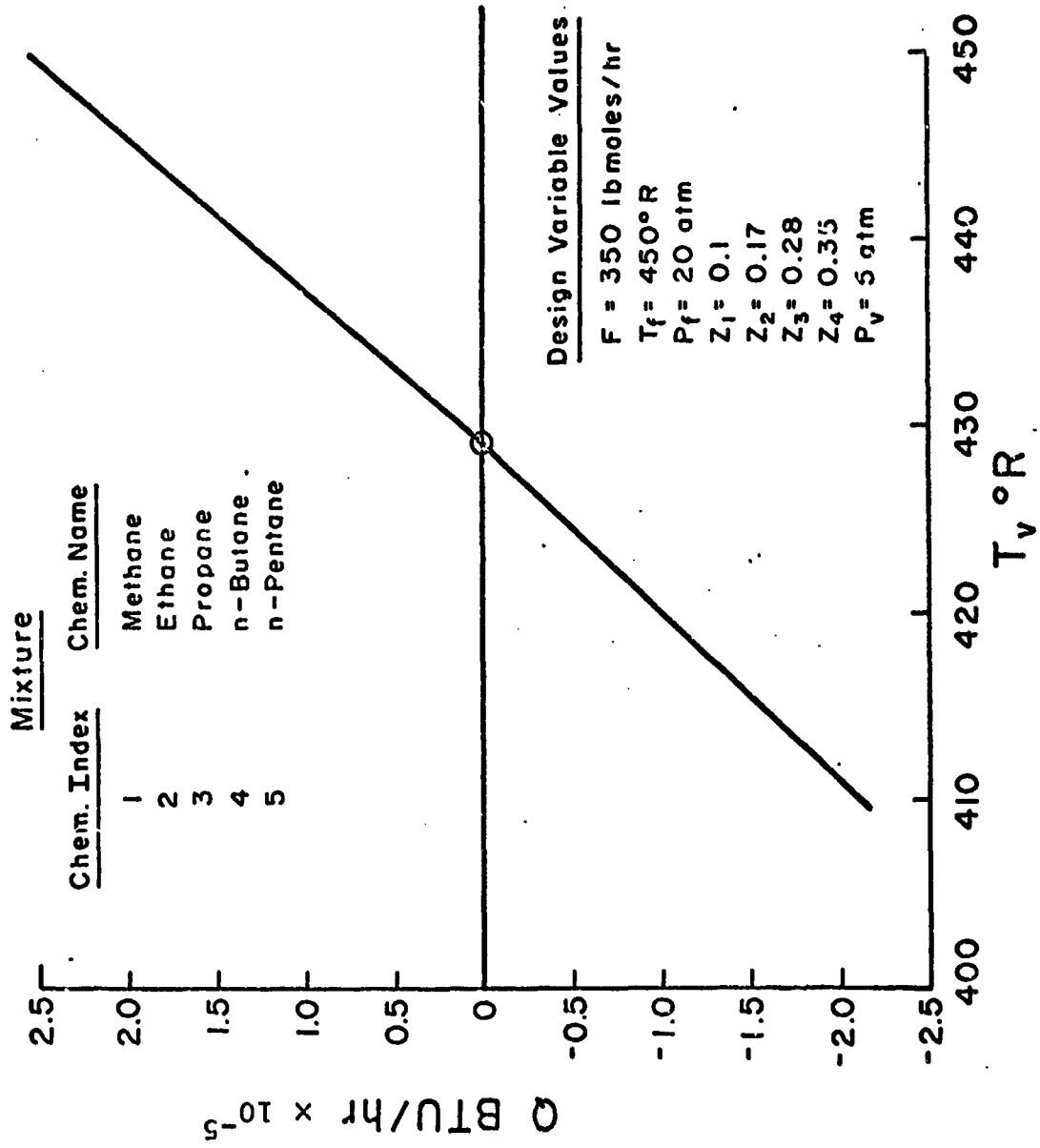
Flash Separator Schematic

Figure 10



Flash Separator Algorithm

Figure 11



Flash Curve
Figure 12

LITERATURE CITED

1. Crowe, C. M., et al., Chemical Plant Simulation, McMaster University, Hamilton, Ontario, Canada (1969).
2. Motard, R. L., et al., CHESS, Chemical Engineering Simulation System, Tech. Publ. Co., Houston, Texas (1968).
3. Carnahan, B., W. D. Seider and D. L. Katz, Computers in Engineering Design Education - Chemical Engineering, Vol. II, The University of Michigan, Ann Arbor, Michigan (1966).
4. Edwards, L. L., "Teaching Optimization: The Best of All Possible Approaches", Chemical Engineering Education, 4, No. 2 (1970).
5. Seider, W. D., "Time Sharing In Engineering Education", Engineering Education, 59, 5 (1969).
6. Seider, W. D., "The Student and His Information Needs", Engineering Education, 60, 5 (1970).
7. Poznanovic, D. S., An Information Storage and Retrieval System for Physical Properties of Chemicals, Master's Thesis, The University of Pennsylvania, Philadelphia, Pennsylvania (1969).
8. Poznanovic, D. S., User's Manual -- Physical Property Data-Base, Storage, Retrieval, and Estimation System, The University of Pennsylvania, Philadelphia, Pennsylvania (1969).
9. An Introduction to Flowtran - Flowsheet Simulation, Monsanto Company, St. Louis, Missouri (1970).
10. Halcon Physical Properties System, Halcon International, Inc., New York, New York (1969).
11. User's Guide - Chemical Engineering Calculation System -- GECEC, Information Service Department, General Electric Co., Bethesda, Maryland (1970).
12. AIChE Physical Property Estimation System, Arthur D. Little, Inc., available from AIChE office, New York, New York (1965).
13. Meadows, E. L., Jr., "Estimating Physical Properties: The AIChE System," Chem. Engr. Progr., 61, No. 5 (1965).
14. Hougen, O. A., K. M. Watson, and R. A. Ragatz, Chemical Process Principles, Part Two: Thermodynamics, Second Edition, J. Wiley, New York (1959).
15. Whittall, P. P. and W. D. Seider, User's Manual - University of Pennsylvania PACER, University of Pennsylvania, Philadelphia, Pennsylvania (1970).
16. Soylemez, S. and W. D. Seider, "The Chemical Engineer - Process Analysis and Design System Interface", presented at the Denver meeting of the AIChE, September, 1970.

LIST OF FIGURE CAPTIONS

Figure No.

1	Physical Property Codes
2	Data Base Component Numbers
3	Component Identification Table
4	Sample Application Program Retrieval Request
5	Phase Determination
6	Property Estimation Procedures
7	The Role of Estimation Procedures
8	Sample Data Records
9	Retrieval Constraint Table
10	Flash Separator Schematic
11	Flash Separator Algorithm
12	Flash Curve

AN INTERACTIVE FLOW CHART GRAPHICS SYSTEM

by

Peter L. Delaney, Jr.

INTRODUCTION

The purpose of the graphics system is to allow a user to draw a flow chart. The graphics system is interactive; that is, it communicates with the user while he is drawing, offering him a choice of several actions that may be performed. The system is designed to be sufficiently fast to keep up with the user; offering no noticeable delay between operations. In addition, the system is modular to allow for easy programming. And finally, the graphics system is implemented on a small computer so as to be inexpensive in operation.

The PDP-8 is the computer selected to house the graphics system. It is a small computer that has 8K of 12-bit words, a 32K disc, a small tape unit, and a 338 cathode ray tube (CRT) display. The 338 is a special purpose computer, controlled by the PDP-8, that executes a "display program" in the PDP-8's core memory.

The "display program" or "display file" resides in a part of memory set aside by the PDP-8 to be executed by the display. The display file is composed of a set of display instructions which tell the 338 how to draw. There are two states of operation, "control state" and "data state". In control state, the display program controls program flow and sets up display registers that contain drawing parameters (for example, beam intensity, drawing scale, and drawing mode). In data state, the display program uses the

data in the display file to position the cathode ray tube display beam. A few of the drawing modes are point, vector, and character. When in point mode and data state, the display will draw points at the coordinates specified in the display file. For example, a display program to draw a line from the point P1:(10,10) to the point P2:(100,100) at scale one and intensity five would be:

<u>Label</u>	<u>Instruction</u>	<u>Comment</u>
PROG1, /	SC1 INT+5	/SET SCALE =1, INTENSITY=5
	EDS POINT	/SET MODE="POINT", ENTER DATA STATE
	10	/Y - COORD
	10+4000	/X - COORD, ESCAPE TO CONTROL STATE
/		
	EDS VEC	/ENTER DATA STATE, MODE="VECTOR"
	100-10+4000	/ΔY=100-10=70 (OCTAL), INTENSIFY
	100-10+4000	/ΔX=100-10=70, ESCAPE
/		
	STOP	/STOP EXECUTION

*Note: All numbers refer to the octal number system.
For details, see the DEC-338 manual.

This display program, when executed, draws a vector from the point P1:(10,10) with components (ΔY=70, ΔX=70) once! In order to keep it on the screen for more than a fraction of a second (since the CRT phosphors relax rapidly), a "JUMP" instruction is used to transfer control to the beginning of the display file. For example:

<u>Label</u>	<u>Instruction</u>	<u>Comment</u>
PROG1,	SC1 INT+5	/SET SCALE AND INTENSITY
/		
LOOP,	EDS POINT	/ENTER DATA STATE "POINT"
	10	/Y - COORD
	4010	/X - COORD + ESCAPE
/		
	EDS VEC	/ENTER DATA STATE "VECTOR"
	70	/ΔY=70
	4070	/ΔX=70,ESCAPE
/		
	JUMP	/TRANSFER CONTROL
	LOOP	/TO ADDRESS "LOOP"

*Note: The display file must be re-executed frequently (about 30 cps) to avoid CRT flicker.

It should be noted that the "JUMP" instruction requires two words of memory; the first denotes the "JUMP" instruction itself, and the second indicates the address to which control is to be transferred. See Figure 1 for an illustration of the picture drawn by this program.

Another important feature of the 338 is its ability to execute calls to display subroutines. A "PJMP", or push jump instruction, is used to put the current display program address plus two onto a hardware push down stack and then transfer control to the address contained in the word following the "PJMP". To return from the display subroutine, a "POP" instruction is used to pop the push down stack and transfer control to the address retrieved from the top of the stack. For example, a display program to draw the three squares in Figure 2 would be:

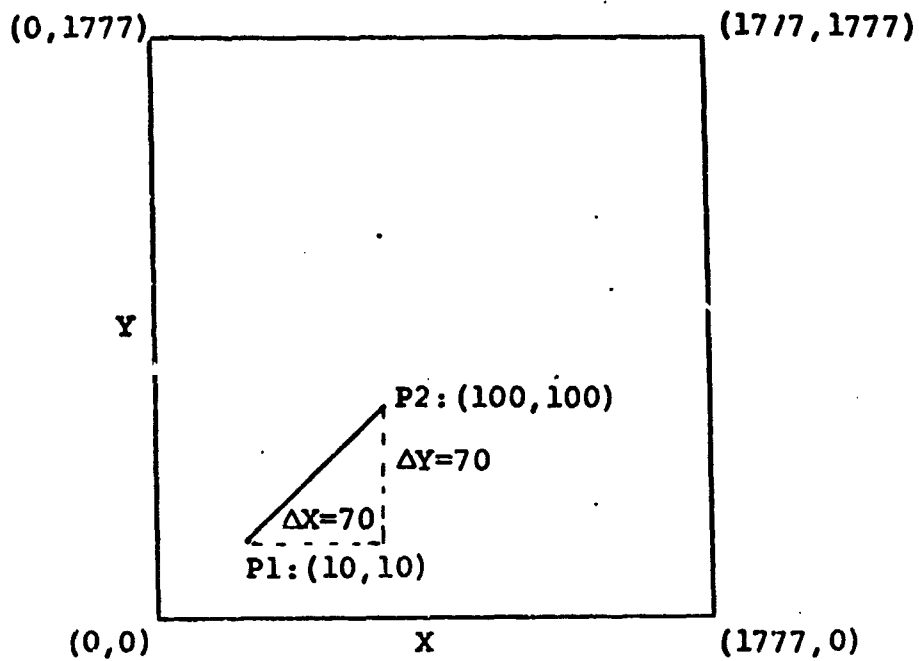


Figure 1

Picture Drawn by "PROG1"

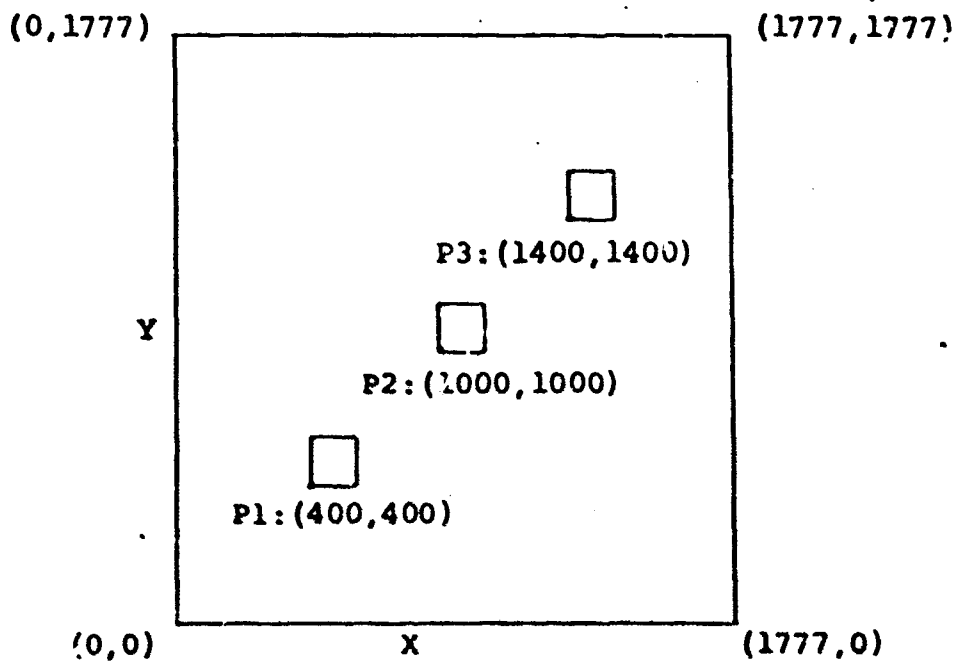


Figure 2

Picture Drawn by "PROG2"

<u>Label</u>	<u>Instruction</u>	<u>Comment</u>
PROG2,	SC2 INT+7	/SET SCALE=2,INTENSITY=7
/		
LOOP,	EDS POINT	/POSITION BOX NO.1.
	400	/Y - COORD
	400+4000	/X - COORD,ESCAPE
	PJMP;SQUARE	/CALL DISPLAY SUBROUTINE TO DRAW
		/A SQUARE.
/		
	EDS POINT	/POSITION SQUARE NO.2
	1000	/Y - COORD=1000
	1000+4000	/X - COORD=100+ESCAPE
	PJMP;SQUARE	/DRAW A SQUARE
/		
	EDS POINT	/POSITION SQUARE NO.3
	1400	/Y=1400
	1400+4000	/X=1400,ESCAPE
	PJMP;SQUARE	/DRAW A SQUARE
/		
	JUMP;LOOP	/REDRAW PICTURE
/		
/		
SQUARE,	EDS VEC	/DRAW 4 SIDES OF SQUARE
	4100;0	/LEFT ($\Delta Y=100, \Delta X=0$)
	4000;100	/TOP ($\Delta Y=0, \Delta X=100$)
	6100;0	/RIGHT ($\Delta Y=-100, \Delta X=0$)
	4000;6100	/BOTTOM ($\Delta Y=0, \Delta X=-100$)
	POP	/RETURN

*Note: 1) A semi-colon (;) is used to concatenate two display instructions on the same line.

2) Vector components with negative length are formed by adding 2000 to the magnitude of the component.

Clearly, the display file for a large flow chart is very large and complex. For this reason, a graphics monitor has been written to simplify display programming.

Before describing the graphics monitor, it is advisable to examine the graphics system as a whole and a few of its important components. See Figure 3. The light pen, as shown in the upper right hand corner of Figure 3, is an important input device in the graphics system. It is a finger operated shutter connected to a fiber optics tube which, in turn, is connected to a photo-multiplier tube housed in the 338. When the light pen is pointed at a line being drawn on the screen, the 338 receives a signal from the light pen. The 338 may then, if conditions permit, stop the display and inform the interrupt handler of the "light pen hit". The pushbuttons, also in the upper right hand corner of Figure 3, are a set of 12 buttons with light indicators which may be set on or off by either the interrupt handler or the user. The user program, shown on the far left of Figure 3, is one of a set of program overlays that call on the graphics monitor to perform functions requested by the user. For example, the user program, after a light pen hit has been recorded (by the interrupt handler), may call upon the graphics monitor to alter the display file. In this way, picture modification can take place interactively.

THE GRAPHICS MONITOR

The graphics monitor is the heart of the graphics system. Its purpose is to simplify programming of the 338. It is composed of two parts, the "interrupt handler" and the "display file monitor" (see Figure 3). The interrupt handler takes care of all I/O activity for the system; for example, loading program overlays, servicing pushbutton and light pen hits, and data phone communications, and will be discussed later.

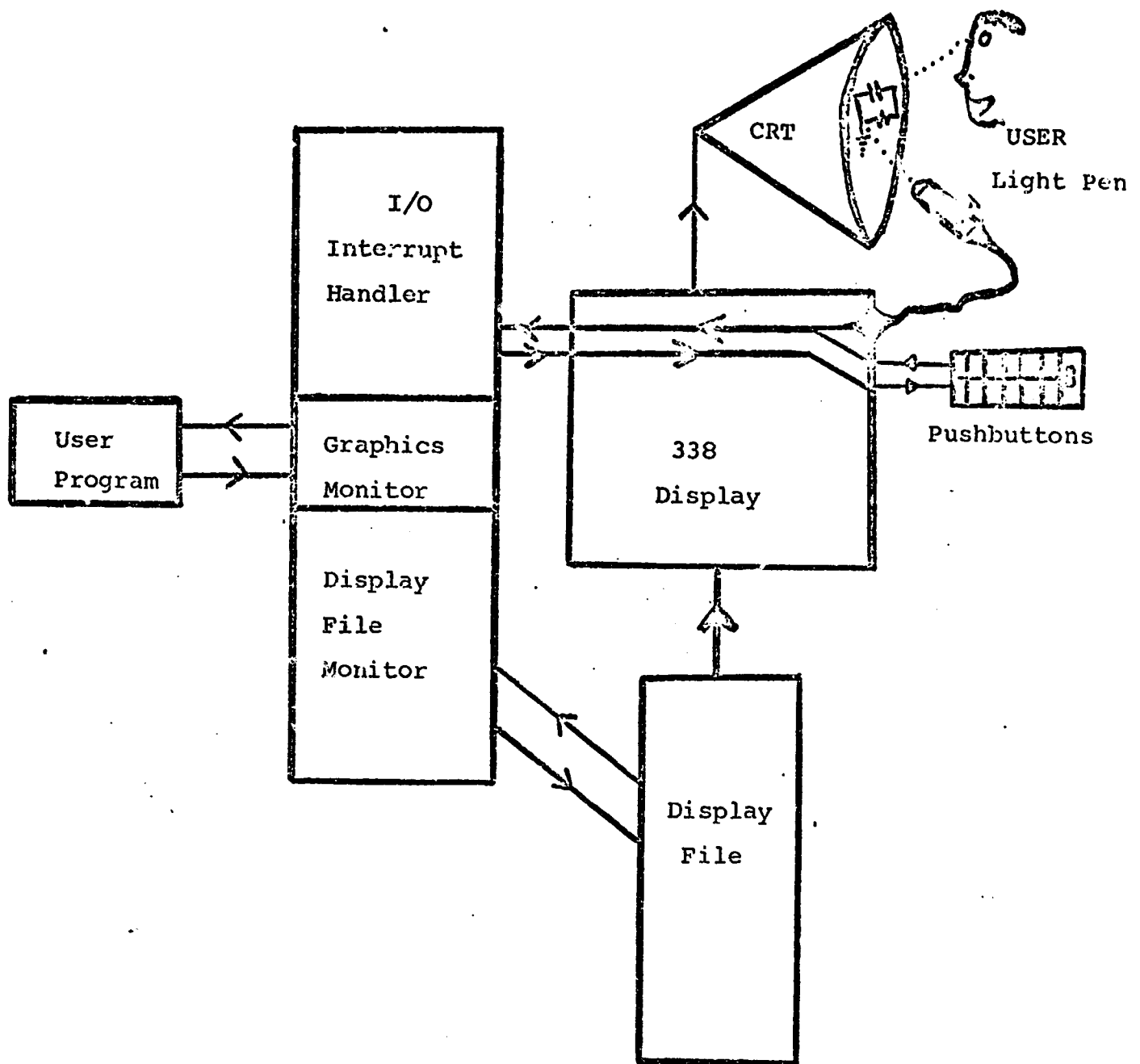


Figure 3
Graphics System Structure

Display File Monitor

The display file monitor handles all display file modifications in the graphics system. To do this efficiently, a display file structure has been designed containing three basic entity types:

- a) Nodes,
- b) Branches, and
- c) Devices.

A node is defined as a point on the screen with a shape (see Figure 4.1). A branch is defined as a vector between two nodes (see Figure 4.2). And a device is defined as a geometric shape with a set of relative positions for "terminal nodes" (see Figure 4.3). The three entity types when assembled form a flow sheet (see Figure 4.4). Any of these entities, node, branch, or device, may also have a "text label" positioned relative to that entity. A fourth entity type called a "light button", or "button" for short, also exists in the graphics system. A light button is a temporary entity that is not a part of the flow chart. It has a shape and a text label and is used to give instructions to user.

The display file is composed of "primary" and "secondary" blocks. Each primary block contains information pertaining to one entity. The primary blocks are linked together using "JUMP" instructions to enable entities to be added or removed from the display file easily, as illustrated below in Figure 5.

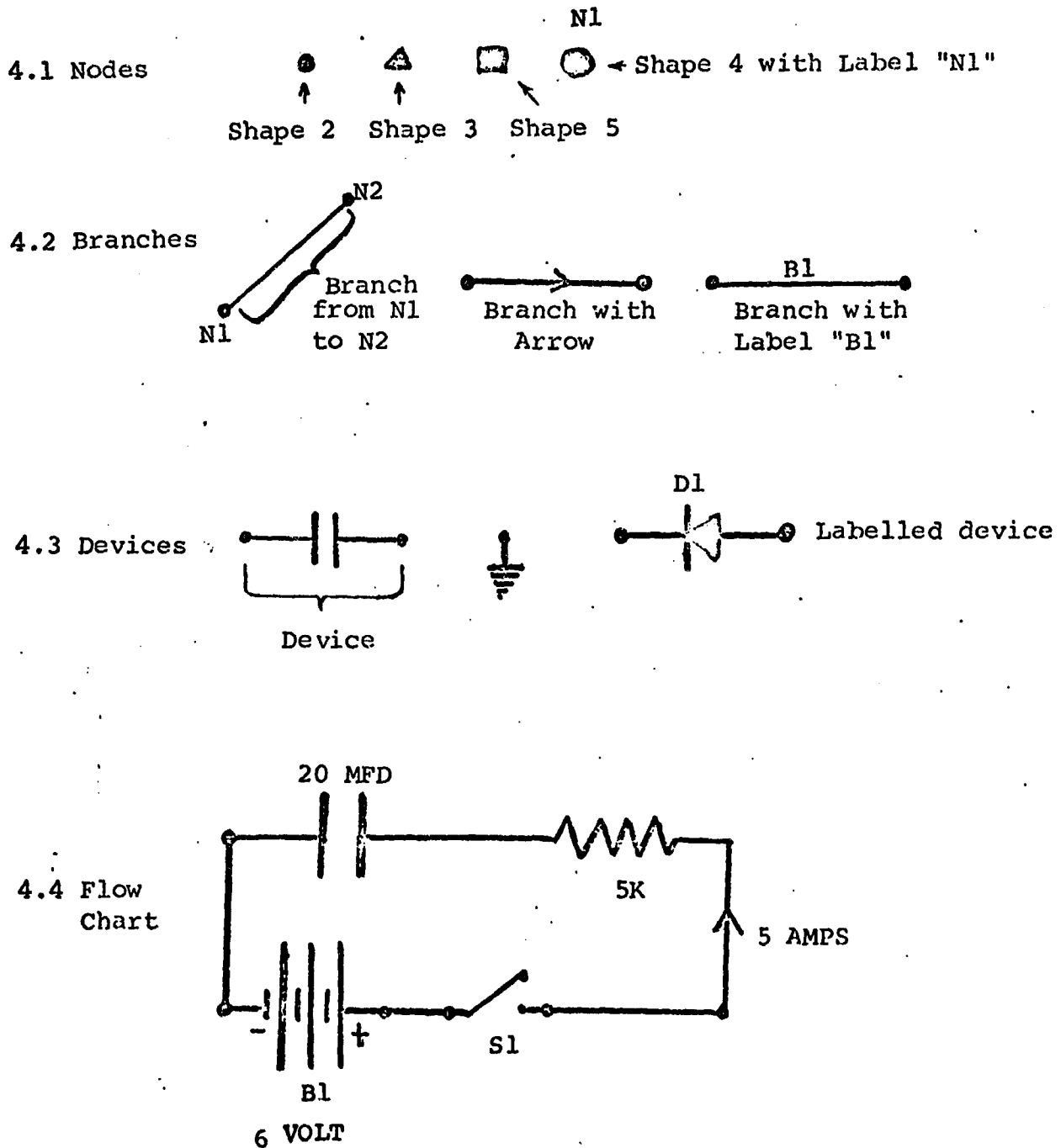


Figure 4
Electrical Network Entities and Flow Chart

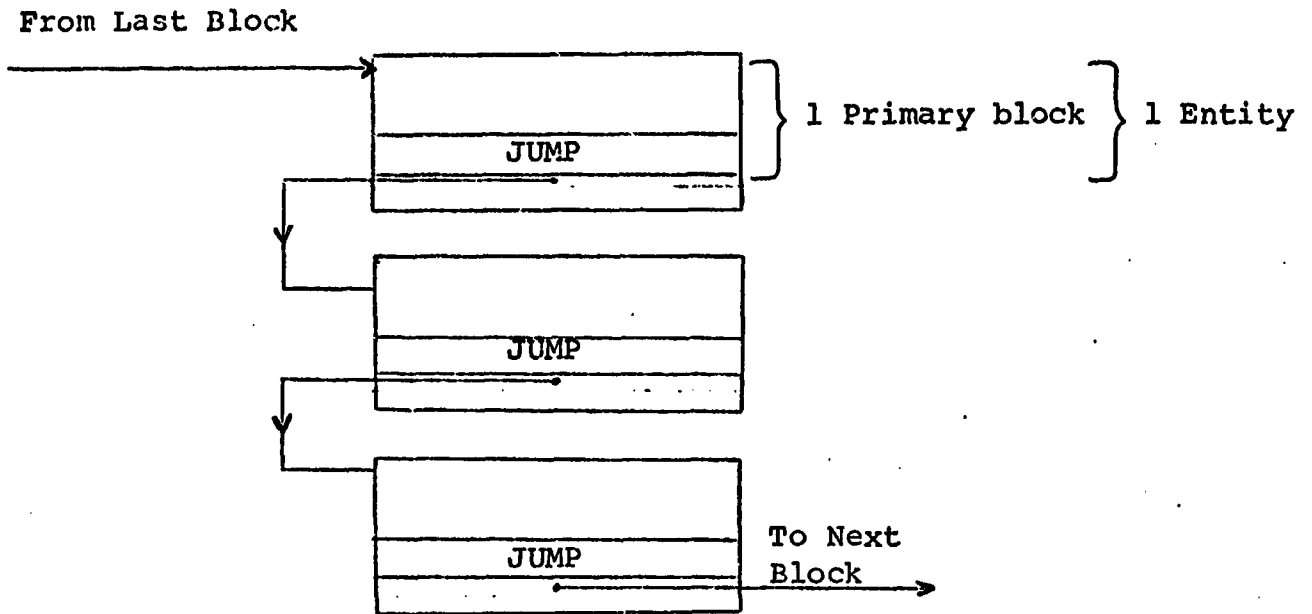


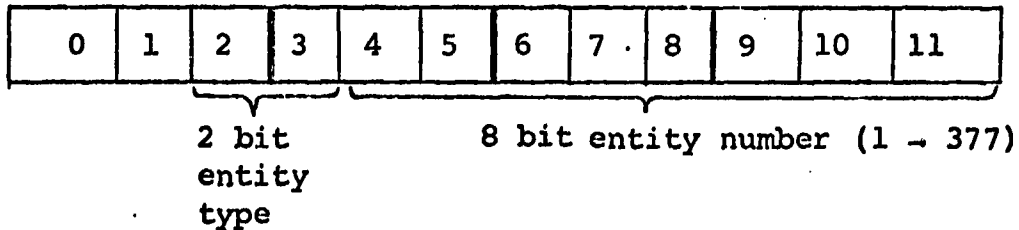
Figure 5

Primary Block Linkage

For identification purposes, an "entity name" is assigned to each block. The entity name is composed of two parts; the entity type (node, branch, device, or button) and an entity number that ranges from 1 through 377, for example:

- a) NODE 5
- b) DEVICE 25
- c) BRANCH 1
- d) BUTTON 10

The entity name is stored in the first word of an entity's primary block. The entity type is stored with the entity number in a 12-bit "full name". The "type" requires 2 binary bits because there are four entity types. The "number" requires eight binary bits because there are 400 possible numbers, with zero reserved for system use. For example:



Note: Bits 0 and 1 hold miscellaneous information and will be described later.

The bit combinations used in bits 2 and 3 have been selected as follows:

<u>Entity Type</u>	<u>Bits 2 and 3</u>	<u>Octal Equivalent</u>
LIGHT BUTTON	00	0000
NODE	01	0400
BRANCH	10	1000
DEVICE	11	1400

The full name is computed by adding the value of the entity to the entity number, for example:

<u>Symbolic Name</u>	<u>Octal Name</u>	<u>Binary Number</u>
NODE 5	0405	00 0 1 00 000 101
DEVICE 25	1425	00 1 1 00 010 101
BRANCH 1	1001	00 1 0 00 000 001
BUTTON 10	0010	00 0 0 00 001 000

type number

Words 2 and 3 contain topological information that describes interconnections between entities; for example, the nodes connected to a branch. This will be described in more detail later.

Each entity has a set of coordinates (X,Y) that positions the entity on the screen. To position the beam at (X,Y), an "EDS POINT" instruction is needed to enter the data state, followed by two words containing the Y and X coordinates, respectively. Words 4-6 of each primary block are reserved for this purpose.

<u>Word</u>	<u>Instruction</u>	<u>Comment</u>
4	EDS POINT	/POSITION ENTITY
5	Y-COORD	/Y - COORD
6	X-COORD+4000	/X - COORD,ESCAPE

In addition, each entity has associated with it two parameter words. The first parameter word, word 7, sets the blink register to "blink on" or "blink off". When blink is on, the entity will blink on and off about once a second. The second parameter word, word 8, sets the scale, intensity, and light pen status registers. The scale, 1,2,4, or 8, indicates the distance between points in vector modes. The intensity is set from 0 through 7, to adjust the intensity of the beam when drawing. The light pen status, on or off, indicates whether an entity may receive a light pen hit should it be pointed to with the light pen. This will be described in more detail in the section about the interrupt monitor. The symbolic contents of words 7 and 8 are illustrated below:

<u>Word</u>	<u>Instructions</u>	<u>Comments</u>
7	BKON /BKOF	/BLINK (ON/OFF)
8	(SC1, SC2, SC4, SC8) +(INT + (0-7)) +(LPON, LPOF)	/SCALE (1, 2, 4, 8) /INTENSITY (0-7) /LIGHT PEN (ON/OFF)

Words 9 and 10 contain a push jump, PJMP, to the display file of the entity. In the cases of branches and devices, the PJMP transfers control to a secondary block containing the appropriate display subroutines. For nodes and light buttons, control is transferred to an appropriate "node" display subroutine stored permanently in the graphics monitor; for example:

<u>Word</u>	<u>Instructions</u>	<u>Comments</u>
9	PJMP	/CALL DISPLAY SUBROUTINE
10	"DISPLAY FILE"	/TO DRAW THE ENTITY

Words 11 and 12 contain an optional "PJMP" to the display file for a label stored in a secondary block. Finally, words 13 and 14 contain a "JUMP" to the next primary block entity point; for example:

<u>Word</u>	<u>Instructions</u>	<u>Comments</u>
11	$\left. \begin{array}{l} \emptyset \\ \emptyset \end{array} \right\} \text{ or } \left\{ \begin{array}{l} \text{PJMP} \\ \text{"LABEL"} \end{array} \right.$	/NOP, or CALL DISPLAY
12		/FILE WITH TEXT LABEL
13	JUMP	/TRANSFER CONTROL
14	"NEXT BLOCK" + 3	/TO NEXT BLOCK ENTITY

Thus we have a 14 word primary block. Figure 6 illustrates a complete primary block.

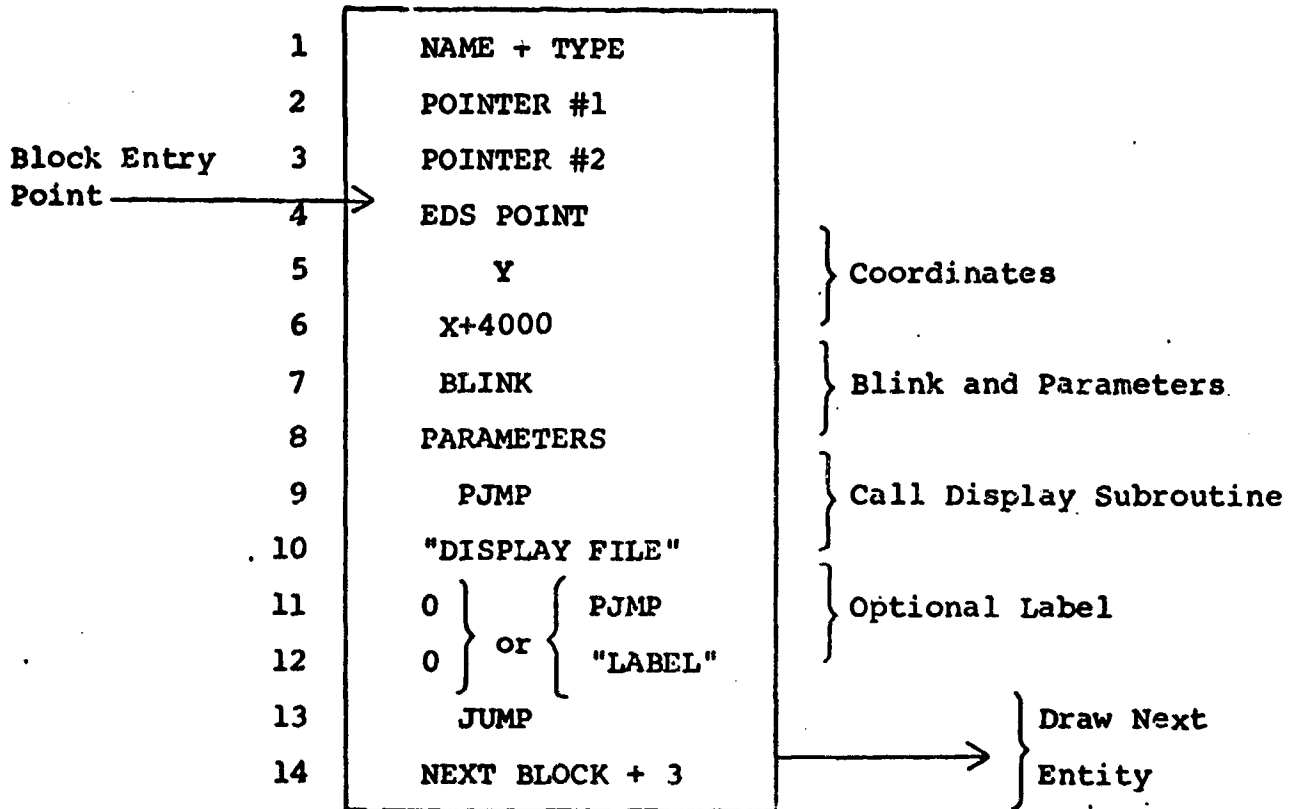


Figure 7

A Complete Primary Block

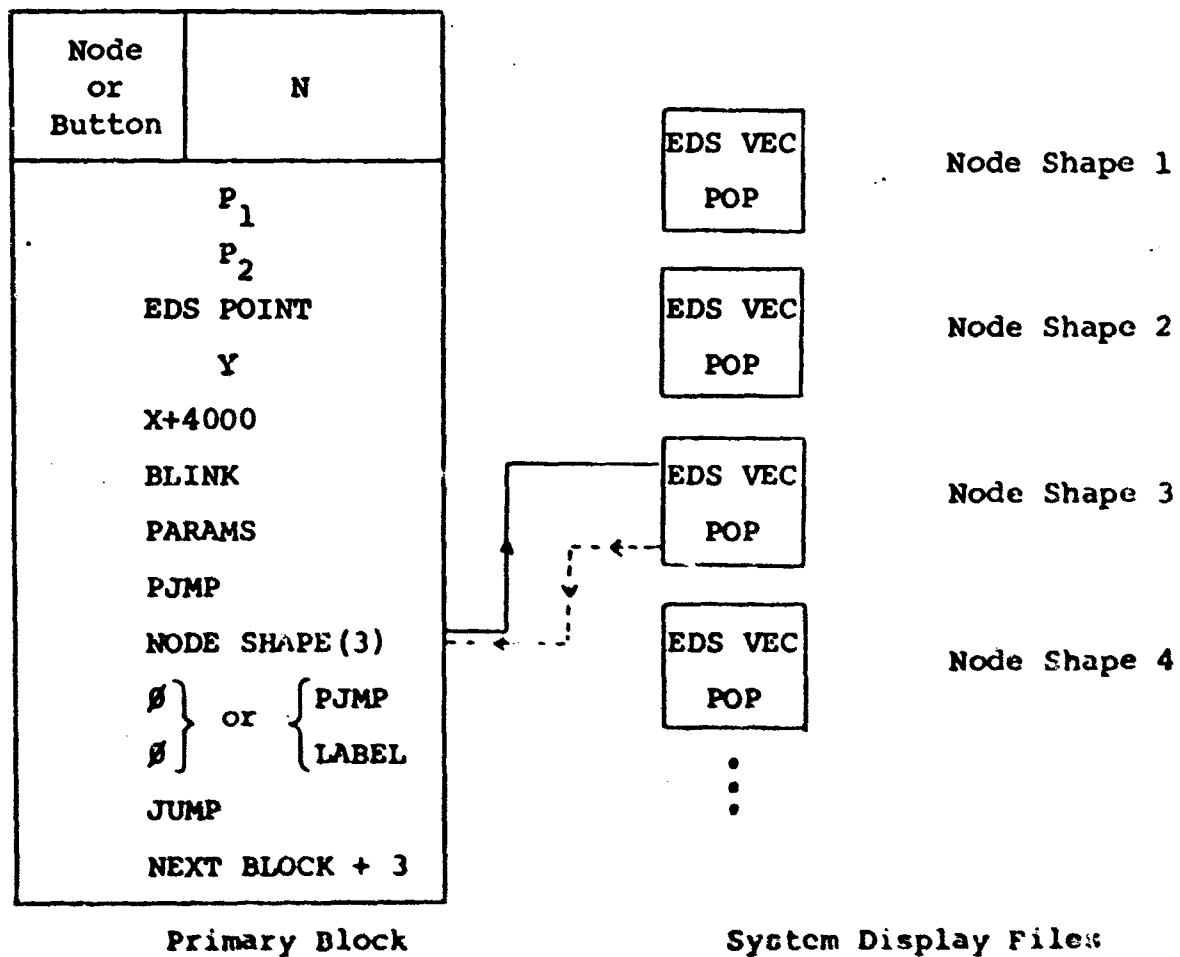
The display files transferred to the primary block differ with entity type as described below.

Nodes and Light Buttons

Each node and light button has associated with it one of the shapes illustrated below, numbered 0 through 7. The display files for the eight node shapes are stored permanently in the graphics system.

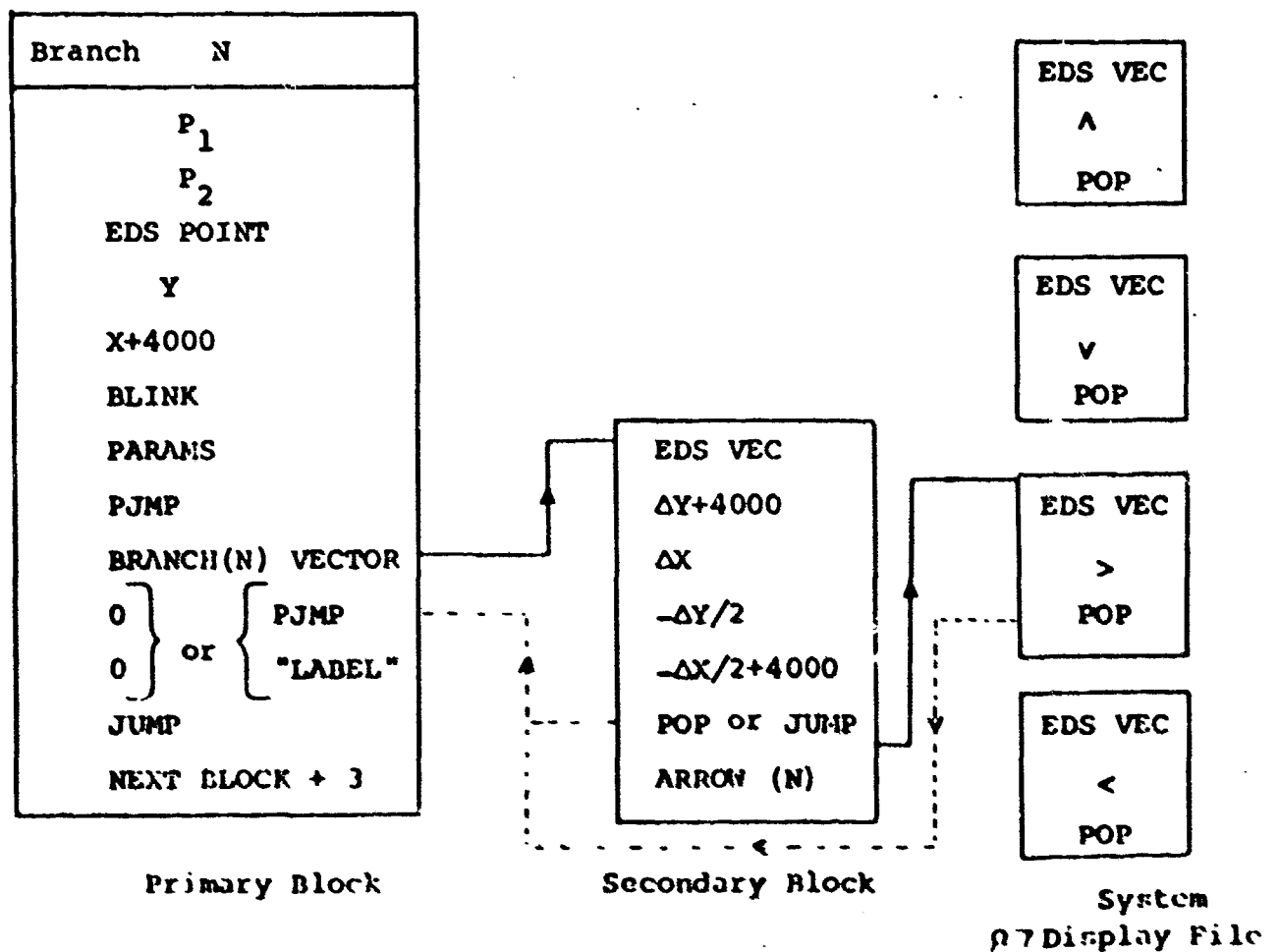
Shape	< NULL >	+	•	▲	●	⊠	⬠	⊙
NO.	0	1	2	3	4	5	6	7

The transfer of control to the node display file from the primary block is diagramed as follows. Upon completion of the display file, control is transferred back to the primary block.



Branches

Since each branch display file depends on the position of its nodes, a separate secondary block is created for each branch. The branch components are computed (by the graphics monitor) as the displacement between the "from node" and the "to node". The beam begins painting the CRT screen at (X,Y) and finishes at (X+ΔX,Y+ΔY). When an optional arrow is displaced at the midpoint of the branch, a return vector half the length and in the opposite direction of the branch vector is drawn invisibly. The branch arrow direction is computed (by the graphics monitor) and the result stored in word 7 of the secondary block. When an arrow is desired, word 6 contains a JUMP instruction; hence, display control is transferred to the appropriate arrow display subroutines. Otherwise, display control returns directly to the primary block (at word 11). For example:



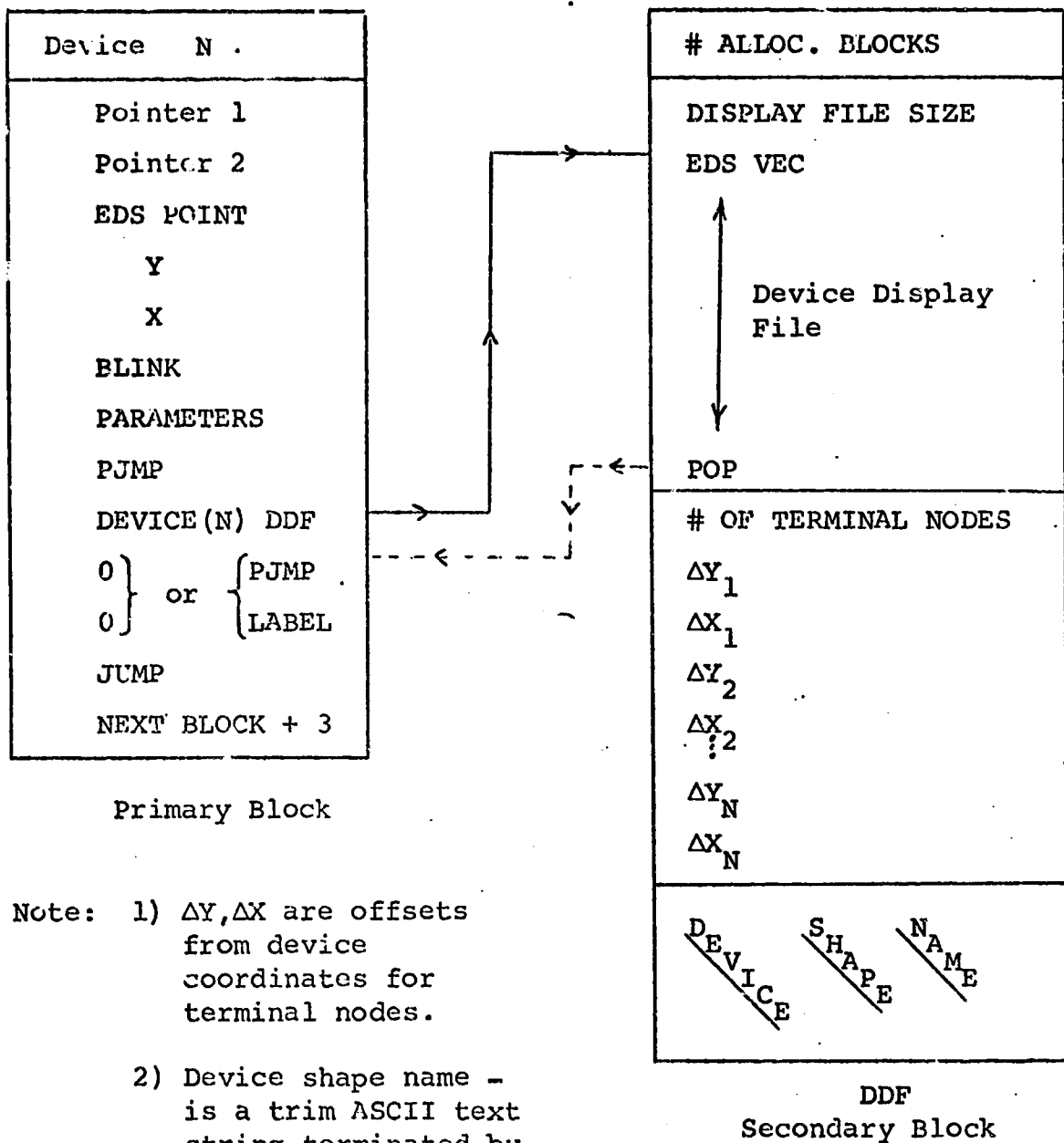
Devices

A device display file (DDF) is of arbitrary size, depending upon the number of vectors required to draw the device shape. Thus the first word of the DDF contains the number of sequential seven word blocks needed to house the DDF. Seven word blocks were chosen to provide efficient storage. Storage allocation methods are discussed later in more detail.

Other words of information stored in each device display file are:

- a) Number of terminal nodes.
- b) Relative positions of terminal nodes.
- c) Name of device display file shape.

For example, a capacitor display file has 2 terminal nodes (see Figure 4.3), 2 sets of offset coordinates for terminal nodes from device coordinates, and a name of "CAPACITOR". The second word in the DDF indicates the display file length, enabling access to aforementioned information stored in the device display file. For example:

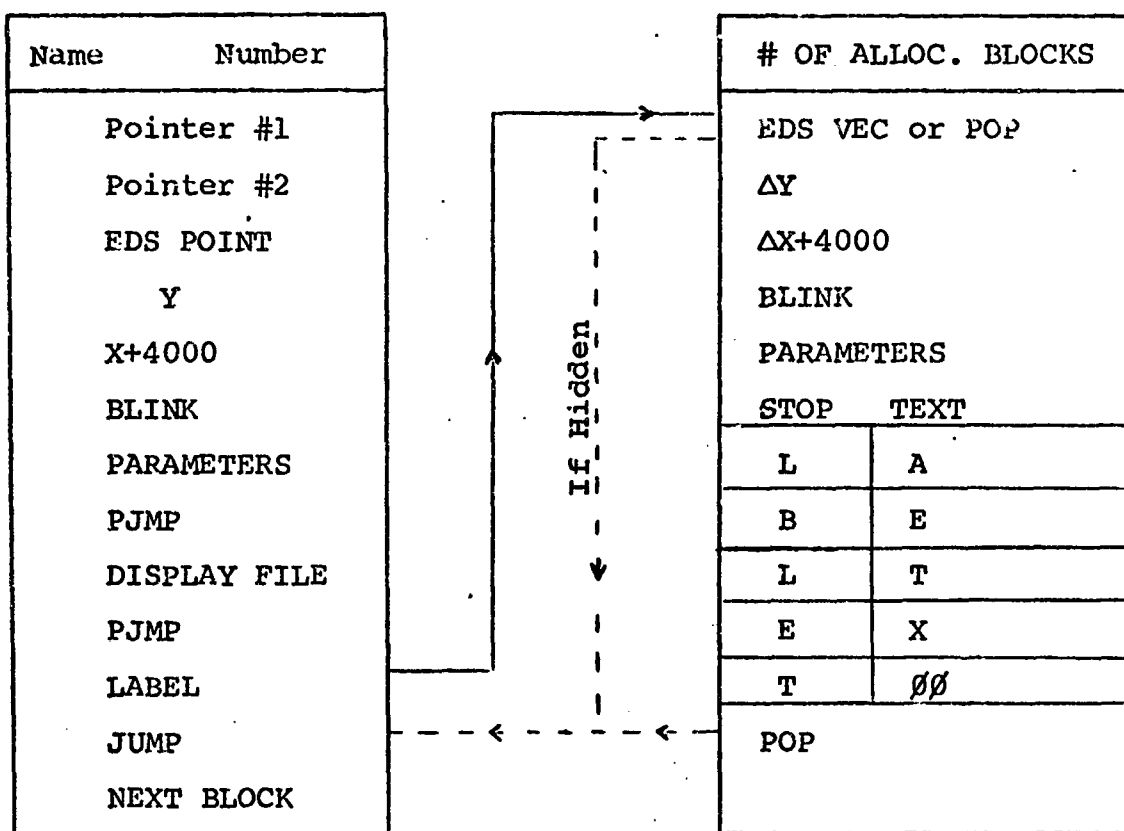


N	A
M	E
S	$\emptyset\emptyset$

Since all information particular to each device is stored in the primary blocks, all devices of the same shape share the same DDF secondary block.

Labels

Labels are optional display files with text positioned relative to the entity. The label block also contains blink and parameters. The text block is of arbitrary length depending on the length of the text string. For example:



It should be noted that word #2 of the text block is either a "EDS'VEC" or a "POP". The purpose of this is to allow the option to "hide" the entity's label, by enabling the display to skip the text label when desired. The ΔY and ΔX are vector components of an "offset vector" from the position left by the display file of the entity. For example, the label for a branch is positioned relative to the midpoint of the branch since the beam is left there after the branch is drawn. The "stop text" instruction is a call to a simulated character generator which takes a six bit character and computes the address of the appropriate display subroutine to draw the character. A null character, $\emptyset\emptyset$, terminates the text string and returns control back to the display file.

We now examine the display file for a complete flow chart of an electronic circuit with a capacitor and diode in parallel. (See Figure 7.) The diode has a label of "D1". The devices are connected by branches to terminal nodes on the devices. The display file is called by a display program called "DISPLAY DRIVER" which limits the display rate to 30 cycles per second. The last entity points to the "link block" which contains a "POP" instruction to return control back to the display driver.

Ring Pointers

Words 2 and 3 of each primary block contain topological information that describes connectivity in the flow chart. A ring structure has been designed for efficient definition of the flow sheet topology. The pointers, 1 and 2, have different meanings for each entity, as follows:

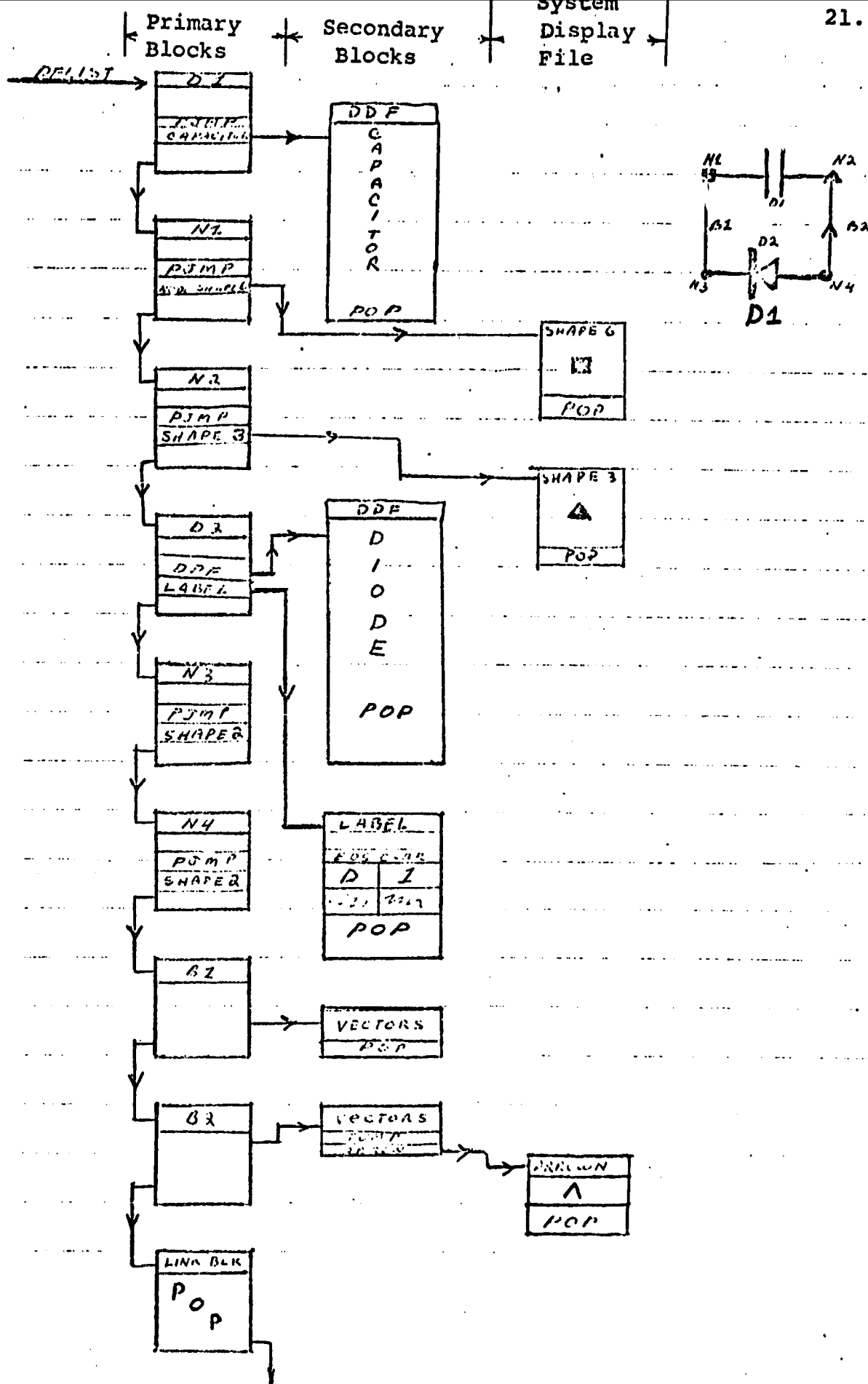
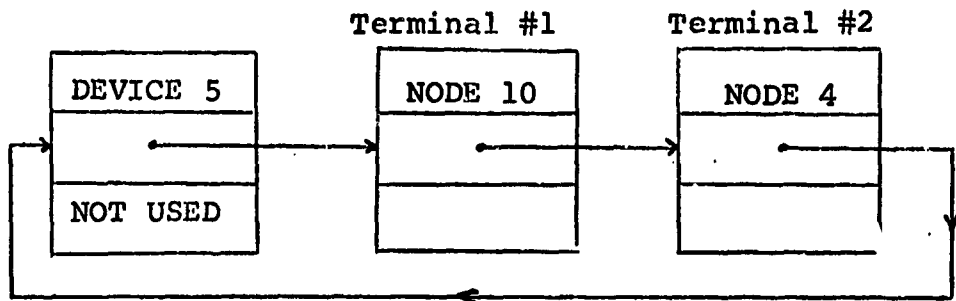


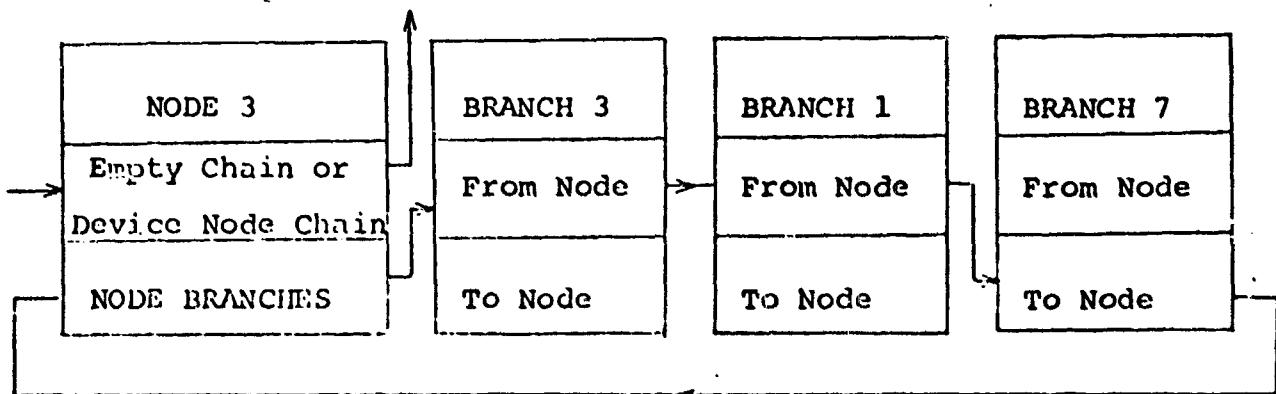
Figure 7
Display File Structure

- Buttons - Since light buttons are not part of the flow chart, the pointers are not used.
- Devices - Pointer #1 is used to point to a chain of terminal nodes. The chain terminates by pointing back to the device; for example:



It should be noted that the order in which nodes are placed in the list corresponds to the terminal number. When node 10 is removed from the display file, node 4 will become terminal #1, and that node can be terminal of only one device. Note that pointer #2 is not used.

- Nodes - Pointer #1, as we have seen, may be part of the device terminal node chain. Pointer #2 points to a branch connected to the node, which in turn points to another branch connected to the node. The last branch terminates the ring by pointing to the node. The entry of the pointer in the branch block indicates whether the node is the "from" or "to" node. For example:



Branches - Both pointers #1 and #2 must be non-empty to completely define a branch. Pointer #1 is part of a node-branch chain that points indirectly to the "from" node. Likewise, pointer #2 points to the "to" node. See the figure on page 22 for example chain rings.

Figure 8 illustrates the ring structure for a circuit containing a device to be deleted. The graphics monitor examines the rings to delete both the nodes and all branches on those nodes. First, the chain from the device to the first terminal node is examined. Next, all branches connected to that node are deleted, as well as the node itself. All other nodes for the device are treated as above. Finally the device is deleted. See Figure 9 for a more complex example illustrating the ring-chain structure.

Storage Allocation

Because this is an interactive graphics system, the display file monitor uses a dynamic storage allocator to allocate 7 word blocks of memory set aside for the display file. A 7 word block was chosen for efficient storage of primary and branch blocks which require 14 and 7 word blocks, respectively. These blocks are linked together to form a list called the "free block list". See Figure 10. This is a directed list in which the first word of the allocation block contains a pointer to the next free block. The list is terminated with the last block pointing to an "end block" with a "Ø" as the next block address. When additional storage is required, for example, to define new entities, the allocator is called upon to find "N" consecutive free blocks, remove them from the list, and return with the address of the

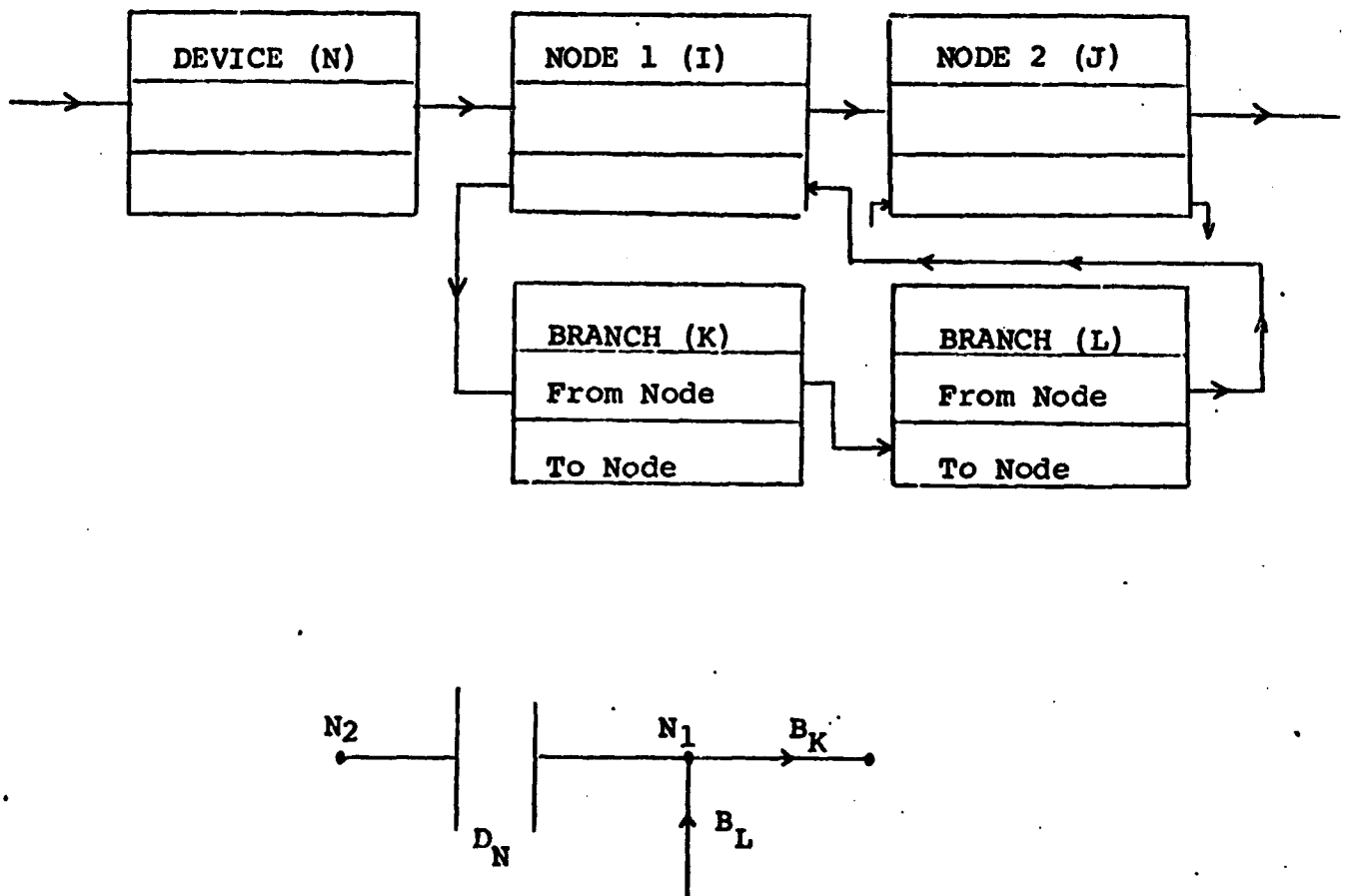


Figure 8

Inter-Block File Structure

first block. When storage is no longer required, for example, when deleting an entity from the flowchart, the blocks that are released and made available for future use. Each block is inserted into the list physically as close to other available blocks as possible.

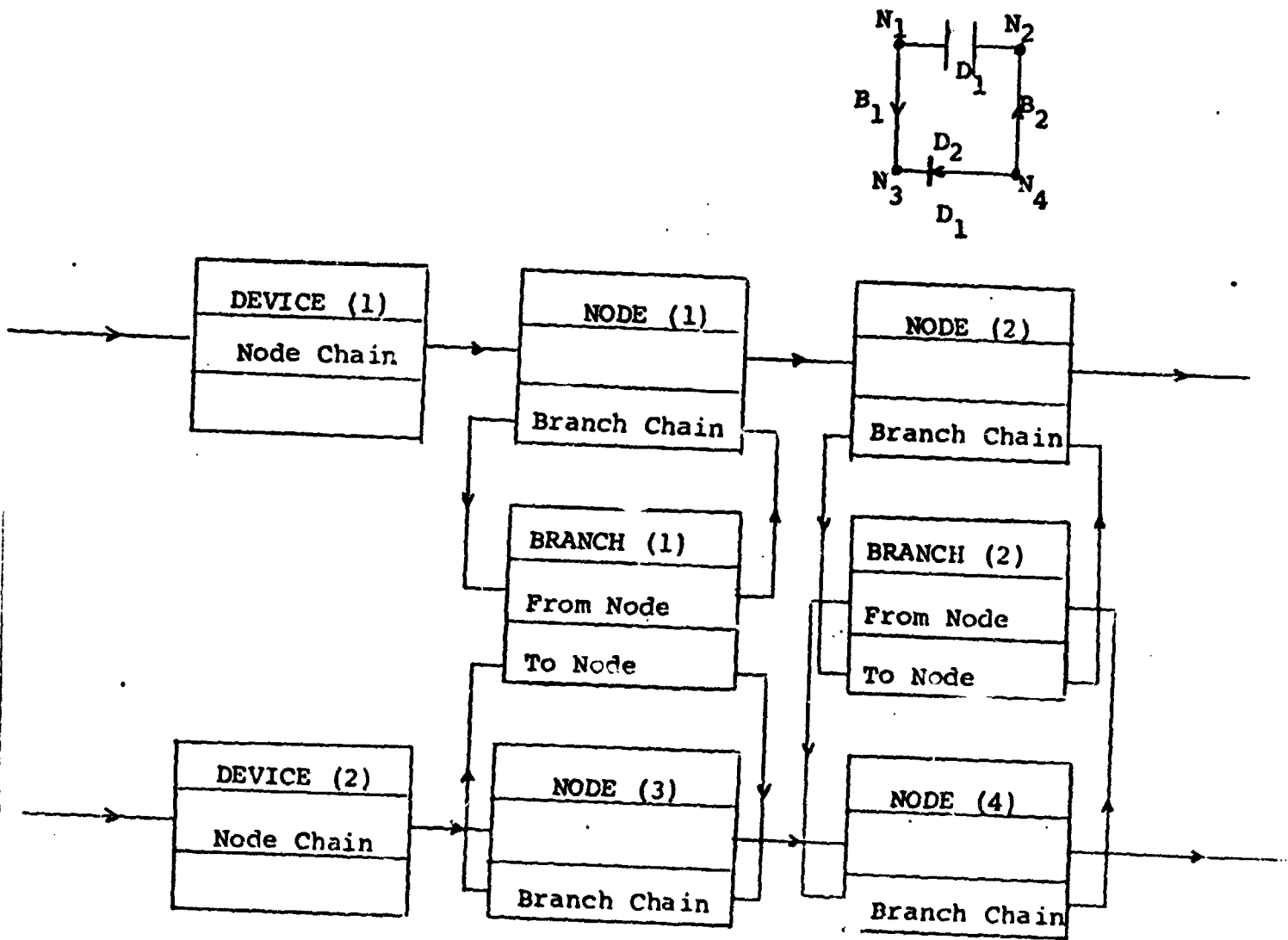


Figure 9

Display File Structure Ring Pointers

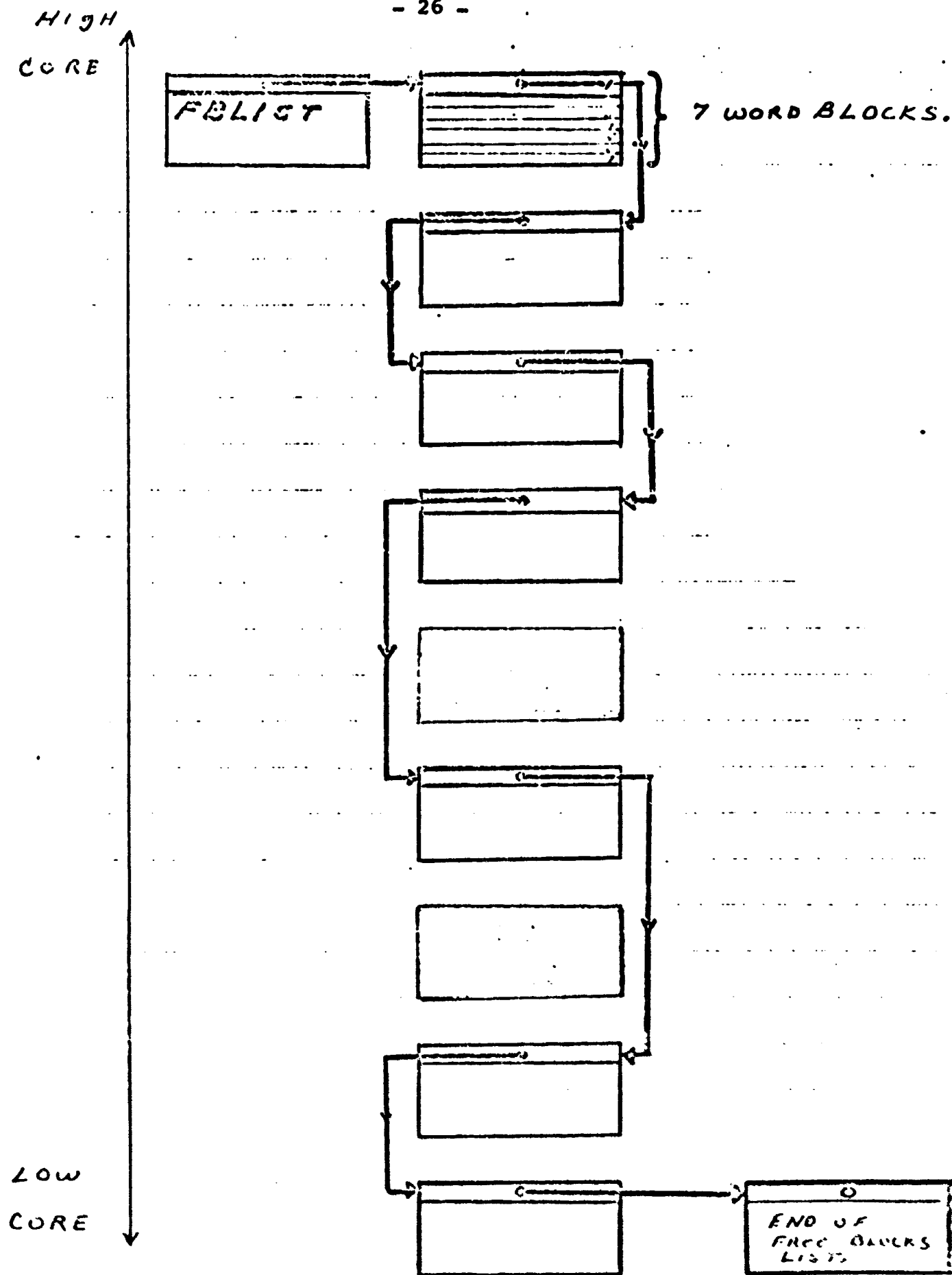


Figure 10
Free Block List

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
Philadelphia, Pennsylvania

REPORT AND RECOMMENDATIONS
ON COMPUTER GRAPHICS FACILITIES
IN THE MOORE SCHOOL

by
Morris Rubinoff
and
Colin West

The Moore School Information
Systems Laboratory
University of Pennsylvania

TABLE OF CONTENTS

	Page
1. Introduction	1
2. Project CAID	3
3. Project CIDS	5
4. Animator Movie System	8
System 1	9
System 2	10
5. DALI	12
6. The Growing Machine Movie System	15
7. Interactive Design of Chemical Process Systems	18
Input System	18
Output System	20
8. MOVIES	21
9. Naval Duel System	24
10. Assembly of Programs for the 338	25
11. Communications between the Spectra 70 and the DEC 338	26
12. Conclusions	28
Hardware	28
Software	30
A General Purpose System	31

1. INTRODUCTION

This report was undertaken in order to examine the use being made of computer graphics facilities in the Moore School, in particular the use of the DEC 338. The aim was to discover what aspects of the existing facilities were likely to limit future research using graphics equipment.

The main body of the report consists of a series of short sections on various projects which are currently using the DEC 338. These sections are included to give a general picture of how the DEC 338 is being used. They, therefore, concentrate on the computer graphics aspects of the projects. These are followed by short sections on communications and program assembly which are of general importance to all users of the 338.

The concluding section contains a number of conclusions and suggestions for improvement of the facilities available. These can be summarized as follows:

1. Although many users are finding that the slow display speed and small memory size of the 338 are disadvantages of the system, these limitations are not severe enough to justify the cost of improving them.
2. The uncertain future of the 7040 computer makes it necessary to produce as soon as possible an assembler for DEC 338 programs that can run on the Spectra 70.
3. The use of the DEC 338 is severely restricted by the lack of a general purpose operating system. Systems that have so far been developed have been designed for particular applications and are not generally useful to new users.

100

4. It is suggested that a general purpose system be developed that could serve a large number of users and so reduce the time taken to develop new applications.

The concluding section contains an outline of one possible system whose primary aim would be to provide a means of using the DEC 338 in conjunction with FORTRAN programs run in the Spectra 70.

2. PROJECT CAID

The CAID project is one designed to investigate a number of problems associated with the training of helicopter pilots. The project developed around a simulator which was designed to investigate problems of formation flying of helicopters. This simulator has been extensively modified. Whereas the original simulator could simulate the flight of up to 25 helicopters, the one used by CAID can only simulate the flight of one. A number of features have been added so that the simulator can accept control inputs from a student pilot and provide him with resultant changes in instrument readings in real time.

The simulator itself is a FORTRAN program that runs in the 7040. The student pilot can operate a number of realistic helicopter controls. These are monitored via an A/D multiplexer system linked to the 338 which can interact with the 7040 via the dataphone PDP8-7040 data channel link.

The main functions of the 338 at present are:

1. To provide the student pilot with a visual monitor of the helicopter performance by displaying a number of instruments for him to read.
2. To allow the experimenter using the CAID system to instruct the pilot to fly a course by displaying the control movements necessary to maintain the helicopter on the course.
3. To input to the simulator the movements of the controls made by the pilot.

The display seen by the student pilot consists of a number of fixed scales representing instruments together with movable pointers and symbols which represent instrument readings and control positions. By operating toggle switches the student pilot can, if he desires, suppress sections of the display.

The program in the 338 is in principle quite simple. A display file has been created that represents all of the elements of the display. Changes of instrument readings received from the simulator of movements of the controls initiate calls to subroutines which produce a rotation or translation of the pointers or symbols by changing a few words in the display file. These routines are the same as the ones used in the DRAW demonstration program.

The most serious problems of the system is maintaining a high rate of data transfer to and from the simulator and computing the airframe equations fast enough so that the system can operate in real time. The state of the helicopter in the simulator is represented by some 300 parameters; real time monitoring of any significant fraction of these could easily swamp the 2400 Baud dataphone link. In fact, information concerning only 20 of the parameters is needed to update the display. The information is transmitted to the 338 in binary form and the amount transmitted is kept within manageable limits by specifying for any particular experiment the relative importance of the parameters. The rate at which each parameter is transmitted can be specified in the range from 10 times a second to once every 10 seconds. The simulator is then made to transmit the data in a pre-determined sequence to achieve the required rates. The 338 routine that receives the data is written in a macro form so that it can be readily changed to accept a different data sequence required for a new experiment.

3. PROJECT CIDS

The CIDS project is an information retrieval system used exclusively for chemical compounds. The problem is, given the structure of part of a chemical compound, to locate in a data bank all compounds that contain the substructure. A number of well defined methods of describing a chemical compound have been developed. One of the most precise and the one that is used in the CIDS project is that in which the structure is defined by giving the nature of the individual atoms and the bonds that link them as a two dimensional network.

Such a representation has obvious disadvantages for computer processing unless a display device can be used for input and output.

The DEC 338 is therefore used for input and output in the CIDS project. The lightpen can be used to draw a structure on the display; suitably coded, the structure can then be included in an information request.

If a request for information is not very specific, many compounds may be located. In this case, the 338 can be used to quickly examine the results of the request, allowing the operator to modify his request and so condense the response thus avoiding excessive hard copy output.

The structure of a compound is input as follows. A menu of the chemical symbols of the most common elements (extendable to include all elements) is displayed on the screen. Pointing at an element with the lightpen identifies it as the one to be added to the structure. The structure is actually drawn on a checkerboard array of squares. Each square can contain one chemical symbol or

can be crossed by lines representing bonds. This method simplifies the display generation, particularly when it is necessary to define closed rings.

The required chemical symbol is added to the structure by pointing at a square with the lightpen and setting a pushbutton. According to which pushbutton is used, lines representing bonds can be produced, linking the symbol to the previous one added to the display. Different types of bonds are represented by various intensities and multiplicities of the lines.

The system includes provisions for modifying, erasing, magnifying and storing on disc any structure drawn on the display. The operation of the pushbuttons and lightpen create a display file and two tables in the 338 which describe the structure. The first contains the x,y display coordinates and element type of each atom in the structure. The second contains the position and nature of each of the bonds. The atom and bond table can be saved on disc. The display file is not saved but can be regenerated from the atom and bond table when needed.

After the structure has been completed, it can be referenced in an information request input via the teletype. The request can consist of several lines of text of a specific form. It can be edited and also saved on disc.

In order to transmit the structure to the 7040, it is represented as a linear string of characters which contains all the information in the atom and bond tables except for the geometrical information used to generate the display.

A response to an information request consists of a series of chemical compounds, each of which is represented in a Dura Mach code. This code is one which can be used to operate a special typewriter which can type the two-dimensional representations of the compounds. The 338 system is capable of producing a display file from the Dura code so that the operator can quickly examine the result of his request by displaying in sequence the compounds found. He may then edit his initial request and resubmit it. He cannot, however, use directly one of the compounds retrieved in a further request as there is no provision for converting from the Dura code to the atom and bond tables from which the input string is generated. Hard copy output can be produced by punching a paper tape in the Dura code which will operate a chemical typewriter.

The 338 display system as it has been written by Andre Gagnoud is complete but has a number of limitations and is being extensively modified. The modifications will provide the user with greater facilities for manipulating the displayed compounds and remove some existing bugs of the system. The formats used to draw compound descriptions to and from the 338 will be made the same.

4. ANIMATOR MOVIE SYSTEM

Before describing this system, it is worthwhile tracing its development. The system was developed as one which could be used to make movies using the SCORS package without the need to do a great deal of laborious FORTRAN programming.

The nucleus of the system is a B.N.F. transmission language which enables someone using the 338 to specify the complete parameters of a movie sequence in a form that can be transmitted to an interpreter program in the 360. The interpreter program produces a sequence of instructions for the SCORS package.

The language permits the user to specify a movie sequence in terms of pictures and motions and was defined in a thesis written by Patti Talbot.

The realization of the system, namely a 338 program which translates a sequence of input operations into the transmission language has been largely the responsibility of Dick Coulter. The program was developed in two stages which we will refer to as systems 1 and 2. System 1 was the first 338 program written by Patti Talbot and Dick Coulter and was introduced as a test of the feasibility of the system. With the experience gained from system 1, the second system, a more complete and better designed program, was started but has yet to be finished.

The interpreter program which translates the transmitted string of characters into instructions for the SCORS package was written by Rosa Hwang for the 360/65. Modifications necessary to make this program run on the Spectra 70 have been started but not as yet completed.

System 1

The first system was one that did not make use of the disc and so was somewhat restricted in terms of what it could do. A movie segment could be defined in terms of pictures and two types of motion, hold and translate. A total of 17 pictures or subpictures could be defined, each one being described by drawing on the display using the lightpen or by typing coordinate information on the teletype. Either method of input generates a 338 display file and a string of characters in the transmission language. Each picture or subpicture could be defined in terms of vectors or names of other subpictures which had been previously defined. The limitation on the number of pictures that could be defined was governed by the DECtape file structure adopted. The complexity of any picture definition was limited by the core size of the 338 to about 300 vectors. Available space in the 338 was rapidly used up because of the need to have in core a working display file, display files for each subpicture being referenced as well as the file containing the transmission language string. Erasing of elements of defined subpictures resulted in removal of intensify flags in the display files and additions to the transmission strings thus consuming more space rather than regaining it.

When definition of the movie sequence was complete, the DEC tape was searched for all files containing the generated transmission code for all picture and motion elements referenced. These were then assembled into one file which was transmitted to the 360/65 in a 4 out of 8 code, this method being used as the 338 had to simulate a 360/20 terminal. Any editing necessitated by erasing or modifying of any of the subpictures was performed in the 360/65.

System 2

The specification of the second system differs from that of the first in the following respects.

1. Up to 126 pictures and subpictures can be defined, the increase being achieved by a change in the file structure.
2. More complex motions can be specified including rotations and parallel and sequential motions.
3. The system is now readily expandable, modular and disc oriented. The sections of the system, namely picture, motion, scene, movie segment, production and transmission programs, are normally resident on disc and are called into core as needed.
4. There are provisions included in the system for later addition of a text facility and also for real time communication with the Spectra 70.

At the time of writing, however, only the motion, scene and production sections of the system are complete. Pictures and subpictures can be defined via the old system in the following way. The old system, kept on disc, is called into core, wiping out the monitor of system 2. Picture files can then be created, the system 2 monitor restored and the created files modified to correspond in format to those created by the second system.

When completed, the second system will be capable of transmitting more complex movie sequences than the first to either the 360/75 or Spectra 70. There will, however, be no facility for generating the movie sequence directly on the 338. This is because the system never actually generates a series of display files that represent the individual frames of the movie,

only the display files that represent pictures or objects depicted in the movie.

A third version of the system which will permit playback of the movie segments on the 338 has recently been started by Phil Rothenstein.

5. DALI

DALI is a graphics language being developed by Tom Johnson. Although primarily intended for teaching graphics techniques, it may have other applications, for example in movie making as it is easy to learn and use and has powerful image manipulation capabilities.

- The language consists of a number of statements which can be processed by SNOBOL to generate FORTRAN IV coding. Executable DALI statements can be freely mixed with FORTRAN statements, adding to FORTRAN's computational power extensive image manipulation capabilities.

The statements of the DALI language can be divided into two classes, declaration and executable statements. The former are used to define PRIMITIVES, OBJECTS and TRANSFORMS. PRIMITIVES are 5-vectors consisting of 4 homogeneous coordinates and a primitive type, namely point, vector or invisible point. A OBJECT can be dimensioned to have a predetermined number of components, each of which can be either a PRIMITIVE or another OBJECT. A TRANSFORM is a 4×4 matrix which can operate on PRIMITIVES or OBJECTS.

The declaration statements separate a large block of common store into 4 sections. The first is an area reserved for the declared transformations. This is followed by a list of addresses specifying subelements of the declared objects, pointers indicating the start of each object in the above list, then the declared primitives. The disadvantage of this layout is that all objects and transforms must be declared at the start of the program. However, it is simpler to use than a more complex general list structure.

The executable DALI statements are used to define and change individual elements of both objects and primitives. They can also be used to define complex transformations in terms of the elementary translation, rotation and scaling transformations defined as part of the system. The use throughout of homogeneous coordinates greatly simplifies manipulation of the objects and primitives using the transformations developed.

The actual production of pictures is done in two stages. Firstly statements USE transform 1, transform 2 ... ON object 1, primitive 2 ... are used. These cause multiplication of the referenced transforms to produce one that is stored on a pushdown, also the objects and primitives are put on an associated pushdown list. No modification of the referenced transforms, objects or primitives takes place.

A statement DISPLAY object 1, object 2, ... etc. initiates a scan of all referenced objects to determine the primitives used; these are then operated on by the transforms linked to them by the pushdowns set up in previous USE statements. The picture can then be output on magnetic tape as a series of primitives. A perspective transform is also defined as part of the DALI system. This is, however, used later.

One overall transformation be it a perspective or simple projection is placed on the output tape with the primitives of the assembly of objects that represent the final picture.

The actual picture is generated by reading in the file from tape, executing the transform and producing the output. It is planned to produce output on the line printer, calcomp plotter or in

the form of a display file that can be transmitted to the 338 and written on DEC tape. Initially this will be done using the Park - Coulter display file transmission system, changing back the display files on tape using the Growing Machine playback program.

Eventually, however, it is planned to assemble each picture in stages for debugging purposes, also to give a user at the 338 console the ability to modify the final transformation and retransmit the picture so that the objects created can be viewed from any angle.

The present status of the system is that it is written but not completely debugged.

6. THE GROWING MACHINE MOVIE SYSTEM

A complete description of the Growing Machine as implemented in the 338 by Noel Bernstein is out of place here. The basic 338 system has been extended by Noel Bernstein and Alan Hayes to produce movies and will be described as it is interesting to compare it with the Animator movies system.

The Growing Machine Movie system has been produced by defining a number of primitives within the framework of the Growing Machine. These primitives can be used to define figures and specify a number of transformations that can be applied to them. The transformation can be translation, rotation or camera zoom. The figures are defined in terms of vectors, either visible or invisible. Three methods of defining figures can be distinguished. Firstly, a figure can be defined as a string of vector primitives input via the teletype. Secondly, it can be defined in terms of an algebraic expression expressed as a sequence of the more general computational primitives available in the Growing Machine. Thirdly, a figure can be drawn directly on the 338 display using the lightpen. For this, the DRAW subroutine is used. This was developed by Jeff Ball for testing routines used by the CAID display program.

Execution of a string to produce a movie sequence creates a series of display files which can be displayed during execution and also saved on DECtape for later playback. The latter results in a more acceptable movie sequence because of the computation time taken during execution.

The Growing Machine and Animator movie system can be compared. The first has a number of advantages. Figures can be defined algebraically where appropriate. Movies generated can be seen essentially when they are produced. However, for someone not too familiar with the Growing Machine, a movie would be relatively difficult to produce. It would also be restricted in complexity as a result of the small size of the 338 memory and be not of very high quality because of the limitations of the 338 display resulting from the digital rather than analog vector generation.

The Animator system could be used to generate some complex movie sequences, would be easier to use and the movies would be of higher quality as a result of using the SC 4020. The absence of an immediate playback facility would be a disadvantage during debugging.

The playback facility of the Growing Machine has proved to be a useful facility. Combined with the program written by Bill Park and Dick Coulter for transmitting 338 display files from the Spectra 70 to the 338 DECTape, it is used by

1. The MOVIES project for playback of sequences generated on the Spectra 70.
2. The Chemical Process Design project for graphical output of simulator results.
3. Bill Park for output of the walking platform simulator.
4. For output of pictures generated by the DALI language.

The playback facility is just a program that will display a series of display files which are written on DECTape. The rate at which the display file sequence is shown can be controlled using the pushbuttons. A feature which has been included to increase the maximum playback speed

is that it is not necessary for a complete display file to be on the input tape if it is partly identical to the previous file on the tape. To be precise, if the second file can be produced from the first by overwriting part of it, only the changed part need to be on the input tape.

7. INTERACTIVE DESIGN OF CHEMICAL PROCESS SYSTEMS

This system is in the early stages of development and can at present be described in two sections. The first is a system intended to provide a graphic input to a chemical process simulator. Most of the system has been written by Jeff Ball, an undergraduate student who has recently left the project. Jeff Kulick has contributed the disc monitor and a number of other I/O facilities used by the system.

The second part of the system, which is being developed by Mike Zaborowski, consists of a means of displaying graphs on the 338 which represent the results of simulator runs. This is at a much earlier stage of development.

Input System

The status of the first part of the system is that it exists in stand alone form. That is to say, it can be used to draw chemical plants in diagrammatic form on the 338 display and to input parameters that define the properties of the plant's components. However, no attempt has been made to interface it to a chemical process simulator of necessity resident in a larger computer.

A chemical plant that can be defined by the system can consist of up to 32 components or units. There are 8 possible unit types, i.e., distillation column, condensor, heat exchanger, etc.

A chemical plant can be defined as follows. The lightpen and pushbuttons are used to indicate the position on the screen where a unit should be placed. The lightpen can then be used to select a unit type from a menu displayed. When the unit type has been established,

it is displayed in position and a request for information defining its parameters is also displayed. This information is typed on the teletype. The units are interconnected by using the lightpen and pushbuttons to define the input and output points on the units to be connected and by drawing the path of the interconnection or stream. There are built in facilities for erasing and modifying any part of the displayed plant and for displaying it at different magnifications.

It is difficult to understand the underlying structure of the system due to the fact that Jeff Ball is no longer here and there is essentially no documentation describing the system.

Briefly the structure is as follows. There is a basic monitor program which is resident in core and other sections are kept on disc and called in as desired according to what the user is doing.

When a particular unit is established on the screen, an entry is created in a unit storage map resident in the lower segment of the core. This entry contains the name of the unit, the number of inputs and outputs and a flag indicating whether or not the display file for the particular unit type is in core or on disc. A 16 word entry is also created in a large display storage map (DSM) which is maintained in the upper segment of core. This contains a number of identification parameters, x,y coordinates of the unit position on the screen and a pushjump to the display routine that represents the unit type. This display routine is normally kept on disc but is placed as 1 or more 16 block entries in the DSM the first time each unit type is referenced. When information specifying the parameters of the unit is typed in, it is stored on disc for later reference. When an

interconnection or stream is defined, two entries are created in the DSM. The first is the parameters that define the input and output connections together with other identifiers. The second, entered by a pushjump from the first, is the display file generated by drawing the path of the interconnection. The actual display of the plant is generated from this DSM by scanning through all the 16 word entries ignoring the parametric information but executing all the display routines for the units and streams in the order they were created.

In order to interface the system to a simulator, it would be necessary to write a program which extracts the topology of the chemical plant from the DSM and retrieves from disc the description of each of the units.

Output System

As stated earlier, the output section of the chemical process design project is at a much earlier stage of development. At present, a dynamic simulator is being used to generate information concerning the time dependence of certain parameters in a chemical plant, for example, the variation of temperature in a specific unit. Such information is generated in an array containing values of temperature at constant time intervals. The temperature and time are connected to a series of x and y coordinates of a graph, which, appropriately scaled and labeled, can be used to generate a display file. The display file representing the graph is generated in the Spectra 70 and transmitted to the 338 using the programs written by Bill Park and Dick Coulter. The graph is at present saved on DEC tape and played back for display using the Bernstein playback program.

8. MOVIES

The MOVIES system which was used to produce the first film in the "Electromagnetic Fields and Waves" series was fully described in a thesis written last year by Don Deily. The system has changed little since then so that it is probably not appropriate to discuss it in great detail.

The system is based on the SCORS package, which is a series of routines made available by the Stromberg Carlson Users Society. These routines permit a FORTRAN programmer to produce images on film using a SC 4020 computer recorder. The SC 4020 comprises a precision cathode ray tube whose face can be photographed, an instruction decoder and a 7 track tape transport system used to input instructions. The cathode ray tube can be used to display printable characters (not useful for MOVIES) and vectors. A single instruction can generate a vector having components of up to 63 raster counts in the x and y directions, there being 1024×1024 addressable points on the tube face.

The SCORS package is used to generate the input tape containing 4020 instructions from a sequence of subroutine calls specified by the programmer. Its most useful functions, as far as the MOVIES project is concerned, are those that provide 4020 camera controls, scaling of the output picture frames, and segmenting of long vectors into sections having components of less than 64 counts that the 4020 hardware can plot.

The task of the programmer is to translate a movie sequence defined in terms of objects and motions into a series of vectors and film advance instructions. There are at present two ways of doing

this. The first makes use of the Animator Movie system described earlier to define a movie sequence in the transmission language which can be interpreted to produce a sequence of calls to the SCORS routines. The second method is to define the movie sequence in a FORTRAN program. Using the latter, more elaborate things can be done; in fact, this is the only way that movie sequences involving general motions of three dimension can be produced. A series of routines exist which can be used to depict the motion of up to 10 objects as seen by a virtual camera. The motions of both objects and camera can be completely general translations and rotations. Provisions have been made for adding fairings to the motions which simulate the effects of inertia. Also hidden edges of certain simple objects can be automatically erased. Perhaps the major disadvantage of the movies system as it exists at present is the lack of a language for defining three dimensional objects which therefore have to be defined on a line-by-line basis.

The actual production of a movie is a slow and costly process. It is therefore essential to have efficient debugging facilities for checking a movie sequence before film is exposed. This can be done for individual frames by providing alternative output on a line printer or Calcomp plotter. A more complete check can be made by producing a sequence of 338 display files that can be transmitted to the 338 and played back as a movie sequence using the Bernstein playback system.

The first movie was produced using the 360/65. The system is not yet completely functional on the Spectra 70. The program developed by Rosa Hwang that accepts output from the Animator system was written in 360 machine language and is not yet working on the 70. The Spectra 70 produces 9 track output tape, so that there are some tape conversion

problems. The tape problem would not exist if an SC 4060 were used to produce the film as this accepts 9 track tape. It is also a more sophisticated device, having greater resolution, full screen vector generation and a programmable processor. In principle, at least, many of the functions at present in the SCORS package might be performed in the 4060. This could produce a significant saving in main computer time but has not yet been fully investigated.

9. NAVAL DUEL SYSTEM

The object of the Naval Duel system is to produce a simulator which can be used to study the interaction of opposing forces at sea. To take an example, the opposing forces might consist of a destroyer and a submarine, the former being assigned the task of destroying the submarine, the second being required to attack a convoy the destroyer is protecting. The simulator is an overall model of the system which runs on the 7040 using the MULTILIST system. The system can be used by two people each representing the commander of one of the vessels. They are able to interact with the simulator by defining their own course of action, requesting sonar readings of their opponent's position, and by launching weapons (e.g., torpedoes) to destroy their opponent. The sonar readings, which are made realistic by simulating occasional false readings, can be used to predict the course of the opponent.

The 338 can be used as a terminal by one of the users of the system. He can define his course by drawing it on the screen using the lightpen and receive predictions of his opponent's course in display form.

The display program has been written by Dave Kristol. The interesting feature of the program is that a basic number of routines have been developed which are in principle of use for a large number of display applications. These consist of routines for creating and deleting display files, transmitting to and from the 7040, lightpen, pushbutton and light button operations, disc I/O as well as many others. These basic routines are always in core and can be called by the more specific part of the program which is segmented and mainly on disc.

10. ASSEMBLY OF PROGRAMS FOR THE 338

At present, essentially all programs developed for the 338 are assembled using a PDPMAP assembler written some time ago by Tom Johnson and Mike Wolfberg. This assembler runs on the 7040 and was written as an extension of the MAP assembler used for that machine.

It includes a number of features which are desirable as aids to writing programs for the 338. For example, in a PDP-8 memory reference instruction only 7 bits are available for generating a direct address so that the memory has to be considered as divided into 200_8 word pages. Only locations on the same page as the instruction or on the first page of a memory can be directly referenced. In general, a location not on the current page can only be referenced indirectly. The PDPMAP assembler is able to generate automatically such an indirect address, thus considerably simplifying the work of the programmer. It also has extensive literal and macro facilities and generates cross reference listings and diagnostics for debugging purposes.

Programs having been assembled are generally punched on the PDP-8 in D.R.L.; using the MULTILIST system, it is possible to transmit them to the 338 for punching there but this is not generally done. There is no simple direct way of assembling a program onto DECTAPE on the 338.

Some months ago, Dave Kristol started to write a similar assembler to run on the Spectra 70 but did not complete it for technical reasons. Recently Len Bosack obtained an assembler that runs on a 360. This is being converted for the Spectra 70 but is not as sophisticated as PDPMAP and existing programs would need extensive modification before they could be assembled using it. There does not appear to be any plans to produce an assembler which could reproduce the PDPMAP facilities on the Spectra 70.

11. COMMUNICATIONS BETWEEN THE SPECTRA 70 AND THE DEC 338

At present, it is only possible to use the dataphone link between the two machines when the Spectra 70 is operating under the TDOS system. The communications programs that are in the Spectra 70 have been written by Bill Park. The facilities that are most widely used are as follows.

A user who has, for example, a FORTRAN program running in the Spectra 70 can transmit a buffer to the 338 by executing a statement

```
CALL      TELCOM (a,b)
```

where b is the name of an INTEGER*2 array and a is the number of words to be transmitted. Each of the 16 bit words can contain 12 bits of information to be transmitted (usually a 338 instruction or data word), right justified and filled out with zeros.

Each word is transmitted as follows. The 12 bits are split into two 6 bit numbers between 0 and 63. These are used as addresses which define printable characters in an EBCDIC translation table. Each 6 bit pattern is thus represented by an EBCDIC character.

The multichannel communications program takes the string of EBCDIC characters generated, translates them into USASCII characters and places them in another buffer. From here they are transferred to a synchronous data buffer in the communications controller, transformed to ASCII with odd parity and transmitted over the dataphone preceded by a start of text (STX) character and followed by an end of text (ETX) character.

The data is generally received in the 338 via the program interrupt. By setting bit 0 of the eight bits to 1, each character can be converted to an ASR-33 character compatible with that generated by the teletype. Otherwise, an original 12 bit word can be produced by packing the least

significant bits minus 40_8 of two successive characters.

This system has evolved as being the simplest method of transmitting 338 display files to the 338, given the way the TDOS system is set up. At present, only a single character message can be transmitted to the Spectra 70 from the 338. This must be of the form STX b ETX where b can be either A, B, C, or D. A signifies that the 338 is ready to receive a message, B that a message has been received, C is A and B combined and D is used to set a logical variable to TRUE in the user program. The last message essentially acts as a sense switch. A call to CRXIT in the user program will set a logical variable to TRUE if the message has been transmitted since the previous call.

The above summarizes the communications facilities that are in general use between the Spectra 70 and the 338. Work is under way to extend the facilities in two ways. Bill Park has been extending the program used for communications under TDOS; Jeff Kulick is trying to develop communications under the TSOS operating system. Under TDOS, a system for transmitting messages in both directions has been developed using special characters after the STX to indicate whether or not the previous message has been successfully received and the type of information contained in the current message. Provision has been made for transmitting 8 bit bytes, 12 bit 338 words and ASCII characters in suitably economical formats. Work on the communications, encoding and decoding routines is essentially complete and they will soon be available for all users.

As far as transmission under TSOS is concerned, an attempt is being made to make the 338 simulate a Video Display terminal. This is still in the very early stages of testing.

12. CONCLUSIONS

In the Introduction, it was stated that the aim of the report was to find out the limitations of the graphics facilities that are available to students and to make some suggestions as to what might be done to improve them.

In the hope that the suggestions will be both practical and useful, two assumptions will be made. The first is that should a need for additional hardware be demonstrated then funds might be available to purchase it. The amount of money that might be found is assumed to be significantly less than the cost of replacing the DEC 338 to emphasize the aim of making the best possible use of what equipment is available.

The second assumption is that the 7040 will not be available after May of next year. The 7040 is currently scheduled to stay until then, after which time its availability cannot be guaranteed. In this case, the Spectra 70/1:6 will have to provide essentially all central computing facilities for graphics users.

With these assumptions in mind, hardware and software aspects of the current facilities are now discussed.

Hardware

The first conclusion that has been made is that although the DEC 338 has a number of disadvantages, they are not serious enough to represent major limitations to research in the immediate future.

The disadvantages most frequently mentioned by users were the slow speed of the display and the limited core and disc memory of the PDP-8. The slow display speed limits the amount of information that can be displayed before flicker is produced. Short of replacing the 338, the

only improvement that could be made would be to add a hardware character generator that would reduce the time taken to display characters by a factor of the order of 2 to 5 depending on circumstances. The overall improvement in the display speed would only be fractional as the average user is not limited solely by the amount of text he is displaying. It does not therefore appear that the addition of a hardware character generator would produce a significant enough improvement to justify its cost.

Problems associated with the small memory size of the DEC 338 are obviously very serious if it is used as a stand-alone machine. Although users of the 338 have always in principle had access to the 7040 via a dataphone link, its use as part of an interactive system has been limited as it is not a time sharing machine. Anyone devising a system has therefore been faced with the problem that most of the time the 7040 is not available for on-line use.

In this situation, more demands are made on the 338 as a processor than if continuous access to a time sharing computer were possible. It may therefore be that future systems will make less stringent demands on the 338 so that the need for extra memory will be reduced.

Other suggested improvements to the hardware which the users suggested included a data tablet and a real time clock. The need for either of these appears less universal than for memory or for a character generator. However a clock would be useful for generating regular interrupts and costs much less than the other alternatives.

It is perhaps worth noting that all of the major improvements suggested would cost of the order of \$6,000-\$8,000. In the event of the 7040 leaving, it might be more worthwhile to purchase a KV graphics

system for the PDP-8 in D.R.L. This would cost \$8,000 and although its use of a storage tube display makes it a less flexible system than the 338, its ability to display flicker free much more text and vectors as well as circles might make it a useful addition to the graphics facilities at a relatively modest cost.

Software

The most urgent software problem is the lack of a good assembler for DEC 338 programs other than the PDPMA1 assembler on the 7040. The continued availability of the 7040 cannot be guaranteed; also, it is inconvenient to use because of the slow turnaround time and the need to punch paper tape.

It therefore appears that the provision of a 338 assembler on the Spectra 70 is an urgent necessity, particularly as it would be some time before it could be the sole means of assembling 338 programs. As PDPMA1 is an extension of the 7040 MAP assembler, it may not be possible to duplicate all of its features on the Spectra 70. In this case, considerable time may be needed to convert existing 338 programs once the assembler is available.

It would be useful if an assembler could be written to generate 338 programs in a page-to-page relocateable form. This would make individual routines useful to more users and reduce the time used by the assembler as it would be no longer necessary to reassemble complete programs in order to make changes.

Once an assembler is available on the Spectra 70, the preparation of DEC 338 programs will be considerably easier. The Spectra 70 File Editing system can be used for creating source programs and assembled programs could be transmitted to the 338 over the dataphone.

An efficient assembler and editing system on the Spectra 70 will greatly speed up the development of new systems for the 338. A further modest improvement might be made by establishing a program library containing well documented programs and routines in source or assembled form. The library might contain character generation, communications routines, etc., and would facilitate much of the interchange of programming techniques between users that already exists.

The interchange of techniques between users and the ease with which new users can adapt existing systems to their own needs are not limited by the lack of a program library, rather by the lack of a common operating system for the DEC 338. This will be discussed in the next section.

A General Purpose System

Reasons for developing a general system

All graphics systems that have been so far developed in the Moore School have been designed to serve particular applications. As a result, they are not in general useful to new users of the graphics facilities, who therefore develop new special purpose systems.

Small parts of existing systems can be adapted for new applications and the first part of this report has indicated a number of cases where this has been done. Because the systems that have been developed are so highly specialized, the interchange of routines between users, even those working on closely related problems, is not really as extensive as it might be. For example, the system developed for chemical plant design performs much the same functions as one that might be developed for electrical circuit design. Its adaptation to do this would represent so much effort that it might be quicker to develop another system.

A general purpose system would promote the interchange of techniques between users and make it much easier to develop new applications of graphics techniques.

Nature of the general purpose system

It would be difficult to substantiate a statement that a single system could be developed that everyone would find useful. The system described here represents one that would serve a large but finite range of applications which appear well suited to the available hardware.

As the 338 is only a small computer, it is desirable that a general purpose system should provide a link between the DEC 338 and a program running in the Spectra 70. The nature of the interface between the two machines would appear to be the most important factor to be taken into account when deciding the type of application that a general purpose system should serve.

It does not appear that applications that require a rapid interchange of messages between the 338 and the Spectra 70 can be effectively implemented. The reasons for this are that under TSOS several seconds may elapse before the Spectra 70 can respond to a message from the 338, and that the dataphone link is slow so that the maximum data transfer rate is 2400 baud.

The general purpose system described here is therefore intended to give a user the facilities necessary for creation of a display, its transmission to the Spectra 70 for processing, and for output of results rather than facilities for implementing real-time applications.

In order to make the system available to as many users as possible, that part of the system resident in the 338 should be such that a user

should not have to do any 338 programming at all. Also the system should provide an interface to FORTRAN programs as FORTRAN is the most widely used language for engineering and scientific problem solving.

Data Structures

In order to provide a well defined set of functions for creating displays and for their interpretation it is necessary to adopt specific data structures for the system. The data structures should have a number of properties.

It should be possible to reference as a single unit the sequence of vectors in a display that represent an object. The display should have a hierarchical structure so that objects can be described in terms of their components. It should be possible to associate with any object or component a series of attributes that describe properties of the object which are not directly associated with the display information. It is also desirable that the data structure should contain topological information such as is needed to describe a network in terms of nodes and arcs. This feature is less important than the others as, in principle, topological information can be derived from the geometry of the display.

It is proposed that the general purpose system be designed to operate on three related data structures, each of which has the first three properties listed above. Two of the data structures would be in the Spectra 70 and would contain two and three dimensional data, respectively. The data structures in the Spectra 70 would contain display information in a form suitable for interpretation by the user program and would need to be flexible enough to be incorporated in a larger data base associated with the user problem. The structure in the

338 would contain display information as actual display instructions that could be executed to produce the display.

In both cases, the data would be accessed via control blocks or groups. In order to illustrate how a data structure could be composed, a possible implementation of the 338 data structure will be described in more detail.

A display structure would be formed by linking together a series of control blocks or groups. Each group would have the following 16 word format.

- | | | |
|----|--|------|
| 1 | ENTER DATA STATE | |
| 2 | Y (NONINTENSIFIED) | |
| 3 | X (ESCAPE) | |
| 4 | PUSHJUMP | |
| 5 | ADDRESS OF DISPLAY DATA | |
| 6 | SKIP ON PUSHBUTTON | |
| 7 | PUSHJUMP | |
| 8 | ADDRESS OF TEXT STRING | |
| 9 | PUSHJUMP | |
| 10 | ADDRESS OF COMPONENT GROUP OR SUBGROUP | |
| 11 | JUMP | EDS |
| 12 | ADDRESS OF NEXT GROUP | Y |
| 13 | SPARE | or X |
| 14 | SPARE | POP |
| 15 | FLAGS | |
| 16 | IDENTIFIER | |

Words 1-3 contain the parameters of an unintensified raster scan to move the 338 beam before starting to display the information associated with this group.

Words 4 and 5 contain a pushjump to a sequence of display instructions in vector or short vector format. These instructions represent an object associated with this group and form a closed figure so that when control is returned to the group by a pop, the total beam displacement is zero. This form enables multiple users of the same instructions to form respective displays and aids display editing.

Words 7 and 8 contain a pushjump to a sequence of display instructions needed to display a text string used to describe attributes associated with the group. Again, control is returned via a pop.

Word 6 contains a skip option so that display of the text can be suppressed.

Words 9 and 10 contain a pushjump to a group which is in a lower level of the data structure such that if the first group represents an object, the lower level group represents a component of the object.

Words 11 to 14 contain a jump to another group at the same level or a vector used to close the sequence of display instructions at this level before returning control to a higher level group.

The last two words contain respectively flags indicating the history of the group, i.e., whether it originated in the Spectra 70 or the 338, whether or not it can be deleted or modified, etc.; and a pointer to a table containing the name associated with this group.

The way in which groups are linked together to form a display structure is best illustrated by the example shown in Figure 1.

Groups 1, 2 and 3 would represent objects at the same level in the structure. Groups 4 and 5 would be components of group 1 and 4 itself has a component group 7. Group 6 would be a component of group 3. Groups 3, 5, 6 and 7 would end in vectors so that the series of groups (1,2,3), (4,5), (6), (7) would each be closed figures so that any object and its components can be added or deleted from the display without disturbing the rest of the display.

This type of display structure has a number of useful features. The actual display instructions are in closed form so that they can be called by any number of groups. Groups can be easily added and removed from the structure. An object can be moved around on the screen easily (for example, group 1 can be moved by adding opposite sign increments to the vectors in groups 1 and 2) while maintaining the same spatial relationship between itself and its components.

A disadvantage of this particular way of implementing the structure is that extensive use is made of the 338 subroutine instructions. A push and pop together take $31.5\mu\text{sec}$ to execute. This time would result in a significant increase in the time taken to display a file containing a great deal of structured information.

Tom Johnson has suggested that this disadvantage might be overcome by extending a technique developed by Dave Kristol for rapid character display. Each sequence of display instructions called by a group would end in a STOP rather than a POP. While the sequence was being displayed, the PDP-8 would execute a routine designed to interpret a group and complete the address of the next sequence of display instructions to be executed. When this computation was complete, the PDP-8 would wait for the internal stop flag and reinitialize the display at the newly computed

address. Stopping and restarting the display would take only about 10 μ sec., so that significant time would be saved provided the necessary calculation could be performed during the display of the previous sequence.

Interpretation rather than execution of the groups would enable them to be much more compact, in fact the number of words in each group could be reduced by a factor of two.

The data structures in the Spectra 70 would have the same structural form as in the 338 in order to make the system simple to use. The representation of control information and coordinate data would of course be different. Coordinates would be in homogeneous form and represented as floating point numbers. The vector (really a linear transformation) at the start of each group would be generalized to a homogeneous transformation that would relate the coordinate system of the data referenced by a group to that of a group higher in the structure. For example in Figure 1, the transformations of groups 4 and 5 would relate their coordinate systems to that of group 1.

Facilities Available to the User

The system would be organized so that that part in the 338 would enable the user to create and view displays and also transmit them to the Spectra 70. Only in the Spectra 70 would there be facilities for interpreting and processing graphics information.

The system would be implemented in stages that would differ in the range of facilities available to the user. An initial system might give the user the following facilities.

In the 338 the user would be able to create, name, and link groups to form a structured display. He would be able to draw, type in, or copy display information and text associated with each group. Editing facilities enabling the user to modify display information, text and the structure itself would also be provided. Routines for rotating and scaling displayed objects would be available; then use would be restricted to avoid inadvertant transformation of instructions referenced by several groups.

The user would be able to transmit a completed display to the Spectra 70, possibly defining it to be an addition to a modification of an existing display structure and also specifying a scaling factor so that a display structure in the Spectra 70 could be built up in stages to have a greater coordinate range than is possible in the 338.

In the Spectra 70 the same facilities for creating a display would be available using subroutine calls for the application program. In addition the user would be able to define the transformations associated with each group and execute them to determine absolute vector coordinates. In general, interpretation of the text strings generated in the 338 would be application dependent. A few interpreters could be provided, for example to enable a user to input floating point numbers.

As far as output of displays from the Spectra 70 is concerned, all output would be from the two dimensional data structure. A user would access data in the three dimensional structure by specifying a transformation to be used to project it into the two dimensional structure. The projection would produce a planar representation of the data of the same structural form that it had in the three dimensional

structure. Output from the two dimensional structure to the 338 would involve windowing and scaling the data. The display resulting from output from the Spectra 70 would be structurally identical to an equivalent display composed at the 338 console so that a user could interact with his applications program to compose a display.

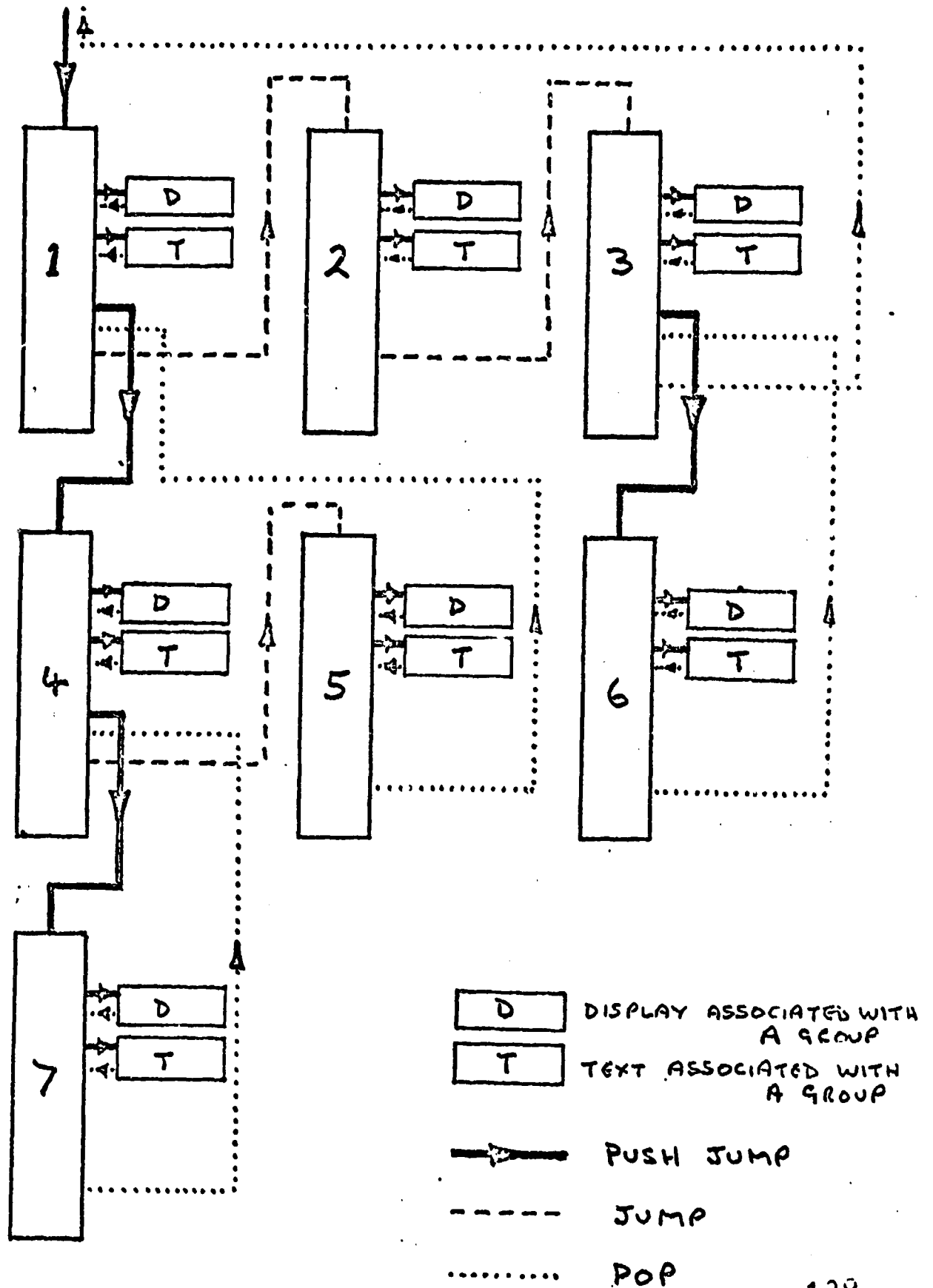
Subsequent versions of the system would give the user facilities for direct input of three dimensional data from the 338 console, permit the applications program to direct the flow of the program in the 338 and provide for transmission of standard curves or objects between the two machines in a compressed form.

The system would be designed to be sufficiently modular and well documented so that a user might add additional features needed for his specific application.

The overall aim would however be to produce a system that enabled users to apply graphics techniques by writing applications programs in FORTRAN rather than by programming the DEC-338.

FIG. 1

TO HIGHER
LEVEL GROUPS



University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING

AN INFORMATION STORAGE AND RETRIEVAL
SYSTEM FOR PHYSICAL PROPERTIES OF
CHEMICALS

Daniel S. Poznanovic

A thesis submitted to the Faculty of The Moore
School of Electrical Engineering in partial
fulfillment of the requirements for the degree
of Master of Science in Engineering (for grad-
uate work in Computer and Information Sciences)

Philadelphia, Pennsylvania
December, 1969

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING

Title of thesis: An Information Storage and Retrieval
System for Physical Properties of
Chemicals

Abstract:

This paper describes a prototype information storage and retrieval system for the physical properties of pure chemicals and mixtures. The system is composed of a data base, a library of property estimation routines and generalized storage and retrieval routines. The data base stores several forms of numeric property data. The retrieval routines are called from within FORTRAN application programs and are capable of retrieving both stored and computed physical property values. Both the structure and use of the system are discussed.

Degree and date of degree: Master of Science in
Engineering (CIS)
December 1969

Daniel S. Poyanov
AUTHOR

Harold D. Seider
FACULTY SUPERVISOR

ACKNOWLEDGEMENTS

I wish to thank my wife for her encouragement and understanding throughout the work leading to this thesis.

I wish to express my gratitude to Dr. Warren D. Seider, my supervisor, for his guidance and encouragement.

I would like to thank the members of the Chemical Engineering Calculation System Project, particularly Messrs. Sezer Soylemez and Peter Ham, for their suggestions and criticism.

I would also like to thank Mr. Hajime Komaki for testing the property system through extensive use.

I would like to express my appreciation to Mrs. Sarah Amos for the typing of this thesis.

Finally, my appreciation to the Office of Aerospace Research of the United States Air Force and to the Esso Education Foundation is acknowledged for their support of this research.

TABLE OF CONTENTS

	<u>Page</u>
LIST OF FIGURES	v
CHAPTER I INTRODUCTION	1
1.1 Objective	1
1.2 Order of Presentation	3
CHAPTER II THE PROPERTY SYSTEM	4
2.1 Design Considerations	4
2.2 Property System Structure	5
2.2.1 Storage Subsystem	5
2.2.2 Retrieval Subsystem	7
CHAPTER III THE DATA BASE	10
3.1 Data Records	10
3.2 Directory	12
3.3 Data Pool	14
3.4 Justification of Directory Levels	16
CHAPTER IV RETRIEVAL	19
4.1 Definition	19
4.1.1 Accepting Retrieval Requests	20
4.1.2 Locating Data Records	20
4.1.3 Returning Property Data and Property Values	24
4.2 Retrieval Routines	24
4.3 Justification of Retrieval Scheme	25
CHAPTER V THE APPLICATION PROGRAM	27
5.1 Definition	27
5.2 Requesting Property Values	27
5.2.1 Pure Substance Property Value Requests	28
5.2.2 Mixture Property Value Requests	33
5.3 Property System Activation	38
CHAPTER VI RUNNING AN APPLICATION PROGRAM	42
6.1 Component Identification Table	42
6.2 Retrieval Constraint Table	44

TABLE OF CONTENTS (continued)

	<u>Page</u>
CHAPTER VII DATA STORAGE	58
7.1 Adding New Personal Property Data	59
7.2 Updating Existing Data	67
7.3 Deleting Existing Data	67
7.4 Storage Control Deck	68
CHAPTER VIII PROPERTY ESTIMATION ROUTINES	71
8.1 Definition	71
8.2 Estimation Routine Types	72
8.3 Estimation Routine Conventions	72
8.4 Requesting Property Values and Data	75
8.4.1 Requesting Property Values	75
8.4.2 Requesting Property Data	76
8.5 Example Estimation Routine	82
8.6 Entering an Estimation Routine Into the Library	85
CHAPTER IX CONCLUSION AND RECOMMENDATIONS	87
9.1 Conclusion	87
9.2 Recommendations	87
Appendix I	90
A Physical Property Codes	91
B Data Base Component Codes	92
C Data Type Codes	92
D Estimation Routines	93
E Property System Messages	100
Appendix II Logic Diagrams	103
Appendix III U of P IBM 360/75 JCL	107
Appendix IV Input Language Specification	108
BIBLIOGRAPHY	111

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
II-1 Storage Subsystem	6
II-2 Retrieval Subsystem	8
III-1 Directory Structure	13
III-2 Directory Elements and Tracks	15
III-3 Data Element and Track Structure	17
IV-1 Directory Search	22
V-1 Component Identification Table	29
V-2 Example Use of the PPCP Routine	31
V-3 Example Use of the PPCF Routine	34
V-4 Example Use of the PPCS Routine	37
V-5 Example Application Program Flowchart	39
V-6 Example Application Program	40
VI-1 Component Identification Table and Deck	43
VI-2 Property Data vs. Property Values	46
VI-3 Retrieval Constraint Table	49
VI-4 Example Retrieval Constraint Table	51
VI-5 Retrieval Constraint Deck	52
VI-6 Variant Information Deck	54
VI-7 Application Program Run Deck	55
VI-8 Example Application Program Results	57
VII-1 Data Record	61
VII-2 Punched Form of Data Records	61
VII-3 Example Data Records	65
VII-4 Storage Control Deck	69
VIII-1 Example Use of the SER1 Routine	78
VIII-2 Example Use of the SER2 Routine	80
VIII-3 Example Use of the SER3 Routine	82
VIII-4 Estimation Routine Logic	83
VIII-5 Example Estimation Routine	84

CHAPTER I

INTRODUCTION

1.1 Objective

Chemical physical property values are of importance in chemical engineering process calculations. Since the mid - 1950's computers have played an increasingly significant role in chemical engineering process simulations. The need for development of computer systems for supplying physical property values has been expressed by Yen (1), Zseleczky (2), and Shannon (3).

A common approach to supplying physical property values to chemical engineering application programs is by reading property data from cards. The property data is often assumed to be constant for the temperature and pressure ranges under consideration. When this assumption cannot be made, that is when variable dependence (for example, temperature and pressure) must be taken into account, property estimation procedures are required. The estimation procedures are often coded into the application program with correlation coefficients, tables, and constants being read from cards by the application program. A more sophisticated approach to providing property estimation procedures is to separate estimation algorithms from application program in the form of subprograms (functions or subroutines) that contain all the information necessary for property estimation (1,2). The application program calls on the property estimation subprograms for property values instead of reading data from cards.

The A.I.Ch.E. (8,9,10) has sponsored development of a large computer system of subroutines for estimation of physical properties. Systems for computer-aided design and simulation (4,5,6,7,14) have been developed that include property estimation subroutines. The system developed by Beirute (11) and that of Johnson (14) are the most general systems of those associated with design systems.

The above approaches to supplying property values are limited in that the estimation procedures used by a given program are fixed; that is, the estimation procedures are linked directly to an application program by program coding. When changes in mixture components and/or temperature and pressure range occur, the property estimation procedures used by the application program often must be changed. The approaches above require modification within the application program especially when property estimation procedures have been incorporated into the coding of the application program. Such changes have practically eliminated the possibility of establishing general chemical process unit calculation material and energy balance, design, and simulation libraries. Maintenance of a general purpose unit program library becomes expensive when general purpose routines must be modified for each variation in chemical mix and operating conditions. Hence, few general purpose unit program libraries are widely used today.

The emphasis in previous work has been in development of systems that generate physical property values and not in the development of information storage and retrieval for physical properties. Most of the systems previously developed lack flexibility; they were designed specifically for use by special user programs. None of the systems contain a data base for storage of physical property data.

147

The objective of this thesis is to present a prototype information storage and retrieval system for chemical physical properties that is unique:

- 1) in its ability to retrieve both stored and computed physical property values
- 2) in its structure that allows any FORTRAN program to request property values during program execution and
- 3) in its method for identifying and requesting physical property values for pure chemicals and mixtures.

1.2 Order of Presentation

Chapter II presents an overview of the property system, identifying the components making up the system, and discussing briefly their function. Chapter III details the purpose and structure of the physical property data base. The concept of a generalized retrieval routine is discussed in Chapter IV together with the retrieval scheme used by the system.

Chapter V, VI, VII and VIII are directed to users of the property system and present system conventions to be followed by the user. Chapter V defines the application program and presents the conventions for using the generalized retrieval routines. Chapter VI explains the procedures for running an application program and discusses the conventions for constraining the estimation of property values to meet the standards set by each individual user. The conventions for entering personal property data into the system's data base are discussed in Chapter VII. The preparation of property estimation routines is discussed in Chapter VIII. The final chapter, Chapter IX summarizes and contains suggestions for future work. The Appendices contain lists and explanation of each code used by the system together with messages generated by the property system.

148

CHAPTER II

THE PROPERTY SYSTEM

2.1 Design Considerations

Physical property values are required in chemical engineering process calculations. The chemical engineer often performs process calculation through the use of FORTRAN programs. Several computer-aided design systems for chemical processes are written in the FORTRAN language (12). Therefore, any system that is to provide property values for pure chemicals and mixtures for process calculations should be compatible with FORTRAN programs if the system is to be useable. The property system is therefore written entirely in the FORTRAN programming language.

Physical property data can take the form of constants, correlation coefficient, or tables, therefore, the property system contains a data base that can store each of the data types. When properties can be expressed as a functional relationship of independent variables such as temperature and pressure, subprograms for estimation of property values are prepared. The property system contains a library into which estimation routines can be entered.

Within a FORTRAN application program requests for property values must be simple. The request must specify the property value to be obtained without indicating the method to be used to retrieve the value from the data base or the method to be used to estimate the value. Once a request for property data has been encoded into an application program, changes must not be required when mixture components and/or independent

variable ranges (such as temperature and pressure ranges) change. Generalized retrieval routines with standardized arguments are provided for use within an application program. The generalized retrieval routines determine at the time of execution of an application program the method for obtaining requested property values.

The engineer must be able to constrain retrieval of property values to meet his own standards. Variant information constraining retrieved property values can be specified by the user externally to an application program.

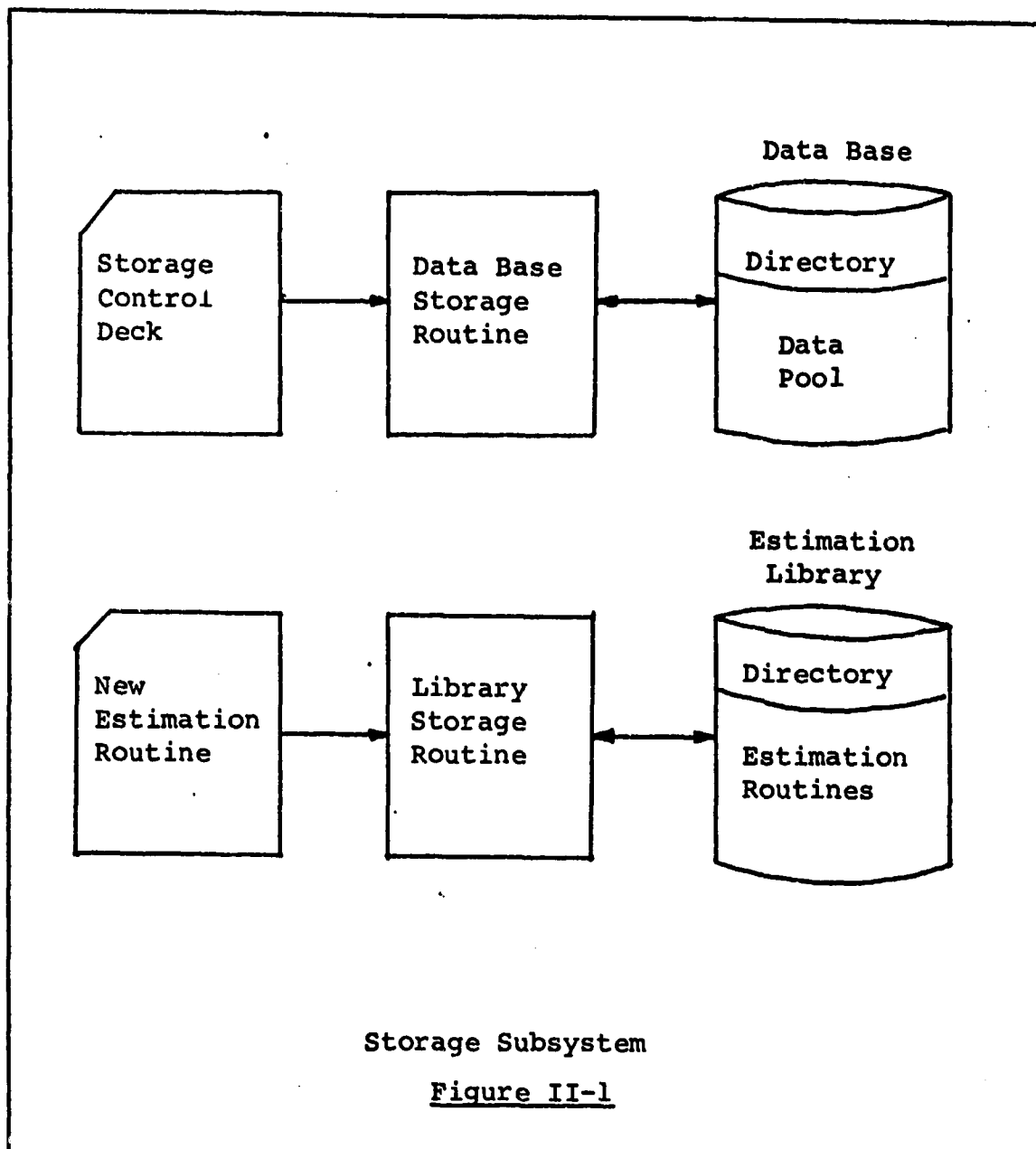
The state of the property estimation field is dynamic (10), obsolescence can be prevented only if the system is open ended. The property system allows additions, updates, and deletions to data stored in its data base and allows insertion of new estimation subprograms into its estimation routine library.

2.2 Property System Structure

The discussion below outlines the structure of the property system, and the information flow among the components of the system. The property system is decomposed into two subsystems (1) the storage subsystem and (2) the retrieval subsystem.

2.2.1 Storage Subsystem

The storage subsystem adds, updates, and deletes stored property data and enters new estimation routines into the system library. Figure II-1 illustrates the components and lines of information flow of the storage subsystem.



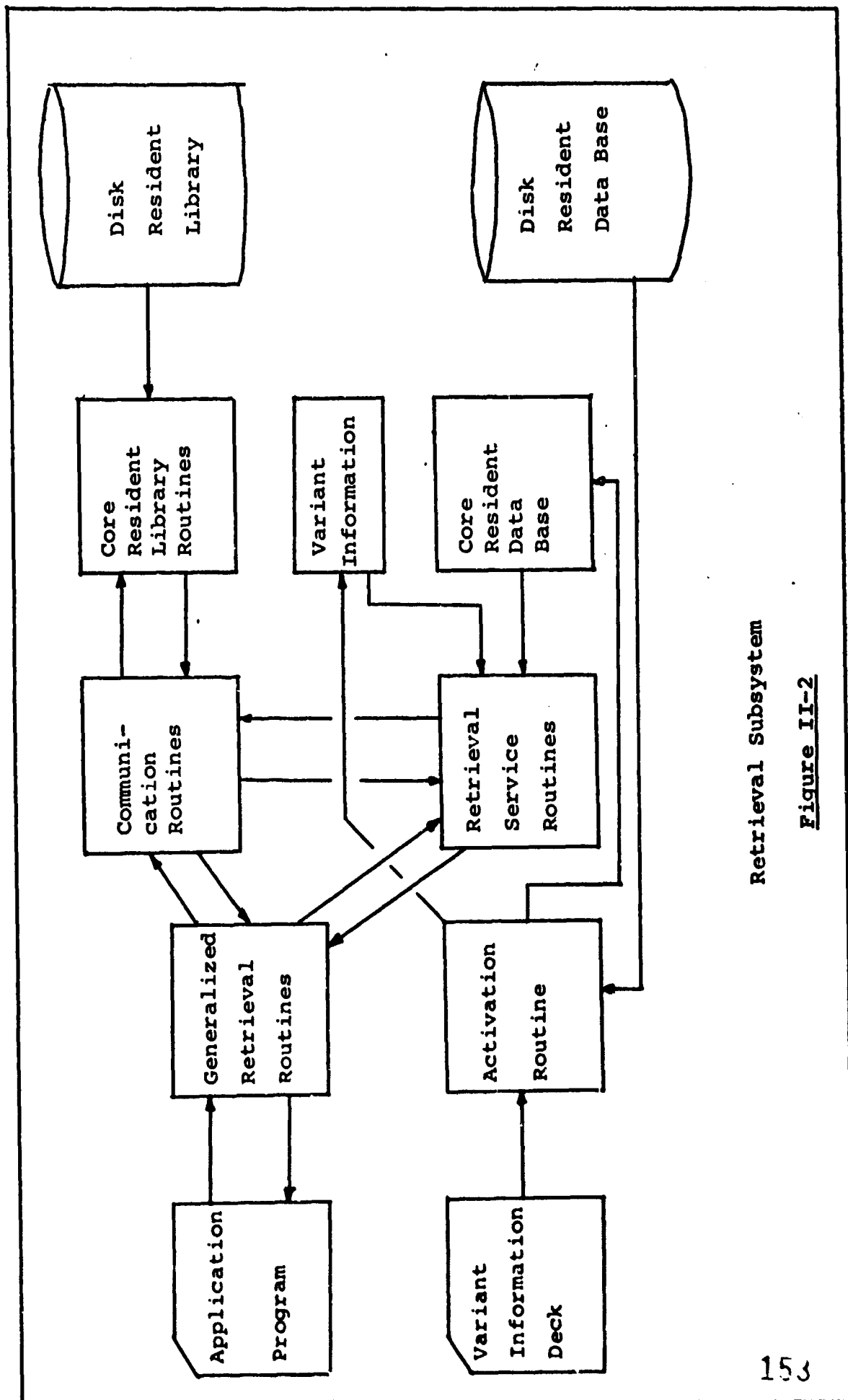
The user modifies the contents of the data base by supplying a storage control deck composed of data records containing identifiers and property data, and commands to the data base storage routine. (The command language specification is presented in Chapter VII.) The data base storage routine reads the storage control deck and performs the specified modifications to the system's data base.

New estimation routines are entered into the estimation library through the use of library storage routines. The library storage routines are dependent upon the particular computer facility used in the implementation of the property system, and are composed of a FORTRAN compiler and in the case of the IBM 360 system the Linkage Editor. An estimation routine preprocessor used to insure that new estimation routines conform to system conventions will be discussed in Chapter VIII.

2.2.2 Retrieval Subsystem

The retrieval subsystem of the property system is composed of routines that are used during the execution of an application program to retrieve property values requested within an application program. Figure II-2 illustrates the components of the retrieval subsystem and the lines of information flow.

A FORTRAN application program containing requests for property values is compiled and an executable module is generated containing the application program, property system routines, and required estimation routines. The executable module is generated by a loader or linkage editor provided within the computing system that supports the implementation of the property system. The application program calls the activation routine to read the disk resident data base into core storage, and to read the variant information deck. The variant information deck contains execution time information constraining property value retrieval to meet the user's particular standards. (Chapter VI contains the command language specification of the variant information deck). Property values are requested during application program execution through calls to the generalized retrieval routines.



Retrieval Subsystem

Figure II-2

The generalized retrieval routines use the retrieval service routines to interrogate the data base in order to locate property data meeting the request. The generalized retrieval routines either return data stored in the data base or call one of the communication routines that select an estimation routine to compute the requested property value. The generalized retrieval routines return to the application program either stored or computed property values.

In subsequent chapters each component routine of the property system is discussed in detail.

CHAPTER III

THE DATA BASE

The property system's data base contains physical property data for pure chemicals and mixtures. Property data is logically organized into data records. Physically the data base is composed of a directory and a data pool. The data base normally resides in disk storage, and is read into core storage by the activation routine during the execution of an application program. Loading a complete property system data base into core storage in the case of this prototype system corresponds to loading essential directories and a segment of the data pool of a full scale implementation of the property system. In Chapter IX recommendations are made concerning the disk resident data base and its purpose in a full scale implementation of the property system.

3.1 Data Records

The data base is logically organized into data records. Each data record is composed of seven attribute values together with property data. The attribute values completely characterize each data record. A data record contains:

1. a physical property code (for example 401 = vapor enthalpy as a function of temperature and pressure).

2. the ranges of the independent variables associated with the property for which the data is valid.
(for example, 200°K-400°K for temperature and 1 atm - 3 atm for pressure),
3. a code specifying the contributor who entered the data record into the data base (for example 427 = John Jones),
4. the data base component code(s) of the pure chemical or mixture for which the data is applicable (for example, 2 = methane),
5. the number of the estimation routine that produces property values from the data contained in the data record. (for example, 20 = third degree polynomial routine),
6. the maximum percentage error expected in the property value produced by the estimation routine (for example 3%),
7. the data type code that indicates the form of the property data contained in the data record (for example, 2 = correlation coefficient),
8. the property data itself (for example, the third degree polynomial coefficients a,b,c,d).

Appendix I contains the lists of established codes for each attribute.

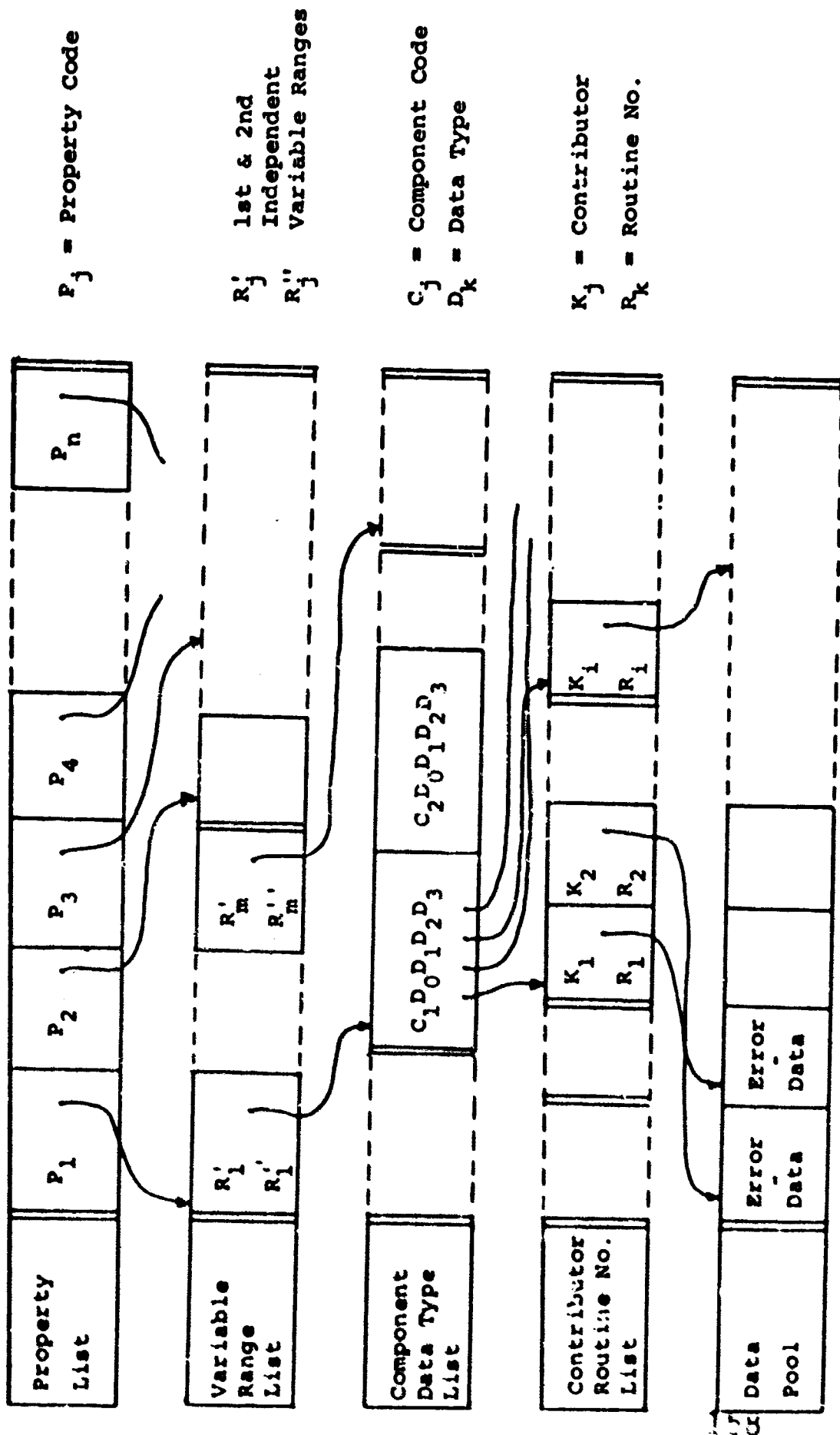
Conceptually data records can be thought of as the basic unit of storage in the data base, but actually the content of a data record is divided into directory elements and data pool elements,

3.2 Directory

Directory elements form the nodes of an inverted tree structure of lists called the directory. The directory provides a mapping from a set of property data attribute values to an element of the data pool containing property data.

The directory is of depth three, that is, there are four nodes including the root in every inverted tree. Each node represents a list of directory elements. Each directory element within a nodal list has a unique content. The branches of each tree represents pointers from a directory element in a list at depth i to the head of a list at depth $i+1$, with the root being a list at depth zero. The terminal points of the branches of the directory (inverted tree) are data pool elements.

The single node at depth zero is a list of directory elements each containing a unique property code and a pointer to a node at depth one. The nodes at depth one are each a list of directory elements, each containing two independent variable ranges and a pointer to a node at depth two. The nodes at depth two are each a list of directory elements, each containing a data base component code and four pointers, one for each data type, to a node at depth three. The nodes at depth three of the directory tree are each a list of directory elements, each containing a contributor code, an estimation routine number and a pointer to an element of the data pool. Figure III-1 illustrates the inverted tree structure of the



Directory Structure

Figure III-1

data base together with the list structure.

In the figure each box represents one directory element and the set of contiguous directory elements between double bars represents a list making up one node. From the figure, one sees also that the directory is made up of four master lists, the property list, the variable range list, the component-data-type list, and the contributor-routine-number list. Each master list represents the union of all nodes at the same depth in the directory tree.

The basic building unit of each master list is a track. A track is a set of contiguous directory blocks. The size of a track can be set to any value for each master list, but once set, each track within a master list is fixed in size. Each sublist making up a node consists of one or more tracks. The tracks in a sublist need not be contiguous since the last two directory elements in each track contains a forward link to the next track in the sublist and a backward link to the previous track in the sublist. Figure III-2 illustrates the layout of directory elements and tracks within each master list.

3.3. Data Pool

The data pool is a master list of data elements. Each data element is a terminal point at a branch from a directory element at depth three. Each data element contains the maximum expected error associated with the data together with property data itself. Figure III-1 illustrates the data pool.

The format of each data element depends upon the data type of the data contained in the element. There are four data types: constant data, (data type 1), correlation

Property List	Property Code	Ptr. to V - R List		Back Link	Forward Link
---------------	---------------	--------------------	--	-----------	--------------

Variable Range List	First Range	Second Range	Ptr. to C-D List	Back Link	Forward Link
---------------------	-------------	--------------	------------------	-----------	--------------

Component Data Type List	Component(s)	Ptr. to C-R List	Ptr. to C-R List	Ptr. to C-R List	Back Link	Forward Link
--------------------------	--------------	------------------	------------------	------------------	-----------	--------------

Contributor Routine No. List	Contributor Code	Routine Number	Ptr. to Data Element	Back Link	Forward Link
------------------------------	------------------	----------------	----------------------	-----------	--------------

Directory Elements and Tracks

Figure III-2

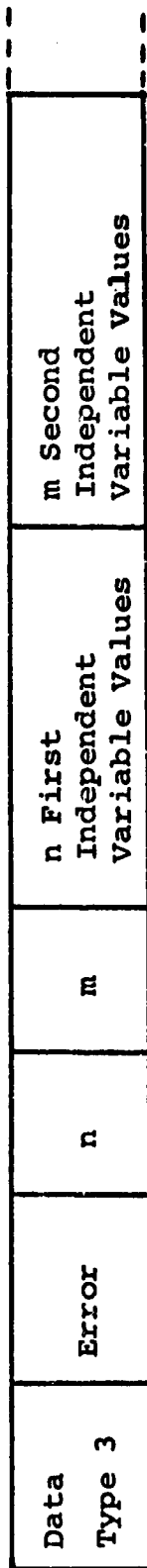
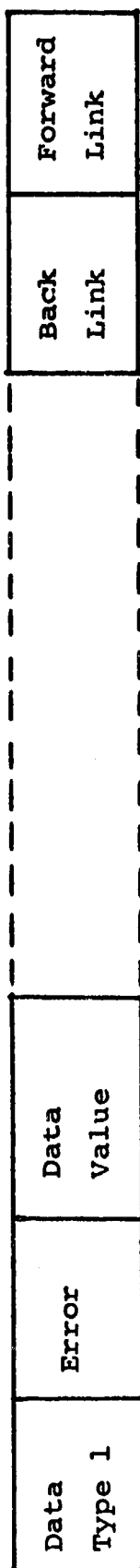
coefficients (data type 2), tabular data (data type 3) and null data (data type 0).

The data pool, like the other master lists, is built of tracks whose size is fixed but can be specified at the time of implementation. Figure III-3 illustrates the layout of data elements and data pool tracks.

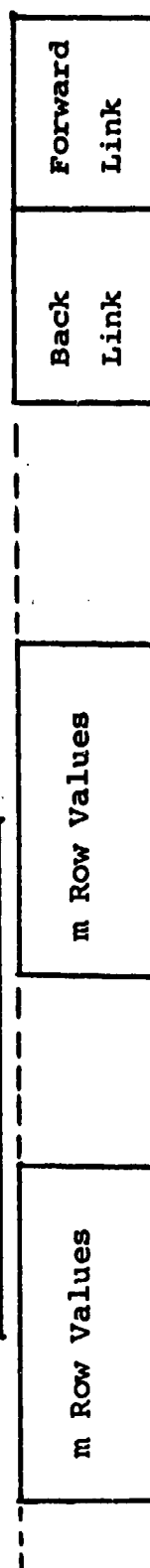
3.4 Justification of Directory Levels

The data base directory provides a mapping from a set of attributes to a data element within the data pool. The mapping is accomplished through traversing the directory tree from the root through connected nodes to the data pool. It will be seen in Chapter IV that the minimum set of attribute that can be specified in a request for property values is a property code, values of two independent variables, and a component code (or codes in case of a mixture). The remaining four attributes may be optionally specified. Therefore, the mandatory attributes were given the highest level nodes in the tree structure to insure that the most "promising" branches of the directory tree are traversed first. Once the directory tree is traversed to a depth of two, any branch leading to the data pool may be taken, when none of the optional attributes are specified.

The combining of attributes at depth two and three saves core storage, since the two additional master lists required if combining had not been done, would be lists made up of tracks of length one or two. Such tracks require more core storage for forward and backward links than for productive directory elements, a very inefficient use of core storage. The maximum expected error is contained within each data element since it can be expected that the values of maximum



n sets



Data Element and Track Structure

Figure III-3

expected error for each data record will vary considerably. If maximum expected error were given a directory level of its own, the effect would be to construct a maximum expected error list (node) for each data element; such a situation constitutes inefficient use of storage.

CHAPTER IV

RETRIEVAL

4.1 Definition

Within the context of the property system, retrieval is a multi-step task composed of:

- (1) accepting a retrieval request that identifies property data or values,
- (2) locating data records within the data base that meet the request,
- (3) retrieving data contained in the data records or values computed by an estimation routine using data contained in the data records and independent variable values.

Before continuing, some terminology must be defined. In the property system a distinction is made between property values and property data. Property values are determined using a property estimation procedure. Property data is raw data in the form of tables, correlation coefficients and constants stored in the system's data base. A property estimation routine uses property data stored in the data base, and/or property values produced by other estimation routines to produce property values. For example, a correlation routine (property estimation routine) uses correlation coefficients (property data) to determine estimated property values.

Only in the case of "constant" data are property values and property data equivalent. No estimation procedure is required to transform a constant, for example, molecular weight, to a property value.

4.1.1 Accepting Retrieval Requests

A request for property values or data is made within a FORTRAN program or subprogram through a call statement to one of the FORTRAN subprograms called the retrieval routines. A request must identify the property data or values desired within the calling program. Property values and data are identified by a minimum of three attribute values; (1) property code, (2) values of the two independent variables associated with the property and (3) the data base component code(s) of the pure chemical or mixture for which the information is desired. The values of these three attributes make up the minimum set of keys required to locate the requested property data or values. This minimum set of keys is called the compact set of retrieval keys. The compact set of retrieval keys are the arguments to the retrieval routines.

4.1.2 Locating Data Records

In order to return requested property data a data record containing the property data must be located in the data base. In order to return requested property values a data record containing required property data and the number of an estimation routine to compute the value(s) using the data must be located. The data base therefore serves the important role of providing property data and information to direct the retrieval routines to an estimation routine when a property value must be computed using stored property data.

A data record is located when a retrieval routine determines the address in the data pool of the data element that contains the data record's data part. The data elements are the terminal points of the inverted rooted tree of attribute values called the directory. Therefore, to locate a data record a retrieval routine must traverse the branches of the tree (directory) until a terminal point is reached. Traversal of a branch of the directory between a node at depth i and a node at depth $i+1$ occurs when a pointer to the head of a nodal list at depth $i+1$ is found within a directory element in a nodal list at depth i .

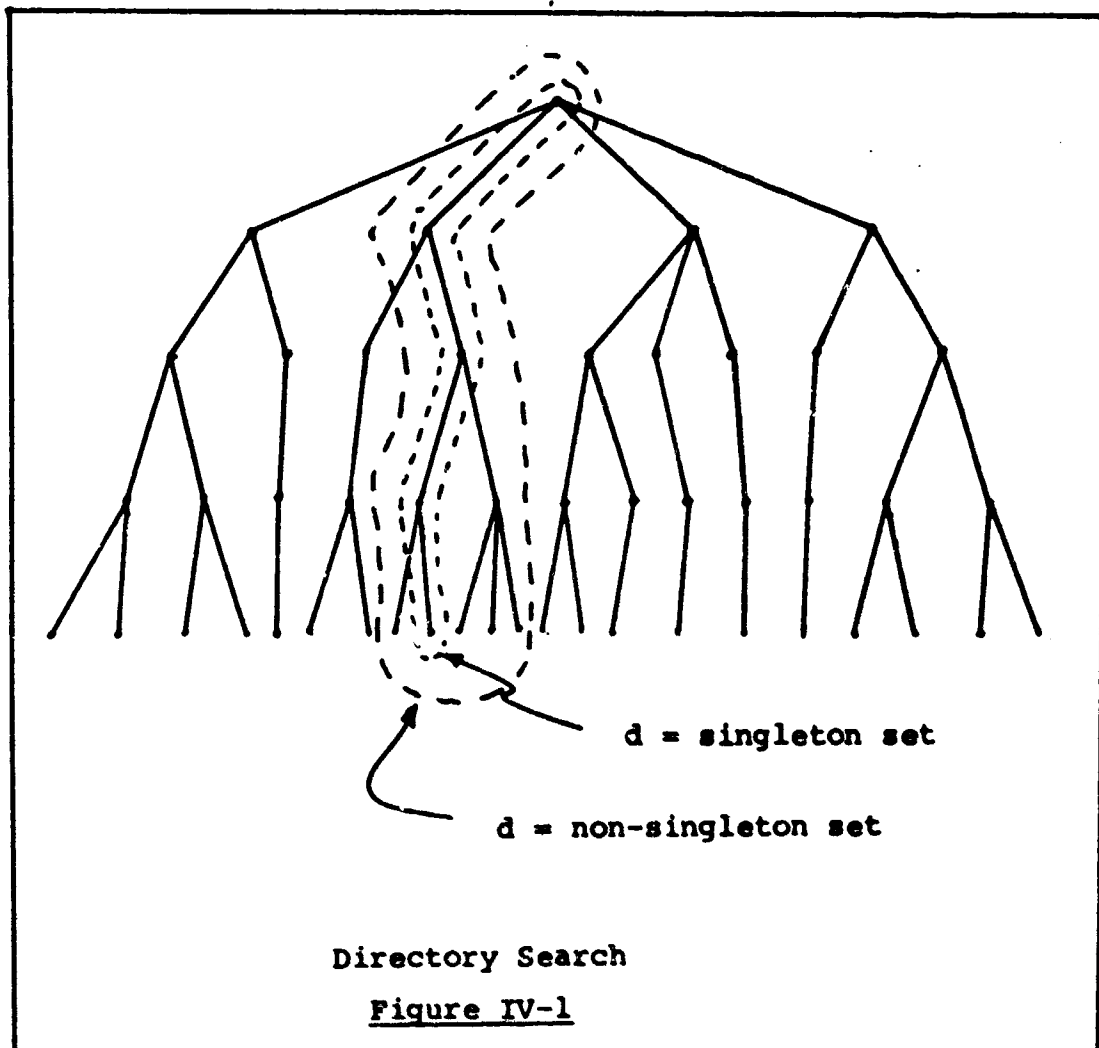
Traversal of the directory can be described in terms of a directory search function F . The domain of F is the set of seven-tuple attribute values, one attribute value for each of the seven attributes that characterize a data record. The range of F is the set of subset of the data elements within the data pool together with zero, the trap address, indicating non-existent data. Symbolically,

$$d = F(p, v, c, t, k, r, e)$$

where

d = set of data element addresses
 p = property code
 v = two independent variable values
 c = data base component code(s)
 t = data type code
 k = contributor code
 r = routine number
 e = maximum allowed error

If each of seven attributes is defined, d is the singleton set containing one data element address. If any t, k, r , or e is undefined d is a set of addresses of data elements whose data records contain the defined attribute values. Figure IV-1 illustrates the two cases.



The directory search function F is further broken down into five nodal list search functions

$$f_i \quad (i=1,5) .$$

Symbolically this can be written

157

$$\begin{aligned}
a_2 &= f_1(a_1, p) \\
a_3 &= f_2(a_2, v) \\
a_4 &= f_3(a_3, c, t) \\
a_5 &= f_4(a_4, k, r) \\
d &= f_5(a_5, e) \\
d &= F(p, v, c, t, k, r, e)
\end{aligned}$$

where a_1, a_2, a_3, a_4, a_5 represent, respectively, addresses of the headwords of nodal lists within the property master list, variable range master list, component-data type master list, contributor-routine number master list and data pool.

Section 4.1.1 explained that the compact set of retrieval keys (p, v, t) was supplied to the retrieval routines to identify requested property values or data. If the directory search function F is applied to the compact set of retrieval keys a set of data element address is obtained,

$$d = F(p, v, t, x, x, x, x)$$

where x indicates undefined attributes values. Since only one data element address is required, one of the addresses contained in d is picked arbitrarily. To insure that the engineer has complete control of the data he wishes to use, the additional four attributes can be specified externally to his application program through the use of the retrieval constraint table. By specifying important attributes the set of data elements d is made smaller, and the retrieval data meets the engineers specification. Chapter VI, "Running an Application Program", discusses the preparation of the retrieval constraint table.

4.1.3 Returning Property Data and Property Values

Once a data element has been located either the data contained in the element must be returned to the calling program or subprogram, or an estimation procedure must be applied to the data to compute a property value. If the retrieval routine called by the application program or estimation routine is of the type that returns property data, the contents of the located data element is returned to the calling program or subprogram. If the retrieval routine called returns property values, the retrieval routine calls the estimation routine whose number is contained in the data record containing the data within the located data element. The property value(s) produced by the estimation routine are then returned to the calling program or subprogram.

4.2 Retrieval Routines

There are three types of retrieval routines: (1) retrieval service routines, (2) communication routines and (3) generalized retrieval routines.

The retrieval service routines retrieve stored property data. The routine named SER1 retrieves constant data (data type 1), the routine named SER2 retrieves correlation coefficients (data type 2), the routine named SER3 retrieves tabular data (data type 3) and the routine named SER4 retrieves the routine number stored within a data record.

The retrieval service routines are used primarily by estimation routines to obtain property data used to compute property values. Chapter VIII, "Property Estimation Routine", discusses the conventions for using the retrieval service routines.

The communication routines sole purpose is to call property estimation routines. Once the routine number of the estimation routine to be used to determine property values has been found a communications routine calls the estimation routine to obtain property values. Communication routines are called by the generalized retrieval routines, and by property estimation routines. An application program never has the need to call a communication routine.

The generalized retrieval routines are called by the application program to request property value(s). There are three generalized retrieval routines: (1) the routine named PPCP for retrieval of a property value for a pure chemical, (2) the routine named PPCF for retrieval of a property value for a mixture, and (3) the routine named PPCS for retrieval of a property value for each component of a mixture. Chapter VI, "The Application Program", discusses in detail conventions for using the generalized retrieval routines.

4.3 Justification of Retrieval Scheme

The retrieval scheme developed for the property system was designed with a primary concern for the user of the system. Such a concern dictated that the system be easy to use and yet completely controllable. The retrieval scheme reflects this.

The retrieval routines fall into three categories (1) routines for retrieving stored data (the retrieval service routine), (2) routines used only by the system itself (communication routines) and (3) routines allowing retrieval of stored or computed property information (generalized retrieval routines). The more sophisticated

user may choose to use the retrieval service routines, and the less sophisticated user may use the generalized retrieval routines; in either case the user has a tool that is easy to use for obtaining property information.

The retrieval scheme allows the user to specify a minimum amount of information in requesting property values, namely a compact set of retrieval keys, and yet allows the user the option of completely directing the retrieval activities of the system by specifying retrieval constraints. The tree structure directory used in the retrieval scheme is very well suited to this mode of retrieval.

CHAPTER V

THE APPLICATION PROGRAM

5.1 Definition

Within the context of the property system, an application program is a computer program that is written in the FORTRAN programming language and that requires property values for pure chemicals and mixtures during its execution. An application program is not restricted in its structure or use of labelled or unlabelled COMMON, since the property system uses no COMMON statements in system or estimation routines. The only restriction placed upon an application program is that requests for property values conform to the conventions established for using the property system. This chapter discusses the conventions for requesting property values and explains the use of the property system from the point of view of the engineer who writes application programs.

5.2 Requesting Property Values

Property values for pure chemicals and mixtures are requested in an application program through FORTRAN function and subroutine call statements that call the generalized retrieval routines. Invariant information identifying the requested property value is supplied to the property system in the form of arguments to the subprogram calls of the generalized retrieval routines. (Variant information specification is discussed in Chapter VI on "Running An

Application Program"). The generalized retrieval routines call upon property estimation routines that combine data (from the data base) to compute property values or retrieve data directly from the data base (when applicable).

There are three generalized retrieval routines that can be called by an application program. The FORTRAN function subprogram named PPCP retrieves pure substance property values requested within an application program. There are two generalized retrieval routines that retrieve mixture property values. The routine named PPCF is a FORTRAN function.subprogram that retrieves a single property value for a mixture (e.g., heat capacity). The PPCS FORTRAN subroutine retrieves a property value for each component of a mixture (e.g., equilibrium coefficients).

The remaining sections of this chapter discuss conventions for calling the generalized retrieval routines, giving examples and explaining the semantics of each request for property values. The reader is encouraged to read Chapter IV for a discussion of the retrieval scheme used by the generalized retrieval routines.

5.2.1 Pure Substance Property Value Requests

The standard request for a pure substance property value such as density or critical pressure uses the PPCP retrieval routine; for example,

PROP = PPCP (MP,V1,V2,INDEX,IC) .

The arguments of the PPCP routine are:

- MP - physical property code (for example, 401 = vapor enthalpy).
- V1 - value of first independent variable, a real number or a real variable.
- V2 - value of second independent variable. a real number or a real variable.
- INDEX - pure substance index number an integer or integer variable.
- IC - completion code variable.

Appendix I of this thesis contains a list of the established property codes. Associated with each property code are two independent variables (for example 401 is the code for vapor enthalpy with temperature the first independent variable and pressure the second). The pure substance index identifies the chemical for which the property value is requested. The pure substance index establishes that component in the component identification table for which a property value is requested. See Figure V-1 for illustration of a sample component identification table.

<u>Pure Substance Index</u>	<u>Data Base Component Code</u>	<u>Component Name</u>
1	2	Methane
2	10	n-Hexane
3	5	i-Butane
4	4	Propane
NCOMP=5	20	n-Hexadecane

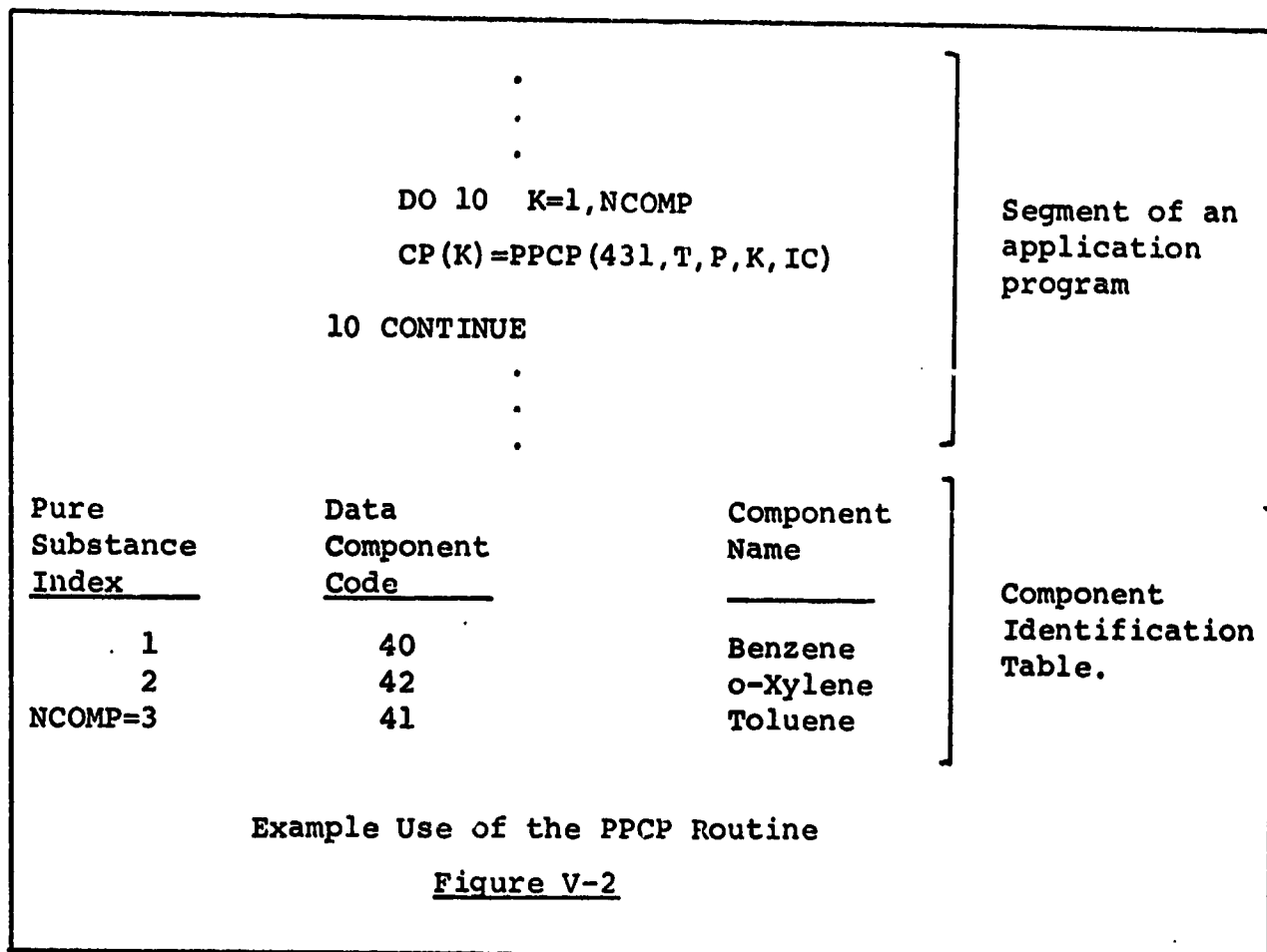
Component Identification Table

Figure V-1

174

The component identification table identifies to the property system the components for which property values are required. Appendix I contains the list of established data base component codes. The completion code variable is used to inform the application program whether all phases of property estimation have succeeded. The integer variable IC is returned with a zero value if retrieval has been successful. A non-zero value indicates that the validity of the returned property value is questionable. Figure V-2 illustrates a typical call to the PPCP routine.

The program segment illustrates retrieval of the vapor heat capacity for each of three chemicals. Notice that the component identification table is provided as data and is not a part of the application program. The identity of the chemicals is classed as variant information and is supplied to the property system outside of the application program (method to be discussed in Chapter VI). Hence, when changing the component mixture no alterations are required in the application program; only the component identification table requires change.



The information provided in the PPCP request is invariant. The property code 431 specified vapor heat capacity as a function of temperature, the first independent variable, and pressure, the second. The program variables T and P provide temperature and pressure values to the PPCP routine. The integer variable K specified the pure substance index.

In the above example the property code indicates an assumed phase, that is, the vapor phase. Property codes ending with 1 (e.g., 431 = vapor heat capacity) specify vapor phase properties while codes ending in 2 (e.g., 432 = liquid heat capacity) specify liquid phase properties, and codes ending in 3 (e.g., 433 = solid heat capacity) specify solid phase properties. When a phase specified property code (that is,

a property code ending in 1,2, or 3) is supplied to a generalized retrieval routine, the phase of the substance for which the property value is requested is taken as that specified in the property code. The property value is obtained using estimation programs and data for the specified phase even if under the conditions indicated by the two independent variables a different phase exists (assuming data is available in the independent variable ranges specified - to be amplified in Chapter VIII). For example, if the vapor heat capacity of water at 300°K and 1 atm. is requested in an application program, the generalized retrieval routine uses vapor data and estimation programs to obtain the pseudo - vapor heat capacity of water (even though the vapor phase does not exist at this temperature and pressure). The generalized retrieval routines perform no phase checking when phase specified property codes are provided in a request for a property value.

Phase unspecified property codes end in 0 (e.g., 430 = heat capacity). If the phase is unspecified, the generalized retrieval routines perform phase determination. (For the present, phase determination takes place only for properties using temperature and pressure as associated independent variables). The pure substance property retrieval routine PPCP requests the vapor pressure of the pure substance. When the phase is determined not to be vapor, the PPCP routine requests the melting point (property code 1007) to determine whether liquid or solid exists. Once the phase is determined, the PPCP routine prepares a property value using the phase specified property code.

5.2.2 Mixture Property Value Requests

There are two routines for retrieving property values for mixtures. The function named PPCF retrieves a single property value for a mixture. The standard form of a request for mixture property data using the PPCF routine is

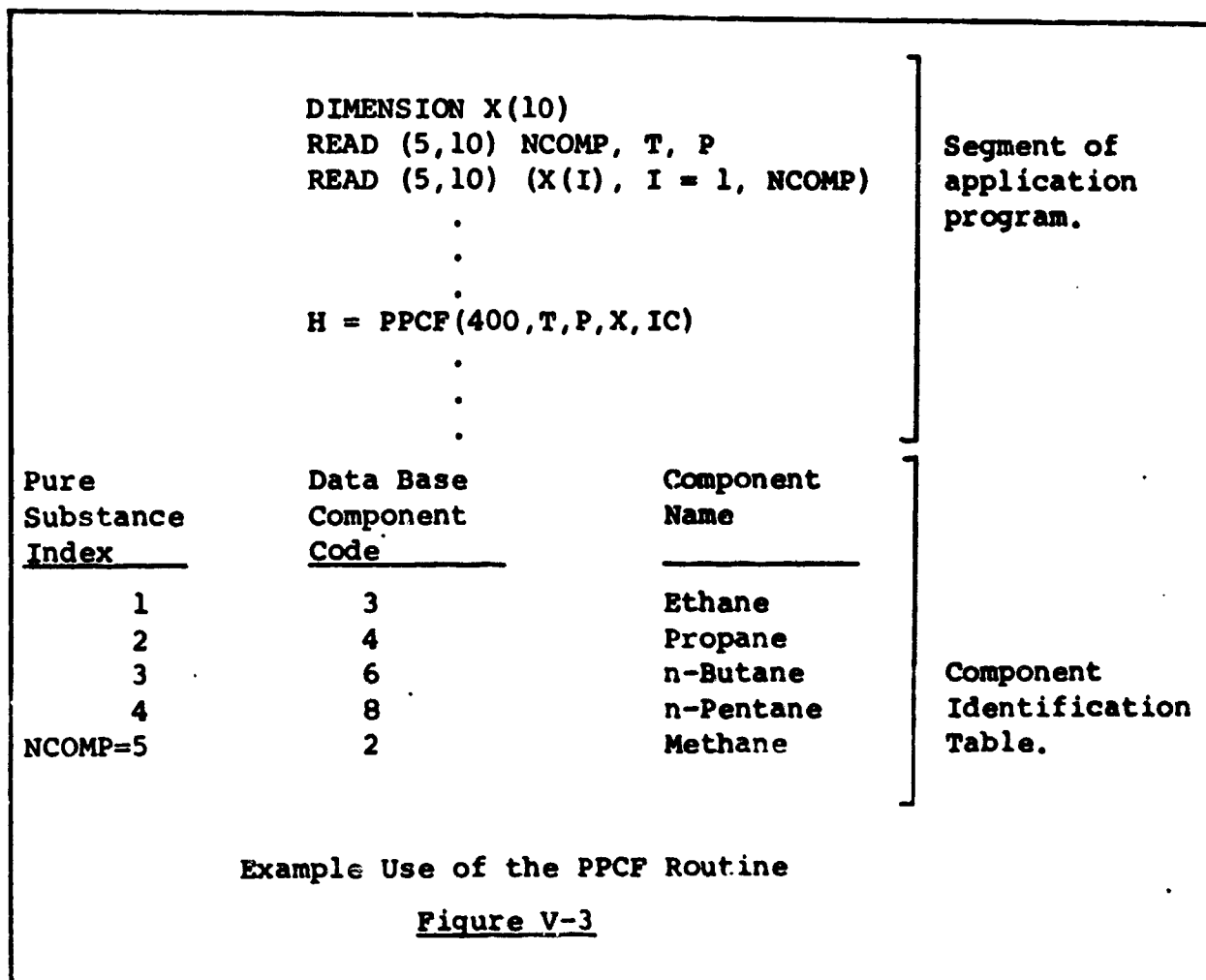
$$\text{PROP} = \text{PPCF}(\text{MP}, \text{V1}, \text{V2}, \text{X}, \text{IC})$$

The arguments of the PPCF routine are:

- MP - physical property code, an integer
- V1 - value of first independent variable associated with the property. V1 is either a real number or real variable.
- V2 - value of second independent variable associated with the property, V2 is either a real number or real variable.
- X - mole fraction array. X is a singly subscripted variable containing the mole fractions of the components listed in the component identification table in the same order as the components appear in the table.
- IC - completion code variable, an integer variable.

The PPCF routine retrieves the requested property value for the mixture whose components are given in the component identification table. Figure V-3 illustrates the use of the PPCF generalized retrieval routine. Here the PPCF routine is used to obtain the enthalpy of the five component mixture specified in the component identification table. In the example, the property code 400 of enthalpy as a function of temperature and pressure is used. The associated independent

variables, temperature and pressure are represented by the program variables T and P, and the mole fractions of the five components are contained in the array X.



Property codes that are a multiple of ten (that is, end in 0 as in the above example) do not specify the assumed phase of a mixture. Property code 401 indicates vapor enthalpy, and property code 402 indicates liquid enthalpy. If the property code provided to a generalized retrieval routine is phase unspecified, a phase determination is performed. The PPCF and PPCS retrieval routines determine the phases of a mixture by requesting the vapor fraction of the mixture (property code 518). (At present phase determination takes place only for properties with temperature and pressure as

associated independent variables. Mixture liquid or solid phases are not distinguished. All mixtures that do not exist in the vapor phase are treated as liquids. All liquids are assumed miscible). Once the vapor fraction is obtained, the phase(s) of the mixture is identified as liquid and/or vapor. When a phase unspecified property value is requested the value of the property returned is given by,

$$\text{PROP} = (\text{VAPOR FRAC.}) \times \text{VAPOR PROPERTY VALUE} + \\ (1 - \text{VAPOR FRAC.}) \times \text{LIQUID PROPERTY VALUE.}$$

Care should be exercised to avoid requesting a property value for a two-phase mixture when using phase unspecified property codes for non-state properties (e.g., compressibility or density of a mixture, either the vapor or liquid phase may exist independently, but not together. When a two-phase mixture is provided, the generalized retrieval routines print a diagnostic and abort.

The third generalized retrieval routine is the PPCS subroutine. The PPCS routine returns a property value for each component of a mixture and is used to retrieve properties such as vapor-liquid equilibrium coefficients.

The standard form of a request for property values using PPCS is

CALL PPCS (MP,V1,V2,X,RES,IC)

1011

The arguments of PPCS are:

- MP - physical property code, an integer.
- V1 - value of the first independent variable associated with the property. V1 is either a real number or real variable.
- V2 - value of second independent variable associated with the property. V2 is either a real number or real variable.
- X - mole fraction array. X is a singly subscripted variable containing the mole fraction of the components specified in the component identification table in the same order as the components appear in the table.
- RES - result array. RES is a singly subscripted variable that is returned with a property value for each component in the order the components appear in the component identification table.
- IC - return code variable.

Figure V-4 illustrates the use of the PPCS routine.

```

DIMENSION X(10)
REAL KVAL (10)
READ (5,10) T,P,NCOMP
READ (5,20) (X(I),I=1,NCOMP)
.
.
.

CALL PPCS(305,T,P,X,KVAL,IC)
IF(IC.NE.0) GO TO 999
.
.
.

```

Segment of
application
program

Pure Substance Index	Data Base Component Code	Component Name
1	4	Propane
2	6	n-Butane
3	8	n-Pentane
4	57	i-Hexane
5	12	n-Octane
NCOMP=6	11	n-Heptane

Component
Identification
Table

Example Use of the PPCS Routine

Figure V-4

Temperature, the first independent variable, is supplied by the program variable T, and pressure, the second independent variable, is supplied by the variable P. The component mole fractions are contained in the array X, and equilibrium coefficients are returned in the array KVAL. Notice that the value of the completion code variable IC is tested before the equilibrium coefficients are used.

5.3 Property System Activation

Before requests for a property value(s) can be serviced by any of the three generalized retrieval routines the property system must be activated. During activation the property system prepares itself for the first retrieval request. The property system is activated by coding

CALL ACTIVE

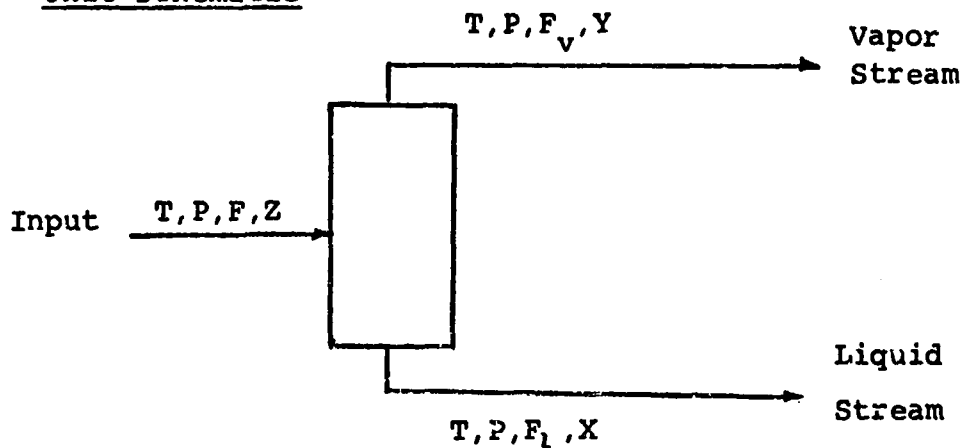
as the first executable statement in the application program.

All the conventions necessary to prepare an application program have been discussed. To further illustrate the preparation of an application program that uses the property system an example program is prepared below.

5.4 Example Application Program

The example application program determines the composition of vapor and/or liquid phases in a process stream. Schematically, the process stream of unknown phase composition may be visualized as entering into a process unit where vapor and/or liquid streams exit in equilibrium. The process unit maintains isothermal and isobaric conditions throughout. The program accepts feed stream temperature, pressure, flow rate, and mole fractions for each product stream. The physical property system is used to calculate vapor fraction, equilibrium constants (K-values), and enthalpy (of the product stream). Figure V-5 presents the application program flow chart. Figure V-6 illustrates the FORTRAN application program discussed above. The results of the example application program are discussed in Chapter VI "Running an Application Program".

Unit Schematic



Application Program Flow Chart

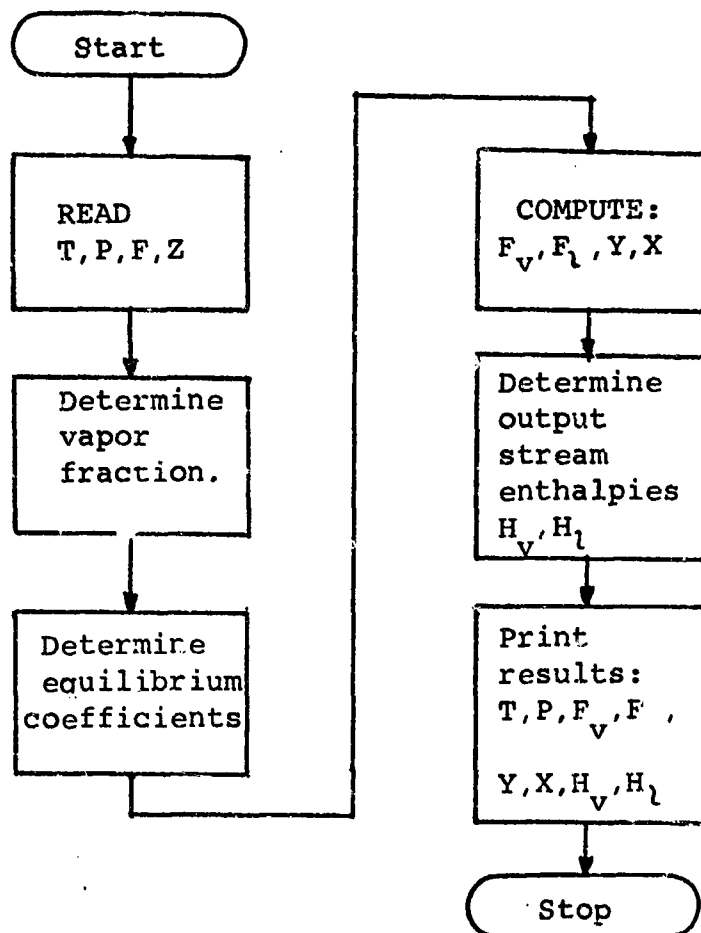


Figure V-5

```

C
C-- EXAMPLE APPLICATION PROGRAM
C
      DIMENSION X(10),Y(10),Z(10)
      REAL KVAL(10)

C-- ACTIVATE THE PROPERTY SYSTEM
C
      CALL ACTIVE

C
C-- READ INPUT STREAM INFORMATION
C
      READ(5,10) T,P,F,NCOMP
      READ(5,10) (Z(I),I=1,NCOMP)

C
C-- DETERMINE VAPOR FRACTION ( PROPERTY 518 )
C
      VFRAC=PPCF(518, T, P, Z, IC )
      IF( IC .NE. 0 ) GO TO 99

C
C-- DETERMINE K-VALUES ( PROPERTY 305 )
C
      CALL PPCS(305, T, P, Z, KVAL, IC )
      IF ( IC .NE. 0 ) GO TO 99

C
C-- COMPUTE OUTPUT STREAM MOLE FRACTIONS
C
      DO 20 K=1,NCOMP
        Y(K) = Z(K)/(VFRAC+(1.0-VFRAC)/KVAL(K))
      20 X(K) = Y(K)/KVAL(K)

C
C-- COMPUTE OUTPUT STREAM FLOW RATES
C
      FV = VFRAC * F
      FL = F - FV

C
C-- DETERMINE ENTHALPIES OF OUTPUT STREAMS
C--          (PROPERTY 401 VAPOR ENTHALPY)
C--          (PROPERTY 402 LIQUID ENTHALPY)
C
      HV = PPCF (401, T, P, Y, IC )
      HL = PPCF (402, T, P, X, IK)
      IF (IC .NE. 0 .OR. IK .NE.0 ) GO TO 999

```

Example Application Program

Figure V-6

```

C
C-- PRINT OUTPUT
C
50  WRITE(6,24)
    WRITE(6,23) T, P, F, (Z(I),I=1,NCOMP)
    WRITE(6,21) T, P, FV, HV, (Y(I),I=1,NCOMP)
    WRITE(6,22) T, P, FL, HL, (X(I),I=1,NCOMP)
    STOP

C
C-- ERROR CONDITIONS
C
99  WRITE(6,30) T, P, F, (Z(I),I=1,NCOMP)
    STOP
999 WRITE(6,31)
    HL = 0.0
    HV = 0.0
    GO TO 50

C
C-- FORMATS
C
10  FORMAT(5G15.7)
24  FORMAT(10X,' EXAMPLE PROGRAM RESULTS'/)
23  FORMAT(' INPUT STREAM :'/,' T=',G15.7,' P=',
$G15.7/' F=',G15.7/(' MOLE FRACTIONS = ',3G12.5))
21  FORMAT('/ OUTPUT VAPOR STREAM :'/,' T=',G15.7
$, ' P=',G15.7/' F=',G15.7,' H=',G15.7/
$(' MOLE FRACTIONS = ',3G12.5))
22  FORMAT('/ OUTPUT LIQUID STREAM :'/,' T=',
$G15.7,' P=',G15.7/' F=',G15.7,' H=',G15.7/
$(' MOLE FRACTIONS = ',3G12.5))
30  FORMAT(' COMPLETION CODE PROBLEM-INPUT',
$' STREAM'/,' T=',G15.7,' P=',G15.7,' F=',
$G15.7/' MOLE FRACTIONS = ',4G15.7)
31  FORMAT(' COMPLETION CODE PROBLEM-OUTPUT',
$' STREAM'/,' T=',G15.7,' P=',G15.7,' F=',
$G15.7/' MOLE FRACTIONS = ',4G15.7)
    END

```

Figure V-6 (continued)

CHAPTER VI

RUNNING AN APPLICATION PROGRAM

Chapter V explained that invariant information identifying requested property values is supplied to the property system as arguments in subprogram calls to the generalized retrieval routines from within an application program.

Variant information such as the mixture components' identities and the identities of estimation techniques and/or data used to obtain property values are supplied to the property system outside of the application program. By separating the invariant and variant information, application programs can be prepared that do not require modification when mixture component changes are made and/or when property estimation techniques must be changed. This section discusses the procedure for specifying variant information to the property system.

Variant information is supplied to the property system in the form of a deck of cards that is read by the system after it is activated and before the first request for property values is serviced. Variant information is organized into two tables: (1) the component identification table, and (2) the retrieval constraint table.

6.1 Component Identification Table

The component identification table (introduced in Chapter V) identifies to the property system the components for which property values are required. The component identification table is punched onto cards according to the following rules:

1. The first card of the component identification deck must contain the words

COMPONENT ID TABLE

The first word may begin in any column of the card.

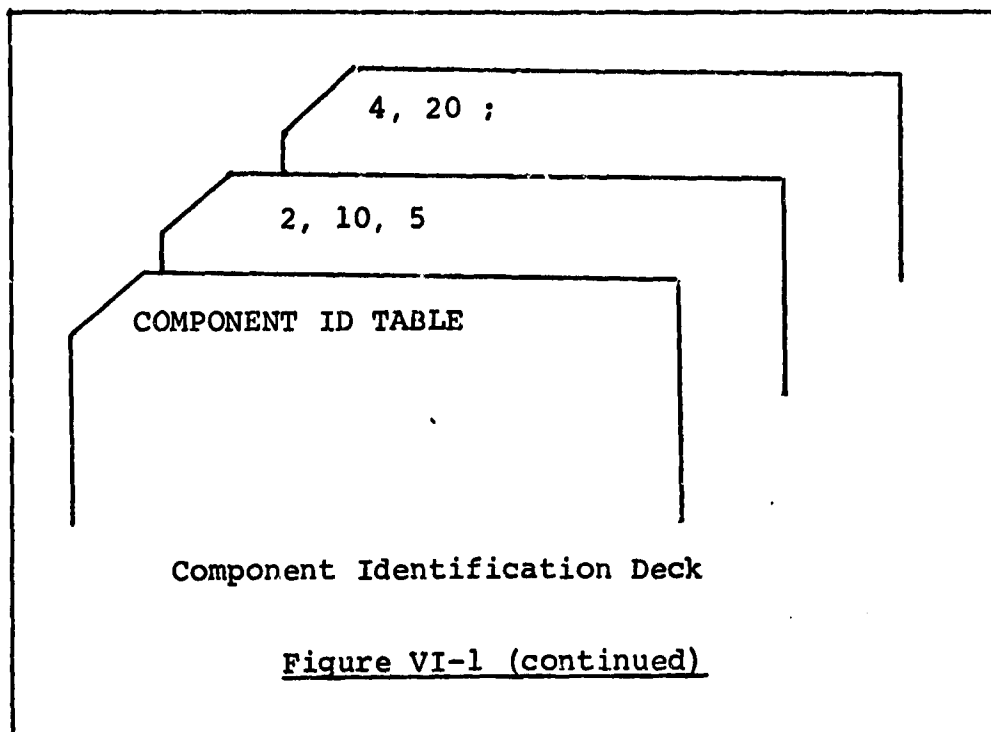
2. The second card of the deck and as many additional cards as are required must contain the data base component codes of the components for which property values are required. The data base component codes must be separated by commas or blanks and can be entered in any column of the cards.
3. The last data base component code must be followed by a semicolon.

Figure VI-1 illustrates a component identification table and the component identification deck corresponding to it.

Pure Substance Index	Data Base Component Code	Component Name	Component Identification Table
1	2	Methane	
2	10	n-Hexane	
3	5	i-Butane	
4	4	Propane	
NCOMP=5	20	n-Hexadecane	

Component Identification Table

Figure VI-1



Notice that the component identification deck is punched in free format and that the list of data base component codes can be continued from one card to another. A component identification table must be supplied to the property system on cards each time an application program is run.

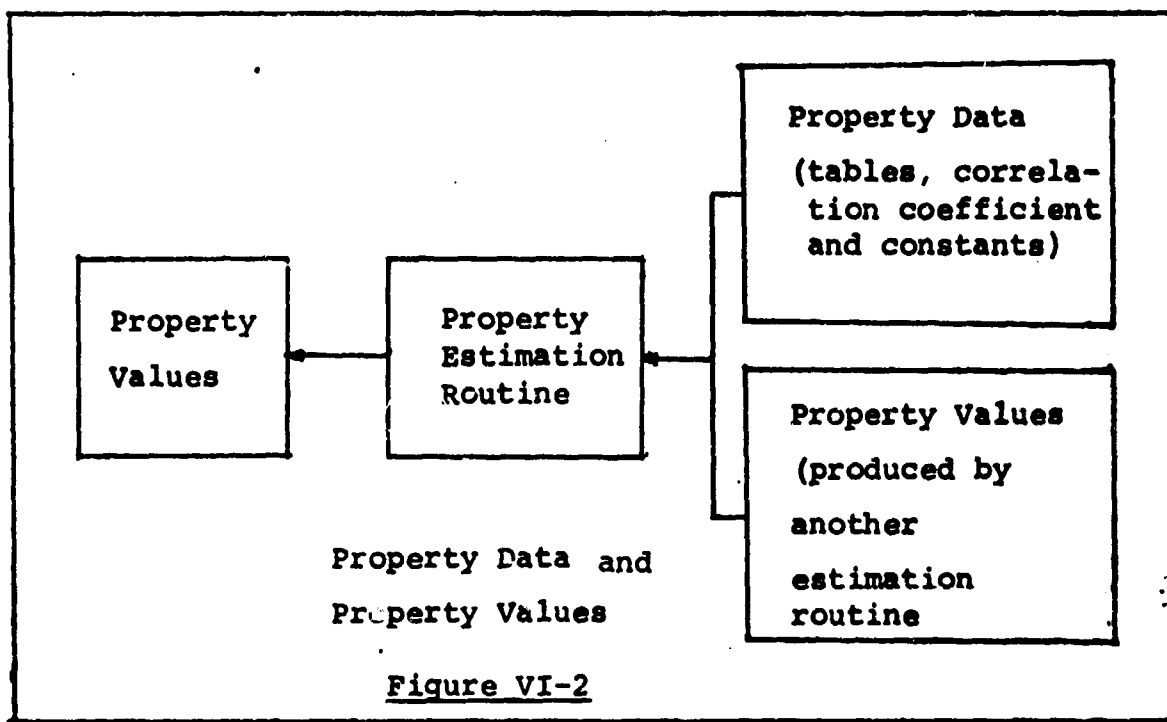
6.2 Retrieval Constraint Table

The retrieval constraint table contains variant information used by the generalized retrieval routines to select estimation procedures for requested property values. Before discussing the preparation of a retrieval constraint table some required background information is presented.

In the property system a distinction is made between property values and property data. Property values are determined using a property estimation procedure. Property data is raw data in the form of tables, correlation coefficients and constants stored in the system's data base. A property estimation routine uses property data stored in the data base and/or property values produced by other estimation routines to produce property values. For example, a correlation routine (property estimation routine) uses correlation coefficients (property data) to determine estimated property values. Frequently, property estimation routines use a combination of property values determined by other estimation routines as well as property data. For example, enthalpy is often determined by evaluating separately zero-pressure enthalpy and a pressure-correction term. The estimation procedure for enthalpy then requires use of two property values determined by other estimation procedures.

Only in the case of "constant" data are property values and property data equivalent. No estimation procedure is required to transform a constant, for example, molecular weight, to a property value.

Figure VI-2 depicts graphically the relation between property data, property values, and property estimation routines.



The generalized retrieval routines accept a request for a property value or values, search the data base for property data required to produce the requested property values, select the proper estimation routine and call on the estimation routine to produce the requested property values.

Property data is stored in the data base in the form of data records. Data records contain property data together with information specifying its characteristics. A data record contains:

1. a physical property code (for example 401 = vapor enthalpy as a function of temperature and pressure),
2. the ranges of the independent variables associated with the property for which the data is valid (for example, 202°K - 400°K for

- temperature and 1 atm - 3 atm for pressure),
3. a code specifying the contributor who entered the data record into the data base (for example, 427 = John Jones),
 4. the data base component code(s) of the pure chemical or mixture for which the data is applicable (for example 2 = Methane),
 5. the estimation routine number that produces property values from the data contained in the data record (for example 20 = third degree polynomial routine)
 6. the maximum percentage error expected in the property value produced by the estimation routine (for example, 3%),
 7. the data type code that indicates the form of the property data contained in the data record (for example, 2 = correlation coefficient),
 8. the property data itself (for example, the third degree polynomial coefficients, a,b,c,d).

A data record is either a pure chemical data record or a mixture data record. Pure chemical data records contain one data base component code and property data applicable to the chemical identified by that code. Mixture data records contain several data base component codes and property data applicable to the mixture whose components are identified by the data base component codes.

Chapter V explained that the invariant information supplied as arguments to the generalized retrieval routines serves to request property values from within an application program. The generalized retrieval routines search the data base for valid data records. A valid data record:

1. contains the property code specified as invariant information.
2. contains independent variable ranges into which the supplied independent variable values fall.
3. contain data base component code(s) specified in the component identification table.

The generalized retrieval routines select a valid data record, and call the estimation routine whose routine number is contained in the selected data record. The estimation routine obtains the necessary property data from the data base and/or property values produced by other estimation routines and produces the requested property value. The generalized retrieval routines return the property value to the application program.

Often the generalized retrieval routines locate several valid data records. The valid data records may differ in the estimation routine used to calculate a property value, the maximum percentage error expected between the true value(s) and the estimated property value(s), and/or the identity (code) of the contributor who entered the data record into the data base. The generalized retrieval routines select one of the valid data records. The retrieval constraint table provides variant information used to direct the selection of a valid data record. If no retrieval constraint is supplied, the first valid data record encountered in a search of the data base is selected.

In the retrieval constraint table the user can specify for any property:

1. the estimation procedure to be used (Appendix I contains the current list of estimation routines).

2. the percentage error within which the difference between true and estimated physical property values must fall,
3. the identity of the contributor of the property data to be used, and
4. the degree of the constraint (to be defined below).

The retrieval constraint table provides the basis for selection of one valid data record from among several competing records. If none of the valid data records meet the specified constraints and the constraints are absolute, a message is printed and the system aborts. In the special case when the constraints are not absolute, a message is printed and the first valid data record is selected (even though the constraints are not satisfied). (Appendix II contains the generalized retrieval routine logic diagrams that illustrate the role of the component identification table and the retrieval constraint table.).

Figure VI-3 illustrates a retrieval constraint table.

Property Code	Allowable Error	Estimation Routine Number	Contributor Code	Constraint Degree
518	-	27	-	*
305	1%	-	-	-
401	-	-	342	*
432	-	C	343	-
-	2%	-	-	-

Retrieval Constraint Table

Figure VI-3

The example retrieval constraint table has the following interpretation:

1. If vapor fraction (property code 518) is requested, estimation routine number 27 is to be used. The asterisk, *, in the constraint degree column indicates that this is not an absolute constraint; therefore, if routine number 27 cannot be located in a valid data record, a message will be printed and the search will continue until a valid data record is located.
2. If equilibrium coefficients (property code 305) are requested, any means for obtaining the data is permitted, so long as the estimated values differ from the true values by less than 1%. The dash in the constraint degree column indicates that this is an absolute constraint. If an equilibrium coefficient data record with 1% or less error cannot be located, a message is printed and the system aborts.
3. If vapor enthalpy (property code 401) is requested, data supplied by contributor 342 should be used if available. This constraint is not absolute (*), therefore, data supplied by other contributors may be used if none is available from contributor 342.
4. If liquid heat capacity (property code 432) is requested, constant data stored by contributor 342 must be used. ("C" in the estimation routine number column indicates that a stored constant is required). This is an absolute constraint.

5. For all property values returned (other than those mentioned in the retrieval constraint table), true values must not differ from estimated property values by more than 2%. This is an absolute constraint; therefore, if this constraint cannot be met, a message will be printed and the system will abort).

In preparing a retrieval constraint table any combination of the five entries (property code, allowable error, estimation routine number, contributor code, and constraint degree) can be used. A property code may appear more than once in the retrieval constraint table. Consider the retrieval constraint table illustrated in Figure VI-4

Property Code	Allowable Error	Estimation Routine Number	Contributor Code	Constraint Degree
305	-	26	343	*
305	-	29	344	*

Example Retrieval Constraint Table
Figure VI-4

The retrieval constraint table in Figure VI-4 is interpreted as follows: if equilibrium coefficients (property code 305) are requested, estimation routine number 26 or 29 is to be used together with data contributed by contributors 343 or 344, respectively (assuming that valid data records can be located). The constraints are not absolute.

Retrieval Constraint Deck

The retrieval constraint table is punched onto cards according to the following conventions:

1. The first card of the retrieval constraint table must contain the words

RETRIEVAL CONSTRAINTS

anywhere on the card (free format).
2. Each row of the retrieval constraint table is punched on a separate card.
3. Each of the five entries on a card (property code, allowable error, estimation routine number, contributor code and constraint degree), is separated by a comma or a blank. A missing entry is replaced by a dash (minus sign).
4. The last entry on the last card is followed by a semicolon (;).

Figure VI-5 illustrates the retrieval constraint deck for the table of Figure VI-3

card column 1					
card 1:	RETRIEVAL CONSTRAINTS				
card 2:	518,	- ,	27,	- ,	*
card 3:	305,	1%,	- ,	- ,	-
card 4:	401,	- ,	- ,	342,	*
card 5:	432,	- ,	C ,	343,	-
card 6:	-	2%,	- ,	- ,	-;

Retrieval Constraint Deck

Figure VI-5

The retrieval constraint table may or may not be used by the engineer when using the property system. It is an optional feature provided for convenience in specification of constraints by the engineer.

The component identification deck and the retrieval constraint table deck (when used) are combined into one deck of cards, hereafter referred to as "the variant information deck". The conventions for making up a variant information deck are:

1. The first card of the deck contains the words

BEGIN VARIANT INFO

anywhere on the card (free format),

2. The component identification table deck follows the first card,
3. The retrieval constraint deck (when used) follows the component identification table deck,
4. The last card of the variant information deck contains the word

END

anywhere on the card (free format).

Figure VI-6 illustrates a variant information deck.

card column 1

card . 1: BEGIN VARIANT INFO

card 2: COMPONENT ID TABLE

card 3: 2, 10, 5,

card 4: 4, 20;

card 5: RETRIEVAL CONSTRAINTS

card 6: 518, -, 27, -, *

card 7: 305, 1%, -, -,

card 8: 401, -, -, 342, *

card 9: 432, -, C, 343, -

card 10: -, 2%, -, -, -;

card 11: END

Variant Information Deck

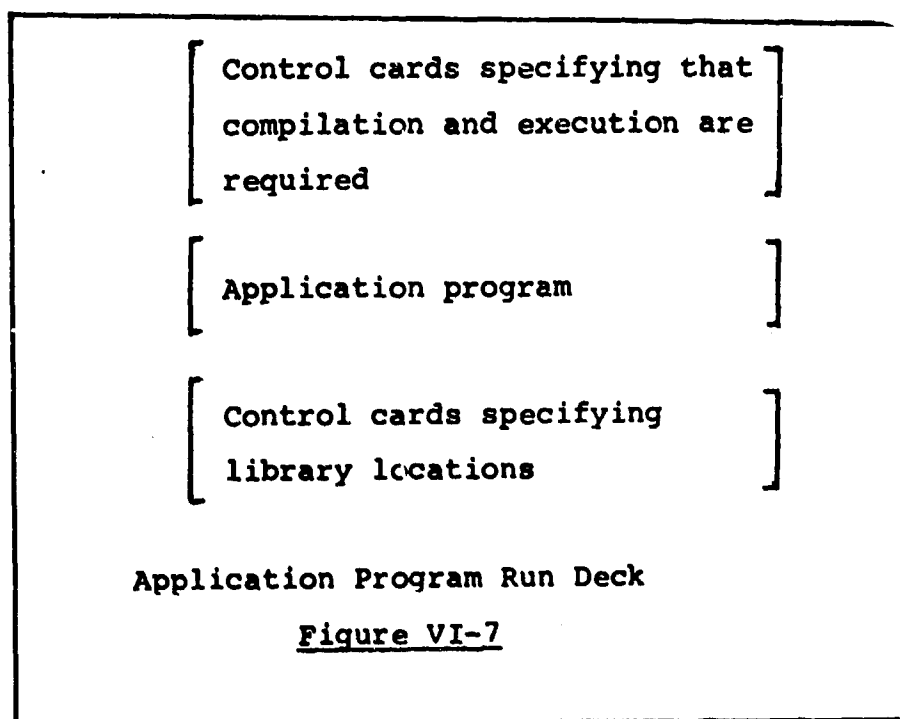
Figure VI-6

6.3 Computer Operating System Control Specifications

The preparation of an application program and a variant information deck is independent of the particular computing system used to compile and execute the application program. However, the procedure to be followed for submitting an application program for execution differs with each computing system. It is therefore not possible to specify in this paper the exact procedure to be used to execute an application program on all computers. Without regard for the computing system used the following information must be supplied (the format for IBM 360/75 at the University of Pennsylvania follows in the next section):

1. Control cards specifying that the application program is to be compiled and executed.
2. Control cards specifying the location (physical storage location) of the library of storage, retrieval and property estimation routines.
3. Control cards specifying the location of the data base (stored on some external device) to be read by the property system using FORTRAN input/output unit number 3.
4. Control cards specifying that the variant information deck is read by the property system using FORTRAN input/output unit number 5.
5. Control cards specifying that the property system writes diagnostic messages on FORTRAN input/output units 6 and 11.

Figure VI-7 illustrates the structure of a typical application program together with data and control cards.



[Control cards specifying
location of data base and
other FORTRAN input/output units]

[Variant information deck]

[Application program data]

Application Program Run Deck

Figure VI-7 (Continued)

6.4 University of Pennsylvania IBM 360/75 Control Cards

Appendix III contains a list of the control cards required for running an application program on the IBM System 360/75 at the University of Pennsylvania Computing Center.

6.5 Example Application Program Results

Figure VI-8 illustrates the printed output of the example application program discussed in Chapter V. A component identification table containing the data base component codes of propane, i-butane, i-pentane, n-pentane, n-butane, and n-hexane was provided as variant information.

```
***** N E W *****
MIXTURE COMPONENT LIST

MIX COMP # | DATA BASE #
      1    |         4
      2    |         5
      3    |         6
      4    |         7
      5    |         8
      6    |        10
END          0

DATA BASE SUCCESSFULLY INITIALIZED

EXAMPLE PROGRAM RESULTS

INPUT STREAM :

T= 380.0000      P= 11.90000
F= 450.5999
MOLE FRACTIONS = 0.32000      0.15000      0.80000E-01
MOLE FRACTIONS = 0.50000E-01 0.10000E 00 0.30000

OUTPUT VAPOR STREAM :

T= 380.0000      P= 11.90000
F= 260.5024      H= 20946.00
MOLE FRACTIONS = 0.43937      0.17904      0.90167E-01
MOLE FRACTIONS = 0.44350E-01 0.83229E-01 0.16369

OUTPUT LIQUID STREAM :

T= 380.0000      P= 11.90000
F= 190.0974      H= 15847.09
MOLE FRACTIONS = 0.15642      0.11021      0.66067E-01
MOLE FRACTIONS = 0.57743E-01 0.12298      0.48679

Figure VI-6
```

CHAPTER VII

DATA STORAGE

A data base containing physical property data records is an integral part of the property system. A master data base containing approved property data is maintained by the property system librarian, and is available to all users.

Often a user wishes to add personal property data to the data base to be used with one of the property estimation routines. To insure the integrity of the master data base only the system librarian may enter new data into it. Personal property data is entered into a user's personal data base using the property system's storage routine.

When an application program requests property values, the generalized retrieval routines search the data base for a valid data record. The user's personal property data is searched first to locate a valid data record. If none is found, the data records contained in the master data base are searched. In this manner preference is given to the user's personal property data.

Through the use of the property system's storage routine the user can:

1. add new personal property data to the user's data base,
2. update property data existing in the user's data base, or
3. delete property data existing in the user's data base.

To insure that a user's personal data base can be accessed by only the user himself, the user selects a password upon entering property data into his personal data base for the first time. The selected password is stored by the system and must be supplied each time data is added, updated or deleted from the user's personal data base.

The engineer generally will have need to add personal property data when working with new chemicals or when obtaining new data from the literature and/or by experiment. The new data can be added to the data base at any time, but cannot be used unless the appropriate property estimation routine is available. Appendix I contains a description of property estimation routines currently stored in the property system library. Often it is necessary to define new property estimation routines. Chapter VIII of this paper describes the procedures for writing new estimation programs and storing them in the property system library.

The procedures for using the property system's data storage routine are discussed below.

7.1 Adding New Personal Property Data

The data storage routine is used to add new data records to a user's personal data base. The data records to be added are punched onto cards to be read by the data storage routine. The deck of data records in card form is hereafter called the addition deck.

Adding new property data to the data base involves entering new directory elements to the directory where necessary together with adding a new data element to the data pool. The storage routine first searches the property code nodal list to determine if the property

204

code of the new data record exists. If it does not, the new directory element containing the property code is added to the list and new lower level nodal lists are created. If the code is found, the branch to the associated variable range nodal list is followed. The storage routine searches that list for the variable ranges of the new data record. If a matching directory element is not found, one is created together with new lower level nodal lists. If a match is found, the branch to the associated component-data type list is searched for a matching entry. If one is not found, a new directory element containing the component code(s) and data type is added to the list and new lower level nodal list are created. If a matching directory element in the component-data type nodal list is found, the branch to the associated contributor-routine number nodal list is followed. If a match is not found in this lowest level nodal list, a new directory element is added to the list together with a pointer to an empty data element of the data pool and the data contained in the new data record is then added to the empty data element. If a match is found in the lowest level nodal list, an error condition exists. All of the attribute values of the new data record have been found in the directory, therefore, data already exists. In such a case new data cannot be added to the data base; the existing data must instead be updated. The storage routine prints a message and goes on to the next data record.

To simplify the preparation of an addition deck the data records are punched in a special way. Each data record is separated into two parts; (1) the characteristic part, and (2) the data part. Figure VIII-1 illustrates the two parts of a data record.

Data Record	1. Property code	-	425	Characteristic Part
	2. Contributor code	-	317	
	3. Validity ranges	-	200°K-400°K (no second range)	
	4. Max. expected % error	-	2%	
	5. Estimation routine no.	-	24	
	6. Data type code	-	2	
	7. Data base component code	-	3	Data Part
	8. Data	-	4.0, 5.2x10 ⁻¹ , 6.7x10 ⁻²	

Data Record
Figure VII-1

Data records to be added that have identical characteristic parts are grouped together. For each such group it is necessary to punch the common characteristic part of the data records only once. Following the common characteristic part of the data records are one or more data parts. Figure VII-2 illustrates the punched form of data records.

```

group 1    (Common Characteristic Part
            '(Data part1) (Data part2) ... (Data partn)'

group 2    (Common Characteristic Part)
            '(Data part1 (Data part2) ... (Data partm)'
            .
            .
            .
            Punched Form of Data Records
            Figure VII-2

```

206

Notice that the common characteristic part of the data records and each data part are preceded by a left parenthesis, "(", and followed by a right parenthesis, ")" and that each set of data parts is preceded and followed by an apostrophe.

Characteristic Part

The rules for punching the characteristic part of a data record onto cards are:

1. The first character must be "(" and can begin in any column of the card,
2. The entries in the characteristic part appear in order:
 - a) property code,
 - b) contributor code,
 - c) 1st Ind. variable validity range,
 - d) 2nd Ind. variable, validity range,
 - e) maximum percentage error,
 - f) estimation routine number,
 - g) data type code,and are punched in free format. Separated by commas,
3. Each validity range is punched as
lower value - upper value
4. Any missing entry is replaced by a dash (minus sign),
5. The last character must be ")",
6. The characteristic part of a data record may be continued from one card to another.

The characteristic part of the data record in Figure VII-1 would be punched as,

{425,317,200.-400.,-,2%,24,2) .

207

The data part of a data record is punched in free format in the form.

(component(s) : data)

In pure chemical data records only one data base component code appears, but in mixture data records several data base component codes appear, each separated from the others by a comma. The component codes are followed by data entries separated by a colon. Data may appear following the colon in the following forms: (1) a constant data value, (2) a set of correlation coefficients, (3) a table, or no data at all when the estimation procedure does not require property data.

Data Part

The rules for punching the data part of a data record are:

1. The first character is a left parenthesis "(",
2. The data base component code(s) follow "(", each separated by commas if more than one appears,
3. A colon (:) follows the data base component codes,
4. The data follows the colon.
 - a) If the data is a constant, the single value is entered in "E" or "F" format. (For example 27.5 or 2.75E+1)
 - b) If the data is in the form of a set of correlation coefficients, the coefficients are entered, in "E" or "F" format separated by commas. (For example, 5.10,6.1E2,7.5E-3)
 - c) If the data is in the form of a table, the values of the first independent variable are entered first, the values of the second independent variable next, and then the body of the table is entered row by row. (each row corresponds to a single value of the first independent variable). For example the table,

208

	x_1	x_2	x_3
y_1	a_{11}	a_{12}	a_{13}
y_2	a_{21}	a_{22}	a_{23}
y_3	a_{31}	a_{32}	a_{33}

(with y_1, y_2, y_3 values of the first independent variable, and x_1, x_2, x_3 values of the second) is entered as,

$(y_1, y_2, y_3), (x_1, x_2, x_3), (a_{11}, a_{12}, a_{13}), (a_{21}, a_{22}, a_{23}), (a_{31}, a_{32}, a_{33})$.

If the table has only one independent variable the missing independent variable entries are replaced by a dash (-). For example, the table,

y_1	a_1
y_2	a_2
y_3	a_3

(with y_1, y_2, y_3 values of the first independent variable) is entered,

$(y_1, y_2, y_3), (-), (a_1), (a_2), (a_3)$.

d) If no data appears, a dash (-) is entered.

The data part of the data record illustrated in Figure VII-1 would be punched as,

(3 : 4.0,5.2E-1,.067) .

When punching numbers in the characteristic part or data part of data records any FORTRAN number representation format may be used (integers have no decimal point and real numbers are represented in F or E format).

Figure VII-3 illustrates four data records and the corresponding card images for addition of the data records into a user's personal data base.

	Data Record 1	Data Record 2	Data Record 3	Data Record 4
Property :	425	424	425	431
Contributor:	153	153	153	153
Validity (1): range	300-400°K	300-400°K	300-400°K	1800-3600°K
(2) :	-	-	-	0.5 --2.0 atm
Max. error :	2%	2%	2%	5%
Estimation : Routine	10	10	10	21
Data Type :	2	2	2	3
Component :	2	3	6	47
Data :	4.75	5.99	5.76	
	1.2×10^2	3.0×10^2	3.5×10^2	1800. 0.656
	5.0×10^{-1}	4.77×10^{-1}	3.5×10^{-1}	2222. 0.701
				2777. 0.743
				3333. 0.771
				3600. 0.782

Example Data Records
Figure VII-3

Punched data records:

card column 1
↓
card 1 : (425,153,300-400,-,2%,10,2)
card 2 : '(2:4.75,1.2E2,0.5)(3:5.99,300.,0.477)
card 3 : (6:5.76,3.5E2,3.5E-1)'
card 4 : (431,153,1800-3600.,0.5-2.0,5%,21,3)
card 5 : '(47: (1800.,2222.,2777.,3333.,3600.),(-),
card 6 : (0.656),(0.701),(0.743),(0.771),(0.782))'

Figure VII-3 (Continued)

Notice that data records 1,2, and 3 have the characteristic part in common; therefore the three data records are grouped together. Card 1 contains the characteristic part, and card 2 and 3 contain the data parts. The fourth data record is punched onto cards 5 and 6. Notice that the dash (-) indicates that the table is represented only as a function of the first independent variable (temperature). This data record will be used only when the second independent variable (pressure) falls within 0.5 and 2.0 atm. The table represents data that is independent of pressure in the validity range specified.

Additions Deck

An additions deck is made up of punched cards containing data records to be added to a personal data base. The control card containing the words,

ADDITION DECK ,

211

in free format is the first card of an additions deck. The punched data records follow the control card.

7.2 Updating Existing Data

Property data contained in data records stored in a personal data base can be altered through the use of the storage routine. Updating existing data is performed similarly to adding new data. The updated version of a record is punched onto cards as previously discussed. The storage routine then locates an existing data record stored in the personal data base that is identical to the data record punched on cards except in the data itself. The storage routine replaces the data in the existing data record with the data in the new data record. The storage routine will update only the data portion of a data record; the other items (property code, contributor code, etc.) cannot be updated.

An update deck is made up of punched cards containing the updated records. A control card containing the words,

UPDATE DECK ,

in free format is the first card of an update deck. The punched data records follow the control card.

7.3 Deleting Existing Data

Data records that exist in a personal data base can be deleted by using the property system's storage routine. To delete an existing data record, the entire data record excluding its data part is punched onto cards. The storage routine searches the users personal data base for an exact

duplicate of the data record to be deleted. When the data record is located in the users personal data base it is deleted. The punched version of the data records to be deleted together with a control card make up a deletion deck.

If data record 1 from Figure VII-3 was to be deleted from a personal data base, the punched version of the data record (without its data) would take the form,

(425,153,300,-400.,-,2%,10,2,(2)) .

Notice that the above consists of the characteristic part of the data record together with the data base component code(s).

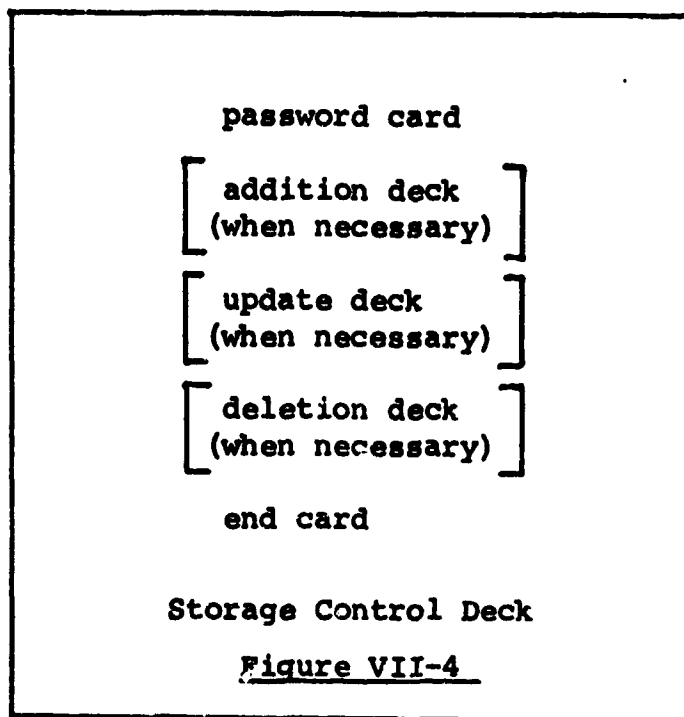
The control card containing the words,

DELETION DECK ,

in free format is the first card of the deletion deck. The punched data records (without their data) to be deleted follow the control card.

7.4 Storage Control Deck

The addition deck, update deck, and deletion deck are assembled into a single deck of cards that direct the data storage routine to perform its function. Figure VII-4 illustrates the storage control deck.



The first card of a storage control deck is the password card. The password card has two forms. One form is used only the first time personal property data is added to a personal data base. The form of the card is,

NEW PASSWORD = user provided password .

Any password may be chosen by the user. The password is stored by the property system and must be supplied by the user each subsequent time that modifications to the user's personal data base are requested. The second form of the password card is

OLD PASSWORD = user provided password .

The last card of the storage control deck is the end card. The end card contains the words,

END STORAGE DECK ,

in any column of the card (free format).

214

Appendix III contains the control cards required to execute the property system's storage routine on the University of Pennsylvania's IBM 360/75 computing system.

Appendix IV contains the Backus Naur Form specification of the storage control deck.

CHAPTER VIII

PROPERTY ESTIMATION ROUTINES

8.1 Definition

A property estimation routine is a FORTRAN function or subroutine subprogram that computes a property value or property values using property data stored in the data base and/or property values generated by other estimation routines. The advanced user frequently has recourse to add new property estimation routines to be used with new property data in the estimation of physical property values. New property estimation routines are prepared in the form of FORTRAN subprograms and entered into the estimation routine library. The new property estimation subprogram is available for use with new and/or updated data records that refer to its routine number.

Property estimation subprograms become a part of the property system when they are entered into the system's library, therefore, their preparation is regulated by a set of conventions much more restricting than those governing the preparation of an application program. The remainder of this chapter discusses the conventions for preparing estimation routines and for entering the routines into the system's estimation routine library.

8.2 Estimation Routine Types

There are three types of estimation routines: (1) pure chemical property estimation routines that compute a single property value for a pure chemical, (2) mixture property estimation routines that compute a single property value for a mixture, hereafter called single-valued mixture property estimation routines, and (3) mixture property estimation routines that estimate a single property value for each component of a mixture, hereafter called multiple-valued mixture property estimation routines. Notice that the three types of estimation routines correspond to the three types of generalized retrieval routines, PPCP, PPCF, and PPCS.

8.3 Estimation Routine Conventions

The estimation routines are an integral part of the property system and must conform to conventions concerning their structure. That is, each estimation routine must follow the conventions outlined below:

1. An estimation routine is a FORTRAN subprogram. Each type of estimation routine has a standard form for its FUNCTION statement or SUBROUTINE statement.

Pure Chemical Property Estimation Routines are FUNCTION subprograms having the following standard FUNCTION statement,

REAL FUNCTION NAME (MP,V1,V2,INDEX,IC)

where the arguments are:

NAME - the name of the estimation routine
MP - physical property code, an integer variable

V1 - first independent variable, a real variable

V2 - second independent variable, a real variable

INDEX - pure substance index, an integer variable

IC - completion code, integer variable

Single-Valued Mixture Property Estimation

Routines are FUNCTION subprograms having the following standard FUNCTION statement,

REAL FUNCTION NAME (MP,V1,V2,NCOMP,IC)

where the arguments are:

NAME - the name of the estimation routine

MP - physical property code, an integer variable

V1 - first independent variable, a real variable

V2 - second independent variable, a real variable

X - mole fraction array, a real singly subscripted variable

NCOMP - number of mixture components, an integer variable

IC - completion code, an integer variable

Multiple-Valued Mixture Property Estimation

Routines are SUBROUTINE subprograms having the following standard SUBROUTINE statement,

SUBROUTINE NAME (MP,V1,V2,X,RES,NCOMP,IC)

where the arguments are:

NAME - the name of the estimation routine
 MP - physical property code, an integer variable
 V1 - first independent variable, a real variable
 V2 - second independent variable, a real variable
 X - mole fraction array, a real singly subscripted variable
 RES - result array, a real singly subscripted variable
 NCOMP - number of mixture components, an integer variable
 IC - completion code, an integer variable

Each argument of the FUNCTION and SUBROUTINE statements has been previously discussed in Chapter V, with the exception of the variable representing the number of mixture components (NCOMP). The number of mixture components is supplied to the estimation routines by the generalized retrieval routines via the variable NCOMP.

2. An estimation routine must not alter the value of any of its arguments with the exception of the result array (RES) and the completion code variable (IC).
3. Each estimation routine must initialize the completion code variable to zero. If the estimation routine performs check procedures during computation and determines that an abnormal condition exists, adversely affecting the property value estimated, the completion code variable is assigned a non-zero

value. Each abnormal condition that can arise in an estimation routine is assigned a unique completion code.

4. An estimation routine contains no WRITE, PRINT, READ, or STOP statements.
5. An estimation routine may not use blank or labelled COMMON.

8.4 Requesting Property Values and Data

An estimation routine may require property values produced by other estimation routines and property data in the form of constants, correlation coefficients, or tables contained in data records stored in the data base.

8.4.1 Requesting Property Values

In preparing property estimation routines, requests for property values are coded as FORTRAN function and subroutine call statements that call the generalized retrieval routines. The PPCP generalized retrieval routine is used to request pure chemical property values, the PPCF generalized retrieval routine is used to request a single mixture property value, and the PPCS generalized retrieval routine is used to request property values for each component of a mixture. Due to the lack of the recursive property in FORTRAN subprograms, (discussed in section 8.6) the calls to generalized retrieval routines coded at the time of preparation of the estimation routine are changed to calls to one of the corresponding communication routines. The modification of calls to the generalized retrieval routines takes place prior to entering the estimation routine into the estimation routine library, and is discussed in the last section of

this chapter. For clarity in the discussion of the preparation of estimation routines, requests for property values of pure chemicals and mixtures are discussed as calls to the generalized retrieval routines. The conventions for calling the generalized retrieval routines are discussed in Chapter V, "The Application Program".

8.4.2 Requesting Property Data

Estimation routines often require property data in the form of constants, correlation coefficients, and/or tables. For example, an estimation routine that evaluates a third degree polynomial requires data in the form of four correlation coefficients associated with the polynomial. Property data for chemicals are requested by and supplied to estimation routines using the property system's retrieval service routines. Within the estimation routines, invariant information identifying the requested property data is supplied to the retrieval service routines in the form of arguments (similar to the generalized retrieval routine arguments).

There are three retrieval service routines that can be called by an estimation routine. The FORTRAN function subprogram SER1 retrieves stored constants, the FORTRAN function subprogram SER2 retrieves a stored correlation coefficient, and the FORTRAN subroutine subprogram SER3 retrieves a stored table.

Retrieval Service Routine SER1

The retrieval service routine SER1 retrieves constant property data (data type 1) contained in a data record that is stored in the data base. The standard request for constant property data such as critical temperature or acentric

factor using the SER1 routine is,

CONST = SER1 (MP,V1,V2,INDEX).

The arguments of the SER1 routine are:

- MP - physical property code, an integer or integer variable
- V1 - value of first independent variable associated with the property, a real number or real variable
- V2 - value of second independent variable associated with the property, a real number or real variable
- INDEX - pure substance index number, an integer or integer variable

Appendix I of this thesis contains lists of the established property codes. Associated with each property code are as many as two independent variables. If the property code does not have two independent variables associated with it (for example 1006 = molecular weight has no associated independent variables) a constant (preferably zero) must be entered in place of each missing independent variable. The pure substance index identifies the chemical for which the property data is requested. The pure substance index is given a value of zero, if the requested property data is for a mixture rather than for a pure chemical. Figure VIII-1 illustrates the use of the SER1 retrieval service routine. In Figure VIII-1, the SER1 routine is used to obtain the critical temperature and pressure for the components of a mixture from within a multiple-valued mixture property estimation routine. Since the SER1 routine is a FORTRAN

function subprogram, a call to SER1 can be incorporated into an arithmetic assignment statement as illustrated in the example.

```
SUBROUTINE FUGL (MP,T,P,X,RES,NCOMP,IC)
  DIMENSION X(25),RES(25)
  IC=0
  .
  .
  .
  DO 20 K=1,NCOMP
    TR=r/SER1(1001,T,P,K)
    PR=P/SER1(1002,T,P,K)
    .
    .
    .
  20 CONTINUE
  RETURN
  END
```

Example Use of the SER1 Routine

Figure VIII-1

The retrieval service routines do not perform phase checking, therefore, only phase specified physical property codes are allowed as arguments to the retrieval service routines.

Retrieval Service Routine SER2

The second retrieval service routine is SER2. The SER2 routine returns one of the coefficients in a set of correlation coefficients contained in a data record. The standard request for a correlation coefficient (data type 2) using the SER2 routine is,

COEFF = SER2 (MP,V1,V2,INDEX,ISUB).

The arguments of SER2 are:

- MP - physical property code, an integer or integer variable
- V1 - value of first independent variable associated with the property, a real number or real variable
- V2 - value of second independent variable associated with the property, a real number or real variable
- INDEX - pure substance index number, an integer or integer variable
- ISUB - subscript of the correlation coefficient, an integer or integer variable

The arguments to SER2 are identical to those of SER1 with the exception of an additional argument ISUB. The argument ISUB indicates the particular correlation coefficient to be retrieved from a data record by SER2. If there are N correlation coefficients, then ISUB can take on the values 1,2,.....,N. If the SER2 routine is called upon with a zero value for ISUB, SER2 returns the value N, the number of correlation coefficients in the set. Figure VIII-2 illustrates the use of the SER2 routine in the estimation routine POLYD3 for evaluating a third degree polynomial using coefficients stored in a data record.

```

REAL FUNCTION POLYD3 (MP,V1,V2,INDEX,IC)
IC=0
C0=SER2 (MP,V1,V2,INDEX,1)
C1=SER2 (MP,V1,V2,INDEX,2)
C2=SER2 (MP,V1,V2,INDEX,3)
C3=SER2 (MP,V1,V2,INDEX,4)
POLYD3=((C3*V1+C2)*V1+C1)*V1+C0
RETURN
END

```

Example Use of the SER2 Routine

Figure VIII-2

Retrieval Service Routine SER3

The third retrieval service routine is the FORTRAN sub-routine SER3. The SER3 routine returns sets of values of two independent variables together with the tabulated property data. The standard request for tabular property data (data type 3) using the SER3 routine is

```
CALL SER3 (MP,V1,V2,INDEX,N1,VIND1,N2,VIND2,TABLE)
```

The arguments of SER3 are:

- MP - physical property code, an integer or integer variable
- V1 - value of first independent variable, a real number or real variable
- V2 - value of second independent variable, a real number or real variable
- INDEX - pure substance index number, an integer or integer variable
- N1 - number of tabulated values for the first independent variable, an integer variable

VIND1 - first independent variable value vector,
a real singly subscripted variable

N2 - number of tabulated values for the second
independent variable, an integer variable

VIND2 - second independent variable value vector,
a real singly subscripted variable

TABLE - tabulated property data array, a real
doubly subscripted variable

The arguments representing physical property code, independent variable values, and pure substance index have been discussed earlier in this section of the manual. The variables N1 and N2 must contain the dimensions of the array TABLE as specified in the DIMENSION statement of the estimation program when the SER3 routine is called. The variable N1 contains the number of rows and N2 contains the number of columns that are specified in the estimation routine's DIMENSION statement for the array TABLE. The SER3 routine replaces the value of N1 with the number of rows found in the array retrieved from a valid data record (not always equal to the specified value of N1), and replaces the value of N2 with the number of columns found in the retrieved array of property data. Upon completion of SER3 retrieval of data from a valid data record, the first independent variable vector VIND1 contains the tabulated values for the first independent variable vector, VIND2 contains the tabulated values for the second independent variable (one value for each column of the retrieved property data array), and array TABLE contains tabulated data, all for subsequent use in the estimation routine. Each row of the array TABLE contains the tabulated property data for a single value of

the first independent variable contained in VIND1, and each column contains the tabulated property data for a single value of the second independent variable contained in VIND2. If the property data is tabulated as a function of only one independent variable, the variable N2 will be returned to the estimation routine with the value zero. Figure VIII-3 illustrates the use of the SER3 retrieval service routine.

```
REAL FUNCTION INTERP (MP,V1,V2,INDEX,IC)
DIMENSION Y(20),X(20),T(20,20)
IC=0
```

```

      .
      .
      .

```

```

NY=20
NX=20
CALL SER3 (MP,V1,V2,INDEX,NY,Y,NX,X,T)
DO 10 I=1,NY
DO 10 J=1,NX

```

```

      .
      .
      .

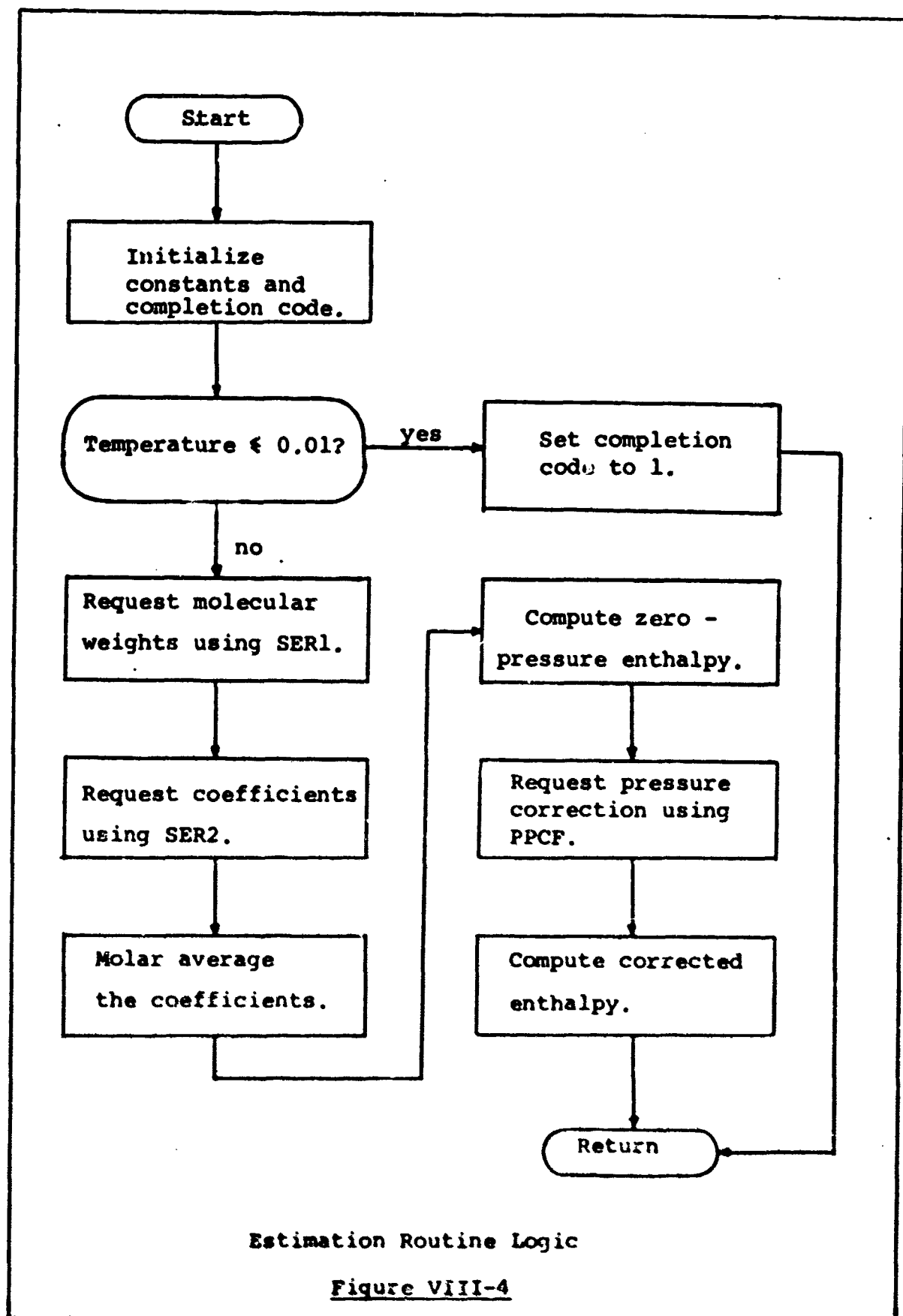
```

Example Use of the SER3 Routine

Figure VIII-3

8.5 Example Estimation Routine

The example estimation routine is a single valued mixture property estimation routine named HLIQ. The routine computes the liquid enthalpy of a mixture—using a correlation to determine a zero pressure enthalpy and using the generalized retrieval routine to obtain an enthalpy pressure correction. Figure VIII-4 illustrates the logic flow diagram of the estimation routine and Figure VIII-5 illustrates the program.



```

      REAL FUNCTION HLIQ(MP,T,P,X,NCOMP,IC)
C
C-- THIS IS AN EXAMPLE SINGLE-VALUED MIXTURE
C   PROPERTY ESTIMATION ROUTINE.
C
      DIMENSION X(20),C(20)
C
C-- INITIALIZE VARIABLES
C
      IC=0
      DO 10 K=1,5
10    C(K)=0.0
C
C-- CHECK TEMPERATURE
C
      IF( T .LT. 0.01 ) GO TO 99
C
C-- REQUEST COEFFICIENTS AND MOLECULAR WT. FOR
C   EACH COMPONENT, AND THEN AVERAGE THE
C   COEFFICIENTS.
C
      DO 20 K=1,NCOMP
      IF(X(K).LT.0.001) GO TO 20
      XMWT=SER1(1006,T,P,K)
      DO 15 I=1,5
15    C(I) = C(I) + SER2(MP,T,P,K,I)*X(K)*XMWT
20    CCNTINUE
C
C-- COMPUTE ZERO PRESSURE ENTHALPY
C
      ZPH = T*(C(1)+T*(C(2)+T*C(3)))+C(4)/T+C(5)
C
C-- REQUEST PRESSURE CORRECTION (PROPERTY 412 )
C
      PCOR = PPCF( 412,T,P,X,1K)
C
C-- COMPUTE ENTHALPY AND RETURN
C
      HLIQ = ZPH - PCOR
      RETURN
C
C-- SET COMPLETION CODE
C
99    IC = 1
      HLIQ = 0.0
      RETURN
      END

```

Example Estimation Routine

Figure VIII-5

8.6 Entering an Estimation Routine Into the Library

All property estimation routines reside in the property system's library. At present only the system librarian can enter an estimation routine into the estimation routine library. The librarian must alter the new estimation routine to modify all calls to generalized retrieval routines. Since estimation routines contain calls to the generalized retrieval routines, the generalized retrieval routines must have recursive capabilities. FORTRAN subprograms cannot directly or indirectly be called recursively. Therefore, modifications to the calls on generalized retrieval routines are made prior to entering the routine into the library.

The modification that is made is a substitution of a call to a communication routine for the call to a generalized retrieval routine. There is a set of communication routines corresponding to each type of generalized retrieval routine. Each communication routine has the capability of calling a set of estimation routines. In setting up the communication routines care is taken to insure that no two estimation routines that require property values computed by each other is called by the same communication routine. By taking such precautions the communication routines partition the estimation routines into sets of unrelated routines. This procedure insures that no estimation routine is called recursively.

At present the responsibility for partitioning the set of estimation routines falls upon the system librarian. A processor system should be developed to partition the set of estimation routines and to modify calls to the generalized retrieval routines from within an estimation routine. If such a system is developed the "recursion" problem will be

solved and the system librarian will be relieved of a tedious task.

Appendix I contains the current list of property estimation routines that are resident in the system's library. A list of completion codes for each estimation routine may also be found in Appendix I.

CHAPTER IX

CONCLUSION AND RECOMMENDATIONS

9.1 Conclusion

A prototype system for storage and retrieval of physical properties of pure chemicals and mixtures has been described. The system allows the engineer to prepare general purpose application programs that do not require programming modification when chemical mix and/or operating conditions change. Variant and invariant information concerning the attributes of requested physical properties are separated, allowing general purpose application programs to be written and allowing limited or complete control over property data and value retrieval. The system has a data base and library thus allowing storage of property data in several forms. Storage routines allow the user to add, update or delete property information.

The property system has been used to provide property values to a large computer-aided design system created by Mr. Hajime Komaki (13). The property system performed successfully in providing property values for design and simulation of a natural gasoline plant studied by Mr. Komaki.

9.2 Recommendations

In using the property system many features that would enhance the system and would allow full scale implementation have been recognized.

1. Provision for allowing several different sets of engineering units to be used should be implemented. (for example, temperature in °K, °R, °C, °F and pressure in atm., psia, mm/Hg)
2. A preprocessor should be developed that would scan an application program before execution to determine the required properties, and then generate a personal property set containing only the property data and estimation routines required.
3. The system should be expanded to allow retrieval of property data from disk together with retrieval from the core resident data base.
4. The application program should be given the capability of altering the retrieval constraint table during execution.
5. A new attribute for estimation routines should be added to the seven existing. The attribute should give indication of time requirements for property estimation.
6. A trace feature should be incorporated into the system in order to determine, after execution, the estimation routine used, and the time required.
7. The system should have a mode of operation for generating tables of estimated property values and then generating a correlation for the property. This would allow considerable savings of time during application program execution.

8. Assembly language recursive generalized retrieval routines should be written to eliminate the need for the communication routines.
9. A system for interrogation of the property system's data base is needed to simplify the task of determining the available property data. The system could be implemented in a time shared environment to allow interaction.
10. The property system should be implemented in a time shared environment to allow storage and retrieval of property data and values interactively at a terminal.

Appendix I

Appendix I contains the lists of property codes, data base component codes, data type codes, and estimation routines. The lists of codes used by the property system are the lists of assigned codes. Physical property data does not exist in the master data base for all property codes and for each data base component code. At present the user must use the property system to determine if property data is available. A system that allows the user to inquire concerning availability of property data is planned.

A. Physical Property Codes

<u>Code</u>		<u>Independent Variables</u>	
100	Density	T	P
101	- vapor	T	P
102	- liquid		
200	Fugacity Coefficient	T	P
201	- vapor		
202	- liquid		
300	Activity Coefficient	T	P
302	- liquid		
305	Equilibrium Coefficients	T	P
400	Enthalpy	T	P
401	- vapor		
402	- liquid		
411	Enthalpy Pressure Correction - vapor	T	P
412	- liquid		
425	Zero Pressure Enthalpy	T	P
430	Heat Capacity	T	P
431	- vapor		
432	- liquid		
515	Bubble Pt. Temperature	P	-
516	Dew Pt. Temperature	P	-
517	Temperature	Enthalpy	P
518	Vapor Fraction	T	P
1001	Critical Temperature	-	-
1002	Critical Pressure	-	-
1003	Acentric Factor		-
1004	Solubility Parameter	-	-
1005	Molar Volume	-	-
1006	Molecular Wt.	-	-
1007	Melting Pt. Temperature	-	-

B. Data Base Component Codes

<u>Compound or Element</u>	<u>Data Base Component Number</u>	<u>Compound or Element</u>	<u>Data Base Component Number</u>
Hydrogen	1	1-Pentene	29
Methane	2	Cis-2-Pentene	30
Ethane	3	Trans-2-Pentene	31
Propane	4	2-Methyl-1-Butene	32
i-Butane	5	3-Methyl-1-Butene	33
n-Butane	6	2-Methyl-2-Butene	34
i-Pentane	7	1-Hexene	35
n-Pentane	8	Cyclopentane	36
neo-Pentane	9	Methylcyclopentane	37
n-Hexane	10	Cyclohexane	38
n-Heptane	11	Methylcyclohexane	39
n-Octane	12	Benzene	40
n-Nonane	13	Toluene	41
n-Decane	14	O-Xylene	42
n-Undecane	15	M-Xylene	43
n-Dodecane	16	P-Xylene	44
n-Tridecane	17	Ethylbenzene	45
n-Tetradecane	18	Ammonia	46
n-Pentadecane	19	H ₂ O	47
n-Hexadecane	20	Ethyl Alcohol	48
n-Heptadecane	21	Acetone	49
Ethylene	22	Nitrogen	50
Propylene	23	Oxygen	51
1-Butene	24	Carbon Monoxide	52
Cis-2-Butene	25	Carbon Dioxide	53
Trans-2-Butene	26	Air	54
i-Butene	27	Argon	55
1,3-Butadiene	28		

C. Data Type Codes

- 0 - Null data
- 1 - Constant data
- 2 - Correlation coefficient data
- 3 - Tabular data

D. Estimation Routines:

Routine number: 1

Name : BWRLD
Type : FUNCTION, mixture routine
Purpose : BWRLD computes liquid density of a mixture (property code 102) using the Benedict-Webb-Ruben equation of state.
Requirements: BWRLD requires the eight BWR coefficients $A_0, B_0, C_0, a, b, c, \alpha, \gamma$ (A_0 the first, γ the eighth) for each component of the mixture
Completion : 0 - OK
Codes
1 - Non-existent
2 - All mole fractions : 10^{-6} , BWRLD = 0.0

Routine number: 2

Name : BWRVD
Type : FUNCTION, mixture routine
Purpose : BWRVD computes vapor density of a mixture (property code 101) using the Benedict-Webb-Ruben equation of state.
Requirements: BWRVD requires eight BWR coefficients $A_0, B_0, C_0, a, b, c, \alpha, \gamma$ (A_0 the first, γ the eighth) for each component of the mixture.
Completion : 0 - OK
Codes
1 - Non-existent
2 - All mole fractions : 10^{-6} , BWRVD = 0.0

Routine number: 3

Name : ZPH
Type : FUNCTION, mixture routine
Purpose : ZPH compute zero pressure enthalpy
(property code 425) using the equation
$$H_0 = c_1T + c_2T^2 + c_3T^3 + c_4/T + c_5$$

Requirements: (1) ZPH requires the five coefficients
 C_1, C_2, C_3, C_4, C_5 , for each component of the
mixture. Reference: API Data Book
(2) molecular weight of each component
(property code 1006)
Completion : 0 - OK
codes
1 - Temperature ≤ 0.0 ,
ZPH = 0.0

Routine number: 4

Name : BUBTP1
Type : FUNCTION, mixture routine
Purpose : BUBTP1 computes the bubble point temperature
of a mixture (property code 515).
Requirements: (1) liquid fugacity coefficient for each
component (property code 202)
(2) liquid activity coefficient for
each component (property code 302)
(3) vapor fugacity coefficient for each
component (property code 201)
Completion : 0 - OK
codes
1 - No mixture components. BUBTP1 = 0.0
2 - Non-existent
3 - Temperature iteration failed. BUBTP1
= last iteration temperature

4 - Equilibrium coefficient iteration
failed. BUBTP1 = last iteration
temperature.

Routine number: 5

Name : DEWTP1
Type : FUNCTION, mixture routine
Purpose : DEWTP1 computes the dew point temperature
of a mixture (property code 516)
Requirements: (1) vapor fugacity coefficient for each
component (property code 201)
(2) liquid fugacity coefficient for each
component (property code 202).
(3) liquid activity coefficient for each
component (property code 302)
Completion : 0 - OK
codes
1 - No mixture components. DEWTP1 = 0.0
2 - Non-existent
3 - Temperature iteration failure
DEWTP1 = current value
4 - Equilibrium coefficient iteration
failure. DEWTP1 = current value.

Routine number: 6

Name : BWRLPC
Type : FUNCTION, mixture routine
Purpose : BWRLPC computes the liquid enthalpy
correction to zero pressure enthalpy
due to pressure (property code 412)
using the Benedict-Webb-Ruben equation
of state

Requirements: (1) eight BWR coefficients $A_0, B_0, C_0, a, b, c, \alpha, \gamma$ for each component

(2) liquid density (property code 102).

Completion : 0 - OK
codes

1 - Non-existent

2 - All mole fractions $\leq 10^{-6}$. BWRLPC = 0.0

3 - Density $\leq 10^{-6}$, BWRLPC = 0.0

Routine number: 7

Name : VFRAC

Type : FUNCTION, mixture routine

Purpose : VFRAC computes the vapor fraction of a mixture (property code 516).

Requirements: (1) liquid fugacity coefficient for each component (property code 202)

(2) vapor fugacity coefficient for each component (property code 201)

(3) liquid activity coefficient for each component (property code 302)

Completion : 0 - OK
codes

1 - Vapor fraction converged. VFRAC $\geq$ 1.0
VFRAC = 1.0

2 - Vapor fraction iteration failed
VFRAC = 0.0

3 - Equilibrium coefficient iteration failed. VFRAC = 0.0

4 - Vapor fraction converged. VFRAC $\leq$ 0.0.
VFRAC = 0.0

Routine number: 8

Name : FUGL1
Type : SUBROUTINE, mixture routine
Purpose : FUGL1 computes the liquid fugacity coefficient for each component (property code 202). The method is that discussed by Chao-Seader (AIChE Journal, 7, 598, 1961). Coefficient by Grayson - Streed. (Sixth World Petroleum Congress, Section VII, Paper 20 - PE7, June 1963).

Requirements: (1) coefficients $C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9$ for each component.
(2) critical temperature (property code 1001).
(3) critical pressure property code (1002)
(4) acentric factor (property code 1003)

Completion : 0 - OK
codes

Routine number: 9

Name : ACTL1
Type : SUBROUTINE, mixture routine
Purpose : ACTL1 computes a liquid activity coefficient for each component (property code 302). The method is that discussed by Chao-Seader. (AIChE. Journal, 7, 598, 1961)

Requirements: (1) molar volume @ 25°C (property code 1005)
(2) solubility parameter (property code 1004)

Completion : 0 - OK
codes

Routine number: 10

Name : FUG
Type : SUBROUTINE, mixture routine
Purpose : FUG computes a vapor fugacity coefficient for each component (property code 201 using the Benedict-Webb-Ruben equation of state.
Requirements: (1) eight BWR coefficients $A_0, B_0, C_0, a, b, c, \alpha, \gamma$ for each component
(2) vapor density (property code 101).
Completion : 0 - OK
codes
1 - All mole fractions $\leq 10^{-6}$, XFUG(I) = 1.0
2 - Density $\leq 10^{-6}$, XFUG(K) = 1.0

Routine number: 11

Name : KVAL
Type : SUBROUTINE, mixture routine
Purposeq : KVAL computes a equilibrium coefficient for each component (property code 305).
Requirements: (1) liquid fugacity coefficient for each component (property code 202).
(2) vapor fugacity coefficient for each component (property code 201)
(3) liquid activity coefficient for each component (property code 302).
Completion : 0 - OK
codes
1 - vapor fraction converged > 1.0
2 - Vapor fraction failed to converge
XK(I) = latest iterated value.

- 3 - Equilibrium coefficient iteration failed. $XK(I)$ = latest iterated value.
- 4 - Vapor fraction converged < 0.0

Routine number: 12

Name : ENTH
Type : FUNCTION, mixture routine
Purpose : ENTH computes the enthalpy of a vapor or liquid mixture (property code 401 or 402).
Requirements: (1) zero-pressure enthalpy (property code 425)
Completion : 0 - OK
codes 1 - property code does not end in "1" or "2".

Routine number: 13

Name : TSUBH
Type : FUNCTION, mixture routine
Purpose : TSUBH computes iteratively the temperature given enthalpy of a mixture (property code 517)
Requirements: (1) bubble point temperature (property code 515)
(2) dew point temperature (property code 516)
(3) vapor fraction (property code 518)
(4) equilibrium coefficients (property code 305)

(5) vapor enthalpy (property code 401)
(6) liquid enthalpy (property code 402)
Completion : 0 - OK
codes
1 - Bubble point failed. TSUBH undefined.
2 - Dew point failed. TSUBH undefined
3 - Vapor fraction failed. TSUBH undefined.
4 - K-valued failed. TSUBH undefined.
5 - Temperature iteration failed. TSUBH =
last iteration value.

E. Property System Messages

1. "DATA BASE SUCCESSFULLY INITIALIZED"

Generating Routine : COMP

Reason : All is well; initialization complete.

2. "BUBBLE PT. FAILURE"

Generating Routine : PPCF or PPCS

Reason : Phase determination was required and
bubble point determination failed.

3. "DEW PT. FAILURE"

Generating Routine : PPCF or PPCS

Reason : Phase determination was required
and dew point failed.

4. "VAPOR FRACTION FAILED"

Generating Routine : PPCF or PPCS

Reason : Phase determination was required
and vapor fraction failed.

5. "ERROR*ERROR*PPnF NOT AVAILABLE"

Generating Routine : PPCF

Reason : Communication routine PPnF (n an integer) is required but has not been entered into the library. Contact system librarian.

6. "ERROR*ERROR*PPCF - COULD NOT DISTINGUISH ROUTINE TYPE"

Generating Routine : PPCF, PPCS or PPCP

Reason : The system library directory contains an error. Each estimation routine should have "S" or "F" associated with it (SUBROUTINE or FUNCTION). The routine in question did not

7. "ERROR*ERROR*PPnS NOT AVAILABLE"

Generating Routine : PPCS

Reason : Same as 5.

8. "ERROR*ERROR*PPnP NOT AVAILABLE"

Generating Routine : PPCP

Reason : Same as 5.

9. "ERROR*ERROR*WRONG PPn $\begin{Bmatrix} F \\ S \\ P \end{Bmatrix}$ CALLED** PPn $\begin{Bmatrix} F \\ S \\ P \end{Bmatrix}$ "

Generating Routine : PPnF, PPnS, or PPnP (n an integer)

Reason : Catastrophic malfunction. Contact system librarian. Data base must be rebuilt.

10. "ERROR*ERROR*REQUESTED ESTIMATION ROUTINE NOT ENTERED***

PPn $\begin{Bmatrix} F \\ S \\ P \end{Bmatrix}$ "

Generating Routine : PPnF, PPnS, PPnP (n an integer)

Reason : Calling statement for estimation
routine has not yet been entered.
Contact system librarian.

11. "***WARNING***COMPLETION CODE = n_1 FOR ROUTINE NUMBER
 n_2 ***"

Generating Routine : PPnF, PPnS, or PPnP (n an integer)

Reason : An estimation routine encountered computa-
tion difficulties. See section in
Appendix I concerning estimation routines
for completion codes.

12. "ERR n ... < TEXT >"

Generating Routine : Data base storage and retrieval
routines

Reason : Each error message is numbered and
contains text explaining the error
that came up. Text is self explanatory.
A "POST MORTEM" dump of important variables
is made and the system aborts.

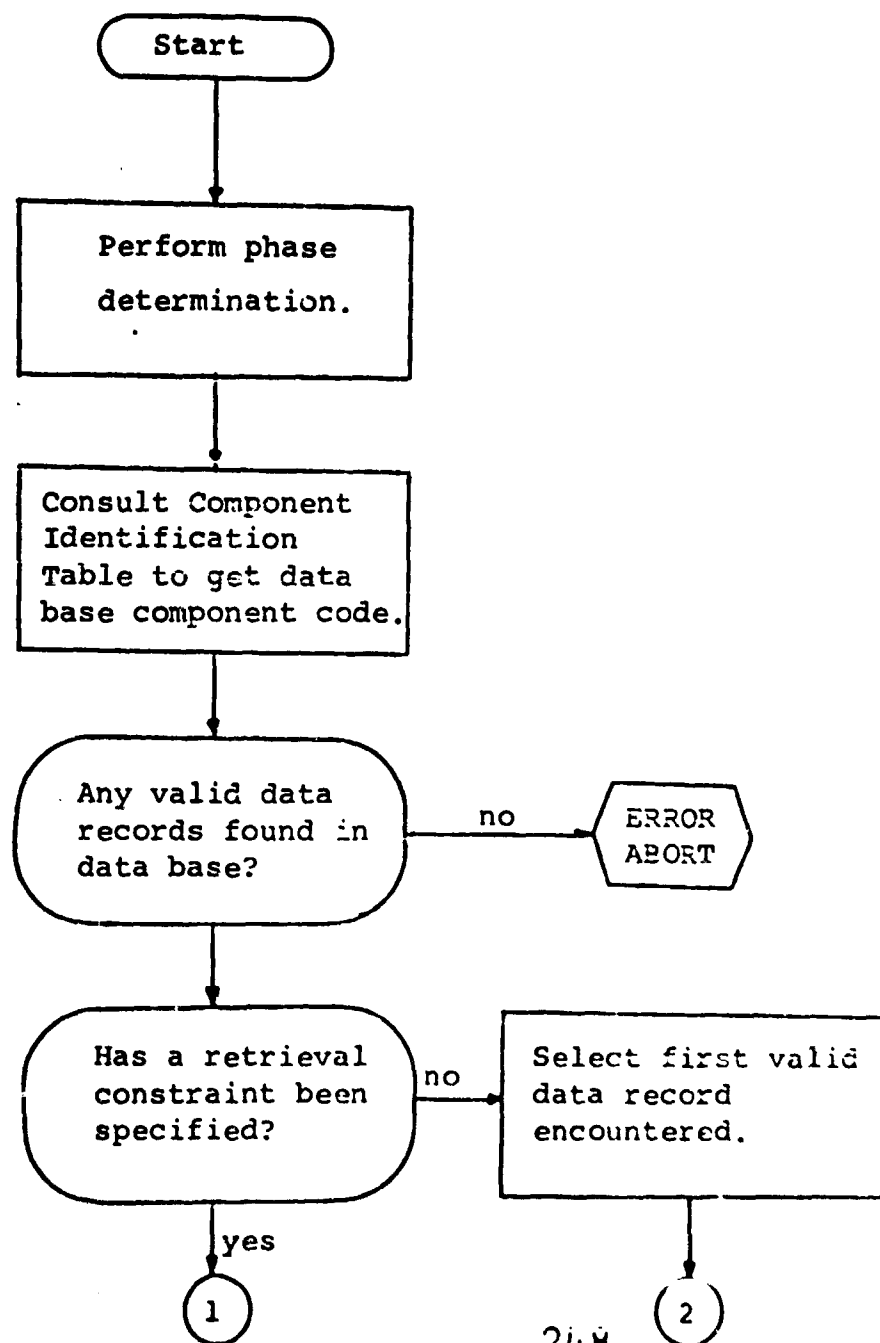
Appendix II Logic Diagrams

A. Pure Chemical Property Value Retrieval Using PPCP

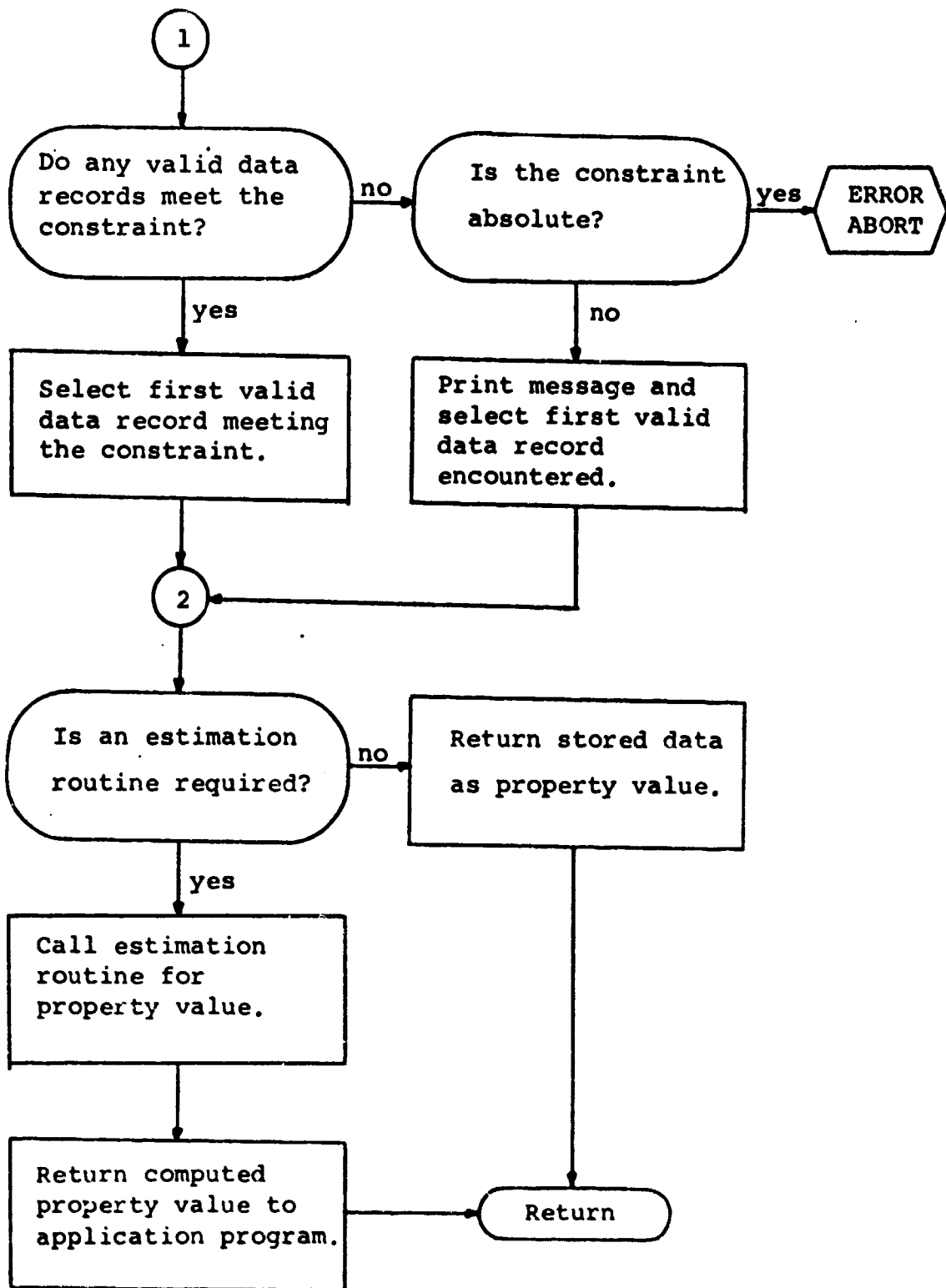
Invariant information provided:

1. property code
2. pure substance index
3. independent variable values

Logic diagram:



248

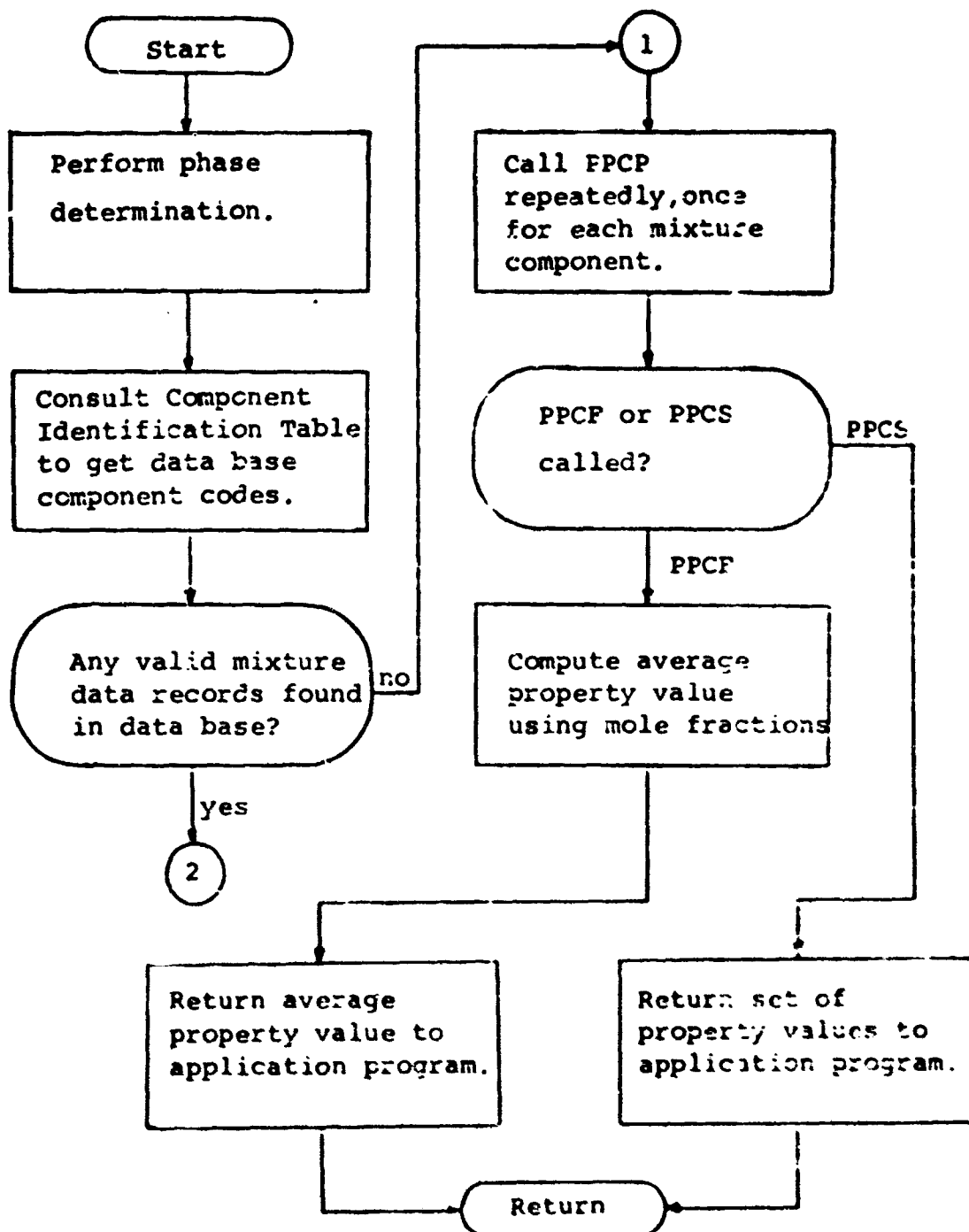


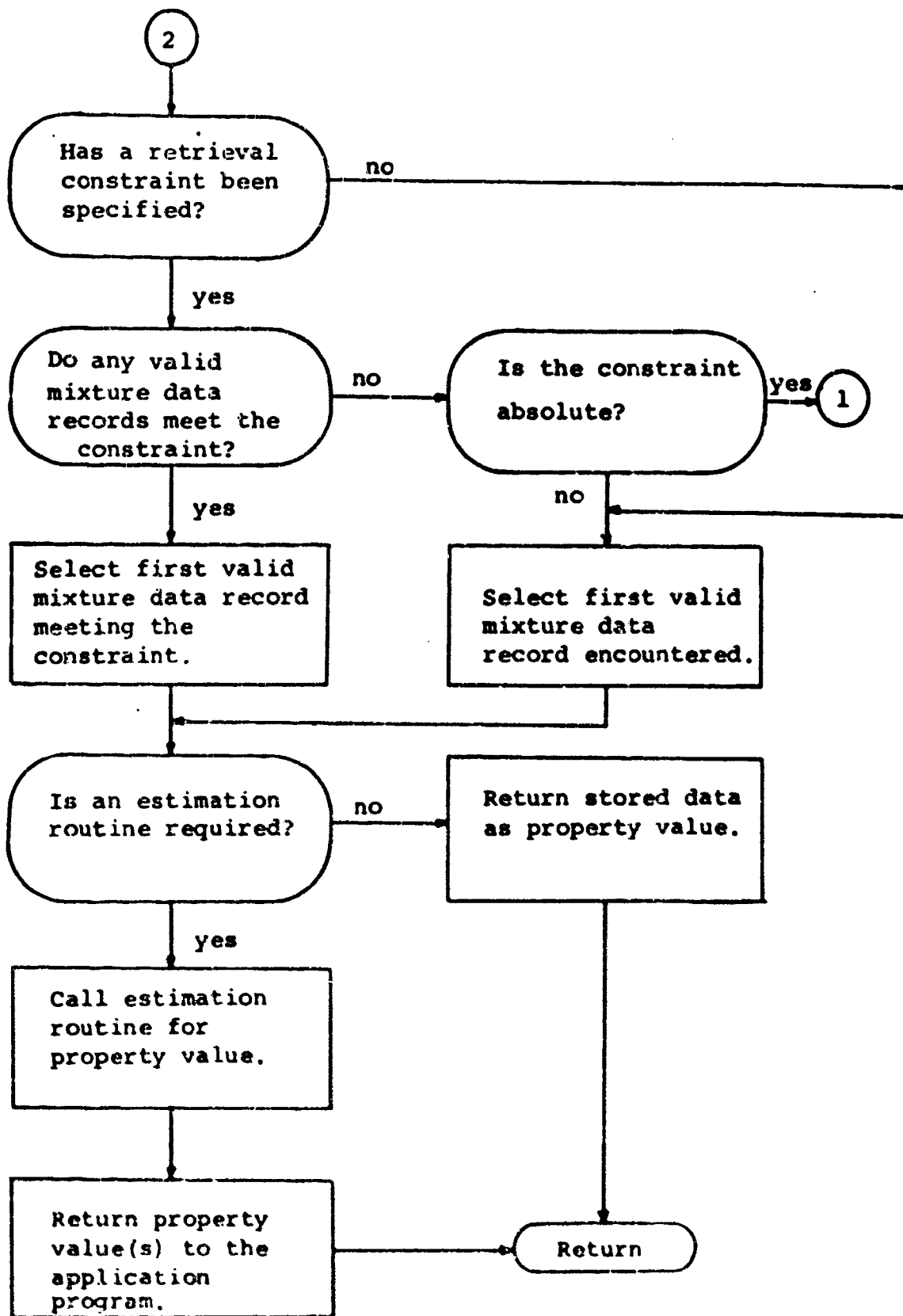
B. Mixture Property Value Retrieval Using PPCF or PPCS

Invariant information provided:

1. property code
2. independent variable values
3. mole fractions

Logic diagram:





Appendix III U of P IBM 360/75 JCL

A. Execution of an Application Program

```
//name JOB (nnnn,bbbb), 'soc. sec. ..', CLASS={J  
//STEP1 EXEC PORTGCLG  
//PORT.SYSIN DD *
```

[Application Program]

/*

```
//LKED.SYSLIB DD DSN=SYS1.FORTLIB, L'SP=SHR  
//  
//GO.FT03F001 DD DSN=(name of property system library), DISP=SHR  
//GO.FT03F001 DD DSN=(name of personal data base), DISP=SHR  
//GO.FT11F001 DD DUMMY  
//GO.SYSIN DD *
```

[Variant Information Deck
Application Program's Data]

/*

B. Execution of Storage Routine

```
//name JOB (nnnn,bbbb), 'soc. sec. no.', CLASS={J  
//JOB LIB DD DSN=(name of property system library), DISP=SHR  
//STEP1 EXEC PGM=STORAGE  
//FT03F001 DD DSN=(name of master data base), DISP=SHR  
//FT04F001 DD DSN=(name of personal data base), DISP=SHR  
//FT06F001 DD SYSOUT=A  
//FT05F001 DD *
```

[Storage Control Deck]

/*

Appendix IV Input Language Specification

This appendix contains the language specification in Backus Naur Form of the variant information deck and the storage control deck.

In the following specification, all terminal symbols (primitives) are underlined; they are to be considered self referencing symbols.

A. Variant Information Deck

```
<variant information deck> ::= BEGIN VARIANT DATA <decks> END
<decks> ::= <component identification deck>|
          <component identification deck><retrieval const.deck>
<component identification deck> ::= COMPONENT ID TABLE
                                   <component list>|
<component list> ::= <data base component code>|
                   <component list><delimiter><databasecomponent>
<delimiter> ::= _ | blank
<retrieval const. deck> ::= RETRIEVAL CONSTRAINTS<constraint list>|
<constraint list> ::= <constraint entry>|<constraint entry><blank>
                   <constraint entry>
<constraint entry> ::= <property entry><delimiter>
                   <allowed error entry><delimiter>
                   <routine no. entry><delimiter>
                   <contributor entry><delimiter>
                   <degree entry>
<property entry> ::= _|<integer>
<allowed error entry> ::= _ | <integer><u>
<routine no. entry> ::= _ | <integer> | C
```

<contributor entry> ::= $_$ | <integer>

<degree entry> ::= $_$ | $\%$

B. Storage Control Deck

<storage control deck> ::= <password card><control deck>

END STORAGE DECK

<password card> ::= OLD PASSWORD = <password> |

NEW PASSWORD = <password>

<password> ::= <integer>

<control deck> ::= <addition deck><update deck><deletion deck>

<addition deck> ::= ADDITION DECK <data record deck>

<data record deck> ::= <data record> | <data record deck>

<data record>

<data record> ::= (<characteristic part>)!<data group>!

<characteristic part> ::= <property entry>_<contributor entry>_

<variable range entry>_

<variable range entry>_

<max. error entry>_

<estimation routine entry>_

<data type>

<variable range entry> ::= $_$ | <real no.> $_$ <real no.>

<max. error entry> ::= $_$ | <integer> $\%$

<data group> ::= $_$ | <data part>|<data group><data part>

<data part> ::= (<component list> ; <data>)

<data> ::= <constant data> | <coefficient data> | <tabular data>

<constant data> ::= <real no.>

<coefficient data> ::= <real no.> | <coefficient data><real no.>

$\langle \text{tabular data} \rangle ::= [\langle \text{variable values} \rangle] (\langle \text{variable values} \rangle)$
 $\langle \text{row values} \rangle$

The $\langle \text{variable values} \rangle$ and $\langle \text{row values} \rangle$ classes cannot be specified in BNF form. The reader is directed to Chapter VII for an explanation of tabular data.

$\langle \text{update deck} \rangle ::= \text{UPDATE DECK } \langle \text{data record deck} \rangle$

$\langle \text{deletion deck} \rangle ::= \text{DELETION DECK } \langle \text{deletion record deck} \rangle$

$\langle \text{deletion record deck} \rangle ::= \langle \text{deletion record} \rangle |$

$\langle \text{deletion record deck} \rangle \langle \text{deletion record} \rangle$

$\langle \text{deletion record} \rangle ::= [\langle \text{characteristic part} \rangle]$

$[\langle \text{component list} \rangle]$

The following classes are considered primitive :

$\langle \text{integer} \rangle$

FORTRAN definition accepted

$\langle \text{real no.} \rangle$

BIBLIOGRAPHY

1. Yen, Y. C., K., R. Cantwell, and B. L. Giles, "General Purpose Data System for Computer Process Calculations," Ind. and Eng. Chem., 60, no. 2, pp. 70-73 (February, 1968).
2. Zseleczky, E. P., "Computers in Process Analysis and Design," Proc. Am. Petrol. Instit., 42, sect. III, pp. 345-350 (1962).
3. Shannon, P. T., A. I. Johnson, C. M. Crowe, T. W. Hoffman, A. E. Hamielec, and D. R. Woods, "Computer Simulation of a Sulfuric Acid Plant," Chem. Engr. Progr., 62, no. 6, pp. 49-59 (June, 1966).
4. Ravicz, A. E., and R. L. Norman, "Heat and Mass Balancing on a Digital Computer," Chem. Engr. Progr., 60, no. 5, pp. 71-76 (May, 1964).
5. "Chemically-Oriented Chips," EDP Weekly (December 5, 1966).
6. Kesler, M. G. and P. R. Griffiths, "A Computer System for Process Simulation," Proc. Am. Petrol. Instit., 43, sect. III, pp. 49-56 (1963).
7. Motard, R. L., "Optimization of Natural Gasoline Plant Operation," Computers in Engineering Design Education, vol. II, pp. 36-70, University of Michigan, Ann Arbor, Michigan (April 1, 1966).
8. Meadows, E. L., Jr., "A. I. Ch. E. Physical Properties Project," Proc. Am. Petrol. Instit., 44, sect. III, pp. 300-303 (1964).
9. Meadows, E. L., Jr., "Estimating Physical Properties: the A. I. Ch. E. System," Chem. Engr. Progr., 61, no. 5, pp. 93-95 (May, 1965).
10. Heitman, R. E. and G. H. Harris, "Estimation of Physical Properties by Minimum Error Analysis," Ind. and Eng. Chem., 60, no. 2, pp. 50-59 (February, 1969).
11. Beirute, R. M., "A Thermodynamic and Physical Property Package," Master's thesis, University of Houston (January, 1969).

12. Evans, L. B., D. G. Steward and C. R. Sprague, " Computer Aided Chemical Process Design," Chem. Engr. Progr., 64, no. 4, pp. 39-46 (April, 1968).
13. Komaki, H., "Computer Aided Chemical Engineering Design," Master's thesis, The School of Chemical Engineering, University of Pennsylvania (August, 1969).
14. Johnson, A. I., "Chemical Plant Simulation - A Manual for the Digital Computer Simulation Programs MACSIM GEMCS," Department of Chemical Engineering, McMaster University (March, 1968).

231