

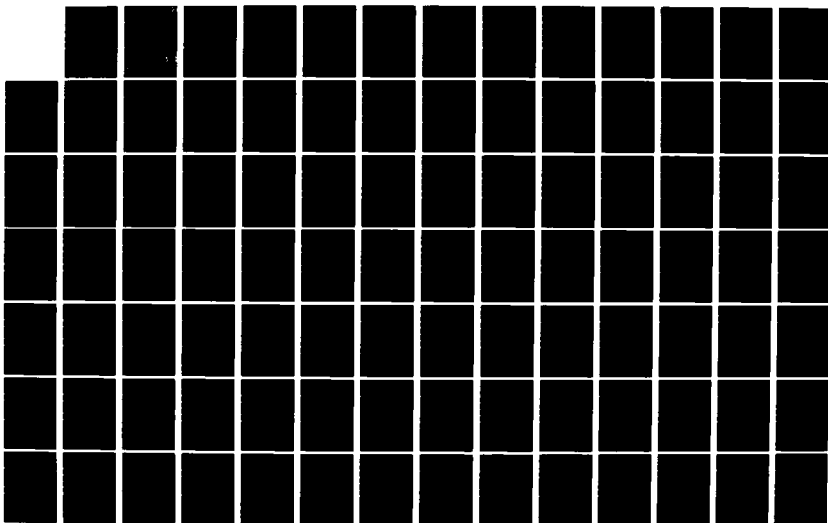
AD-A124 984

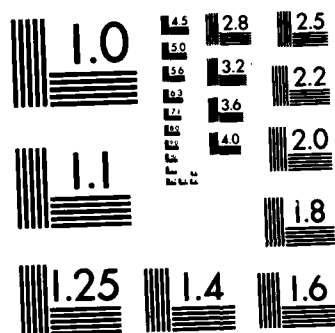
DEVELOPMENT OF COMPUTER PERFORMANCE EVALUATION TOOLS
FOR VAX-11/780 COMPUTERS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... E C GILPIN
DEC 82 AFIT/GCS/EE/82D-15 F/G 9/2

1/2

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A124904



DEVELOPMENT OF
COMPUTER PERFORMANCE EVALUATION TOOLS
FOR VAX-11/780 COMPUTERS

THESIS

AFIT/GCS/EE/82D-15

Eugene C. Gilpin, Jr.
Captain USAF

DTIC
ELECTE
FEB 25 1983

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A

TIC FILE COPY

028

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE 82D-15	2. GOVT ACCESSION NO. AD A124 904	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Development of Computer Performance Evaluation Tools for VAX-11/780 Computers		5. TYPE OF REPORT & PERIOD COVERED MS THESIS
7. AUTHOR(s) Eugene C. Gilpin, Jr., Captain, USAF		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE December, 1982
		13. NUMBER OF PAGES 230
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Approved for public release; IAW AFR 190-17 4 JAN 1983 LYNN E. WOLAVER Dean for Research and Development Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Monitors Computer Performance Evaluation Digital computer software Computer Programming		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) See Reverse		

~~UNCLASSIFIED~~

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

20. Abstract

This thesis reports on an effort to develop a computer performance software monitor for Digital Equipment Corporation's VAX-11/780 computers. The role of the monitor developed by this effort will be to address long-range, as well as short-range performance metrics for a VAX system.

The scope of this effort was limited to a top-level design of the Performance Monitor Tool (PERMTOOL) coupled with a complete design and development of some components of the system. The system is designed to provide maximum user support through a wide range of possible measurement requirements, ranging from job parameter measurements to analyses of device utilizations within the VAX. The system has been designed to permit the user to specify activities in three major areas: data collection, data reduction, and data formatting/reporting. Once started, any such activity will proceed independently of the rest of the PERMTOOL system.

Both the design approach (top-down, modular) and the program methodology (structured) facilitate modifications to the PERMTOOL as changing requirements dictate. Several extensions of this effort are identified which seem worthy of additional research and development.

UNCLASSIFIED

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



DEVELOPMENT OF
COMPUTER PERFORMANCE EVALUATION TOOLS
FOR VAX-11/780 COMPUTERS

THESIS

AFIT/GCS/EE/82D-15 Eugene C. Gilpin, Jr.
Captain USAF

AFIT/GCS/EE/82D-15

DEVELOPMENT OF
COMPUTER PERFORMANCE EVALUATION TOOLS
FOR VAX-11/780 COMPUTERS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air Training Command
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Eugene C. Gilpin, Jr. BGS

Captain USAF

Graduate Electrical Engineering

December 1982

Approved for public release; distribution unlimited.

Preface

This thesis represents the initial effort to define a suite of performance measurement tools for Digital Equipment Corporation's VAX-11 computers. As such, this report is most concerned with defining requirements, and less concerned with specifying functional mechanisms. However, this thesis does present an initial design and implementation of a User Interface module, as well as the design of a data collection module.

Throughout this effort, I have received a great deal of help and support from several people; I want to take this opportunity to thank them. First, Dr. Gary Lamont, my thesis advisor, who has kept my feet on the proper road for this thesis. The members of my thesis committee, Dr. Thomas Hartrum and Major Michael Varrieur, have also been a great help, not only reading drafts of this paper, but also providing suggestions and background information which were essential to my project. On several occasions, members of the Digital Equipment Corporation Support Team at the Aeronautical Systems Division computer center have provided needed advice and direction. Finally, of course, I must thank my family: Beverly, Ellen, and Kit, who have distracted me when I needed distracting, and left me alone when I needed to concentrate.

PREFACE	i
LIST OF FIGURES	ii
LIST OF TABLES	iii
LIST OF ACRONYMS	iv
GLOSSARY	v
ABSTRACT	vi

CHAPTER 1 INTRODUCTION

1.1 PURPOSE	1-1
1.2 BACKGROUND	1-1
1.3 PROBLEM DEFINITION	1-3
1.4 SCOPE	1-4
1.5 THESIS ORGANIZATION	1-8

CHAPTER 2 REQUIREMENTS ANALYSIS

2.1 INTRODUCTION	2-1
2.2 PERFORMANCE MEASUREMENTS	2-3
2.3 PERMTOOL USER INTERFACE REQUIREMENTS	2-11
2.4 PERMTOOL SYSTEM INTERFACE REQUIREMENTS	2-12
2.5 SUMMARY	2-13

CHAPTER 3 SYSTEM DESIGN

3.1 INTRODUCTION	3-1
3.2 OVERALL SYSTEM DESIGN	3-2
3.3 PERMTOOL CAPABILITIES	3-5

CHAPTER 4 EXECUTIVE CONTROL MODULE DESIGN

4.1 INTRODUCTION	4-1
4.2 ORDER OF OPERATIONS	4-2
4.3 EXECUTIVE CONTROL DATA STRUCTURES	4-3
4.4 EXECUTIVE CONTROL SOFTWARE STRUCTURES	4-4
4.5 SUMMARY	4-6

CHAPTER 5 USER INTERFACE SOFTWARE DESIGN

5.1 INTRODUCTION	5-1
5.2 FRAME PROCESSING	5-2
5.3 SOFTWARE DATA STRUCTURES	5-8
5.4 SOFTWARE FUNCTIONAL ORGANIZATION	5-10
5.5 SUMMARY	5-19

CHAPTER 6 DATA COLLECTION SOFTWARE DESIGN

6.1	INTRODUCTION	6-1
6.2	DATA COLLECTION DATA STRUCTURES	6-3
6.3	DATA COLLECTION SOFTWARE STRUCTURES	6-4
6.4	SUMMARY	6-8

CHAPTER 7 RUN-TIME RESOURCE UTILIZATION

7.1	INTRODUCTION	7-1
7.2	MEMORY USE	7-1
7.3	CPU USE	7-2
7.4	I/O DEVICE USE	7-3
7.5	SUMMARY	7-3

CHAPTER 8 SOFTWARE DEVELOPMENT, TESTING,
AND INSTALLATION

8.1	INTRODUCTION	8-1
-----	------------------------	-----

CHAPTER 9 CONCLUSIONS AND RECOMMENDATIONS

9.1	CONCLUSIONS	9-1
9.2	RECOMMENDATIONS	9-2
9.3	SUMMARY	9-4

BIBLIOGRAPHY	Bib-1
------------------------	-------

APPENDIX A VAX OPERATIONAL ESSENTIALS

A.1	INTRODUCTION	A-1
A.2	MEMORY MANAGEMENT	A-2
A.3	PROCESS SCHEDULING	A-4
A.4	PRIORITIES	A-5

APPENDIX B PERFORMANCE MEASURES AND ASSOCIATED
DATA ELEMENTS

B.1	PREFACE	B-1
-----	-------------------	-----

APPENDIX C	DIALOGUE FRAMES USED BY PERMTOOL	C-1
------------	--	-----

APPENDIX D USER INTERFACE DATA STRUCTURES D-1

APPENDIX E USER INTERFACE ORGANIZATIONAL STRUCTURES E-1

APPENDIX F DATA COLLECTION ORGANIZATIONAL
STRUCTURES F-1

APPENDIX G SUMMARY OF EFFORT G-1

INDEX Index-1

VITA

List of Figures

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3-1	Major Components and Functional Relationships in the PERMTOOL system	3-5
4-1	Function Diagram of the PERMTOOL Executive and its Subordinate Functions	4-2
4-2	PERMTOOL Executive order of Operations	4-3
5-1	The Dialogue/Response process	5-2
5-2	Components of a Frame_Item	5-3
5-3	Control data and layout of frame MENU01	5-7
5-4	Structured English description of the User Interface Operational Control routine	5-10
5-5	General Operation of validation routines	5-11
5-6	Operation of bad-action-routines	5-12
5-7	General operation of good-action-routines	5-13
6-1	Data Flow through the PERMTOOL collection system	6-2
6-2	Function diagram of major data collection functions	6-5
B-1	Major Subject Area of Data Items	B-1

List of Tables

<u>Table</u>	<u>Title</u>	<u>Page</u>
2-1	"Fields of Interest" to performance improvement teams	2-2
2-2	Data elements used to quantify system reliability	2-4
2-3	Categories of Data listed in Appendix B	2-11
2-4	PERMTOOL Requirements summary	2-14
8-1	Data Items Which can be collected using PERMTOOL	8-6
B-2	Data Items for System Response Time	B-4
B-3	Data Items for System Workload	B-6
B-4	Data Items for Resource Usage Accounting	B-8
B-5	Data Items for Workload Scheduling	B-10
B-6	Data Items for Fine-Tuning Control	B-11
B-7	Consolidated List of Data Items	B-13
C-1	List of Dialogue Frames	C-1

List of Acronyms

DEC - Digital Equipment Corporation, the manufacturer of the VAX line of computers.

DECNET - DEC Network. A package of hardware and software which permits installations to be connected together via a communications network.

PERMTOOL - PERformance Monitoring TOOL

SPSS - Statistical Package for the Social Sciences. An extensions software package which permits users to perform complex statistical reductions of experimental data.

VAX - Virtual Address extension to the PDP-11 line of DEC computers

VMS - Virtual Memory System. This is the principal DEC supported operating system used on the VAX, and is the operating system which is the basis of this thesis effort.

Glossary

Dialogue - A question/answer session between the computer and someone using an interactive terminal connected to the computer. In the context of this paper, the computer makes a statement, followed by a question which the user must answer.

Frame - A set of one dialogue question and its associated answer. Each frame includes a specification of the computer's statement and question, as well as designation of the routines to validate a user response, and take either good, or bad action.

Mailbox - A computer device, similar to conventional post-office boxes. These devices are established on some computer systems to allow computer users to send messages to each other, through the computer.

Abstract

This thesis reports on an effort to develop a computer performance software monitor for Digital Equipment Corporation's VAX-11/780 computers. The role of the monitor developed by this effort will be to address long-range, as well as short-range performance metrics for a VAX system.

The scope of this effort was limited to a top-level design of the Performance Monitor Tool (PERMTOOL) coupled with a complete design and development of some components of the system. The system is designed to provide maximum user support through a wide range of possible measurement requirements, ranging from job parameter measurements to analyses of device utilizations within the VAX. The system has been designed to permit the user to specify activities in three major areas: data collection, data reduction, and data formatting/reporting. Once started, any such activities proceed independently of the rest of the PERMTOOL system.

Both the design approach (top-down, modular) and the program methodology (structured) facilitate modifications to the PERMTOOL as changing requirements dictate. Several extensions of this effort are identified which seem worthy of additional research and development.

CHAPTER 1

INTRODUCTION

1.1 PURPOSE

This thesis records the results of an effort to develop a suite of computer performance measurement tools called PERMTOOL. These tools are intended to be used on Digital Equipment Corporation's VAX-11/780 computers. The tools can provide system managers with information they might need to accurately characterize the work being done by their computer system and permit the manager to fine-tune system operations.

1.2 BACKGROUND

In the late 1970's, the Digital Equipment Corporation (DEC) introduced the VAX family of computers. The design intent of the VAX was to provide customers with a mini-computer which did not labor under the severe memory constraints which limited earlier minicomputers. Small memory space had proved time and again to be one of the most significant performance limiting factors in the world of mini's. DEC

INTRODUCTION

chose to overcome the memory space limitation with a Virtual Addressing extension (hence the acronym VAX) to their PDP-11 line of computers. The final VAX design was radically different from the older PDP-11 design. The VAX had a new instruction set as well as a different architecture. In order to make the VAX upward compatible with the PDP-11, Digital's designers included a compatibility mode in the VAX. When operated in the compatibility mode, the VAX can execute almost all PDP-11 instructions. However, floating point operations proceed much more slowly than in the VAX/VMS native mode [DEC#10, p 1-2].

DEC provided a new operating system to take advantage of the VAX architecture. This operating system, called VMS (for Virtual Memory System) included most of the PDP-11 operating system's features, as well as many new capabilities. This operating system is designed to support many online users while simultaneously processing batch jobs. The VAX's effectiveness with this mixed workload is determined by many factors, including:

The amount of real memory available.

Memory management policies.

The amount, distribution, and use patterns of disk space.

The kinds of work being done (e.g. compilations,
file editing sessions, statistical
reductions, file maintenance)

INTRODUCTION

The relative mix of on-line and batch jobs at any given time.

1.3 PROBLEM DEFINITION

Unfortunately, at the time of problem conception, DEC provided very few tools which installation managers could use to investigate the performance of their computers. This lack of tools means that it is virtually impossible for a manager to:

- o assess the impact of different batch/on-line mixes;
- o assess the impact of different numbers of simultaneous on-line users;
- o measure the effect of different resource configurations on system productivity.

Currently, there are only four performance measurement tools available from DEC. The first, an online program called DISPLAY [DEC#7, pp 38-39], presents information about how the system has performed during some recent time interval (e.g. the last ten seconds). Regrettably, DISPLAY does not have a logging facility. Consequently, it cannot be easily used for long-term trend analyses.

The second tool provided by DEC is a set of system log

INTRODUCTION

files [DEC#7, pp 38-39] [DEC#9, pp 203-209]. These logs contain information about some system activities, including system and component errors, operator activities, and event terminations (job completions, terminal logoffs). DEC has also provided a facility whereby privileged users can add (programmatically) information to these logs.

DEC's third and fourth tools (programs called ACC1 and SY1) provide a limited capability to analyze the contents of accounting and error log files. Unfortunately, these programs are incapable of dealing with many of the kinds of records which may be on that file. In particular, a log file which has been augmented with user entries could not be analyzed.

Because these tools are so limited, system managers have almost no means of gathering useful statistics about the operation of their computer system. Since, as Yourdon points out [YOU72, 150], performance statistics are a most important source of feedback for future revisions to a system, it must be inferred that VAX managers do not have the tools (other than intuition) with which to fine-tune their computer system.

1.4 SCOPE

This research effort is an attempt to define and develop computer performance measurement tools for VAX-11/780

INTRODUCTION

computers. Since this is a new effort, the analysis and design of several tools will be detailed. Equally important, methods for interfacing these tools with the operating system so that they impose a minimum of overhead on the computer will be specified. As time permits, some of the tools will be developed and installed in the VAX/VMS system at the Air Force Institute of Technology. The goals and limitations expressed in the following paragraphs establish in more detail work being done, and what boundaries are placed on the range of that work.

1.4.1 Goals - As mentioned earlier, the goal of this project is to provide a suite of tools (PERformance Monitoring TOOLS, or PERMTOOL) for VAX system managers. These tools must be integrated with the operating system in an efficient manner so the computer doesn't spend most of its time doing the work of the measurement tools. Even so, PERMTOOL can be expected to introduce unavoidable degradation into the operation of the system. Consequently, methods must be developed to quantify and isolate these artifacts. Finally, the results of this project must include suitable documentation for the system manager, who must not be expected to "discover" how to use the tools or how to interpret their output.

INTRODUCTION

1.4.2 Limitations - Because very little documented work has been done in the area of VAX performance monitoring, more effort must be spent defining the tools and the context in which they work than in developing individual tools. This approach must be followed to establish a firm foundation for later work. Furthermore, no effort will be made to address network (DECNET) operations or their measurement (see DEC#5, DEC#1, DEC#7 for more details about DECNET). Neither will any time be spent developing firm-ware monitors for installation in devices such as disk controllers, even though such an approach can produce very informative data. Where a choice must be made (because of time constraints, for example), data collection tools will receive priority over data reduction or presentation tools. This limitation is imposed because the value of tools in the latter groups, without any data to process, seems very limited. In fact, such tools today can do little more than the existing program ACC1, which summarizes system log tapes.

1.4.3 Methods -

The investigation will procede in a fashion of stepwise refinement, always working from the most general level of concepts through successive levels until the desired degree of detail has been reached. This approach will be used to:

<> Determine information needs. Different elements of information will be needed to

INTRODUCTION

answer different categories of questions. The requisite information must be identified and characterized. For example, if the purpose of a specific tool is to provide information about interactive users and the kinds of work they do, what information is needed? What can be used to characterize that group of users?

<> Establish performance measurement requirements.

Generally, define what tools will be of use to system managers as they monitor and (presumably) fine-tune their computer systems. These tools will first be defined in very general terms, such as memory monitors, and then will be redefined several times to include as many known requirements as possible.

<> Define Data manipulations. Data elements must be identified which can provide the required information. Once data elements have been associated with an intended use, specific transformations to put the data into a useful form must be defined.

<> Define data gathering methods. Establish points in the operating system where the

INTRODUCTION

required data can be gathered; consider what events must occur to make a data element valid.

- <> Design program modules. Develop software design specifications for data capture and data manipulation modules.
- <> Code program modules. Use top-down methods and structured programming techniques.
- <> Test program modules. Use methods as appropriate for specific modules, including exhaustive testing, path analysis, black box testing, and boundary value testing/analysis.

1.5 THESIS ORGANIZATION

This thesis is organized into eleven chapters which describe the analysis, design, and development of the PERMTOOL performance measuring system for VAX computers. In addition, several appendices are included which provided extended information about several topics which are discussed in the body of the paper. Chapter two provides a definition of the functional requirements for the PERMTOOL. Chapter three describes how the PERMTOOL system is composed from its component modules. Chapters four through eight present discussions of the design developed to

INTRODUCTION

support each major subsystem of the PERMTOOL:

- Executive Control subsystem
- User Interface subsystem
- Data Collection subsystem
- Data Reduction subsystem
- Data Presentation subsystem

Chapter nine presents a discussion of resource utilization by PERMTOOL, and relates this utilization to the accuracy of performance studies undertaken with PERMTOOL. Chapter ten describes the development and implementation of PERMTOOL. Finally, Chapter eleven summarizes the project, draws some conclusions about the approach, results, and timeliness of the effort, and makes several recommendations for future efforts.

CHAPTER 2

REQUIREMENTS ANALYSIS

2.1 INTRODUCTION

In order to create useful Computer Performance Evaluation (CPE) tools, one must first answer the question: How is the tool to be used? Unfortunately, there is no single answer to that question. Each installation manager has a unique equipment configuration and workload to manage as well as specific performance goals to establish. Since the intent of this effort is to develop a general purpose performance measurement tool (PERMTOOL), the tool must be useful in a variety of circumstances. With this in mind, nine general reasons for conducting computer performance measurements are listed in Table 2-1, which is an extract from Borovits [BOR79]. Certainly this list is not complete, since many installation managers might think of additional reasons for measuring the performance of a computer system. This list does touch on most major reasons for undertaking such an effort, and as such will be used in this effort to help focus attention on the scope of the problem.

REQUIREMENTS ANALYSIS

Table 2-1. "Fields of interest" to performance improvement teams [BOR79]

- <> Identify System Costs.
- <> Quantify System Reliability.
- <> Quantify System Response times.
- <> Measure the System Work load.
- <> Facilitate Resource Usage Accounting.
- <> Aid System Operations.
- <> Help Workload Scheduling.
- <> Facilitate Fine-tuning Control.
- <> Provide data for Trouble reports.

In order to effectively address any of these reasons, a system manager must have access to certain information. Some of this information must be gathered manually, from other parts of his/her organization. The annual maintenance costs of the system are a good example of this type data. Other pieces of information, such as CPU idle time, may be obtained directly from the computer. Since the purpose of PERMTOOL is to help gather this latter type of information, the rest of this chapter will identify:

- o Specific data items which can be garnered from the computer and which will help a manager extend his understanding of the fields of interest listed above.
- o The requirements for an interface between the PERMTOOL and its user.
- o The requirements for an interface between the PERMTOOL and the computer system.

REQUIREMENTS ANALYSIS

2.2 PERFORMANCE MEASUREMENTS

2.2.1 Introduction - It is not possible to evaluate a system without data which reflects the performance of that system. Therefore, this section identifies data elements which can be used to measure performance in each of the nine fields of interest listed earlier. In some cases, data must be collected manually. In other cases, data can be collected using the existing VMS accounting system. Finally, there are many data elements which cannot be gathered using existing methods. These data elements must become the primary target of PERMTOOL data collection. Furthermore, it is very conceivable that only specific periods of an operational cycle are of interest to the system manager. Consequently, PERMTOOL must be able to identify such data collection windows, starting data collection at the beginning of a specified window, and stopping collection at some designated later time.

2.2.2 System Costs - Many factors relating to system costs must be manually collected. Some factors which affect system costs may be logged by the computer. For example, if system or component lease/maintenance costs are dictated by system or component availability, then the VAX error log or the operator log may contain very useful information. Since it seems unlikely that any contracting organization would undertake such an arrangement unless it was sure that all

REQUIREMENTS ANALYSIS

required data was available, we will not attempt to extend data collection in this area.

2.2.3 System Reliability - An unreliable system is one which cannot be depended upon to satisfy customer requirements. This may happen for any of several reasons, including system or component hardware or software failure, and overuse of either the system or some of its components. In cases of the former type, measures of reliability must be derived from site trouble logs and repair sheets. Such logs at other installations have included (in the author's experience) the information shown in table 2-2:

Table 2-2. Data elements used to Quantify System Reliability.

Data Element Collected
1. Device identification
2. Date/time of failure
3. Date/time of report to management
4. Date/time of report to maintenance organization
5. Date/time of repair
6. Nature of problem
7. Impact of device failure

In the latter type of error: overuse of the system, measures of component usage can be very important indicators

REQUIREMENTS ANALYSIS

of bottlenecks in the system. Such measures will be discussed in greater detail later in this section.

2.2.4 System Response Times - This measure of system effectiveness is particularly important to the interactive user. In effect, the system response time reflects how long a person must wait after typing a carriage return at the end of a command before the system responds to that command. Table B-2 identifies many data elements which may be of value when establishing the responsiveness of a system. This list (and several of the following) are quite long in order to permit the manager to measure performance (with respect to other aspects of the system) such as workload. It is expected that a real analysis effort would initially specify collection of some data items from this (or other) lists. Based upon analysis of that data, subsequent efforts might dictate the collection of different data elements. In either case, it is hoped that the analyst will attempt to restrict the number of data elements collected. This approach will minimize the impact of the PERMTOOL, itself, on the operation of the system.

2.2.5 System Work Load - Measures of work load in a system reflect how busy the system is: how many interactive users are active, how many batch processes are active, and how components of the system (disks, compilers, editors, etc). are being used. Work load measures are used to support a

REQUIREMENTS ANALYSIS

variety of performance analyses. Typically, analysts will expect to observe a degradation in the performance of the system as the workload increases. Establishing an acceptable level of degradation then becomes, in part, a tradeoff between other factors such as system responsiveness and the total amount of useful work being done by the system. As with other interest areas, many data items may be of value. These data items are shown in table B-3.

2.2.6 Resource Usage Accounting - All VAX computers have a variety of resources, both hardware (tapes and disks, for example) and software (such as compilers) which may be used to satisfy user requirements. Keeping track of how these resources are used can serve several purposes, as listed below:

- o Charging customers for their use of the computer.
- o Establishing resource usage patterns for job scheduling purposes.
- o Establishing resource usage trends for configuration planning purposes.
- o Identifying bottlenecks to improved system performance.

Data items which support resource usage accounting are listed in table B-4.

REQUIREMENTS ANALYSIS

2.2.7 System Operations - The computer system operators and support personnel are charged with sustaining the effective operation of the computer system. In order to do this job properly, they make use of several tools to monitor the hour-by-hour performance of the equipment. Tools provided by DEC simplify the operator's job in several ways. Naturally, the system reports component failures to the operators, as well as requests for operator intervention (such as to mount a tape). In addition, through the use of the utility program DISPLAY, DEC provides a capability for the operator to observe the dynamic performance of the computer. With experience, the operator can use this information to modify various system parameters, effectively fine-tuning the system as it runs. No additional information is required.

2.2.8 Workload Scheduling - Making decisions about what combination of jobs should be processed during a given period of time is called workload scheduling. This scheduling process can have a great impact on the computer, affecting its performance for better or worse, depending upon the quality of the schedule. Many of the data elements listed in the preceding paragraphs can help the system manager produce effective schedules. For example, if it is known that the computer workload follows a week-long cycle, then resource utilization figures from the preceding week(s) may be used as predictors of projected workloads. Workload

REQUIREMENTS ANALYSIS

measurement data items are shown in table B-5.

2.2.9 Fine-tuning Control - Fine tuning of the system can be viewed from at least two perspectives. The first is a short-term perspective which is concerned with balancing the system to daily or hourly fluctuations in the workload. The second perspective is long-term and tries to improve the performance of the system in all predicted workload situations.

In either case, controlling the fine-tuning process can be viewed as a feedback situation wherein the system's current performance is compared to its performance at some earlier point in time. If unacceptable performance degradations are observed, then modifications are made to the system. If this tuning is effective, then the modified system should be performing better than its predecessor. Based upon results of such a comparison or upon observations of the modified system, tuners may then install additional modifications to the system, and the feedback process is repeated.

Short-term tuning can be handled by operations personnel using the utility program DISPLAY. Data necessary to support long-term fine-tuning operations is shown in table B-6.

REQUIREMENTS ANALYSIS

2.2.10 Trouble Reports - Trouble reports are most often used to indicate system problems which require maintenance action to correct. Such maintenance action can be expedited if the repairman has a good idea about the nature of the problem. Consequently, system management personnel should make the trouble report as complete as possible. They have several tools available as aids when they make a report. The principal aids will fall into four categories, based upon the source of the data provided:

- * User reports and complaints
 - * Messages on the operator log
 - * Messages on the system error log
 - * Results of diagnostic program runs

User reports and complaints must be recorded manually, or through the use of a central clearing house such as a GRIPE mailbox. The author is familiar with several installations that use the mailbox method: the user prepares his report at a computer terminal and sends it (by computer) to the complaints manager. In either case, the reports must be reviewed manually and appropriate action taken. Experience has shown that user reports can be a very significant early indication of trouble.

Several categories of messages on the operator log may be useful as indicators of trouble. For example: device

REQUIREMENTS ANALYSIS

status messages, which indicate that a device keeps dropping offline, can be used by maintenance personnel to isolate a problem.

The system error log is used by VMS to record any problems it encounters with various components of the hardware and system software. As an example, a common event on the error log is a disk read error. This log is frequently used by DEC technicians who must isolate and repair a variety of hardware problems. Similarly, the log may be used by performance monitoring personnel to help identify failure trends and areas with chronic weaknesses.

In addition to the logs described above, another source of information is the package of diagnostic programs that DEC personnel use to help trace various problems. As with the error log, it may prove worthwhile to maintain a history of results produced by such programs.

2.2.11 Summary - As the discussion of the previous ten sections indicates, many data items are of (potential) value to a system manager. Although some information must be collected manually, much more can be captured automatically. Such information has been provided in tables B-2 through B-7, according to subject area as shown in table 2-3:

REQUIREMENTS ANALYSIS

Table 2-3. Categories of data listed in Appendix B.

Table nbr	Title
B-2	Data Items for System Response Time
B-3	Data Items for System Workload
B-4	Data Items for Resource Usage Accounting
B-5	Data Items for Workload Scheduling
B-6	Data Items for fine-tuning Control
B-7	Consolidated List of Data Items

With these lists in mind, the specific requirements which must be satisfied in order to insure successful data collection are:

- o Provide a procedure which users may follow to specify the data items to be collected.
- o Allow users to establish the bounds of a data collection window.
- o Record any data collected for future analysis and manipulation.

2.3 PERMTOOL USER INTERFACE REQUIREMENTS

In order to effectively deal with a user, PERMTOOL must be able to determine the user's data collection requirements. To do this, PERMTOOL must have the following capabilities:

- o provide clear instructions to the user
- o determine data items to be collected
- o determine time frame of the data collection effort
- o determine the maximum number of events to be collected (if appropriate)

REQUIREMENTS ANALYSIS

- o establish sampling interval desired (if appropriate)
- o show status of a data collection effort in progress
- o stop an active data collection process
- o initiate data reduction or presentation activities
- o permit use of other VMS sub-systems without forcing the user to leave PERMTOOL

Finally, this interface must deal with the user in an easy to follow manner. In other words, it must be a user-friendly interface.

2.4 PERMTOOL SYSTEM INTERFACE REQUIREMENTS

In order to achieve the goals established by a user, PERMTOOL must establish several links to the rest of the operating system. First, it must set up mechanisms to capture the data specified by the user. This may involve either modifying the operating system or enabling certain types of accounting data collection, or both. Second, it must provide a means of recording any data collected. Finally, the PERMTOOL system interface must be able to remove any signs of its presence when a data collection effort is complete.

REQUIREMENTS ANALYSIS

2.5 SUMMARY

A useful performance monitoring tool must provide capabilities to satisfy the diverse needs of many different installation managers. In general, these needs include capturing, manipulating, and reporting information about the performance of a VAX computer system as presented in this chapter. Specific requirements which will guide the development of the PERMTOOL are shown in Table 2-4. Throughout the rest of this document, references to specific requirements will be with regard to the requirement numbers shown in Table 2-4.

REQUIREMENTS ANALYSIS

Table 2-4. PERMTOOL Requirements summary.

Req't nbr	Description
1	Establish user requirements
1-1	Determine data items to be collected.
1-2	Determine time frame of the data collection effort.
1-3	Determine the data reductions required by the user.
1-4	Establish the sampling interval desired.
1-5	Determine reporting methods to be used with the data.
1-6	Maintain a friendly user-interface.
2	Initiate data collection activities.
3	Log collected data items on a permanent medium for later use.
4	Provide facility to manipulate data items according to standard statistical techniques.
5	Provide capability to produce reports which reflect either the data collected, statistical reductions of that data, or both.
6	Provide clear instructions to the user
7	Show the status of data collection efforts which are in progress.
8	Stop any data collection process which is active.
9	Initiate data reduction or presentation activities.
10	Initiate VMS utility activities.

CHAPTER 3

SYSTEM DESIGN

3.1 INTRODUCTION

In order to satisfy the various requirements identified in the previous chapter, PERMTOOL has been designed as a loose structure consisting of several components. The design of each of these components will be discussed in chapters 4 through 6. However, this chapter presents an overview of the PERMTOOL system, and details the methods used to integrate the various components into a system. Alternative configurations were not considered for long because the functional requirements of the system segregate very simply into the following components:

- * Executive Control Module - Initiates and monitors PERMTOOL activities; satisfies requirements 2, 7, 8, 9 and 10, and controls all other PERMTOOL activities.
- * User Interface Module - Handles all interactive communications with users. The design intent of

SYSTEM DESIGN

this module is to satisfy requirements 1-1, 1-2, 1-3, 1-4, 1-5 and 1-6.

- * Data Collection Program - Collects data, as required by the user, about the operational computer system. This module operates to satisfy requirement 3.
- * Data Reduction Program - Designed to meet requirement 4, this program manipulates data elements gathered during an earlier collection effort. Calculates various statistical measures about performance parameters.
- * Data Presentation Program - In order to satisfy requirement 5, this program will present both raw and reduced data in forms such as tables, graphs, and charts.

The integration of these components into a useful software system is discussed in the following section.

3.2 OVERALL SYSTEM DESIGN

The preceding section described the PERMTOOL system as a loose structure of several components. This description was chosen because there is no rigid order or structure which dictates an operational sequence for PERMTOOL's components. Instead, different parts of the system may be simultaneously

SYSTEM DESIGN

active at any given time. For example: the User Interface module may be helping a user at the same time that an independant data collection effort is in progress. Or, data collection may be taking place while data from earlier efforts are being reduced and printed. This is possible because each of the components are designed to be independant programs, whose activity is initiated by the Executive Control Module. By using this approach, PERMTOOL can:

1. Establish the system manager's needs when it is convenient for him/her, and perform the work at another time. Alternatively, the system could only initiate work processes immediately upon receipt of a user command to do so. This latter approach would impose work-time constraints on the user and engender a hostile (or at least unfriendly) attitude towards PERMTOOL on the part of the user. In keeping with requirement 1-6, the former approach will be taken.
2. Collect data when necessary, as dictated by requirements 2 and 3, without requiring the presence of the manager at his terminal. As discussed in the last paragraph, this approach is much more user-friendly. Furthermore, it permits the user to specify, at one time, several data

SYSTEM DESIGN

collection efforts.

3. Perform data reductions and report productions at times when the computer system is least heavily used. Since the data manipulations which may take place here are quite CPU- and I/O-intensive, this permits the system manager to avoid excessive computer use during peak operational periods.
4. Present the status of any other PERMTOOL activities to the user.
5. Provide access to system utilities such as TALK and WATCH. This capability overcomes a serious drawback of other DEC sub-systems, such as the DISPLAY: one must leave the tool to use any other system capabilities.

The major components of PERMTOOL are shown in figure 3-1. Control communications between these components is accomplished by way of VMS system mailboxes, which function as virtual sequential I/O devices that may be read from and written into by different (concurrent) processes.

SYSTEM DESIGN

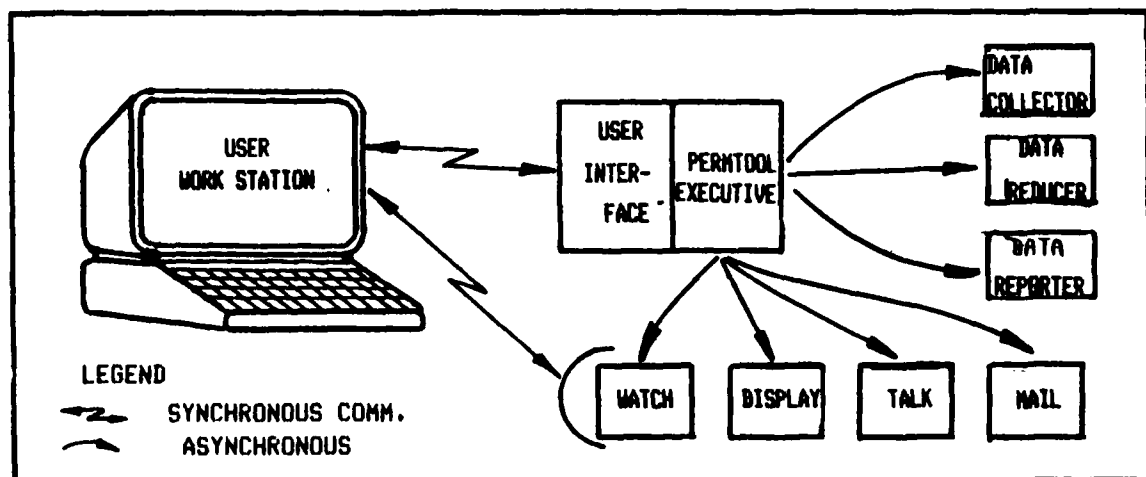


Figure 3-1. Major components and functional Relationships in the PERMTOOL system.

There is no requirement for data sharing or interactive exchanges between modules; however, data files produced by one module may be used by another module. For example, data files produced by the data reduction software may be used by the data presentation program.

3.3 PERMTOOL CAPABILITIES

Each of the components of PERMTOOL operate to provide one or another service to the user. The capabilities to be provided by PERMTOOL were mentioned briefly above, and are presented now in more detail.

3.3.1 System Utility Functions - These functions are provided as a service to the user, and are currently limited to four utilities. This is a step towards satisfying requirements 1-6 and 10. First is the TALK utility. It enables one user to "talk" to other users with the computer

SYSTEM DESIGN

system as an "intercom". In large operations such as the ASD computer center, management personnel are physically isolated from each other. TALK permits them to discuss issues (such as system performance) directly from their computer work stations.

The second utility is the VAX/VMS mail facility. This permits users to send and receive messages from other users. The operating system stores any messages until the receiver wants to read them. This facility is valuable in many ways, including sending memoranda to one's self.

The final two utilities are both on-line performance monitoring utilities: WATCH and DISPLAY. WATCH, which is not supported by DEC, provides a list of all jobs/processes currently using any part of the computer. Included in this list are a dozen data elements relating to each job, such as owner name, job name, execution module name, current state, and accumulated CPU time. This can provide information to operators/system management for short-term fine-tuning or workload planning.

DISPLAY is supported by DEC (See reference [DEC07]) and provides most of the information available from WATCH. In addition, it provides many additional kinds of information, on-line, about the current state and recent historical context of the computer. Although DISPLAY makes much more information available, it is more complicated to use.

SYSTEM DESIGN

Consequently, we expect WATCH will be requested whenever possible.

When a user asks for one of these four utilities, PERMTOOL creates a running process consisting of the specified utility, relinquishes control of the terminal to the utility, and puts itself into hibernation. When the user ends his/her session with the utility, the operating system revives PERMTOOL, which then returns the user to the first dialogue frame, MENU01.

3.3.2 Data Collection Functions - The data collection function is of central importance to the PERMTOOL system. It makes available several kinds of data, collected during the normal operation of the system, for subsequent analyses. This data collection has been grouped into four categories, indicated by the four columns in tables B-2 through B-7, based upon the nature of the data being gathered. These categories include memory, CPU, job/process, and peripheral device data collections. Each of these categories makes available many possible data elements, collectively over one hundred. The user, through the medium of a computer terminal, must select both the category desired and the data elements within that category. Although he/she may select as many categories and data elements as s/he wants, s/he must always limit his/her selections as much as practical. Unrestrained selections will almost always result in

SYSTEM DESIGN

cumbersome data collection processes. This, in turn, will introduce a strong bias into any statistical results derived from the data collected under such conditions. The consequence of such a bias would ultimately be analyses of questionable validity.

3.3.3 Data Reduction Functions - The data reduction functions are intended to provide statistical aggregations of the raw data collected by the data collection module. These will include counting functions, summing and averaging functions, and more sophisticated statistical manipulations as provided by SPSS, the Statistical Package for the Social Sciences. The results of these manipulations are stored, and may be used by either the report production package, or by user-developed application programs. It is expected that both this and the data presentation functions described in the next paragraph will continue to grow, as users determine additional ways of reducing and presenting data.

3.3.4 Data Presentation Functions - Data presentation is a critical part of any study; without effective methods for presenting information, much of the value of the study may be lost through lack of understanding. It is the task of the data presentation functions to provide the user with a variety of graphing tools to facilitate effective presentation of information. These tools operate on data files prepared by the data collection routines, the data

SYSTEM DESIGN

reduction routines, or by other programs as developed by the user. As mentioned above, these functions will probably be extended over and over, as user needs and system configurations change.

3.3.5 Summary - This chapter has outlined the overall system design for PERMTOOL, and has discussed each of the major functional elements of the tool, including:

- o Executive Control Module
- o User Interface Module
- o Data Collection program
- o Data Reduction program
- o Data Presentation program

These components function to satisfy specific requirements identified in Chapter 2. In the following four chapters, the design of each of these components will be discussed in greater detail.

CHAPTER 4

EXECUTIVE CONTROL MODULE DESIGN

4.1 INTRODUCTION

This chapter provides a detailed design description of the PERMTOOL Executive Control Module (ECM). This module is designed to provide a capability to initiate all detached processes which run under PERMTOOL. This includes all functions specified in requirements 9 and 10 [Table 2-4]. The ECM has also been designed to satisfy requirement 8 by permitting a user to stop a process which was started earlier. Furthermore, the ECM coordinates the activities of all other PERMTOOL components. In order to do this, it must determine what a user wants done and take steps to satisfy the user's requirement.

As mentioned before, a user may request PERMTOOL to initiate a data collection, perform data reductions on previously collected data, produce reports from data collected earlier, or provide support from any of several VMS utility programs. When a request is received, the ECM

EXECUTIVE CONTROL MODULE DESIGN

must insure that it is not redundant or impractical. For example, multiple simultaneous data collection efforts are considered impractical because they would impose an excessive load on the computer system. Such a load would, in turn, significantly affect many computer performance parameters and adversely affect the accuracy of any performance statistics generated. Consequently, the ECM must not permit more than one data collection process to operate during the same time frame.

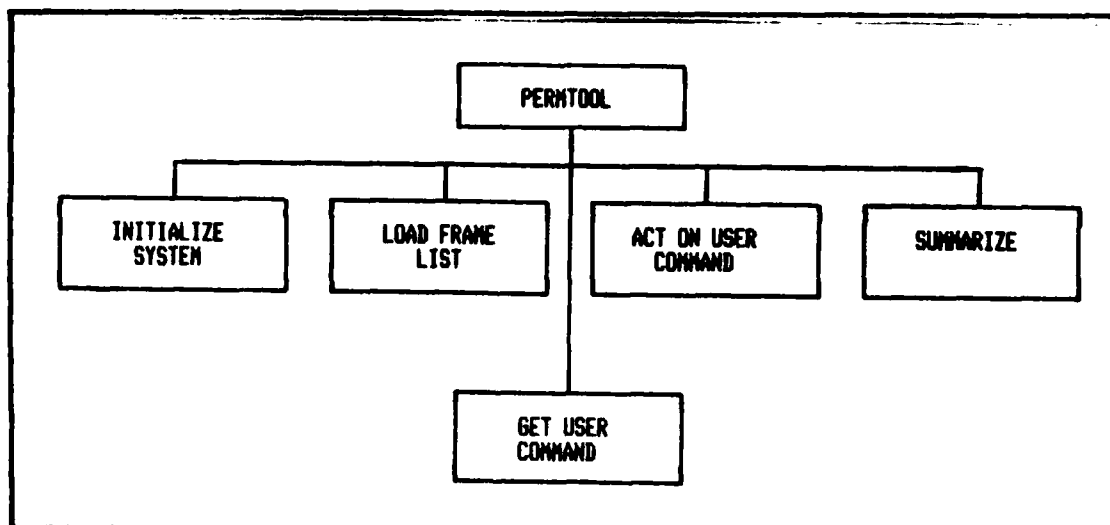


Figure 4-1. Function Diagram of the PERMTOOL Executive and its subordinate functions.

4.2 ORDER OF OPERATIONS

When the PERMTOOL system is invoked by a user, the Executive Control Module initiates several actions. These activities include all necessary housekeeping, all user support activities, all system monitoring activities, and ultimately, all system shutdown activities. The general

EXECUTIVE CONTROL MODULE DESIGN

relationship of these functions is depicted in the function diagram of figure 4-1. Figure 4-2 reflects the logical order in which these operations are performed.

1. Initialize all system variables
2. Load the dialogue frame list
3. While the user is not finished
 - a. Get a user command from the User Interface Module
 - b. Perform the desired operation
 - c. Update PERMTOOL resource use statistics
4. Display summary statistics

Figure 4-2. PERMTOOL executive order of operations.

4.3 EXECUTIVE CONTROL DATA STRUCTURES

Several data structures are of significant importance to the PERMTOOL ECM because of the central role they play in the operation of the system or its components. These data structures, and their use are described below:

- * **FRAME_FILE** - The library file which contains the complete set of dialogue frames. This file is designed to be maintained by the PERMTOOL utility program FRAMEDIT. However, the library is compatible with the VAX/VMS EDT text editor. Consequently, narratives in the file can be refined or clarified without using FRAMEDIT. The ECM loads the framelist (see next entry) from this file.

EXECUTIVE CONTROL MODULE DESIGN

- * **FRAME_LIST** - A memory resident copy of all dialogue frames which is loaded by the ECM when the PERMTOOL system is invoked. This list provides the text of all computer responses to the user, as well as control information which indicates where and how the text is to be displayed, and whether or not the user must reply to the frame being displayed.
- * **SYS_CONFIG_TBL** - This table keeps track of the current PERMTOOL configuration. It is used to prevent overlapping data collections or reductions from occurring. It is loaded when PERMTOOL is invoked.
- * **PERM_IN_MBX** - A VMS mailbox through which PERMTOOL may receive messages from asynchronous processes. It is used to keep PERMTOOL abreast of all changes to Data Collection activity.
- * **USER_COMMAND** - Contains flags to represent the current user command, as well as any options associated with that command.

EXECUTIVE CONTROL MODULE DESIGN

4.4 EXECUTIVE CONTROL SOFTWARE STRUCTURES

The ECM is part of a single compilation unit which consists of a main routine and several dozen subroutines. The subroutines include all those used by the functions shown in figure 4-1. Together, these routines and the ECM constitute the portion of the PERMTOOL system which must be available whenever a user asks for support by running PERMTOOL.

Most of the routines in the ECM will be discussed in later chapters. However, several housekeeping routines function solely to support the Executive Control Module, and are discussed in this chapter. These routines include:

- System calibration routine
- Dialogue frame loading routine
- DEC mail processing routine
- Resource utilization monitor

The system calibration routine determines the characteristics of the current operating system. This includes physical parameters about the computer (such as memory size and available devices) and the identification of any independant PERMTOOL processes currently running or scheduled to be run. This information is needed so that the ECM can prevent redundant or overlapping operations.

EXECUTIVE CONTROL MODULE DESIGN

The dialogue frame loading routine has the job of locating the dialogue frame library and loading it into the memory-resident frame list. In addition, it builds a frame dictionary from the frame list. This dictionary contains the identification of each frame, and a pointer to its actual location in the list. Use of this dictionary speeds the task of finding frames to display.

Both before and during the operation of the ECM, other, independant, PERMTOOL processes may have need to send messages to the ECM. For example, the ECM needs to know when a data collection process finishes. Otherwise, it would not be able to start another data collection process, since (normally) only one such process is permitted at a time. As mentioned in the preceding chapter, this interprocess communication is accomplished through the use of system mailboxes. ECM will use a mail processing routine to periodically examine its mailbox. Whenever a message is found, this routine will update the system configuration table to reflect the change indicated by the message.

When PERMTOOL runs, it uses computer resources, just as all other jobs do. For many kinds of analyses, a researcher must be able to identify any bias imposed upon the system by the monitor software. To help him do this, the ECM (and every other independantly operating module of PERMTOOL) keeps track of the computer resources it uses. A resource

EXECUTIVE CONTROL MODULE DESIGN

utilization monitor will perform this job, and ultimately report how much resources were used during the execution of the process.

4.5 SUMMARY

This chapter described the functions and design of the PERMTOOL Executive Control Module and its component data structures and software structures. These structures are documented further, in the form of data flow diagrams, in appendices D, E, and F.

CHAPTER 5

USER INTERFACE SOFTWARE DESIGN

5.1 INTRODUCTION

This chapter describes the design of the User Interface Module (UI), and follows the format of the previous chapter: it will present a general discussion of the UI, followed by a description of the data structures and software structures which are part of the design.

In order satisfy requirements 1-1 through 1-6 and 6 [Table 2-4], and to enhance the design of the PERMTOOL through the use of functionally independant modules, the User-Interface (UI) module is responsible for all PERMTOOL interactive dialogues with the user. Basically, the UI asks the user what he/she wants to do, checks the legitimacy of the user's response, and passes valid requests back to the executive for action. The method chosen for this is a conventional computer statement/user response sequence. In this paper, these will be called a sequence of dialogue frames.

USER INTERFACE SOFTWARE DESIGN

5.2 FRAME PROCESSING

Each frame in the dialogue asks a specific question for which there are but a few valid responses (and a variety of invalid responses, of course). Each response leads to one (and only one) associated follow-on frame. Invalid responses lead to an error message frame, followed by the original frame statement. Valid responses lead either to the next level of detail in the dialogue, or to an action initiation return to the executive (NOTE: except when the user has finished a session, the executive will ultimately return to the first frame of the dialogue). This process is illustrated in figure 5-1.

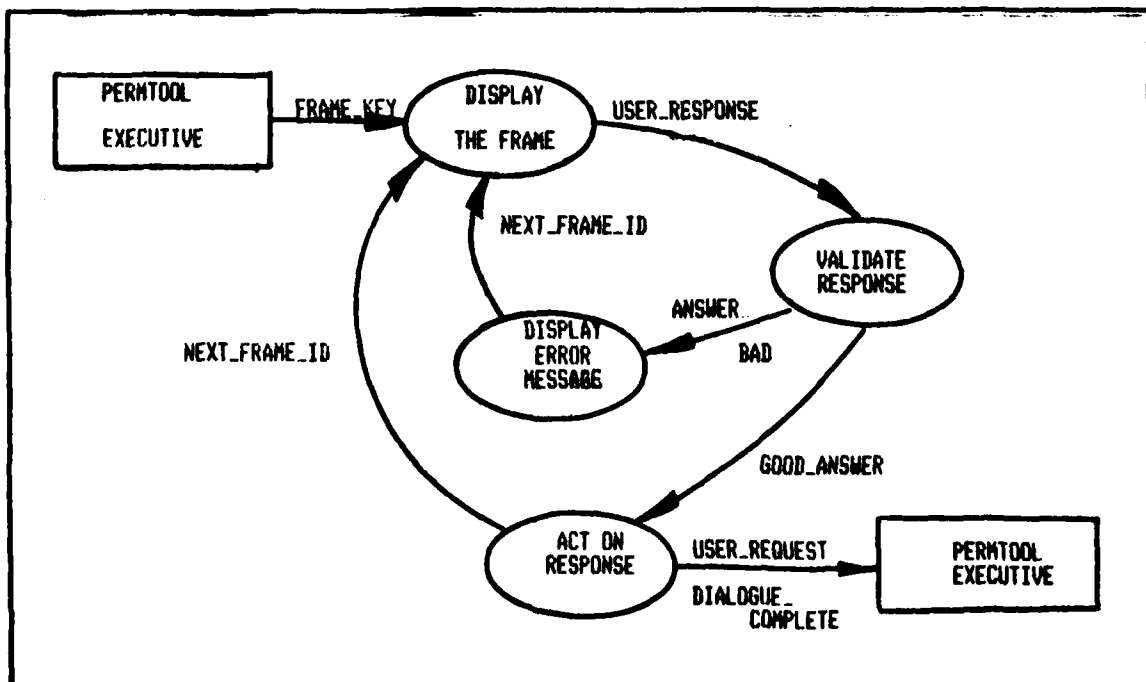


Figure 5-1. The Dialogue/Response process.

USER INTERFACE SOFTWARE DESIGN

5.2.1 Frame Control Structure - Associated with the dialogue is a control structure called the <frame_item_list>. Each item in this list represents the control structure for a single dialogue frame. One item is called a <frame_item> and consists of three principle components and several sub-components, as shown in figure 5-2 and described below.

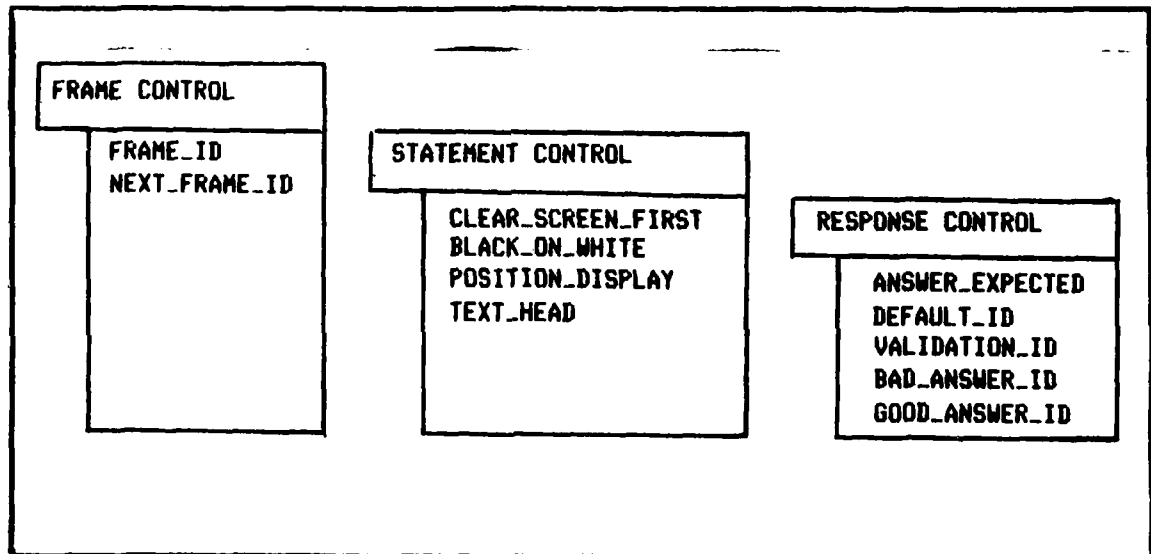


Figure 5-2. Components of a Frame_Item.

This structure was chosen because it segregates data elements into groups according to their function, and reflects the functions in the normal order for their processing. Other structures which contain the same information are possible. For example, another structure which was considered contained a list of good_answer_id's and a list of specific correct responses to associate with each good_answer_id. This would reduce the requirements for logic embedded in response processing routines, since

USER INTERFACE SOFTWARE DESIGN

intermediate routing decisions would effectively be made by the frame control structure. However, this could only be done at the expense of a more complicated data structure. Another alternative involved hard-coding each frame into a display-and-process routine. This method was rejected because it would complicate the program logic and increase the size of the program. Since the first method satisfies the initial set of requirements, and since it was the simpler structure, it was selected.

5.2.1.1 Dialogue Frame Control - The frame control component contains those items which are needed to locate a specific item in the list. The `<frame_id>` is a unique label given to each frame. the `<next_frame_id>` is a pointer to the next frame item in the list.

5.2.1.2 Dialogue Statement Control - The statement control portion of a `frame_item` contains the information required to properly display a computer dialogue statement. The elements of this part dictate whether or not the screen should be cleared before displaying the statement; they determine if the text should be displayed as a normal reverse video image (that is, dark letters against a light background); they determine if the text is to be placed in a window on the terminal screen and where such a window should be located; finally, the statement control elements specify the actual text to be displayed on the screen.

USER INTERFACE SOFTWARE DESIGN

5.2.1.3 Dialogue Response Control - The response control portion of the frame_item determines what the computer should do with a user's response. This portion establishes whether or not any user response to the frame is expected. For example, an error message often has no associated user response. Normally, the error message will be displayed, and then control will pass to the frame_item which originally received the invalid response. In this case, the <answer_expected> flag will be false, and the next frame to be processed is specified by the <default_id> frame key.

All remaining items are effective only when the user is expected to respond to the frame. Each of the remaining items is a symbolic vector to a processing routine. One, <validation_id>, names the boolean function routine which must validate the user's response. <bad_answer_id> specifies the routine which processes an invalid user response, and <good_answer_id> dictates the routine to process a valid answer. Both good and bad answer routines are expected to specify the dialogue frame which follows.

5.2.2 Frame Generation And Storage - When the PERMTOOL development is complete, the system will use several hundred dialogue frames. These frames are maintained in a disk library and loaded into the PERMTOOL frame list whenever the system is invoked. Maintenance of the frame library is handled by a utility program called FRAMEDIT. This routine

USER INTERFACE SOFTWARE DESIGN

permits a user to interactively create, change, or delete items from the frame library. Configuration control of the frames must be managed by the system manager. However, a second utility program (FRAMEFIG) has been defined to facilitate this management task. FRAMEFIG provides a formatted list of the frame library and a cross-reference of all processing routines specified in the various frames. Such a listing is shown in Appendix C.

5.2.3 User Dialogues. - As already mentioned, the User Interface Module is the only way a user can give directions to PERMTOOL. Consequently, the UI dialogue must permit the user to completely specify his requirements. This is achieved by frame sequences which gather the user specifications one step at a time, beginning with the most general information: all possible frame sequences begin with the frame MENU01 which is shown in figure 5-3.

USER INTERFACE SOFTWARE DESIGN

```
frame name: MENU01

clear screen - NO          reverse video - NO
user answer required - YES

answer processors:  validation - VMEN01
                   bad answer  - BMEN01
                   good answer - GMEN01
                   default next - MENU01

frame positioning required - NO
frame position  - left column - 0
                  width       - 0
                  first line  - 0

FRAME LAYOUT

      1      2      3      4      5
.....5.....0.....5.....0.....5.....0.....5.....0

PLEASE SELECT AN OPTION FROM THE FOLLOWING LIST:

ANALYZE      HELP      STOP
COLLECT      MAIL      TALK
DISPLAY      SHOW      WATCH
EXIT

YOUR CHOICE:
```

Figure 5-3. Control data and layout of frame MENU01.

From this starting point, the dialogue diverges according to the responses of the user. For example, if the user selects collect, then he/she is next asked to identify the category of information required. This process continues until the UI has a complete request set which can be returned to the executive for action.

USER INTERFACE SOFTWARE DESIGN

5.3 SOFTWARE DATA STRUCTURES

The User Interface module uses three different groups of data structures to support specific functional requirements of the UI. These groups were chosen to help isolate different functional processes and to insure that data was not used in a context other than the one for which it was intended. All three groups are described in the following paragraphs. Their structure is specified in Appendix D.

5.3.1 Frame Use Data Structures - Frame use data structures include those which permit off-line storage of dialogue frames and on-line use of the frames. These structures, which are detailed in Appendices C and D, and discussed in some detail in the following paragraphs, are of singular importance to User/PERMTOOL dialogues. In addition, several variables are used to keep track of the last frame processed, the current frame, and the next frame selected for processing.

5.3.2 User Response Interpretation Structures - The principal data structure used to interpret a user response is called <user_says>, and consists of three parts. The first part contains the last line typed by the user, and is used as the starting point for command interpretation. The second part contains one word taken from the user command line. This is used for the current level of command decoding (when a command with options is given) and is

USER INTERFACE SOFTWARE DESIGN

updated as the user command is parsed from left to right. The final part of <user_says> is an integer which represents the command-code which will be returned to the executive. This integer represents the major function requested, as well as some of the options associated with that function. When a new dialogue is started, the value of the integer is set to zero, and then augmented by coded values associated with the user responses to the dialogue. This data element will be used by the executive to select the action taken by PERMTOOL.

5.3.3 User Option Storage Structures - As already mentioned, part of the <user_says> data structure contains an encoded value representing the major function requested. However, in several cases, many more parameters are needed than can be encoded into a single integer. Consequently, a control list has been adopted to pass additional (often optional) parameters. For example, if the user requested data collection, s/he may want ten different data items to be collected for one hour, starting at 0900 today. The control list, in this case, will contain the start time and duration, as well as a list of true/false items which indicate which specific data items are desired by the user. This list will be used by the execution module: that module which actually provides the service requested by the user.

USER INTERFACE SOFTWARE DESIGN

5.4 SOFTWARE FUNCTIONAL ORGANIZATION

5.4.1 Operational Control - Operational control of the User Interface is maintained by a looping process which continually displays selected dialogue frames until a valid command has been completely specified. The logic of this process is shown in figure 5-4.

1. While a user command is not complete do:
 - a. Display the next frame selected.
 - b. If an answer is needed
then
 - (1) Get the user's answer
 - (2) Validate the answer
 - (3) Take appropriate action *
else
 - (1) Select the default follow_-on as the next frame to process
- * This process will ultimately set a flag to indicate a completed command is available. In addition, this process always selects the next dialogue frame.

Figure 5-4. Structured English description of the User Interface Operational Control routine.

5.4.2 Operational Routines - In the User Interface module, routines are required to validate the answer to each dialogue frame and take action either in the event of an invalid response or when the user response to a frame is valid. By using a stored dialogue approach, it is possible to keep each of these routines very simple since they must only be associated with one possible response. At the same time, it will be possible to use some routines in numerous

USER INTERFACE SOFTWARE DESIGN

instances. For example, validating user-entered date-time groups, which always conform to the DEC standard format, can be done by one routine, rather than several. Each type of operational routine conforms to a standard format, and thus further simplifies the design, coding, and testing of individual routines.

Whenever a user-response to a frame has been received, the UI will call the validation executive which calls, in turn, the validation routine associated with the specific dialogue frame just displayed (remember, the dialogue frame data structure contains a pointer to the proper validation and action routines). This validation routine then validates the user's answer, and passes a true/false indicator back through the validation executive to the UI. The generalized operation of the validation routines is shown in figure 5-5:

1. Receive user response from calling routine.
2. Set validation flag to true
3. If user-response in list of valid responses,
then
 - a. Set command-code to associated valueelse
 - a. Set command-code to invalid value
 - b. Set validation flag to false

Figure 5-5. General Operation of validation routines.

Whenever the designated validation routine flags a user response as invalid, the UI executive will call the bad action executive. This routine determines which specific

USER INTERFACE SOFTWARE DESIGN

bad action routine is appropriate (again, based upon indicators in the dialogue frame data structure) and passes control to that routine for action. The general operation of bad action routines is shown in figure 5-6:

1. If a bad action routine exists for this frame then call that routine to:
 - a. Take frame specific action
 - b. Select proper error message display frameelse
 - a. Generate unknown system error message
 - b. Reset PERMTOOL to MENU01
2. Return to User Interface to display the frame

Figure 5-6. Operation of bad-action-routines.

Ultimately, of course, the user is expected to type a correct response to a frame. When he does, the User Interface executive will call the good action executive. As can be seen from figure 5-7, this routine works in a fashion

1. If a good action routine exists for this frame then call that routine to:
 - a. Recode User response into PERMTOOL system format.
 - b. Save reformatted user-response as part of user-request record
 - c. select next frame to displayelse
 - a. Generate unknown system error message
 - b. Return PERMTOOL to MENU01
2. Return to User Interface to display frame

Figure 5-7. General Operation of good action routines.

USER INTERFACE SOFTWARE DESIGN

very similar to the bad action executive: it decides which specific action routine should be used, and then calls that routine. This may complete a user command sequence, in which case the User Interface will return control to the PERMTOOL executive so that the user request may be executed. Alternatively, the user response may take him to another frame of dialogue, in order to gather more information about his request.

5.4.3 Support Routines - The operational routines listed above frequently will require help from generalized support routines. These routines fulfill a variety of functions which can be grouped into three distinct categories:

- * Screen Support Routines
- * Data Conversion Routines
- * Frame Support Routines

The screen support routines provide a repertoire of functions which includes screen control operations such as clearing the screen, and cursor manipulation operations which permit calling routines to move the cursor to any location on the screen, as well as remember where the cursor originated. The routines which are designed to satisfy these ancillary functions are listed below:

- o CLEAR_SCREEN - This routine permits the

USER INTERFACE SOFTWARE DESIGN

calling program to clear the user's terminal screen. As an adjunct to this operation, the cursor is left in the upper left-hand corner of the screen.

- o SAVE_CURSOR - When this routine is called, it instructs the terminal to save the current position of the cursor.
- o MOVE_CURSOR - This routine uses X- and Y-coordinates provided by the calling program as a destination specification, and moves the cursor to that position.
- o RESTORE_CURSOR - This routine directs the terminal to move the cursor back to the last 'remembered' position. If no position has been saved, it will move the cursor to the home (upper-left) position.
- o HIGHLIGHT - this routine, and its companion NOHIGHLIGHT, were provided by Major G. Gembarowski, and function to place the terminal in either reverse or normal video mode. In reverse mode, selected by HIGHLIGHT, letters on the terminal screen appear dark against a light background. NOHIGHLIGHT complements this to light letters on a dark

USER INTERFACE SOFTWARE DESIGN

background.

Data conversion routines are needed to convert user-typed input character strings to other forms. Procedures are available to convert date information to DEC standard format (four-word binary), and numeric strings to corresponding integer values. This last capability is essential since Pascal, the language to be used for initial implementation, will abort program execution if it encounters a non-numeric digit during such a conversion. Further functions include lexical functions to skip over blanks, find a word, and check for a "yes" or "no" word in the user's response. The capability to convert a string from lower-case to upper-case letters is also available. Each of these functions is described further in the following list:

- o SKIP_BLANKS - This function is designed to scan a one line text buffer which contains the user's last response. It passes all blank characters, beginning at the position specified by the calling program, and returns a pointer which indicates the first non-blank character.
- o UP_CASE_SHIFT - This routine will translate an 80-character line to upper case letters. Any

USER INTERFACE SOFTWARE DESIGN

character other than a letter is left in its original state.

- o CHECK_YES_NO - This routine tests the current word in the input buffer to determine if it is either the word "YES" or the word "NO". If one of these conditions is satisfied, it returns a boolean value of true.
- o GET_WORD - This function finds and returns to the calling routine the next 'word' in the input string. A word in this context is defined as follows:

```
<word> ::= <word_start> |  
          <word_start><word_end>
```

```
<word_start> ::= A | B | C | D | E | F | G |  
                  H | I | J | K | L | M | N |  
                  O | P | Q | R | S | T | U |  
                  V | W | X | Y | Z | 0 | 1 |  
                  2 | 3 | 4 | 5 | 6 | 7 | 8 |  
                  9 | 0 |
```

```
<word_end> ::= <word_start> | : |  
               <word_start><word_end> |  
               : <word_end>
```

USER INTERFACE SOFTWARE DESIGN

- o CHECK_DATE - This function is provided so that other functions can easily validate that a user-entered date/time group is in the proper format, which consists of a string:

DD-MMM-YYYY HH:MM:SS.NN

Where DD is the day of the month, MMM is the first three letters of the month name, YYYY is a year. These date fields are optional. However, if one is omitted, they must all be omitted. HH is an hour between 00 and 24, MM is minutes within the hour, SS is seconds within the minute, and NN is hundredths of a second.

- o CHARS_TO_NBR - This function will convert a numeric user input string to its corresponding integer value. Any input string larger than ten digits will result in a system error being generated by the routine.

Finally, several functions are defined to facilitate manipulating dialogue frames. This facility provides a capability to locate a specific frame data structure, to position and display a frame on the terminal, to find and display a frame on the terminal,

USER INTERFACE SOFTWARE DESIGN

to display a sequence of frames, or to get a user's response to a frame. In addition, two functions are defined for use during testing. These functions will provide a capability to dump the frame list and the associated frame dictionary.

- * `FRAME_LOOK_UP` - Determines the pointer value needed to locate a specific dialogue frame, given the six-character name of the frame. This is accomplished by searching the frame dictionary until a match is found on the frame name.
- * `DISPLAY_FRAME` - Will position and display a frame on the user's terminal, in either reverse- or normal-video, as specified by the control information associated with the frame being displayed. The frame to be displayed is identified by a pointer to the frame data.
- * `DISPLAY_LABELED_FRAME` - Combines the functions of the preceding two routines.
- * `DISPLAY_FRAME_LIST` - Displays a sequence of frames on the screen, with no delay between frames. This is useful when sending error messages, then clearing part of the screen, before displaying a statement/question frame.

USER INTERFACE SOFTWARE DESIGN

5.5 SUMMARY

The User Interface module (UI) is the single component of the PERMTOOL system which communicates with a user. As such, the UI is responsible for determining all user requirements, as specified in Chapter 3. In addition, the UI must take appropriate informative action whenever the user makes a mistake or needs help. In order to accomplish these functions, the UI deals with the user by means of a dialogue. The UI's side of the dialogue is stored in a series of dialogue frame data structures. These frames contain information about how to display the frame, and how to process the user's response to the frame. This processing ultimately leads either to another frame, or back to the PERMTOOL executive for action.

The data structures used by the UI are defined in Appendix D, and fall into two categories. One category supports the actual dialogue process, including frame manipulation and user request storage. The second category supports communication with the PERMTOOL executive to the extent needed to specify a complete user request for action.

In order to do its job, the UI operates in a loop which displays a frame, accepts and validates a user response, and selects the next frame for processing.

USER INTERFACE SOFTWARE DESIGN

This loop continues until the UI has completed a valid user request, at which point, the request will be passed back to the PERMTOOL executive for action. Note that the PERMTOOL executive will always return to the UI after acting on the user's command, unless the command was 'EXIT', which indicates that the user has no more work to do with the PERMTOOL.

CHAPTER 6

DATA COLLECTION SOFTWARE DESIGN

6.1 INTRODUCTION

Perhaps the most important aspect of any performance measuring tool is its ability to efficiently collect all data required for a specific analysis. In the PERMTOOL, data collection is mandated by requirement 3 of chapter 2 [Table 2-4], and is collected under the direction of the PERMTOOL executive software. This software determines (through the User Interface Module) what data is to be collected, when, and for how long. The executive uses this information to initiate execution of the appropriate data collection process. This chapter discusses the data collection processor and describes both the data structures and software structure needed to support this function.

The overall process of data collection can be viewed as a period of waiting, followed by the events shown in figure 6-1, and ending in another period of waiting.

DATA COLLECTION SOFTWARE DESIGN

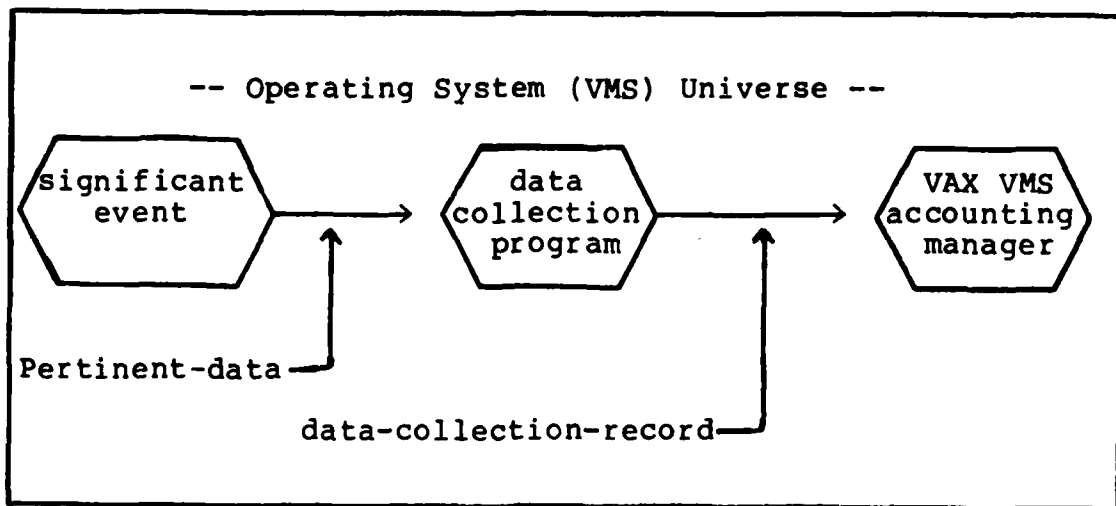


Figure 6-1. Data Flow through the PERMTOOL collection system.

Prior to the initiation of data collection activities, the user of PERMTOOL must specify his data collection interest(s). This specification is translated into a list of significant events and an associated list of data items to collect when an event occurs. The collection process begins when an event of significance occurs. This event might be a page fault or device request or (in the case of several sampling techniques) the termination of a time interval. When such events happen, the list of pertinent data items is used to select data from various sources within the operating system. The data collection program is subdivided into four sections, each of which is designed to process a different kind of data:

- <> Memory data
- <> Processor data
- <> I/O data
- <> Job/process data

DATA COLLECTION SOFTWARE DESIGN

This is done to simplify processing, since each of these groups' data can be gathered from different system internal tables and/or using different collection techniques. The data is saved until enough has been collected to warrant building a VMS accounting system record (several hundred characters, see DEC04, pages C-1 through C-13), which can then be passed to the VMS Accounting system manager. The Accounting system manager will take the actions required to place the record on the accounting log file: ACCOUNTNG.DAT. Finally, the data collection program begins waiting for the next significant event to take place.

6.2 DATA COLLECTION DATA STRUCTURES

6.2.1 VMS System Tables - The VMS system tables (called VMS Internals) contain all the data needed to provide any of the information specified in appendix B. As with most large operating systems, these tables are extensive and difficult to understand. Because of this, and because the author was unable to discover any documentation on the internals, they will be considered outside the scope of this research effort.

6.2.2 PERMTOOL Data Structures - The data collection program uses three major data structures to capture and save data gathered from the VMS operating system:

- <> DATA_COLL_LIST
- <> DATA_CAPTURE_RECORD
- <> DATA_LOG_REC

DATA COLLECTION SOFTWARE DESIGN

Although the information in these structures could be maintained in other ways (for example a single data structure), these structures will be chosen because they enhance functional isolation between the three roles which use these structures. The first of these, DATA_COLL_LIST, is a record received from the PERMTOOL executive which specifies all parameters for the data collection effort. The DATA_CAPTURE_RECORD is used to pass data items to the data logging facility (DATA_LOGGER) of PERMTOOL. The third data structure, DATA_LOG_REC, serves as a buffer and may hold information pertaining to one or more collection events. The structure of these records is defined in Appendix F.

6.3 DATA COLLECTION SOFTWARE STRUCTURES

Several different software structures comprise the data collection module. These structures provide overall control of the data collection process, data and procedural initialization for the process, data collection for each of the four data groups available, data aggregation and logging for all data collected by the process, and finally termination processing when the data collection effort is finished.

The data collection program begins its existence when it is started, as an asynchronous process, by the PERMTOOL executive. Logical control of the program is represented in

DATA COLLECTION SOFTWARE DESIGN

the structure chart of Figure 6-2:

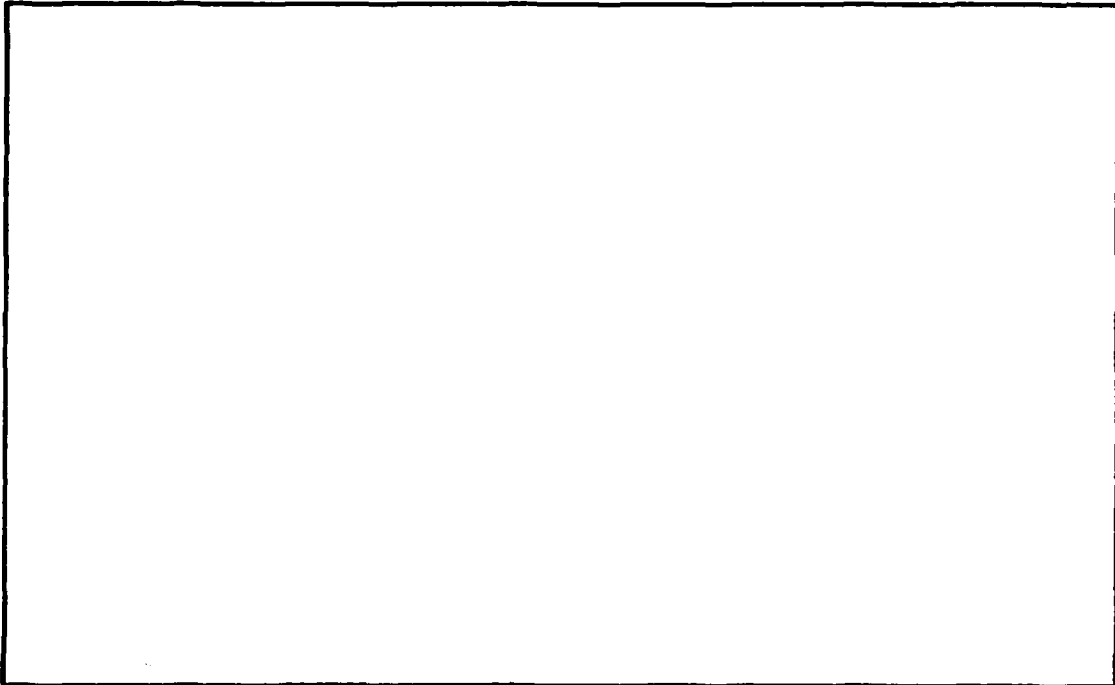


Figure 6-2. Function diagram of major data collection functions.

As its first activity, the data collection program performs a series of housekeeping tasks, under control of the collection initiation routine. This is followed by a period of active data collection. During this time, the data collection program responds to a series of collection events (initially the end of time intervals) by gathering the data specified by the user. After gathering a set of data, the routine will place itself in hibernation until the time specified for the next sample. Periodically, all data collected will be passed to the VMS Accounting system which will log the data onto the system accounting file. Finally, the data collection program performs some termination

DATA COLLECTION SOFTWARE DESIGN

housekeeping and stops running. The design for each of the functions shown in figure 6-2 is discussed in the following sections. Structure charts for the complete data collection process are shown in Appendix G.

6.3.1 Collection Initiation Routine - This routine completes several actions in preparation for data collection. These are shown in the following list:

- Get DATA_COLL_LIST from the PERMTOOL mailbox. This list specifies all characteristics of the data collection effort. It includes parameters which specify the collection interval, as well as the data items to be collected.
- Clear all buffers to initial collection conditions. This overcomes the need for several "first-time" flags which would otherwise be required by subsequent processing routines.
- Place a record on the accounting log which flags the beginning of a data collection effort. This simplifies the work of data reduction programs which will extract data collection information from the accounting log, and must be able to distinguish the recorded boundaries of a collection effort.
- If collection is not to start immediately, then wait until time for the first collection event.

6.3.2 Data Collection Routine - This routine is called whenever data must be collected from the operating system. It uses the DATA_COLL_LIST to determine what data elements are needed. Based upon this list, the procedure invokes one or more of the following four procedures:

DATA COLLECTION SOFTWARE DESIGN

1. JOB_COLL - Job/process data collection routine
2. MEMORY_COLL - Memory data collection routine
3. DEVICE_COLL - I/O device data collection routine
4. PROCESSOR_COLL - CPU data collection routine

As their names imply, these routines are designed to collect data from specific parts of the operating system environment. The JOB_COLL is the only procedure which will be implemented during the initial stages of this effort. It functions with the help of the VMS system routine GETJPI to gather the data it requires. The three remaining routines all obtain their data directly from various control tables maintained by the VMS and stored either in main memory or on disk. When any of these routines has finished gathering data for one event, it passes the data to the data recording routine, and then returns to the data collection executive.

6.3.3 Data Recording Routine - The data recording routine is responsible for two functions:

1. Accepting data from the data collection routines
2. Passing data to the VMS accounting routines

However, the VMS accounting routines deal with larger records than those which may be created during a single data collection event. Consequently, the recording routine functions as an intelligent data buffer. It saves data until it has enough, and then sends a record to the

DATA COLLECTION SOFTWARE DESIGN

accounting manager for processing.

6.3.4 Collection Termination Routine - The collection termination routine is called by the data collection executive immediately before the data collection program stops. The function of this routine is to insure that all data collected is logged onto the accounting log, and that the log has been marked to indicate the end of a data collection effort. This is necessary so that the data reduction program can easily identify the range of data collection records on a (potentially) much longer accounting log. To do this, the termination routine sends a termination message to the recording routine. This forces the recording routine to include the termination message in its current buffer, and send that buffer to the accounting manager for action. When this action is complete, control is returned to the collection executive which stops execution of the module.

6.4 SUMMARY

The Data collection module was designed to satisfy the need specified in requirement 3 of chapter 2. It functions as an independant module which is started by the PERMTOOL executive. Directions from PERMTOOL are received through a VMS system mailbox, and specify both the collection period and the items to be collected. These items, which fall into one of four categories: job data, memory data, I/O data, or

DATA COLLECTION SOFTWARE DESIGN

processor data, are collected and passed to the VMS accounting manager which places the data on the system accounting file. Collection continues until the data collection period expires, at which time the job is finished, and it stops executing.

CHAPTER 7

RUN-TIME RESOURCE UTILIZATION

7.1 INTRODUCTION

One of the fundamental concerns of any performance evaluation effort is to measure the system under normal workload conditions. Yet, any software monitor, such as PERMTOOL, uses computer resources and thereby adds an abnormal portion to the system workload. This additional workload will be reflected in any measures taken of the system, and will have an adverse effect on the accuracy of any evaluation of the system. This chapter will address the impact PERMTOOL might have on various performance measures, and suggest ways to minimize, or at least quantify, that impact.

7.2 MEMORY USE

PERMTOOL memory use (at least during data collection) can be fixed by specifying at load time that all pages of the data collection module must be memory resident. Since the memory requirements of the data collection program are identified

RUN-TIME RESOURCE UTILIZATION

at the time the program is linked to build an execution module, then the analyst can subtract this value from total memory use to measure normal workload use. If the data collection effort is gathering job related data (see chapter 6, paragraph 6.3.2) then the actual memory used can be gathered directly, rather than taking the steps previously described. If the analyst requests working-set-size, then this data will be collected for the data collection process just as it will for all other processes in the system. Unfortunately, more important questions cannot be answered so easily:

- * What is the impact of PERMTOOL memory use on the availability of memory for other programs?
- * Does PERMTOOL prevent or delay the assignment of memory to jobs in the system?
- * How does PERMTOOL memory use affect other system parameters such as degree of multiprogramming?

These questions must be reviewed in light of the specific system and workload being measured; no general answer can be given.

7.3 CPU USE

Measuring the CPU use by PERMTOOL can be done in the same fashion just described for memory use. As before, problems occur when one tries to assess PERMTOOL's impact on other jobs through competition for the CPU. This may have a significant effect on several factors such as measured

RUN-TIME RESOURCE UTILIZATION

degree of multiprogramming, and a lesser effect on other factors such as calculated CPU utilization.

7.4 I/O DEVICE USE

PERMTOOL makes no demand on the systems I/O resources, except periodic writes to the system accounting file. Since the number of records written can be counted and the elapsed time of the data collection effort is specified by the user, then the mean time between successive writes can be calculated. Using common methods [COFF73], the mean time for each write can be calculated, if the characteristics of the disk drive are known. The results of these calculations can then be deducted from estimations of device utilization and performance, as necessary to reflect a real workload.

7.5 SUMMARY

There is a definite need to quantify the resource requirements of the PERMTOOL data collection module. Depending on the nature of the computer system and the requirements of the data collection effort, different methods may be used to estimate PERMTOOL's CPU, memory, and I/O resource uses. Unfortunately, these measures can only be of limited value when assessing the competitive impact of using PERMTOOL. These issues are left to be solved by future researchers.

CHAPTER 8

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

8.1 INTRODUCTION

The purpose of this effort has been to develop at least part of a performance measurement tool for use with VAX computers running under control of the VMS operating system. The approach used was to:

1. Identify a set of information requirements which could be satisfied through the medium of a software monitor.
2. Define a means whereby:
 1. A user can identify the information he/she wants;
 2. PERMTOOL can take steps to collect, reduce, and present the required data to the user.
3. Develop programs which implement step 2, above.
4. Test the software resulting from step 3.
5. Install the software as tested.

This chapter will compare these intended steps with the actual results, discuss the problems encountered during this

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

effort, and report on the current status of PERMTOOL.

As already mentioned, the first step was to define the information goals of a system manager who is concerned about the performance of her/his system. This is a complex issue which involves the workload of the computer system, the components of the computer system, and the specific measurement goals of the system manager [BOROV79, BROWN75, GIAMM76, etc]. Consequently, the goal of PERMTOOL became one of making as many data elements as possible available to the user, so that he/she could choose as needed. Identification of these data elements is "reasonably complete" and is consolidated in the lists of Appendix B.

Allowing a user to tell PERMTOOL what he/she needed was the second step to be addressed. In this part of the effort, there were two concerns:

- Design a human interface which was convenient to use.
- Design an interface which was easy to maintain.

The dialogue frame or slide concept is one which is commonly used. It lends itself particularly well to situations in which a user must be given instructions or information in order to provide a response [GAL81, pp 1 - 2, 14, 79]. Furthermore, the need for a strict question/answer sequence can easily be eliminated for the experienced user by allowing him/her to enter several responses at one step.

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

Designing this dialogue becomes nothing more than:

1. Deciding what to tell/ask the user.
2. Deciding whether a response is valid or not.
3. Deciding what action must be taken for each valid response.
4. Determining a proper action for each invalid user response.
5. Repeating steps 1, 2, 3, and 4 until no further information is needed.

The implementation method for the user dialogues involved creating a list of all possible dialogue statements in a form which specifies:

- * How to display the statement on the user's terminal.
- * Whether or not a user response is necessary.
- * What procedure should validate the user's response.
- * What routine should process a valid response.
- * What routine should process an invalid response.

This method was chosen in the belief that it would simplify the algorithms needed to maintain a user dialogue. At the same time, this approach gave the user much greater ability to modify the computer statements, since they are stored as a separate file, with their own maintenance program. This would be of great value as shortcomings in the original text were discovered. It also provided the manager with a means

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

to disable/enable options without recompiling the program. He/she would only need to change a statement and the name of its associated processing routines.

Unfortunately, this approach fell victim to several problems during implementation. First, the Pascal language has no direct way of processing a symbolic vector such as the stored processing routine names. This would be the ideal way to handle the named processing routines stored in the dialogue frames. During the design period, the only available high-level languages which could do this were LISP and C. Both languages are incompatible with Pascal on the VAX, and the author has almost no experience with either language. Consequently, many long Pascal case statements were used to interpret vector values and transfer to the proper routines.

The second problem developed as a result of limiting the information contained in each `FRAME_ITEM`. Instead of a transfer vector consisting of one validation, one bad-processor, and one good-processor, a better method might have been to permit a series of good processors. In retrospect, this would have simplified several aspects of the coding effort (by reducing confusion) and resulted in a more compact set of procedures. As implemented, there is probably little or no savings over a direct coding approach to the dialogue process.

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

Designing means of collecting data for the user has also turned out to be a mixed bag of successes and failures. This process involves three steps:

- Find out data collection specifications from the user.
- Do the data collection as per the user requirements.
- Log all data collected on a permanent file.

The first step has been accomplished, by way of the user interface, although it is currently limited to the specification of data relating to job/process flow. User requirements are stored in a compact form for transmission to the data collection program, an asynchronous, independent program which performs the data collection. The medium to be used for communication is a VMS mailbox. Unfortunately, using these has proved to be difficult. In Pascal, a mailbox can be defined as a file from which records may be read or written. However, after a month of experimentation, the programmer was unable to achieve a reliable one-way information flow. Further experiments using the create mailbox (CREMBX) and Queue I/O (QIO) system services should solve this problem, since the VAX System Services Manual leaves no doubt that these services do work. However, in order to avoid delaying the project further, the mailbox was replaced by a conventional disk file.

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

Once the data collection program parameters are identified, the program can establish its collection cycle and begin operation. Originally, we intended to gather some data from the system tables (VMS internals) and get other data by using a modified version of the DEC DISPLAY program. Neither of these methods proved practical in the time available. Simply put, the author overestimated how much he could learn about the operation of VMS with its supporting data and software structures. Consequently, it is not currently possible to collect all data elements shown in Appendix B. Those which can be collected are shown in table 8-1, and are only reported on a process-by-process basis.

Table 8-1. Data Items which can be collected using PERMTOOL

ACCOUNT	CURPRIV	FINALEXC	PID	TQCNT
APTCNT	DFPFC	FREPOVA	PPGCNT	TQLM
ASTACT	RFWSCNT	FREPIVA	PRCCNT	UIC
ASTEN	DIOCNT	GPGCNT	PRCLM	USERNAME
ASTLM	DIOLM	GRP	PRCNAM	VIRTPEAK
AUTHPRIV	DIRIO	IMAGNAME	PRI	VOLUMES
BIOCNT	EFCS	IMAGPRIV	PRIB	WSAUTH
BIOLM	EFCU	LOGINTIM	PROCPRIV	WSPEAK
BUFIO	EFWM	MEM	STATE	WSQUOTA
BYTCNT	EXCVEC	OWNER	STS	WSSIZE
BYTLM	FILCNT	PAGEFLTS	TERMINAL	
CPULIM	FILLM	PGFLQUOTA	TMBU	

As stated earlier in this thesis, no attempt has been made to develop data reduction or data presentation programs. They were deemed, at the outset, to be beyond the scope of this project.

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

Developing and testing programs became a much more haphazard process than would be considered ideal. This is due principally to the prolonged research periods trying to learn about VMS and how to use its tables, languages, and system services. None the less, a top down approach was still followed for development and testing. Testing has been completed at the module level through static analysis of the code (desk-checking), boundary-value tests, and path testing. Since most of the modules are quite small, (typically less than 40 lines of code) comprehensive tests of this kind were fairly easy to accomplish. Integration testing is incomplete, and will remain so until such time as the complete system is built. However, all completed modules of the user interface and the PERMTOOL executive have undergone integration tests. These tests have identified several errors which are not yet fixed. In some cases, the errors affect frame processing, while in other cases they seem to be due to erroneous use of the create process (CREPRC) and Hibernate (HIBER) system services. A complete list of identified deficiencies will be included in the program listing of PERMTOOL.

As of this writing, most of the PERMTOOL must be considered incomplete. Those parts which have been completed include the executive control module and the part of the user interface which is needed to define process-related data collection requirements or

SOFTWARE DEVELOPMENT, TESTING, AND INSTALLATION

user-required VMS utility support. The PERMTOOL executive will accept and process user requests in either of these categories. However, the VMS utilities do not return to PERMTOOL when the user is done with them. Access to the VMS HELP system has not been established. Existing dialogue frame processing provides complete explanations (almost the same text which will be used in the HELP system) that minimize the need for a separate HELP facility.

CHAPTER 9

CONCLUSIONS AND RECOMMENDATIONS

9.1 CONCLUSIONS

This thesis represents an effort to develop a software monitor for VAX computers. As such, it is far from complete. However, the thesis has served to underscore several important points for the author. These points are:

1. At least six months of dedicated effort should be allowed to learn a new operating system.
2. Design and planning efforts are of paramount importance to a project of this nature (an academic point which has been driven home very firmly).
3. It is very easy to over-estimate the amount of software which can be developed in a fixed period of time.

If for no other reasons than reemphasizing these points, this thesis effort has been worthwhile.

In addition, this thesis can serve as a starting point for the development of a comprehensive performance monitor. It begins to fill a void which several VAX installations have tried to plug with a variety of isolated tools [DEC82].

CONCLUSIONS AND RECOMMENDATIONS

DEC has packaged these various tools into an integrated system called MONITOR, which is a standard part of version 3.0 of the VMS operating system. This package does a very elegant job collecting and displaying data for site operators and managers. Unfortunately, MONITOR is designed for online, real-time display of data which has been reduced and formatted in predefined ways. The design goal of PERMTOOL (at least this effort) has been to collect and save data of interest in a wide variety of situations. Data reductions can then be defined and performed in fashions most acceptable to site management. For example, MONITOR provides no facility which would permit a user to observe any system parameters (e.g. throughput) over a long period of time such as a week, or a month, or more. PERMTOOL, on the other hand, will support such a method of watching system performance. Consequently both tools are of value; they support different domains of interest.

9.2 RECOMMENDATIONS

The PERMTOOL development effort has laid the groundwork for several follow-on efforts. In some cases, however, the researcher should first become very well grounded in VAX/VMS operating system methods and structures. This background can probably best be obtained by taking the sequence of three short courses, culminating with VMS Internals, which are taught by DEC to VAX system programmers. Although these

CONCLUSIONS AND RECOMMENDATIONS

courses will cost about \$4000 and three weeks of time, they are probably the best quick way of building the essential background knowledge.

With an appropriate background in VAX/VMS, the following topics should be considered for further investigation:

1. Extend PERMTOOL data collection to a full set of performance parameters.
2. Investigate the current data collection and logging methods used by MONITOR; quantify the resources used by the MONITOR itself.
3. Investigate network performance measures for DECNET; extend this investigation to include the DELNET or the DELNET/VAX interface.
4. Building on 1, 2, and 3, above, develop an integrated set of statistical reduction tools which can be used through a broad range of performance efforts.
5. Extend the PERMTOOL so that it can be used with other VAX operating systems such as UNIX and RSZ-11M.

Without a comprehensive VAX/VMS background, several allied areas of research/development are available:

1. Develop a comprehensive data presentation package, using video terminals, printers, and graphic plotters as output media.
2. Develop and publish a complete User's Manual for the PERMTOOL system.

AD-A124 984

DEVELOPMENT OF COMPUTER PERFORMANCE EVALUATION TOOLS
FOR VAX-11/780 COMPUTERS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. E C GILPIN
DEC 82 AFIT/GCS/EE/82D-15 F/G 9/2

2/2

UNCLASSIFIED

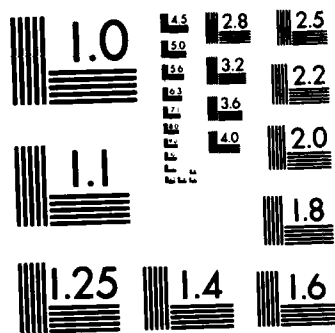
NL

END

FILED

10

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

CONCLUSIONS AND RECOMMENDATIONS

9.3 SUMMARY

Developing PERMTOOL has shown that there is a need for several types of performance measurement tools on the VAX line of computers. Some performance measures can be provided by tools such as MONITOR, yet other measures may only become available through the advent of PERMTOOL, or other similar tools. PERMTOOL, in its present state, is of limited value. However, continued development of the package can yield a very valuable system for evaluating either long- or short-range performance on VAX computers.

BIBLIOGRAPHY

- Ball, Duane Ray, "Software Monitors: Principles of Design and Use", Computer Performance Evaluation, 12th Meeting of the Computer Performance Evaluation Users Group, Nov, 1976, pp 215-219.
- Bell, C. Gordon and J. Craig Mudge, John E. McNamara, Computer Engineering, a DEC View of Hardware Systems Design, Digital Press, Bedford, MA, August, 1979.
- Birch, Harry K., A Management System for Computer Performance Evaluation, MS Thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December, 1981.
- Borovits, Israel and S. Neumann, Computer Systems Performance Evaluation, Lexington Books, Lexington, MA, 1979. [Wright-State Library]

"It should be noted that computer performance ... cannot be discussed but in the context of a defined application or group of applications."

Indices of Performance

Capacity	Response time
Throughput rate	Overhead percentage
Component overlap measure	Software time measures
Software transition measures	Tuning
User satisfaction	Reliability measures
System utilization measures	Raw speed
Ease of use	Availability

Purposes of evaluation (includes descriptions)

- o Selection evaluation
- o Performance projection
- o Performance monitoring
- o Evaluating research objectives

CPE techniques

- | | |
|---------------------|---------------------------------------|
| o Audio-Visual | o Hand timing |
| o Simple formulae | o Periodic Charts and Reports -- |
| o Instruction mixes | - Adams's Charts |
| o Kernal programs | - Auerbach Charts |
| o Models | o Numerical scoring (Figure of merit) |
| - Analytic | o Benchmarks |
| - Statistical | o Synthetic Programs |
| - Graphical | |
| - Algorithmic | |
| o Simulation | |

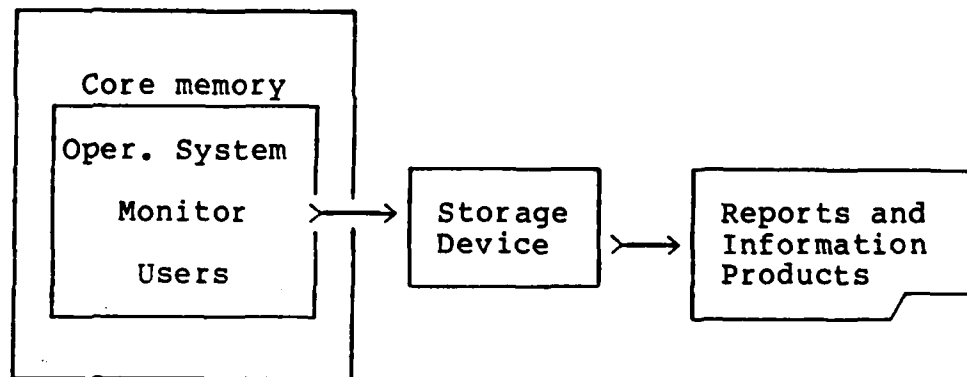
BIBLIOGRAPHY

Monitors

The authors discuss the pro's and con's of the two types of monitors (hardware and software). They also present, as a series of lists, a variety of procurement considerations.

Their view of a software model is as indicated:

Computer Being
Measured



Software Monitor -- Conceptual View

Table 3-1: Characteristics of Hardware and Software Models

Characteristic	H/W	S/W
		-- --
Cost	High	Low to Medium
System Overhead	None	Variable
Hardware/OS Dependant	No	Yes
Precision	Monitor Dependant	Host Sys Depend.
Capture of qualitative Information	Limited	yes
Training required	Extensive	Limited
Flexibility	Good	poor
Ease of use	Poor	Good
Portability	Poor	Good
Peripheral Monitoring	Yes	No
Interrupt Considerations	None	Yes
Expected life	Good	Fair

Adapted from: EDP Performance Review, "Performance Monitors -- Hardware or Software", 1:5 (May, 1973)

BIBLIOGRAPHY

"The application of accounting data to computer performance evaluation is feasible, as a rule, in the evaluation of working systems, and not for selection or performance forecasting purposes."

Purposes of an accounting system

1. Measure modifications to a system and evaluate influence on behavior.
2. Use with hardware and software monitors to improve system performance.
3. Collect information to help direct improvement efforts.
4. Gather load and job characteristic data required by management.
5. Detect trends in system behavior changes.

Brown, James W., "Controlling the Complexity of Menu Networks", Communications of the ACM, July, 1982, pp 412-418.

Brown, Shellman H., Jr., "EDP Capacity Management", Computer Performance Evaluation: Proceedings of the 1977 SIGMETRICS/CMG VIII Conference on Computer Performance: Modeling, Measurement and Management, ACM SIGMETRICS and the Computer Measurement Group, 1977, pp 301-305.

BROWNE, J. C. "An Analysis of Measurement Procedures for Computer Systems," Performance Evaluation Review, Association for Computing Machinery, Jan 1975, 29-32.

Discusses the steps in a Computer Performance Evaluation Study:

1. Goals and purposes which performance measurement and analysis (PMA) project is to further.
2. Definition of the model of the system which will be necessary to support accepted goals and purposes.
 - a. Operation of system must be understood to build the model.
 - b. Defines the data which must be gathered, including workload specifications, performance parameters,

BIBLIOGRAPHY

and process identification data.

The basic goal of a PMA project is to identify a cause-effect relationship between some factor (e.g. hardware speed, algorithm selection, program structure, etc.) and chosen performance metrics.

Make a conscious decision about whether to sample from the real (non-reproducible) world or an artificial world.

Definition of measurements:

- A. Factors
 - 1. Level of definition and resolution of measurements.
 - 2. Retention of correlations in data.
- B. Levels.
 - 1. Executing processes.
 - 2. System level.
 - 3. Device level.
- C. Correlations.
 - 1. All at one level.
 - 2. Between levels.

Be sure to gather data which can validate model and identify any sensitivities which metrics have to system or model environment.

Measurement Tools Further Role of System Models

If well done and formalized, then the model can be extended to hypothetical configurations, with reasonable confidence that the model will be a valid predictor of the proposed system.

Measurements: State-of-the-art and further needs

Identifies things which have been accomplished, as well as topics which still need investigation/development.

Buzen, Jeffrey P., "A Note on Operational Assumptions", Performance Evaluation Review, summer, 1981, pp 76-79.

BIBLIOGRAPHY

- Clark, Jon D. and Robert M. Golladay, "Empirical Investigation of the Effectiveness of Several Computer Performance Evaluation Tools", Performance Evaluation Review, Fall, 1980, pp 31-36.
- Coffman, Edward g. and Peter J. Denning, Operating Systems Theory, Prentice-Hall, Inc, Englewood Cliffs, N. J., 1973.
- Coombs, M. J. and J. L. Alty, editors, Computing skills and the User Interface, Academic Press, New York, NY, 1981.
- Davis, Richard M., Thesis Projects in Science an Engineering, St. Martin's Press, New York, 1980.
- Digital Equipment Corporation, DECnet-VAX System Manager's Guide, Digital Equipment Corporation, Maynard, MA, 1982.
- Digital Equipment Corporation, Introduction to VAX-11 Record Management Services, Maynard, MA, March, 1980.
- Digital Equipment Corporation, Technical Summary, VAX-11/780, Maynard, Ma, 1978.
- Digital Equipment Corporation, VAX-11 Architecture Handbook, Digital Equipment Corporation, Maynard, MA, 1979.
- Digital Equipment Corporation, VAX/VMS Primer, Maynard, Ma, August, 1978.
- Digital Equipment Corporation, VAX-11/RSX-11M Programmer's Reference Guide, Digital Equipment Corporation, Maynard, MA, 1982.
- Digital Equipment Corporation, VAX Software Handbook, Digital Equipment Corporation, Maynard, MA, 1980.
- Digital Equipment Corporation, VAX/VMS Summary Description and Glossary, Maynard, MA, March, 1980.
- Digital Equipment Corporation, VAX/VMS System Services Reference Manual, Maynard, Ma, March, 1980.
- Digital Equipment Corporation, VAX-11/RSX-11M User's Guide, Digital Equipment Corporation, Maynard, MA, 1982.
- Digital Equipment Corporation, Discussion with field representatives, subj: DEC efforts to develop performance measuring tools.
Currently there exists a hodge-podge of tools.

BIBLIOGRAPHY

many of which have been developed by various VAX users. DEC has been trying to consolidate and test these tools. At this time, the tools work, but are sufficiently unfriendly that only DEC field representatives will be allowed to use them. DEC expects the tools to be available for general use in 12 - 18 months. This package of programs includes the following:

- * CERT - Is very similar to CONFIG, a DEC utility which documents the configuration of the system.
- * DVA - Disk Volume Analyzer: this looks for disk fragmentation and is easy to run.
- * COLLECT2 - A program which collects and reports system data.
 - o File primitive statistics
 - o ACP slots
 - o Page list sizes - monitors both free and modified page list characteristics.
 - o Process and balance set count
 - o memory use by group: paged, non-paged, user, ...
 - o CPU mode statistics
 - o I/O rates on each device, paging, swapping, user, ...
 - o SWAP statistics
 - o Page fault statistics
 - o State transitions
 - o Disk/MASSBUS controller utilization
- * PROCESS2 - Formats and prints data gathered by COLLECT2.
- * IMAGE-LEVEL-PROGRAM-ACCOUNTING - This program patches VMS to account by image rather than by process, and produce reports based upon the

BIBLIOGRAPHY

data gathered.

- * FORTRAN PROFILER - At execution time, this program produces system timing diagrams and profiles for a target FORTRAN program.
- * System Profiler - Provides detailed CPU usage data by routine and processor mode.
- * SWCONNECT - Tool designed to allow development of customized tools through connection of the software to software interrupts.

Dinh, Vincent, "Current Needs in the Field of CPE", Proceedings of the 1977 SIGMETRICS/CMG VIII Conference on Computer Performance: Modeling, measurement, and Management, Association for Computing Machinery, New York, 1977 [DINH77].

Discusses CPE measurement and modeling tools:

- * Their history and development since the mid 60's.
- * Problems associated with the use of these tools.
- * Analytically identifies what system users look for during a performance evaluation.

The author discusses early models of computer systems and subsystems, and their subsequent growth through circa 1974. He then talks about the development of measurement tools for both hardware performance and software performance. In this portion of the paper, he also highlights some of the problems, including human resistance to change, associated with various tools.

Dujmovic, Jozo, J., "Criteria for Computer Performance Evaluation", Performance Evaluation Review, Fall, 1979, pp 259-267.

Edgecomb, Gordon D., "Documentation of Computer Evaluation Studies", Computer Performance Evaluation, 12th Meeting of the Computer Performance Evaluation Users Group, Nov, 1976, pp 135-139.

BIBLIOGRAPHY

"System Evaluation Methodologies: Combined Multidimensional Scaling and Ordering Techniques", Performance Evaluation Review, ACM, Spring, 1980, pp 52-58.

Evaluation Review 1975 - 1977, FEDSIM, Washington, DC., 1978.

Giammo, Thomas, "Towards a Sound Computer Performance Evaluation Methodology", Computer Performance Evaluation, Proceedings of the 12th Meeting of the Computer Performance Evaluation Users Group, New York, Nov, 1976, pp 67-76.

"The essence of computer performance evaluation is one of comparative analysis -- that is the comparison of the measured performance of an observed system with the performance to be expected across a range of feasible alternatives under consideration." [page 67]

Some essential characteristics:

1. Range of feasible alternatives must exist in context of different combinations of the hardware, software, workload, operational environment, operational protocols, etc.
2. Performance must be defineable as a concept
3. Performance measures must be directly relatable to the performance concept.
4. 4. Must have a basis for predicting the performance of unmeasured alterantives.

External Performance -- Relates Mgt objectives to computer system functions.

Internal Performance -- Relates computer system functions to workload and configuration factors.

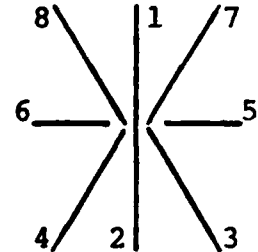
Ref Buzen, Chandy, Reiser, Giammo, et al:

Computer system is represented as a network of processors with exponential service times (Exponential Queuing Network, EQN). Each processor is a service center for (memory, CPU, I/O, etc). Workload is the set of jobs of various classes which place demands on these service centers.

BIBLIOGRAPHY

Kiviat Graphs. One way of representing system status. It keeps track of:

1. CPU busy
2. CPU idle
3. any channel busy
4. CPU idle & any channel busy
5. CPU busy & any channel busy
6. CPU busy & no channel busy
7. CPU busy - problem state
8. CPU busy - supervisor state



Hughes, James H., "DIAMOND A Digital Analyzer and Monitoring Device", Performance Evaluation Review, ACM, Summer, 1980, pp 27-34

Jensen, Kathleen and N. Wirth, Pascal User Manual and Report, 2nd Edition, Springer-Verlag, New York, NY, 1974.

Knudson, Michael E., "A Computer Performance Evaluation Operational Methodology", Performance Evaluation Review, Winter, 1981-1982, pp 74-79.

Koffman, Elliot B., Problem Solving and Structured Programming in Pascal, Addison-Wesley Publishing Co, Reading, MA, 1981.

Kuck, David, "Computer System Capacity Fundamentals", Performance Evaluation Review 1975 = 1977, FEDSIM, Washington, D. C., 1978.

Levy, Henry M and Richard H. Eckhouse, Jr., Computer Programming and Architecture the VAX-11, Digital Press, Bedford, Mass, 1980.

MacDougall, M. H., "The Event Analysis System", Computer Performance Evaluation: Proceedings of the 1977 SIGMETRICS/CMG VIII Conference on Computer Performance: Modeling, Measurement and Management, ACM SIGMETRICS and the Computer Measurement Group, 1977, pp 91-100.

Morris, Michael F. and Paul F. Roth, Tools and Techniques Computer Performance Evaluation for Effective Analysis, Van Nostrand Reinhold Co., New York, 1982.

Nemeth, Thomas A., "An Approach to Interactive Performance Analysis in a Busy System (NOS/BE)", Performance Evaluation Review, Winter 1981-1982, pp 57-73.

BIBLIOGRAPHY

- Oliver, Edward F., "Management Control Criteria for Testing and Evaluating Performance", Computer Performance Evaluation, 12th Meeting of the Computer Performance Evaluation Users Group, Nov, 1976, pp 125-132.
- Orchard, R. A., "A new Methodology for Computer System Data Gathering", Performance Evaluation rfeview, Fall, 1977, pp 27-41.
- Pearson, Sammy W. and James E. Bailey, "Measurement of Computer User Satisfaction", Performance Evaluation Review, Spring, 1980, pp 59-68.
- Probert, R. L., "Optimal Insertion of Software Probes in Well-Delimited Programs", IEEE Transactions on Software Engineering, January, 1982, pp 34-42.
- Rosen, Saul, Lectures on the Measurement and Evaluation of the Performance of Computing Systems, Society for Industrial and Applied Mathematics, Philadelphia, Pa, 1976.

Monograph which covers several general topics:

- a. Conceptual Model of an Operating System;
- b. Accounting Systems;
- c. Software Probes;
- d. Hardware Monitors;
- e. Benchmark Workload models;
- f. Simulation;
- g. Mathematical Models

Rosen limits his discussions of performance to three categories: throughput, responsiveness, and quality. Similarly, he limits his considerations of computer systems to general purpose computers whose hardware and software can support multiprogramming, multiprocessing, and on-line or network processing.

The author points out that human engineering factors can have a significant impact on performance, and uses the layout of a computer room as an example.

While discussing validation of performance data, he paraphrases Sir Arthur Eddington by saying: "...you cannot believe in data about the performance of large computing systems unless they can be explained in terms of a conceptual model of the system." Later, he makes the point that one

BIBLIOGRAPHY

must also be careful to insure that data collected over time has a consistent meaning over the sampling interval.

Two kinds of software probes are used today: sampling probes and tracing probes. In either case, care must be taken to avoid a deluge of data; however this risk is higher with tracing probes.

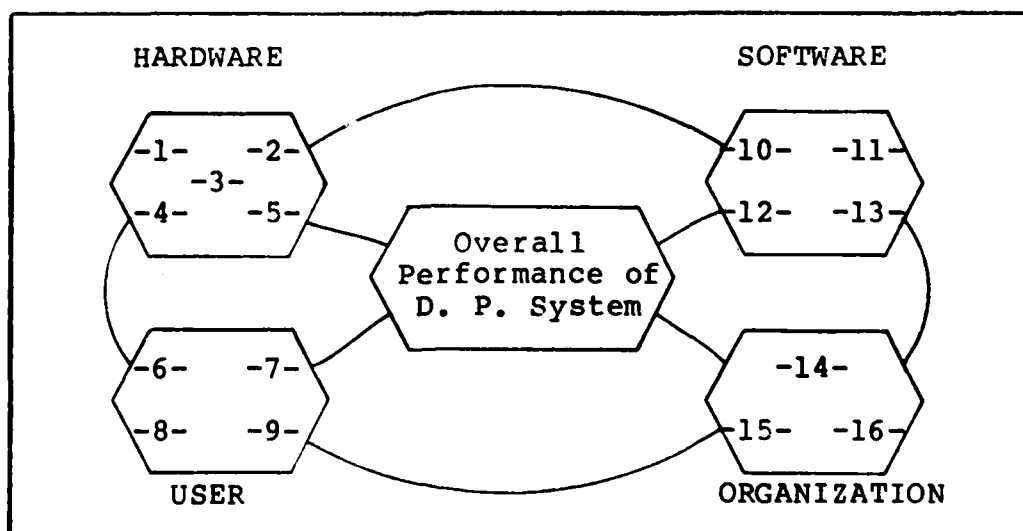
Sauer, Charles H. and K. Mani Chandy, Computer Systems Performance Modeling, Prentice-Hall, Englewood Cliffs, N. J., 1981.

Shneiderman, Ben, "Control Flow and Data Structure Documentation: Two Experiments", Communications of the ACM, January, 1982, pp 55-63.

Svobodova, I., Computer Performance Measurement and Evaluation Methods: Analysis and Applications, American Elsevier Publishing Company, Inc., New York, NY, 1976.

Terplan, Kornel, "Cost Optimal Reliability of Data Processing Systems," Performance Evaluation Review, Association for Computing Machinery, April, 1975, 1-12.

The author presents a model of a data processing operation:



In this figure, the numbers have the following meaning:

BIBLIOGRAPHY

HARDWARE

1. CPU
2. Main Storage Space
3. I/O devices
4. Direct Access Storage
5. I/O channel

ORGANIZATION

14. External Organization
15. Internal Organization
16. Human elements

SOFTWARE

10. Control Program
11. Utilities
12. Program trials
13. Processing program

USER

6. Problem to solve
7. Source program
8. Process methods
9. Data to process

Wang, Y. T., "On the VAX/VMS Time-Critical Process Scheduling", Performance Evaluation Review, Fall, 1981, pp 11-18.

Weinberg, Victor, Structured Analysis, Yourdon Press, New York, NY, 1980.

APPENDIX A

VAX OPERATIONAL ESSENTIALS

A.1 INTRODUCTION

The purpose of this appendix is to provide some understanding of VAX operations. In this light, we will look at three aspects of the VAX:

- <> Memory management.
- <> Process scheduling.
- <> Priority Processing.

The methods employed to effect control of these operations are central to VAX/VMS operation, and are therefore of paramount importance in any system performance data collection or analysis effort.

VAX OPERATIONAL ESSENTIALS

A.2 MEMORY MANAGEMENT

Since the VAX is a virtual memory computer, it has mechanisms in both hardware and software to facilitate swapping pages of code or data between primary and secondary memory. These mechanisms operate within the framework of a **balance set** of active processes, each with its own **resident set** (DEC uses the term **working set**) of primary memory pages. Within the computer, pages are defined to be a fixed size, such as 512 words. Each active process (or job, if you will) in the computer has a resident set size associated with it which defines the number of pages of primary memory available to that process (In fact, the process will not be activated until sufficient free pages are available to satisfy the resident set size requirements of the process).

This resident set size may well be smaller than the memory space of the process. If this is so, then the process will periodically make reference to regions of its memory which are not in primary memory (its resident set). When this happens, a hardware page fault occurs. Such an event forces the operating system (VMS) to eliminate a page from the current resident set of the process in order to make room for the page being referenced. VMS uses a first-in-first-out strategy to decide which page to discard. However, VMS does not actually remove the page (or destroy

VAX OPERATIONAL ESSENTIALS

its contents). Instead, it removes the page from the resident set of the process and inserts it into one of two page lists.

One of these lists, the **free page list**, keeps track of available pages which were not altered during their last use. The other, the **modified page list**, keeps track of pages which were altered during their last use. Before VMS initiates a disk read operation to get a page on disk, it looks to see if the page is in either of the two page lists which collectively make up the **shared page cache**. If it finds the desired page, it is removed from the appropriate list and connected back into the process' resident set, thereby saving the time needed for a disk read.

If the necessary page is not in the page cache, then VMS will try to use memory space taken from the free page list to store the new page being read from disk. Only in the event that the free page list is empty will VMS use space from the modified page list. When it uses this memory space, it must write the current contents of the page to disk (because the page was modified during its last use) before it fetches the new material from disk into that memory page. Under normal circumstances, VMS will try to avoid some of this latter overhead by writing modified pages to disk whenever the system is idle or whenever the list grows too long. Pages which have been saved on disk in this

VAX OPERATIONAL ESSENTIALS

manner are then moved to the free page list for subsequent use.

A.3 PROCESS SCHEDULING

In VAX terminology, a process is any activity which requires CPU and possibly peripheral resources to do its job. A process is the fundamental entity used by VMS for resource allocation; it is also the most elementary non-hardware unit identified by the accounting system. When a process enters the system, it will not be permitted to enter execution until VMS can supply enough pages of memory to complete the process resident set. When the process receives its memory, it joins other processes in the balance set -- those processes which may be scheduled for either I/O or CPU operations.

When a process gains control of the CPU, it retains control until one of several events occur:

- o A higher priority process arrives;
- o The process puts itself to sleep while it waits for some event to occur;
- o The process must wait for an I/O operation to finish;

VAX OPERATIONAL ESSENTIALS

- o The process' residency quantum (guaranteed time in which the process may retain its resident set of memory without being swapped out) expires.

The action taken whenever one of these events occurs is a scheduling activity which depends on both the event and the current state of the system. As such, scheduling involves several factors, prime among them, the relative priority of the process being scheduled.

A.4 PRIORITIES

In general, process priorities range from the lowest, which is 0, to the highest, 31. Within this range of priorities, there exist two groups. The first group, whose legal priorities extend from 0 to 15, is reserved for those processes which are not time-critical. For example, a batch process such as a Pascal compile would normally be assigned a priority in this range. On the other hand, a very time-critical process like the controller for a high speed device would probably receive a priority in the range 16 to 31, which is reserved for real-time processes.

Any process in the computer will have two priorities: a base priority and a current priority. The base priority, which never changes, is used for I/O scheduling, while the

VAX OPERATIONAL ESSENTIALS

dynamically changing current priority, is used for CPU scheduling. CPU scheduling is handled on a preemptive basis. That is, if a process becomes eligible for the CPU, and its priority is higher than the process which currently "owns" the CPU, then the current process will be preempted in favor of the new process. On the other hand, I/O scheduling uses the base priority and will not interrupt an operation which is already in progress.

APPENDIX B
PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

B.1 PREFACE

This appendix contains a list of data items and parameters which can be of value to VAX performance measuring personnel. This list consists of several tables of data items. Within each table, the data items are expected to contribute information about the computer in the area of the major subject heading. These groups are shown in figure B-1.

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table	Subject Area
B-1	Data Groups used in tables B-2 through B-7
B-2	Data Items for Response Times
B-3	Data Items for Workload Measurement
B-4	Data Items for Resource Usage Accounting
B-5	Data Items for Workload Scheduling
B-6	Data Items for Fine-tuning Control
B-7	Consolidated Data Item list

Figure B-1. Major subject areas of data items listed in Appendix A.

Associated with each item is a series of flags which indicate which of the VAX log files and record types contain the item.

Data Group	
Identifier	Meaning
O	Processor related data
P	Process/job related data
M	Memory related data
D	Device related data

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-2.

Data Items for System Response Time

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
allocations	system tables				
application_speed	calculated				
availability	calculated				
blocked_time	calculated				
buffered_io_count	Accounting log	X	X		X
cache_hits	system tables				
collided_page_wait	system tables				
common_event_flag_wait	system tables				
compiler_speed	calculated				
cpu_service_time	calculated				
cpu_time	Accounting log	X	X		X
cycle_time	calculated				
degree_of_multiprogram	system tables				
direct_io	system tables				
direct_io_count	Accounting log	X	X		X
disk_reads	system tables				
disk_writes	system tables				
file_lookups	system tables				
file_opens	system tables				
free_page_wait	system tables				
interarrival_time	calculated				
internal_priority	system tables				
io_service_time	calculated				
loc_ev_flag_wait	system tables				
loc_ev_flag_wait_oswap	system tables				
memory_utilization	calculated				
multi_program_effic	calculated				
mutex_misc_resource_wt	system tables				
quantum_size	system tables				
response_time	calculated				
service_wait_time	calculated				
suspend_wait	system tables				
suspend_wait_outswap	system tables				
time_in_compat_mode	system tables				
time_in_exec_mode	system tables				
time_in_idle_mode	system tables				
time_in_kernal_mode	system tables				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-2 (continued).

Data Items for System Response Time

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
time_in_super_mode	system tables				
time_in_user_mode	system tables				
time_on_interrupt_stk	system tables				
total_inswaps	system tables				
wait_time_cpu	calculated				
wait_time_io	calculated				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-3.

Data Items for System Workload

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
channel_utilization	calculated				
cpu_service_time	calculated				
cpu_tics	system tables				
cpu_time	Accounting log	X	X		X
cpu_utilization	calculated				
current_nbr_processes	system tables				
cycle_time	calculated				
degree_of_multiprogram	system tables				
demand_zero	system tables				
device_busy_time	calculated				
device_idle_time	system tables				
direct_io_count	Accounting log	X	X		X
disk_reads	system tables				
disk_writes	system tables				
fcg_calls	system tables				
file_lookups	system tables				
file_opens	system tables				
interarrival_time	calculated				
io_service_time	calculated				
job_transit_time	calculated				
memory_requests	system tables				
memory_utilization	calculated				
multi_program_effic	calculated				
number_of_online_users	system tables				
number_volumes_mounted	Accounting log	X	X		X
op_sys_overhead	calculated				
page_fault_frequency	calculated				
pages_read	system tables				
pages_written	system tables				
processor_idle_time	system tables				
queue_length	system tables				
read_io_cnt	system tables				
service_wait_time	calculated				
system_up_time	system tables				
time_in_compat_mode	system tables				
time_in_exec_mode	system tables				
time_in_idle_mode	system tables				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-3.

Data Items for System Workload (continued)

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
time_in_kernal_mode	system tables				
time_in_super_mode	system tables				
time_in_user_mode	system tables				
time_on_interrupt_stk	system tables				
time_queued_to_printer	Accounting log	X	X		X
transaction_rate	calculated				
turnaround_time	calculated				
wait_time_cpu	calculated				
wait_time_io	calculated				
work_input	calculated				
work_output	calculated				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-4.

Data Items for Resource Usage Accounting

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
account_name	Accounting log	X			
allocations	system tables				
buffered_io_count	Accounting log	X	X		X
collided_page_wait	system tables				
component_overlap	calculated				
cpu_time	Accounting log	X	X		X
cpu_utilization	calculated				
degree_of_multiprogram	system tables				
demand_zero	system tables				
device_busy_time	calculated				
device_idle_time	system tables				
direct_io_count	Accounting log	X	X		X
disk_reads	system tables				
disk_writes	system tables				
fcf_calls	system tables				
file_lookups	system tables				
file_opens	system tables				
free_list	system tables				
interarrival_time	calculated				
internal_priority	system tables				
io_req_packets_left	system tables				
mailbox_writes	system tables				
memory_requests	system tables				
modified_list	system tables				
mutex_misc_resource_wt	system tables				
nbr_blocks_lte_32	system tables				
nbr_holes_in_pool	system tables				
nbr_print_lines	Accounting log				
number_of_online_users	system tables				
number_volumes_mounted	Accounting log	X	X		X
page_fault_frequency	calculated				
page_faults	Accounting log	X	X	X	
pages_read	system tables				
pages_written	system tables				
peak_paging_file_size	Accounting log	X	X	X	
peak_working_set_size	Accounting log	X	X	X	
print_job_name	Accounting log	X	X		X
print_queue_name	Accounting log	X	X		X

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-4.

Data Items for Resource Usage Accounting (continued)

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
priority	system tables				
process_cpu_time	system tables				
process_direct_io_cnt	system tables				
process_id	Accounting log	X	X		
process_name	system tables				
process_owner	Accounting log	X	X		
process_size	system tables				
process_user_id_code	system tables				
processor_idle_time	system tables				
quantum_size	system tables				
queue_length	system tables				
read_io_cnt	system tables				
record_sizes	calculated				
smallest_block	system tables				
symbiont_get_count	Accounting log				
symbiont_page_count	Accounting log				
symbiont_qio_count	Accounting log				
termination_time	Accounting log	X	X		
time_in_compat_mode	system tables				
time_in_exec_mode	system tables				
time_in_idle_mode	system tables				
time_in_kernal_mode	system tables				
time_in_super_mode	system tables				
time_in_user_mode	system tables				
time_on_interrupt_stk	system tables				
time_queued_to_printer	Accounting log	X	X		X
total_space_left	system tables				
transaction_count	calculated				
window_turns	system tables				
working_set_size	system tables				
write_io_cnt	system tables				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-5

Data Items for Workload Scheduling

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
account_name	Accounting log	X			
application_speed	calculated				
batch_job_name	Accounting log		X		
compiler_speed	calculated				
cpu_time	Accounting log	X	X		X
degree_of_multiprogram	system tables				
disk_reads	system tables				
disk_writes	system tables				
file_opens	system tables				
job_id	Accounting log	X	X		
job_queue_name	Accounting log	X	X		
job_transit_time	calculated				
nbr_print_lines	Accounting log				
print_job_name	Accounting log	X	X		X
print_queue_name	Accounting log	X	X		X
priority	system tables				
process_cpu_time	system tables				
process_id	Accounting log	X	X		
process_name	system tables				
process_owner	Accounting log	X	X		
process_size	system tables				
process_user_id_code	system tables				
termination_reason	accounting log				
termination_status	Accounting log	X	X		
termination_time	Accounting log	X	X		
transaction_count	calculated				
transaction_rate	calculated				
turnaround_time	calculated				
user_name	Accounting log	X			
write_io_cnt	system tables				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-6.

Data Items for Fine-Tuning Control

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
account_name	Accounting log	X			
allocations	system tables				
availability	calculated				
blocked_time	calculated				
buffered_io_count	Accounting log	X	X		X
cache_hits	system tables				
channel_utilization	calculated				
collided_page_wait	system tables				
component_overlap	calculated				
cpu_service_time	calculated				
cpu_tics	system tables				
cpu_time	Accounting log	X	X		X
cpu_utilization	calculated				
current_nbr_processes	system tables				
cycle_time	calculated				
degree_of_multiprogram	system tables				
demand_zero	system tables				
device_busy_time	calculated				
device_idle_time	system tables				
direct_io	system tables				
direct_io_count	Accounting log	X	X		X
disk_reads	system tables				
disk_writes	system tables				
free_list	system tables				
free_page_wait	system tables				
interarrival_time	calculated				
internal_priority	system tables				
io_req_packets_left	system tables				
io_service_time	calculated				
job_transit_time	calculated				
mailbox_writes	system tables				
memory_requests	system tables				
memory_utilization	calculated				
modified_list	system tables				
multi_program_effic	calculated				
nbr_blocks_lte_32	system tables				
nbr_holes_in_pool	system tables				
nbr_print_lines	Accounting log				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-6.

Data Items for Fine-Tuning Control (continued)

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
number_of_online_users	system tables				
number_volumes_mounted	Accounting log	X	X		X
op_sys_overhead	calculated				
page_fault_frequency	calculated				
pages_read	system tables				
pages_written	system tables				
peak_paging_file_size	Accounting log	X	X	X	
peak_working_set_size	Accounting log	X	X	X	
process_cpu_time	system tables				
process_direct_io_cnt	system tables				
process_faults	system tables				
process_state	system tables				
processor_idle_time	system tables				
quantum_size	system tables				
queue_length	system tables				
read_io_cnt	system tables				
record_sizes	calculated				
response_time	calculated				
service_wait_time	calculated				
smallest_block	system tables				
system_up_time	system tables				
termination_reason	accounting log				
time_in_compat_mode	system tables				
time_in_exec_mode	system tables				
time_in_idle_mode	system tables				
time_in_kernal_mode	system tables				
time_in_super_mode	system tables				
time_in_user_mode	system tables				
time_on_interrupt_stk	system tables				
transaction_count	calculated				
transaction_rate	calculated				
turnaround_time	calculated				
wait_time_cpu	calculated				
wait_time_io	calculated				
window_turns	system tables				
working_set_size	system tables				
write_io_cnt	system tables				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Table B-7.

Consolidated List of Data Items

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
account_name	Accounting log	X			
allocations	system tables				
application_speed	calculated				
availability	calculated				
batch_job_name	Accounting log		X		
blocked_time	calculated				
buffered_io_count	Accounting log	X	X		X
cache_hits	system tables				
channel_utilization	calculated				
collided_page_wait	system tables				
common_event_flag_wait	system tables				
compiler_speed	calculated				
component_overlap	calculated				
cpu_service_time	calculated				
cpu_tics	system tables				
cpu_time	Accounting log	X	X		X
cpu_utilization	calculated				
current_nbr_processes	system tables				
cycle_time	calculated				
degree_of_multiprogram	system tables				
demand_zero	system tables				
device_busy_time	calculated				
device_idle_time	system tables				
direct_io	system tables				
direct_io_count	Accounting log	X	X		X
disk_reads	system tables				
disk_writes	system tables				
fcp_calls	system tables				
file_lookups	system tables				
file_opens	system tables				
free_list	system tables				
free_page_wait	system tables				
hibernate_wait	system tables				
hibernate_wait_outswap	system tables				
interarrival_time	calculated				
internal_priority	system tables				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Consolidated List of Data Items (continued)

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
io_req_packets_left	system tables				
io_service_time	calculated				
job_id	Accounting log	X	X		
job_queue_name	Accounting log	X	X		
job_transit_time	calculated				
largest_block	system tables				
loc_ev_flag_wait	system tables				
loc_ev_flag_wait_oswap	system tables				
login_time	Accounting log	X	X	X	X
mailbox_writes	system tables				
memory_requests	system tables				
memory_utilization	calculated				
modified_list	system tables				
multi_program_effic'y	calculated				
mutex_misc_resource_wt	system tables				
nbr_blocks_lte_32	system tables				
nbr_holes_in_pool	system tables				
nbr_print_lines	Accounting log				
number_of_online_users	system tables				
number_volumes_mounted	Accounting log	X	X		X
op_sys_overhead	calculated				
page_fault_frequency	calculated				
page_faults	Accounting log	X	X	X	
pages_read	system tables				
pages_written	system tables				
peak_paging_file_size	Accounting log	X	X	X	
peak_working_set_size	Accounting log	X	X	X	
print_job_name	Accounting log	X	X		X
print_queue_name	Accounting log	X	X		X
priority	system tables				
process_cpu_time	system tables				
process_direct_io_cnt	system tables				
process_faults	system tables				
process_id	Accounting log	X	X		
process_name	system tables				
process_owner	Accounting log	X	X		
process_size	system tables				

PERFORMANCE MEASURES AND ASSOCIATED DATA ELEMENTS

Consolidated List of Data Items (continued)

Data Item	Source of Data	--DATA GROUP--			
		O	P	M	D
process_state	system tables				
process_user_id_code	system tables				
processor_idle_time	system tables				
quantum_size	system tables				
queue_length	system tables				
read_io_cnt	system tables				
record_sizes	calculated				
response_time	calculated				
service_wait_time	calculated				
smallest_block	system tables				
suspend_wait	system tables				
suspend_wait_outswap	system tables				
symbiont_get_count	Accounting log				
symbiont_page_count	Accounting log				
symbiont_qio_count	Accounting log				
system_faults	system tables				
system_up_time	system tables				
termination_reason	accounting log				
termination_status	Accounting log	X	X		
termination_time	Accounting log	X	X		
time_in_compat_mode	system tables				
time_in_exec_mode	system tables				
time_in_idle_mode	system tables				
time_in_kernal_mode	system tables				
time_in_super_mode	system tables				
time_in_user_mode	system tables				
time_on_interrupt_stk	system tables				
time_queued_to_printer	Accounting log	X	X		X
total_inswaps	system tables				
total_space_left	system tables				
transaction_count	calculated				
transaction_rate	calculated				
turnaround_time	calculated				
user_name	Accounting log	X			
wait_time_cpu	calculated				
wait_time_io	calculated				
window_turns	system tables				
work_input	calculated				
work_output	calculated				
working_set_size	system tables				
write_in_progress	system tables				
write_io_cnt	system tables				

APPENDIX C

DIALOGUE FRAMES USED BY PERMTOOL

The following list of frames was produced by the PERMTOOL utility FRAMEFIG and represents the frames developed as of 1 September, 1982. For the most part, these frames are those required to gather job/process data collection parameters from the user. Some of the slides produce error messages or provide high-level responses to user requests for non-operational services.

Table C-1. Dialogue Frame Index

FRAME	Page nbr	FRAME	Page nbr
INTRO1	C-2	CJB004	C-15
MENU01	C-3	BCJB04	C-16
BMEN01	C-4	CJB04A	C-17
NAVAIL	C-5	CJB005	C-18
FINISH	C-6	CJB05A	C-19
COL001	C-7	GCJB05	C-20
BCOL01	C-8	GCJB5A	C-21
CJB001	C-9	GCJB5B	C-22
BCJB01	C-10	BCJB05	C-23
CJB002	C-11	BCJB5A	C-24
BCJB02	C-12		
CJB003	C-13		
CJB03A	C-14		

DIALOGUE FRAMES USED BY PERMTOOL

frame name: INTI01

clear screen - YES reverse video - YES
user answer required - NO

answer processors: validation -
bad answer -
good answer -
default next - MENU01

frame positioning required - YES
frame position - left column - 1
width - 80
first line - 1

FRAME LAYOUT

1 2 3 4 5 6
....5....0....5....0....5....0....5....0....5....0....5....

PPPPPP	EEEEEEE	RRRRRR	M	M	TTTTTT	00000	00000	L
P	P	E	R	R	MM	MM	T	O
P	P	E	R	R	M	M	M	T
PPPPPP	EEEE	RRRRRR	M	M	M	T	O	O
P	E	R	R	M	M	T	O	O
P	E	R	R	M	M	T	O	O
P	EEEEEEE	R	R	M	M	T	00000	00000

>>> VAX PERFORMANCE MONITORING/ANALYSIS TOOL <<<

COPYRIGHT 1982 G. B. LaMont, E. C. Gilpin

DIALOGUE FRAMES USED BY PERMTOOL

frame name: MENU01

clear screen - NO reverse video - NO
user answer required - YES

answer processors: validation - VMEN01
bad answer - BMEN01
good answer - GMEN01
default next - MENU01

frame positioning required - NO
frame position - left column - 0
width - 0
first line - 0

FRAME LAYOUT

 1 2 3 4 5 6
....5....0....5....0....5....0....5....0....5....0....5....0

PLEASE SELECT AN OPTION FROM THE FOLLOWING LIST:

ANALYZE	HELP	STOP
COLLECT	MAIL	TALK
DISPLAY	SHOW	WATCH
EXIT		

YOUR CHOICE:

DIALOGUE FRAMES USED BY PERMTOOL

frame name: BMEN01

clear screen - NO reverse video - YES
user answer required - YES

answer processors: validation - VMEN01
bad answer - BMEN01
good answer - GMEN01
default next - MENU01

frame positioning required - YES
frame position - left column - 50
width - 29
first line - 18

FRAME LAYOUT

	1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0						

SORRY, THAT COMMAND IS NOT
LEGAL. PLEASE SELECT ONE
FROM THE LIST.

DIALOGUE FRAMES USED BY PERMTOOL

frame name: NAVAIL

clear screen - NO reverse video - YES
user answer required - YES

answer processors: validation - VMEN01
 bad answer - BMEN01
 good answer - GMEN01
 default next - MENU01

frame positioning required - YES
frame position - left column - 50
 width - 29
 first line - 18

FRAME LAYOUT

	1	2	3	4	5	6
.....	5.....	0.....	5.....	0.....	5.....	0.....

SORRY, THAT COMMAND IS NOT
AVAILABLE AT THIS TIME.
PLEASE MAKE ANOTHER CHOICE.

DIALOGUE FRAMES USED BY PERMTOOL

frame name: FINISH

```
clear screen - YES          reverse video - NO
      user answer required - NO
```

```

answer processors: validation -
                  bad answer  -
                  good answer -
                  default next - FINISH

```

```

frame positioning required - YES
frame position  - left column  - 1
                  width         - 79
                  first line    - 1

```

FRAME LAYOUT

.....1.....2.....3.....4.....5.....6.....7.....
5.....0.....5.....0.....5.....0.....5.....0.....5.....0.....5.....0.....5.....0.....5.....0.....

[illegible]

DIALOGUE FRAMES USED BY PERMTOOL

frame name: COL001

clear screen - YES reverse video - NO
user answer required - YES

answer processors: validation - VCOL01
bad answer - BCOL01
good answer - GCOL01
default next - COL001

frame positioning required - YES
frame position - left column - 1
width - 79
first line - 4

FRAME LAYOUT

1 2 3 4 5 6 7 8
....5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0

DATA COLLECTION REQUESTED. DATA COLLECTION OPTIONS FALL INTO FOUR CATEGORIES.
PLEASE SELECT ONE OF THOSE CATEGORIES FROM THE LIST OR TYPE 'QUIT' TO RETURN
TO THE MAIN MENU:

TO SELECT CATEGORY:

TYPE:

MEMORY PARAMETER AND USE DATA
JOB/PROCESS RELATED DATA
PROCESSOR UTILIZATION DATA
I/O DEVICE UTILIZATION DATA
RETURN TO MAIN MENU

MEMORY
JOB
CPU
IO
QUIT

YOUR CHOICE:

DIALOGUE FRAMES USED BY PERMTOOL

frame name: BOOL01

clear screen - NO reverse video - YES
user answer required - NO

answer processors: validation -
bad answer -
good answer -
default next - COL001

frame positioning required - YES
frame position - left column - 50
width - 29
first line - 18

FRAME LAYOUT

	1	2	3	4	5	6
....5....	0....5....	0....5....	0....5....	0....5....	0....5....	0

SORRY, THAT COMMAND IS NOT
LEGAL. PLEASE SELECT ONE
FROM THE LIST.

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB001

clear screen - YES reverse video - NO
user answer required - YES

answer processors: validation - VCJB01
 bad answer - BCJB01
 good answer - GCJB01
 default next - CJB002

frame positioning required - YES
frame position - left column - 1
 width - 80
 first line - 5

FRAME LAYOUT

1	2	3	4	5	6	7
....5....0....5....0....5....0....5....0....5....0....5....0....5						

IN ORDER TO COLLECT DATA ABOUT JOBS WE NEED SEVERAL PIECES OF INFORMATION.
THIS WILL BE COLLECTED DURING THE FOLLOWING DIALOGUE.

IF, AT ANY TIME, YOU WANT TO QUIT, TYPE THE WORD 'QUIT'.
NOTE, HOWEVER, THAT THIS WILL CAUSE PERMTOOL TO FORGET ANY
INFORMATION ALREADY PROVIDED. NOW, FOR THE FIRST QUESTION:

WHAT TIME DO YOU WANT DATA COLLECTION TO START
(DD-~~MMM~~-YYYY HH:MM:SS.nn)?

DIALOGUE FRAMES USED BY PERMTOOL

frame name: BCJB01

clear screen - NO reverse video - YES
user answer required - NO

answer processors: validation -
 bad answer -
 good answer -
 default next - CJB001

frame positioning required - YES
frame position - left column - 40
 width - 39
 first line - 13

FRAME LAYOUT

	1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0						

^ SORRY, THAT ANSWER IS NOT VALID.
| EITHER TYPE 'QUIT' OR A DATE/TIME
| IN THE FORMAT SHOW ABOVE.

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB002

clear screen - NO reverse video - NO
user answer required - YES

answer processors: validation - VCJB02
bad answer - BCJB02
good answer - GCJB02
default next - CJB003

frame positioning required - YES
frame position - left column - 1
width - 80
first line - 5

FRAME LAYOUT

 1 2 3 4 5 6 7
....5....0....5....0....5....0....5....0....5....0....5....0....5....

DO YOU WANT DATA COLLECTION TO RUN UNTIL A CERTAIN TIME? IF YOU DON'T, THEN
COLLECTION WILL OCCUR FOR A SPECIFIED NUMBER OF SAMPLES.

YOUR CHOICE (ANSWER 'TIME', 'SAMPLES' OR 'QUIT'):

DIALOGUE FRAMES USED BY PERMTOOL

frame name: BCJB02

clear screen - NO reverse video - YES
user answer required - NO

answer processors: validation -
 bad answer -
 good answer -
 default next - CJB002

frame positioning required - YES
frame position - left column - 30
 width - 35
 first line - 10

FRAME LAYOUT

	1	2	3	4	5	6
....5....	0....5....	0....5....	0....5....	0....5....	0....5....	0

SORRY - YES/NO/QUIT ARE THE ONLY
ACCEPTABLE ANSWERS.

^
|
|

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB003

clear screen - NO reverse video - NO
user answer required - NO

answer processors: validation -
 bad answer -
 good answer -
 default next - CJB03A

frame positioning required - YES
frame position - left column - 1
 width - 80
 first line - 5

FRAME LAYOUT

1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0					

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB03A

clear screen - NO reverse video - NO
user answer required - YES

answer processors: validation - VCJB03
bad answer - BCJB03
good answer - GCJB03
default next - CJB004

frame positioning required - YES
frame position - left column - 1
width - 80
first line - 5

FRAME LAYOUT

 1 2 3 4 5 6
....5....0....5....0....5....0....5....0....5....0....5....0

ENTER THE DATE/TIME WHEN YOU WANT DATA COLLECTION TO STOP
(DD-MMM-YYYY HH:MM:SS.nn)

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB004

clear screen - NO reverse video - NO
user answer required - NO

answer processors: validation -
 bad answer -
 good answer -
 default next - CJB04A

frame positioning required - YES
frame position - left column - 1
 width - 79
 first line - 5

FRAME LAYOUT

1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0					

DIALOGUE FRAMES USED BY PERMTOOL

frame name: BCJB04

clear screen - NO reverse video - YES
user answer required - NO

answer processors: validation -
bad answer -
good answer -
default next - CJB04A

frame positioning required - YES
frame position - left column - 15
width - 40
first line - 7

FRAME LAYOUT

	1	2	3	4	5	6
....5....	0....5....	0....5....	0....5....	0....5....	0....5....	0

THAT IS NOT A VALID ANSWER. EITHER
TYPE 'QUIT' OR A NUMBER BETWEEN
1 AND 9,999,999.

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB04A

clear screen - NO reverse video - NO
user answer required - YES

answer processors: validation - VCJB04
bad answer - BCJB04
good answer - GCJB04
default next - CJB005

frame positioning required - YES
frame position - left column - 1
width - 79
first line - 5

FRAME LAYOUT

1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0					

HOW MANY EVENTS DO YOU WANT TO COLLECT (< 9,999,999)?

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB005

clear screen - NO reverse video - NO
user answer required - NO

answer processors: validation -
bad answer -
good answer -
default next - CJB05A

frame positioning required - YES
frame position - left column - 1
width - 79
first line - 5

FRAME LAYOUT

	1	2	3	4	5	6
....5....	0....5....	0....5....	0....5....	0....5....	0....5....	0

DIALOGUE FRAMES USED BY PERMTOOL

frame name: CJB05A

clear screen - NO reverse video - NO
'user answer required - YES

answer processors: validation - VCJB05
 bad answer - BCJB05
 good answer - GCJB05
 default next - INTROL

frame positioning required - YES
frame position - left column - 1
 width - 79
 first line - 5

FRAME LAYOUT

1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0					

NOW YOU MUST SELECT THOSE DATA ELEMENTS TO BE COLLECTED.

IN A MOMENT A LIST OF POSSIBLE ELEMENTS WILL BE DISPLAYED,
AND YOU WILL BE ASKED TO PICK THOSE ELEMENTS DESIRED. TO DO
THIS, SIMPLY TYPE THE NAME (AS SHOWN) OF THE DATA ITEM.
THE ITEM NAME WILL THEN BE HIGH-LIGHTED TO CONFIRM YOUR CHOICE.

WHEN YOU HAVE FINISHED YOUR SELECTIONS, TYPE THE WORD 'DONE'.

IF YOU ARE UNCERTAIN ABOUT A DATA NAME, TYPE 'HELP' FOLLOWED
BY THE DATA NAME. YOU WILL GET A BRIEF EXPLANATION OF THE
MEANING OF THE DATA ELEMENT.

REMEMBER - YOU MAY TYPE 'QUIT' AT ANY TIME.

PRESS ANY KEY TO CONTINUE.....

DIALOGUE FRAMES USED BY PERMTOOL

frame name: GCJB05

clear screen - NO reverse video - NO
user answer required - NO

answer processors: validation -
bad answer -
good answer -
default next - GCJB5A

frame positioning required - YES
frame position - left column - 1
width - 79
first line - 3

FRAME LAYOUT

1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0					

DIALOGUE FRAMES USED BY PERMTOOL

frame name: GCJB5A

clear screen - NO reverse video - NO
user answer required - NO

answer processors: validation -
bad answer -
good answer -
default next - GCJB5B

frame positioning required - YES
frame position - left column - 1
width - 79
first line - 15

FRAME LAYOUT

	1	2	3	4	5	6
....5....	0....5....	0....5....	0....5....	0....5....	0....5....	0

DIALOGUE FRAMES USED BY PERMTOOL

frame name: GCJB5B

clear screen - NO reverse video - NO
user answer required - YES

answer processors: validation - VGCJB5
bad answer -
good answer -
default next - GCJB5B

frame positioning required - YES
frame position - left column - 1
width - 79
first line - 15

FRAME LAYOUT

1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0					

TYPE THE NAMES OF THE DESIRED DATA ELEMENTS:

DIALOGUE FRAMES USED BY PERMTOOL

frame name: BCJB05

clear screen - NO reverse video - YES
user answer required - NO

answer processors: validation -
 bad answer -
 good answer -
 default next - BCJB5A

frame positioning required - YES
frame position - left column - 40
 width - 35
 first line - 18

FRAME LAYOUT

	1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0						

UNRECOGNIZED DATA NAME. PLEASE
RE-ENTER THE DATA NAME.

DIALOGUE FRAMES USED BY PERMTOOL

frame name: BCJB5A

clear screen - NO reverse video - NO
user answer required - NO

answer processors: validation -
bad answer -
good answer -
default next - BCJB5A

frame positioning required - YES
frame position - left column - 1
width - 79
first line - 16

FRAME LAYOUT

1	2	3	4	5	6
....5....0....5....0....5....0....5....0....5....0....5....0					

APPENDIX D

USER INTERFACE DATA STRUCTURES

D.1 DIALOGUE FRAME SUPPORT STRUCTURES

DIALOGUE FRAME DEFINITIONS —

The following definitions are used for the dialog frames.
Included are definitions of:

1. Pointer to the text of a computer remark
2. Pointer to a user response validation routine
3. Pointer to an invalid response processing routine
4. Pointer to a valide response processing routine
5. Indicator to turn on reverse video for the computer remark.
6. Indicator to clear the screen before displaying the computer remark.
7. Indicator to wait for a user response after remark
8. Display positioning indicators
9. Frame identification

USER INTERFACE DATA STRUCTURES

```
{
{ SCREEN_FRAME LIBRARY RECORDS — These records are
{ formatted for storage on a disk file when the
{ PERMTOOL system executive is not in use. The intent
{ of this format is to facilitate modifications to a
{ frame through the use of text editors. Otherwise,
{ it would be necessary to rebuild the entire frame
{ using the FRAMEDIT utility.
{ }
```

screen_lib_rec = record

```

case rec_type: two_way_sw of
1: (screen_id : frame_key;
    clr_first,
    black_white,
    answer_exp : two_way_sw;
    def_id,
    valid_id,
    bad_id,
    good_id : action_key;
    position : two_way_sw;
    left,
    wide,
    down : integer);
2: (text_data : text_80);

{ rec type 1 - a frame header
{ rec type 2 - a frame text line
{ symbolic id of this frame
{
{ true —> clear screen then
{ display text
{ true —> reverse video display
{ true —> user must respond
{ before next frame is displayed
{
{ process id of validation proc
{ process id of invalid user
{ response procedure
{ process id of valid user
{ response procedure
{
{ true —> screen positioning
{ is required for this frame
{ left-most column of screen,
{ in range 1..MAX_NBR_COLUMNS
{ width of screen in range
{ 1..MAX_NBR_COLUMNS
{ starting line nbr of frame
{ in range 1..MAX_NBR_ROWS
{
{ one line of text for the frame}

end; (* SCREEN_LIB_REC record def'n *)
```

USER INTERFACE DATA STRUCTURES

```

{
ACTIVE FRAME USE FORMAT — This type describes the
frame format used by the PERMTOOL system during
operation of the executive. Many of these records
are stored as a linked list, and control how the
user-interface module operates. In effect, these
frames define a complete state_transition table for
that module.
}

```

```

frame_item = record
  frame_id          : frame_key; { Name of the frame }
  next_frame_ptr    : ^frame_item; { Links this frame to the next }
  clr_screen_first, { TRUE —> screen must be
                    { cleared before display }
  black_on_white    : boolean; { TRUE —> Reverse video display }
  text_head         : ^text_rec; { first line of text for the
                    { screen. }
  answer_expected   : boolean; { TRUE —> User must respond }
  default_id        : frame_key; { Processing routine names }
  validation_id,    { . . . . . }
  bad_answer_id,    { . . . . . }
  good_answer_id    : action_key; { . . . . . }

  case position_display: boolean of { TRUE —> preplanned position }
    { for display on screen }
    false : ();
    true  : (left_column, { Location parameters }
             width,
             row : integer);

end; (* FRAME_ITEM record description *)

```

USER INTERFACE DATA STRUCTURES

```
{
  FRAME DICTIONARY DEFINITIONS — Used to support the
  frame dictionary. This data structure is used to
  find frames which have been stored in dynamic
  memory.
}
```

```
frame_ptr = ^frame_item;
```

```
f_dict_item = record      { storage format for a frame dictionary }
  f_name : frame_key;      { entry. It contains the frame name,   }
  f_loc  : frame_ptr;      { and a pointer to the frame's location }
end;  (* F_DICT_ITEM record def'n  *)
```

```
dict_entry_range = 1..last_dict_entry; { index range of frame diction. }
```

D.2 OPTION STORAGE STRUCTURES

```
command_rec = record
  cmd_code : prm$_f_range; { Numeric encoding of command plus }
                                { options. }
  cmd_name : text_12;      { Current response }
  cmd_parms : text_line;   { Complete user input command line }
end;  (* COMMAND_REC record def'n  *)
```

USER INTERFACE DATA STRUCTURES

D.3 REQUEST COMMUNICATION STRUCTURES

```
{ This record structure is used to tell the data }  
{ collection module what work must be done. }  
{ }
```

```
ctl_dc_rec      = record  
  start_time    : quad_word;  
  case end_switch : dc_limit_type of  
    stop_tm     : (stop_time      : quad_word);  
    event_cnt    : (event_limit    : integer);  
  end;          (* ctl_dc_rec record def'n  *)  
  
data_flags      = packed record  
  ctl_flags     : ctl_dc_rec;  
  cpu_flags     : packed array [1..n_cpu_items] of boolean;  
  mem_flags     : packed array [1..n_mem_items] of boolean;  
  io_flags      : packed array [1..n_io_items] of boolean;  
  proc_flags    : packed array [1..n_proc_items] of boolean;  
end;            (* data flags record definition *)
```

APPENDIX E

USER INTERFACE ORGANIZATIONAL STRUCTURES

USER INTERFACE ORGANIZATIONAL STRUCTURES

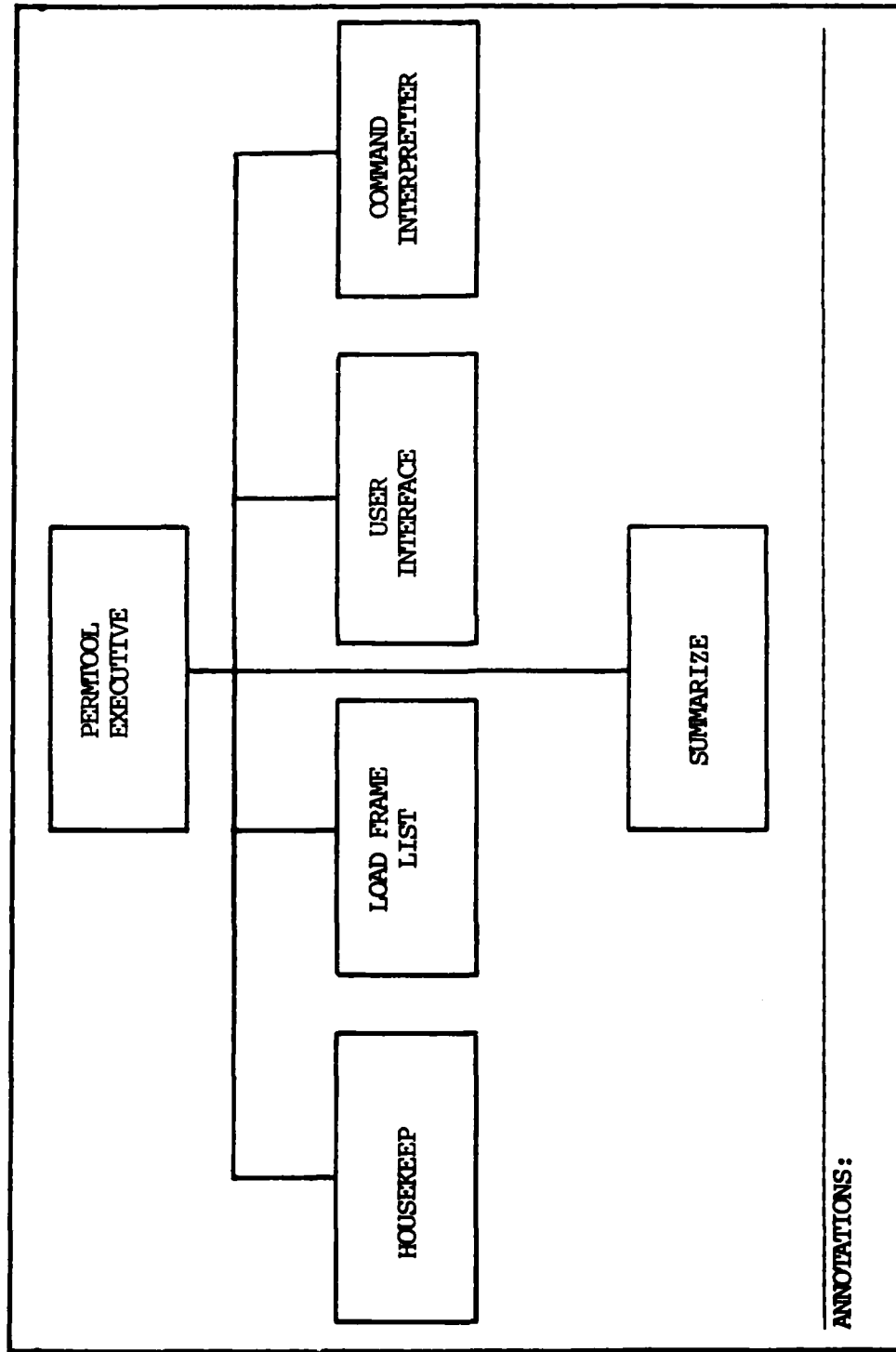


Figure E-1. Structure Diagram - PERMTOOL Executive

USER INTERFACE ORGANIZATIONAL STRUCTURES

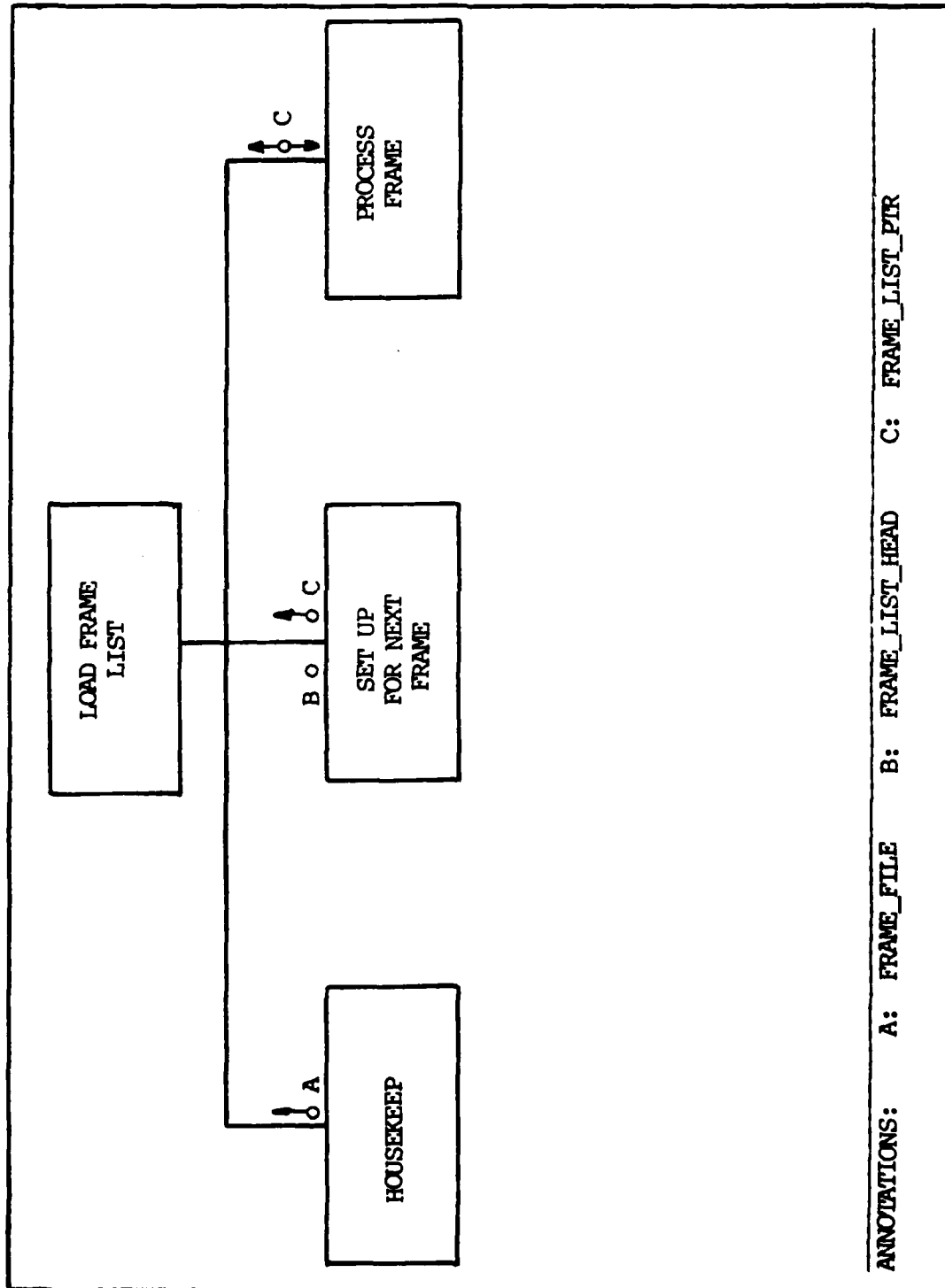


Figure E-2. Structure Diagram - Load Frame List

USER INTERFACE ORGANIZATIONAL STRUCTURES

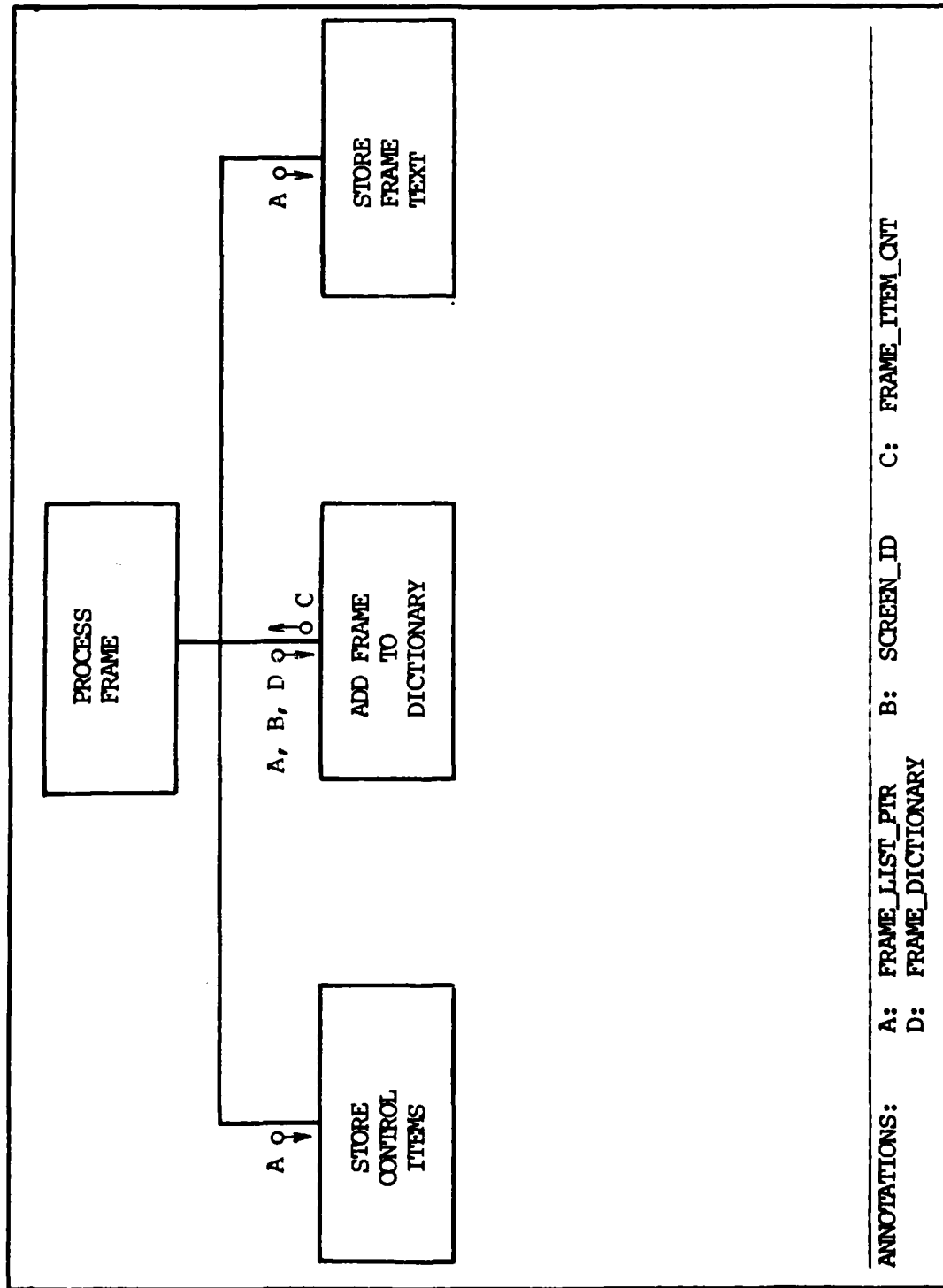


Figure E-3. Structure Diagram - Process Frame

USER INTERFACE ORGANIZATIONAL STRUCTURES

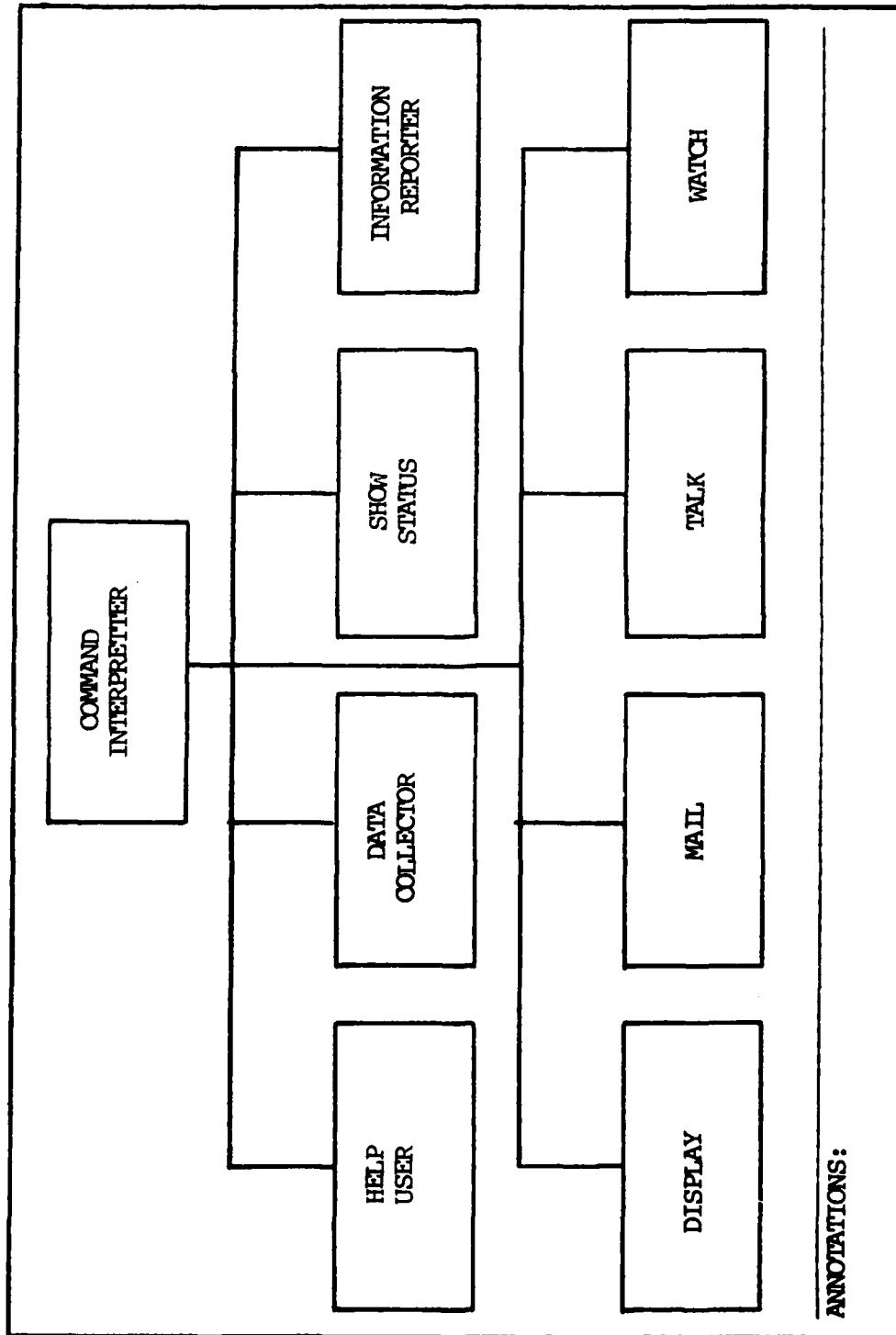


Figure E-4. Structure Diagram - Command Interpreter

USER INTERFACE ORGANIZATIONAL STRUCTURES

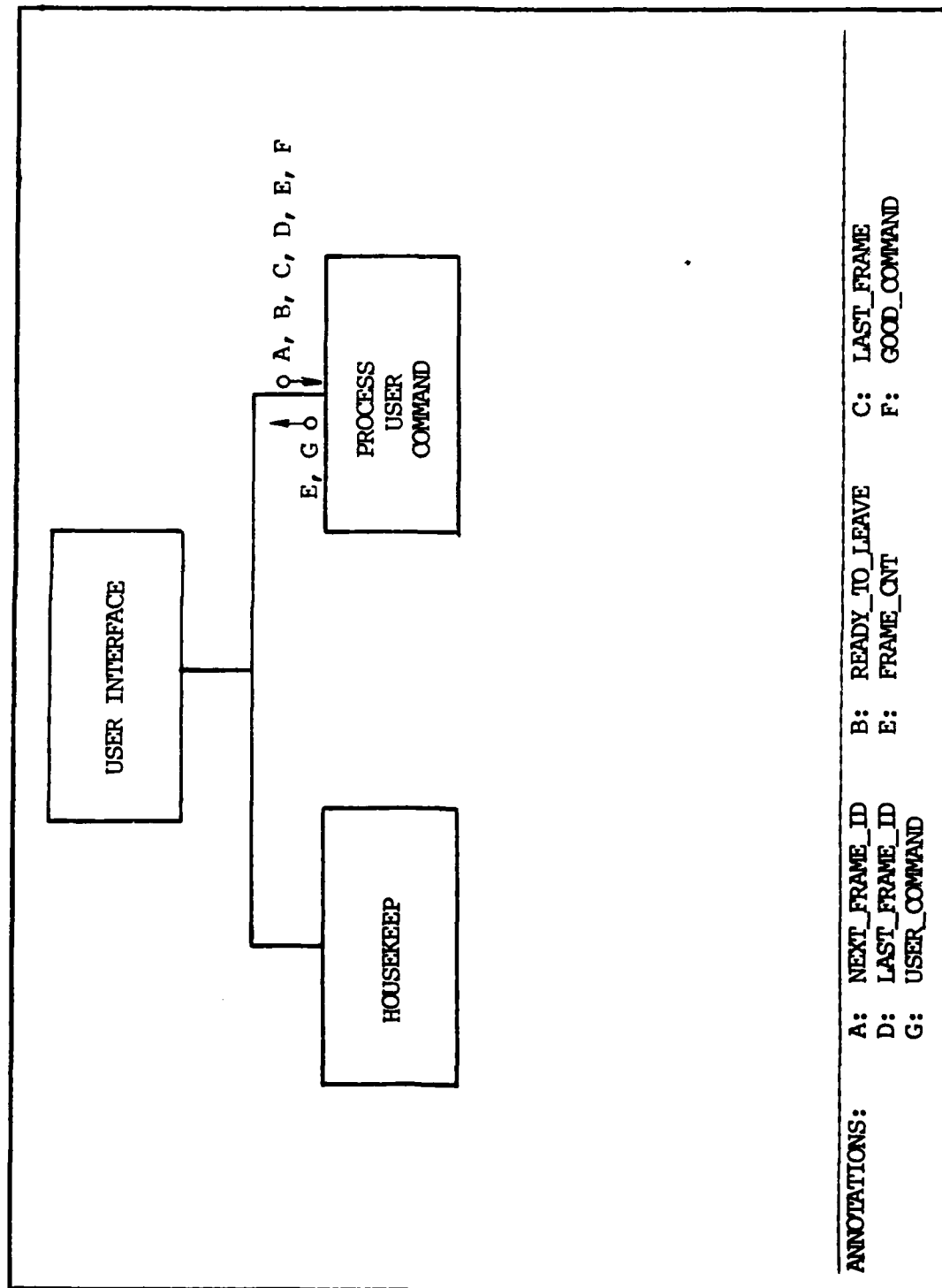


Figure E-5. Structure Diagram - User Interface

USER INTERFACE ORGANIZATIONAL STRUCTURES

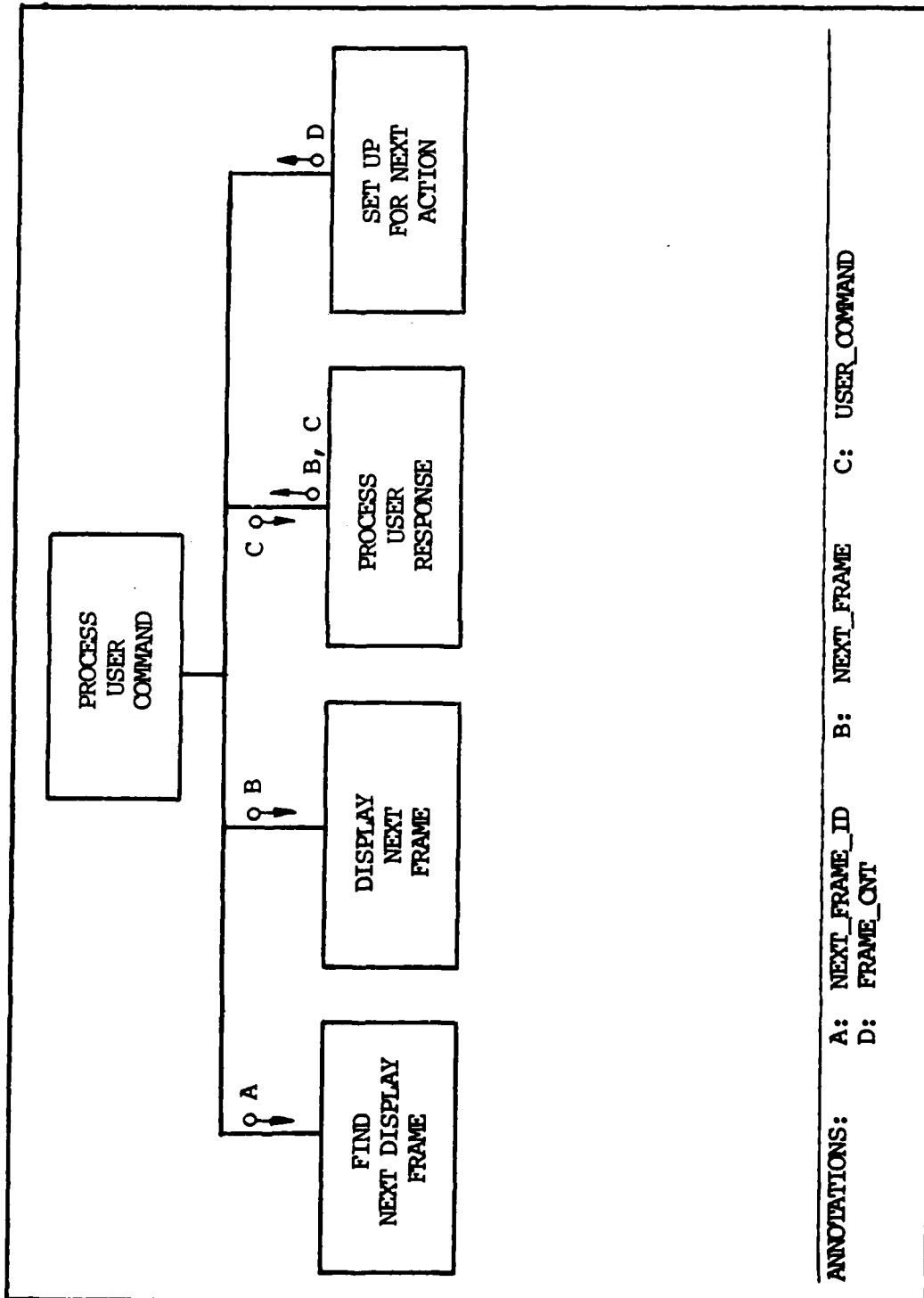


Figure E-6. Structure Diagram - Process user Command

USER INTERFACE ORGANIZATIONAL STRUCTURES

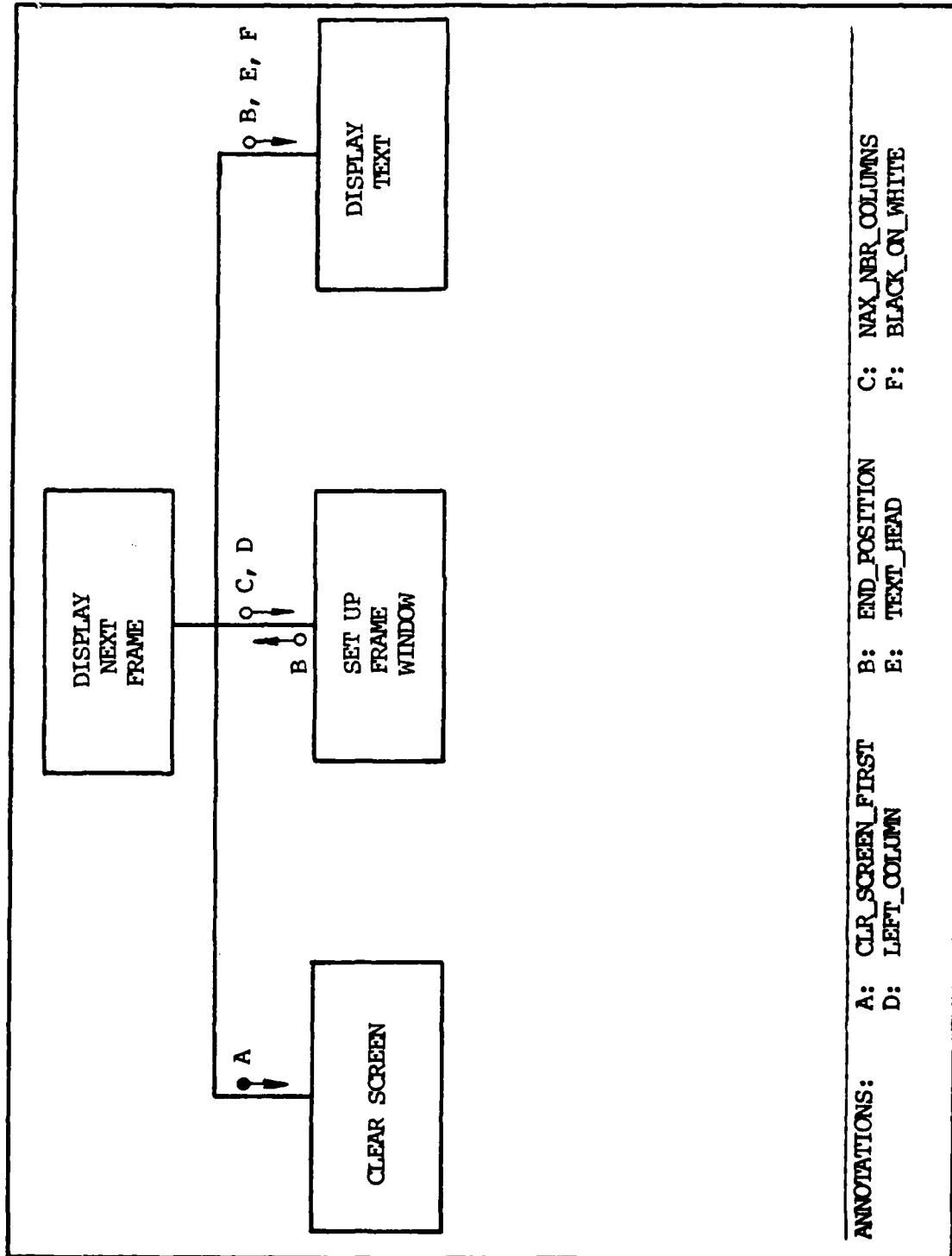


Figure E-7. Structure Diagram - Display Next Frame

USER INTERFACE ORGANIZATIONAL STRUCTURES

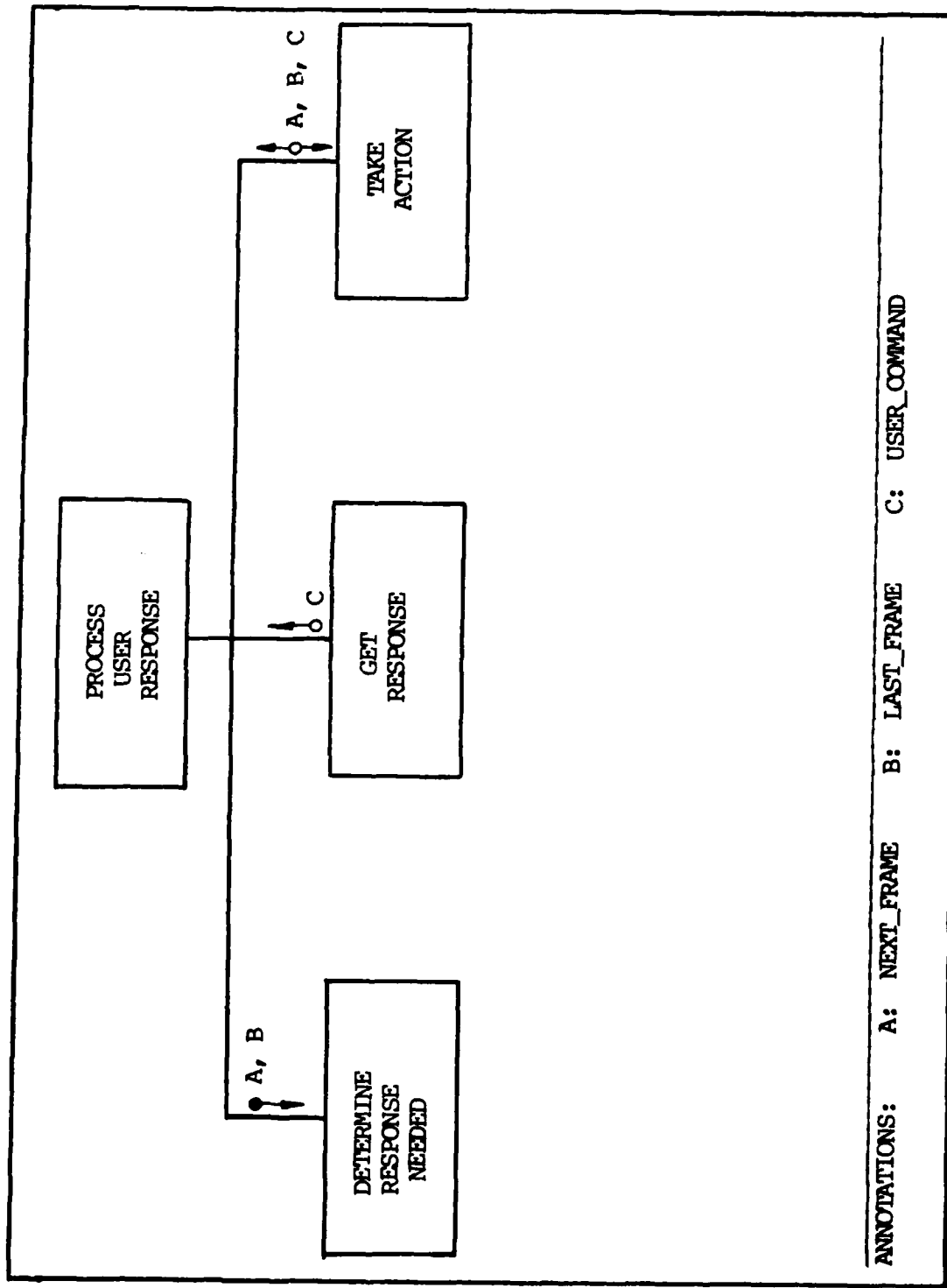


Figure E-8. Structure Diagram - Process User Response

USER INTERFACE ORGANIZATIONAL STRUCTURES

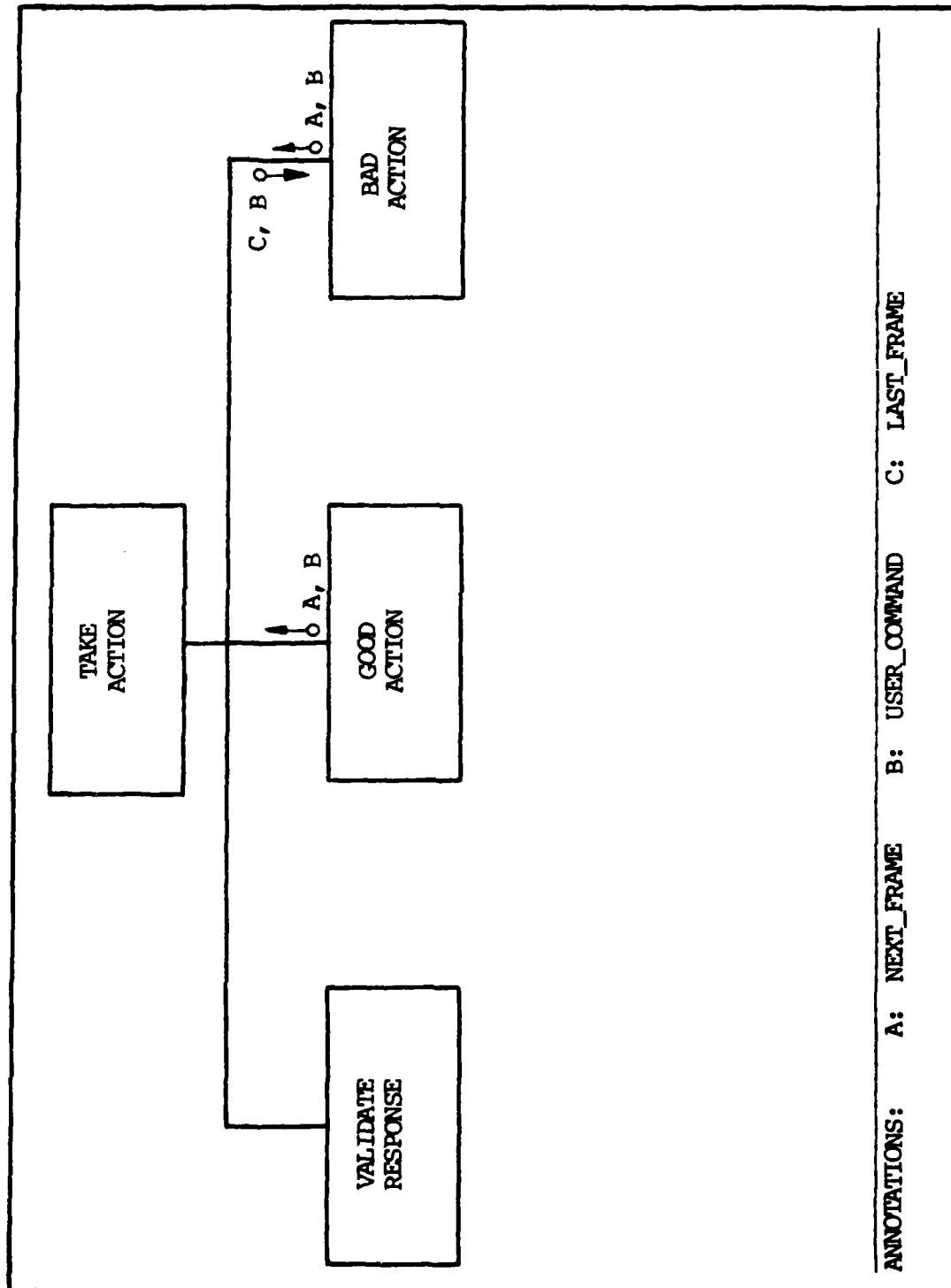


Figure E-9. Structure Diagram - Take Action

USER INTERFACE ORGANIZATIONAL STRUCTURES

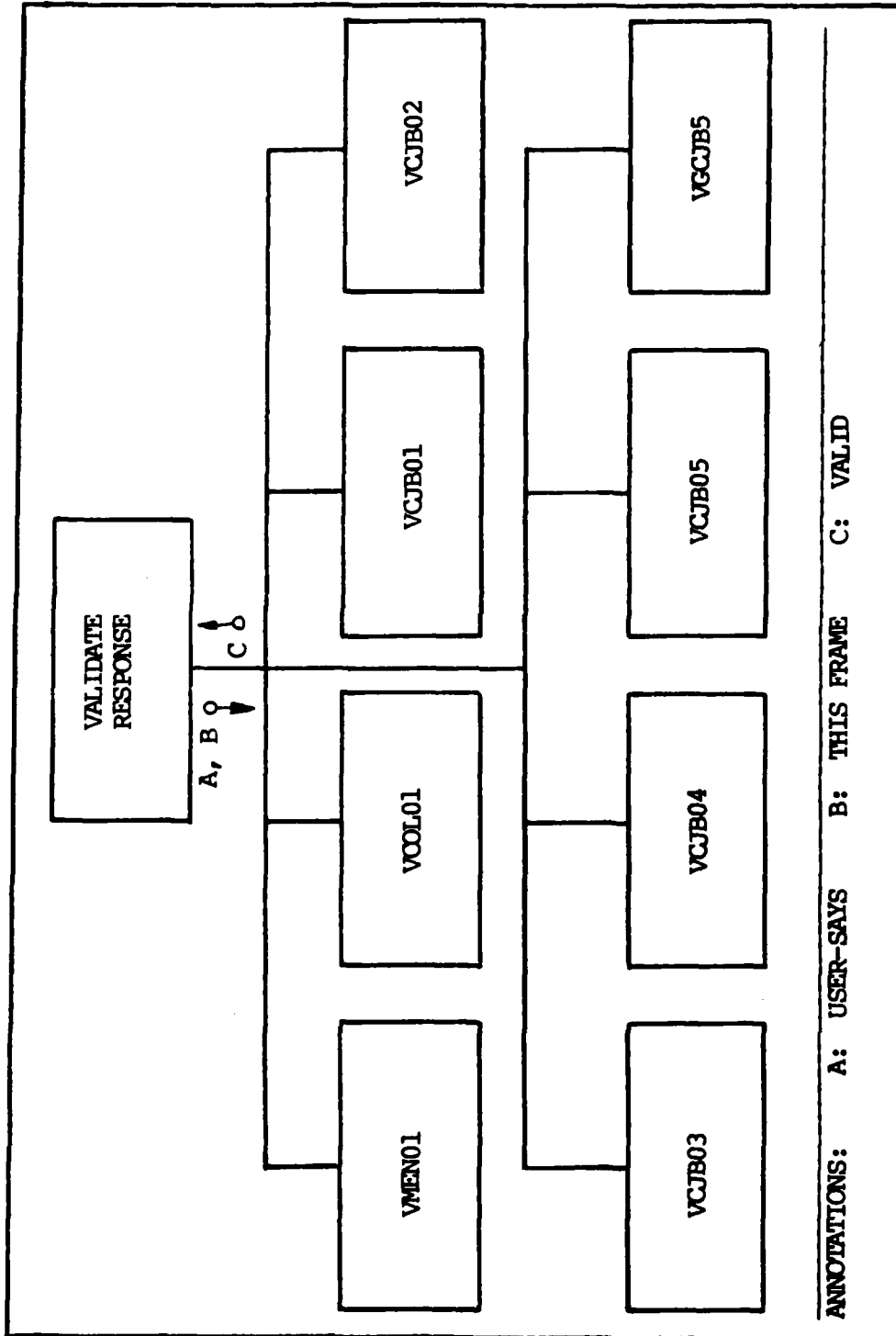


Figure E-10. Structure Diagram - Validate Response

USER INTERFACE ORGANIZATIONAL STRUCTURES

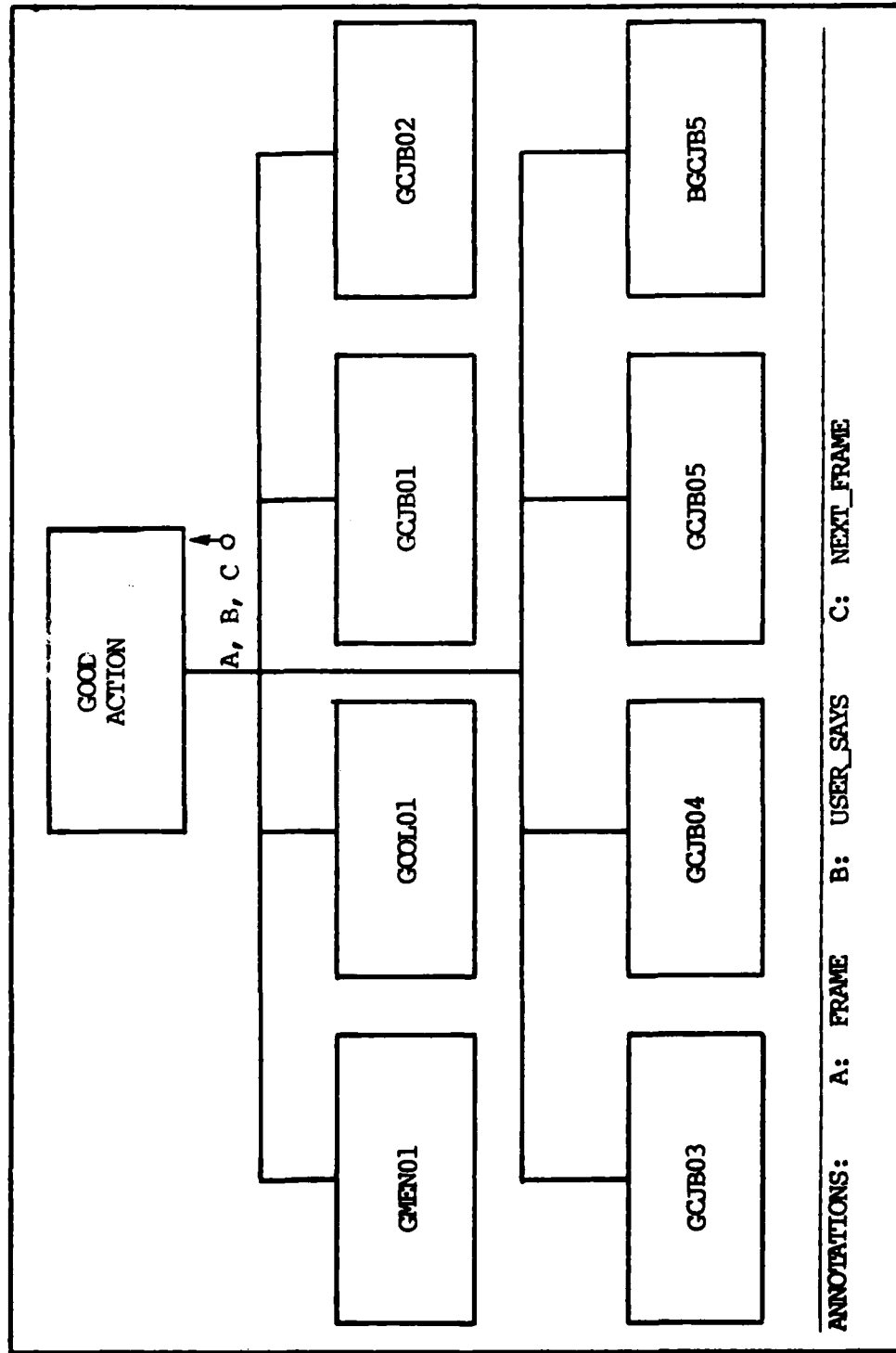


Figure E-11. Structure Diagram - Good Action

USER INTERFACE ORGANIZATIONAL STRUCTURES

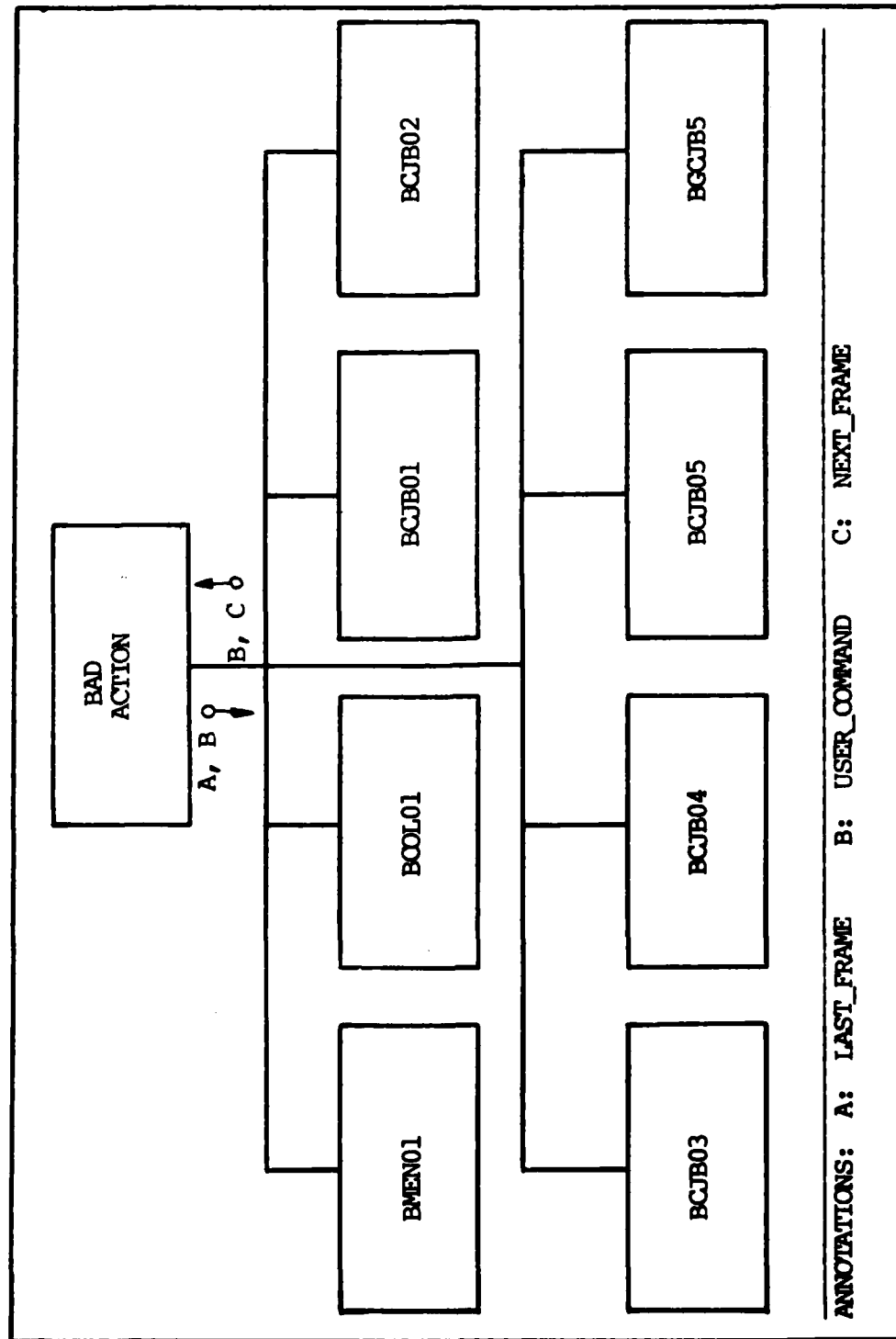


Figure E-12. Structure Diagram - Bad Action

USER INTERFACE ORGANIZATIONAL STRUCTURES

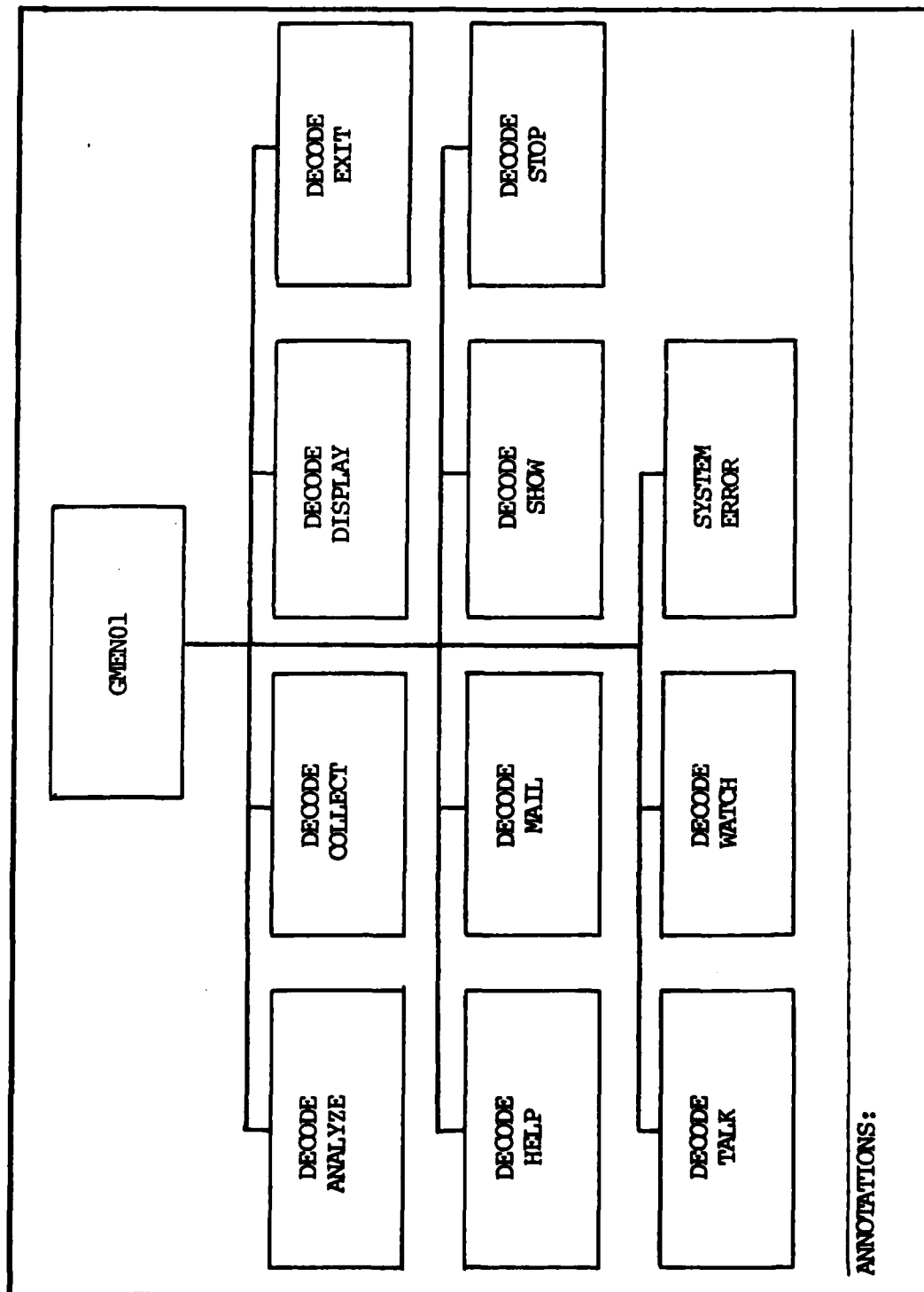


Figure E-13. Structure Diagram - GMEN01

USER INTERFACE ORGANIZATIONAL STRUCTURES

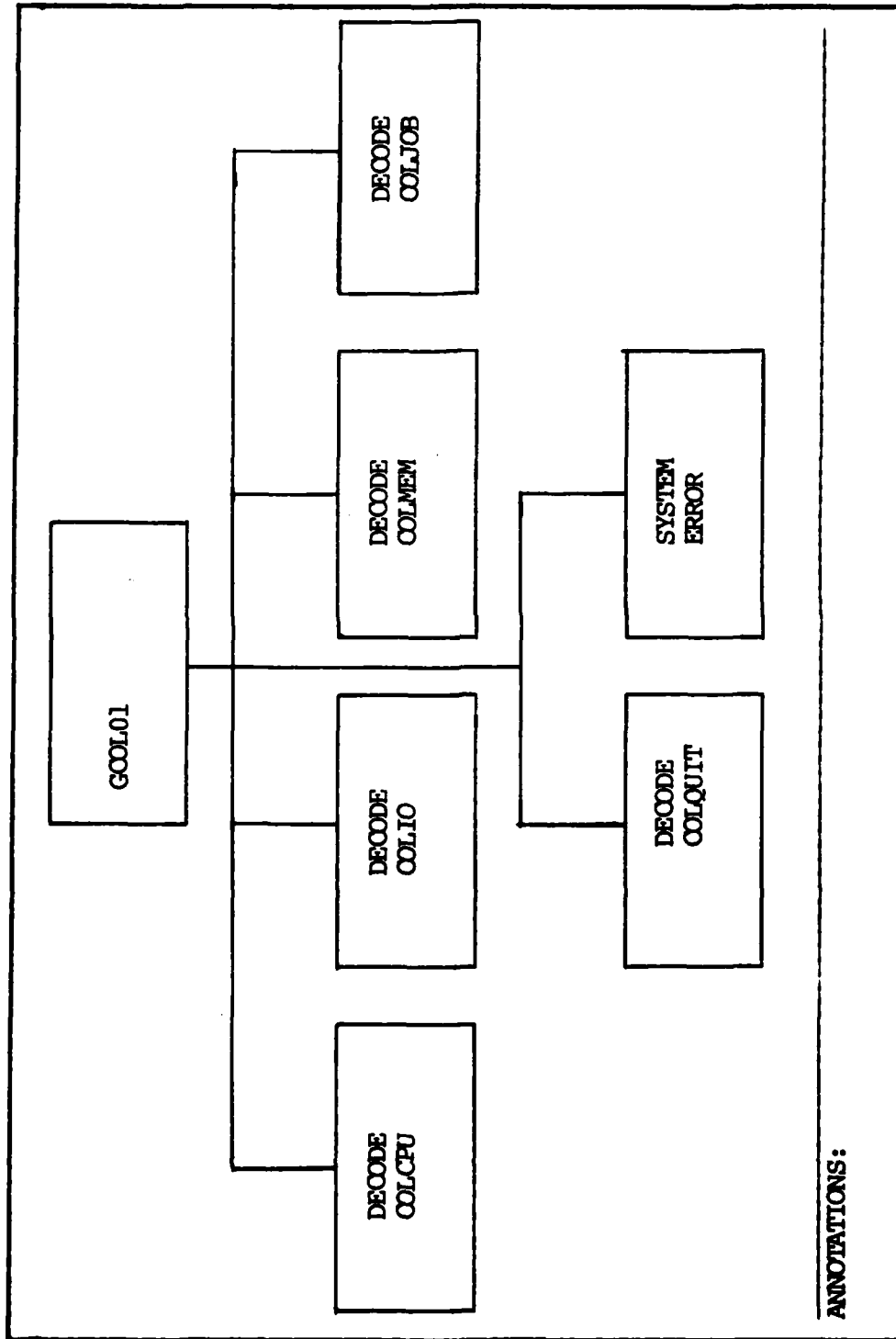


Figure E-14. Structure Diagram - GCOL01

USER INTERFACE ORGANIZATIONAL STRUCTURES

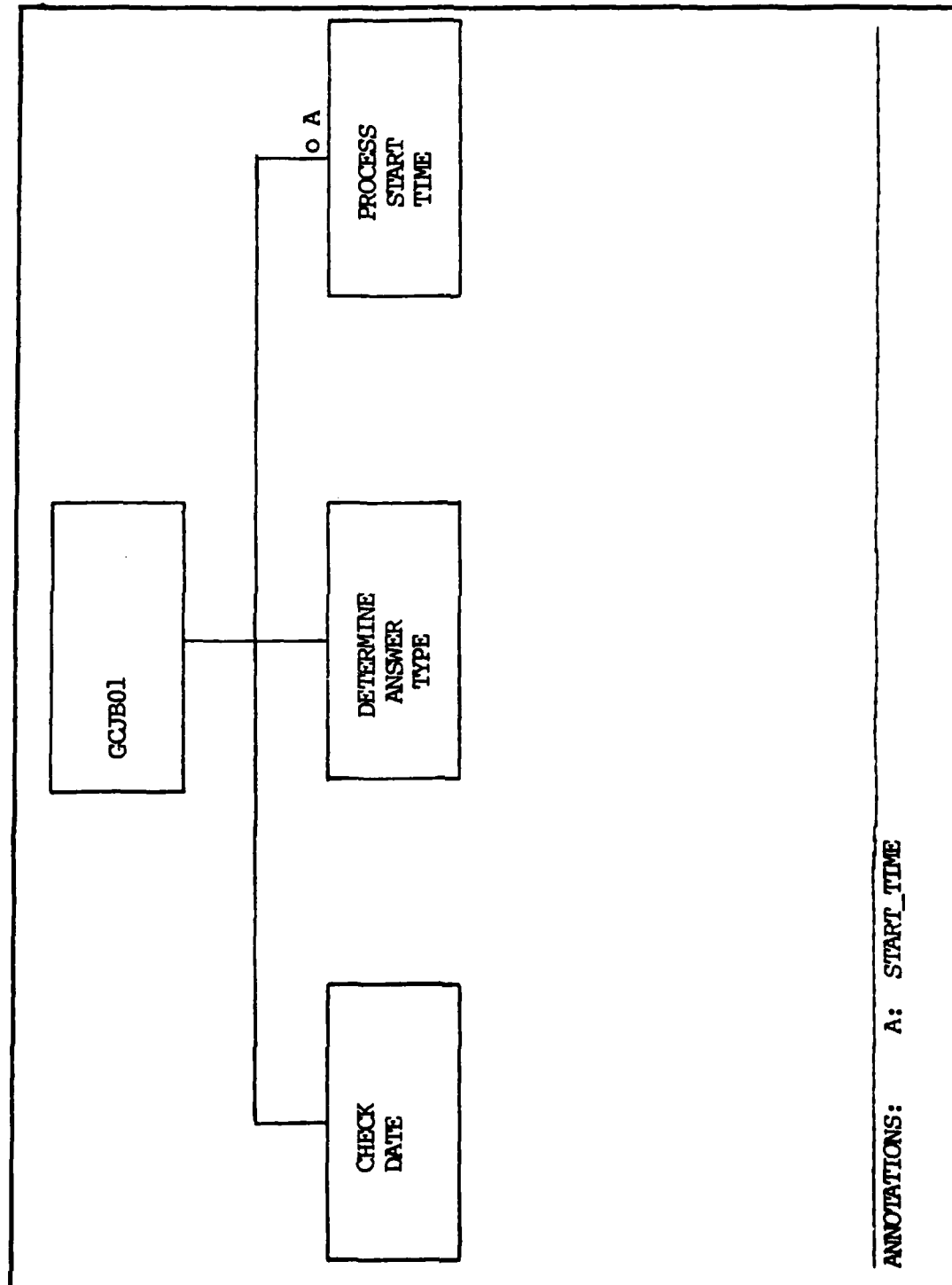


Figure E-15. Structure Diagram - GCJB01

USER INTERFACE ORGANIZATIONAL STRUCTURES

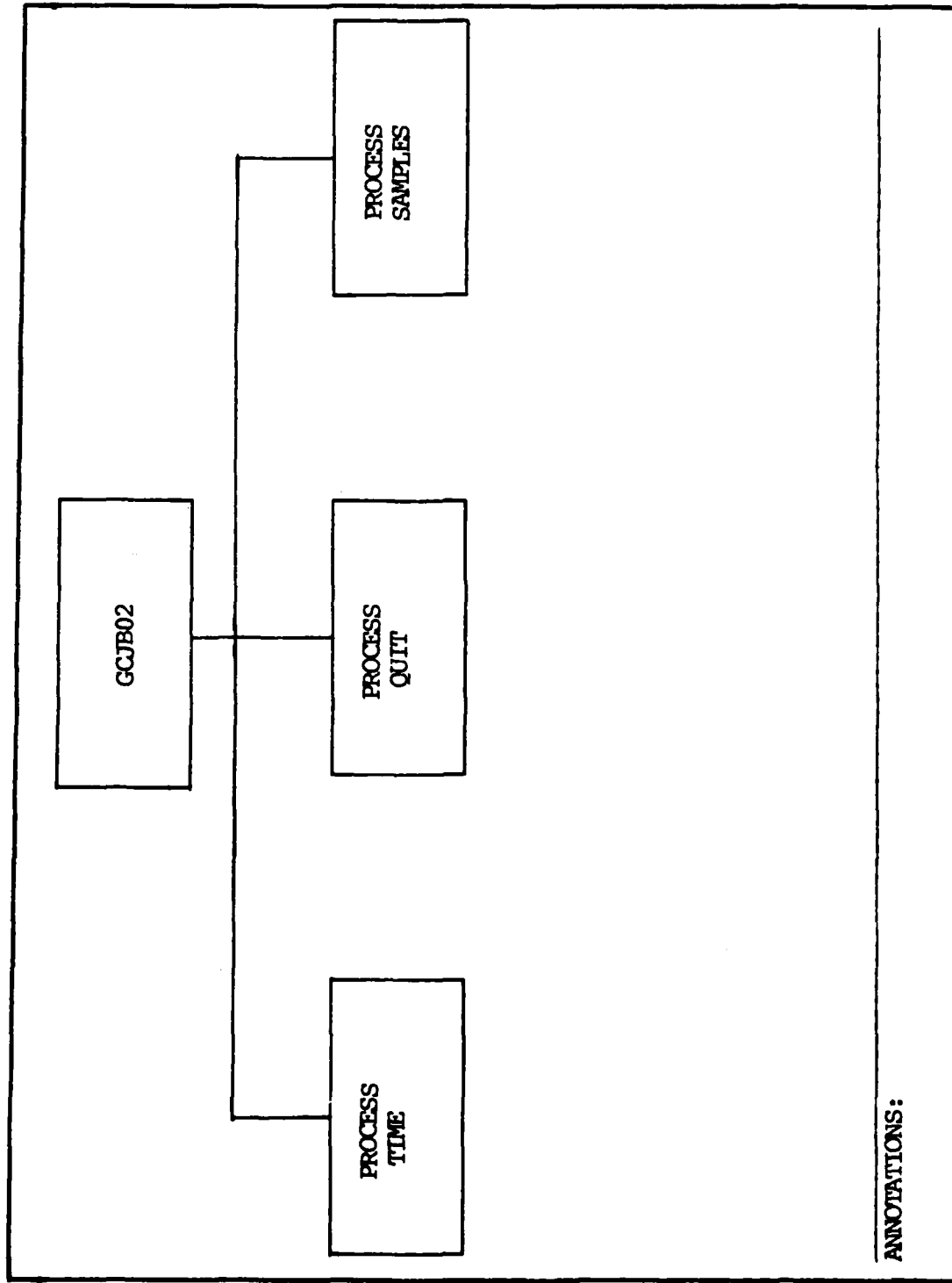


Figure E-16. Structure Diagram - GCJB02

USER INTERFACE ORGANIZATIONAL STRUCTURES

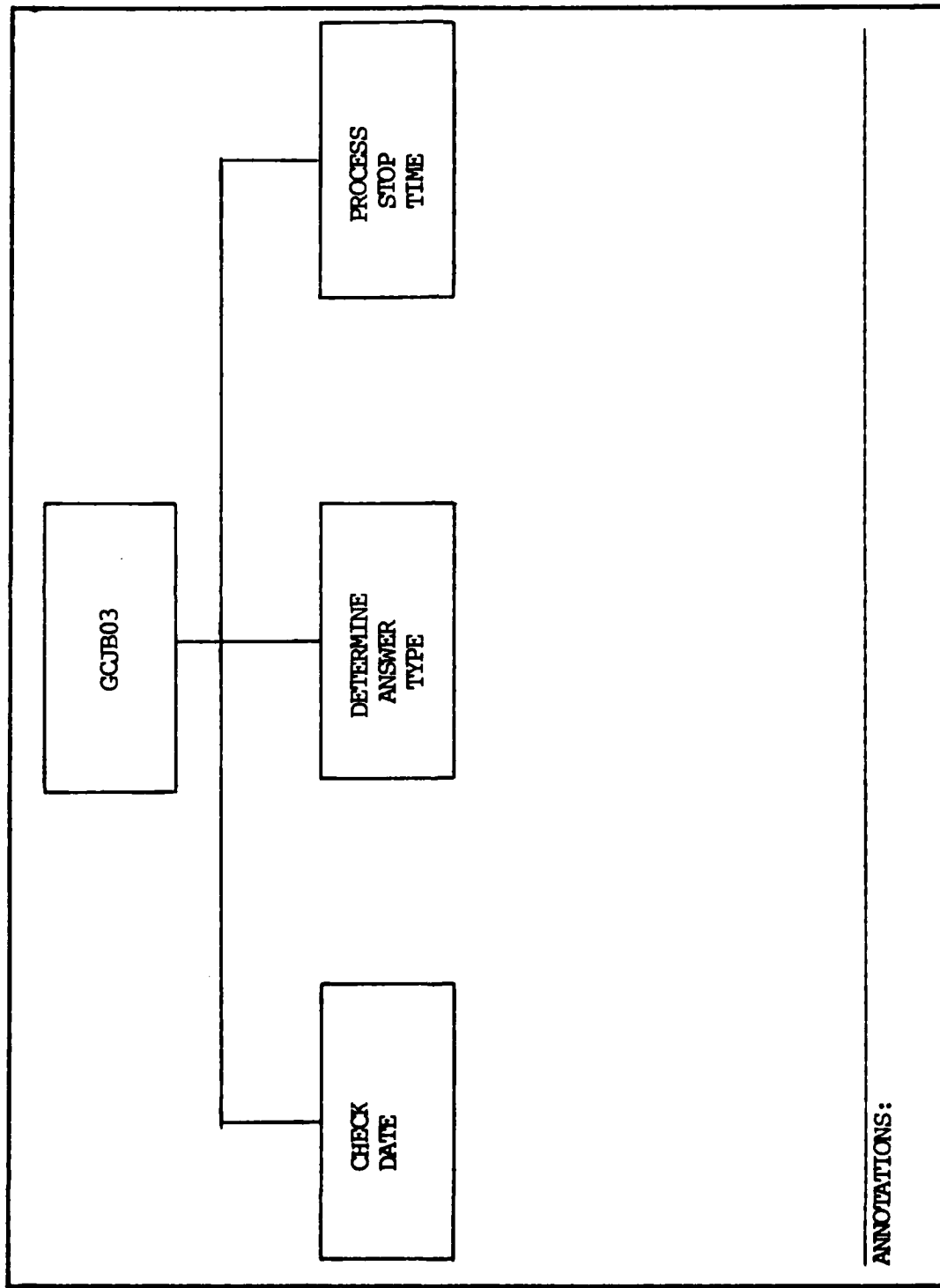


Figure E-17. Structure Diagram - GCJB03

USER INTERFACE ORGANIZATIONAL STRUCTURES

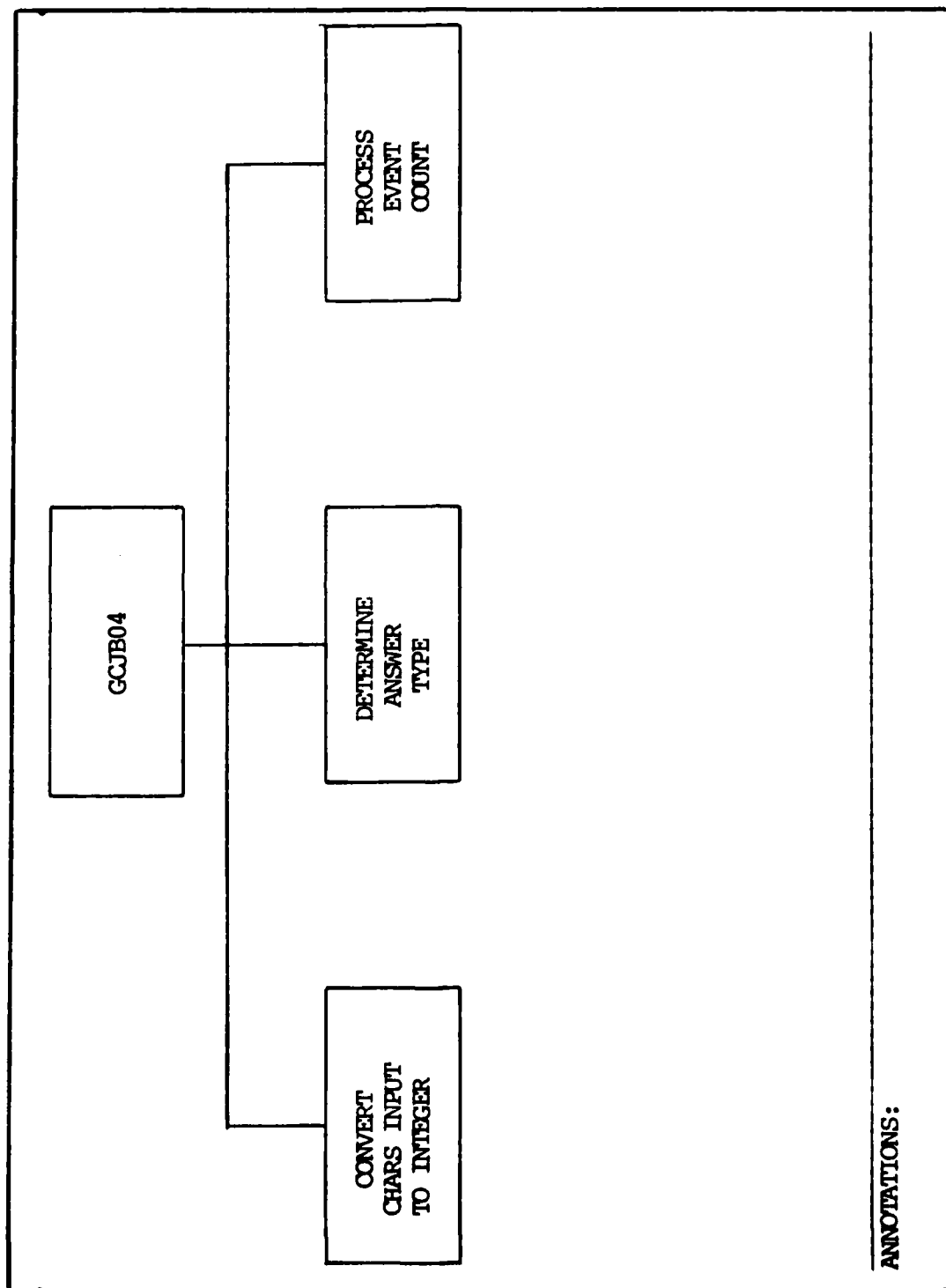


Figure E-18. Structure Diagram - GCJB04

USER INTERFACE ORGANIZATIONAL STRUCTURES

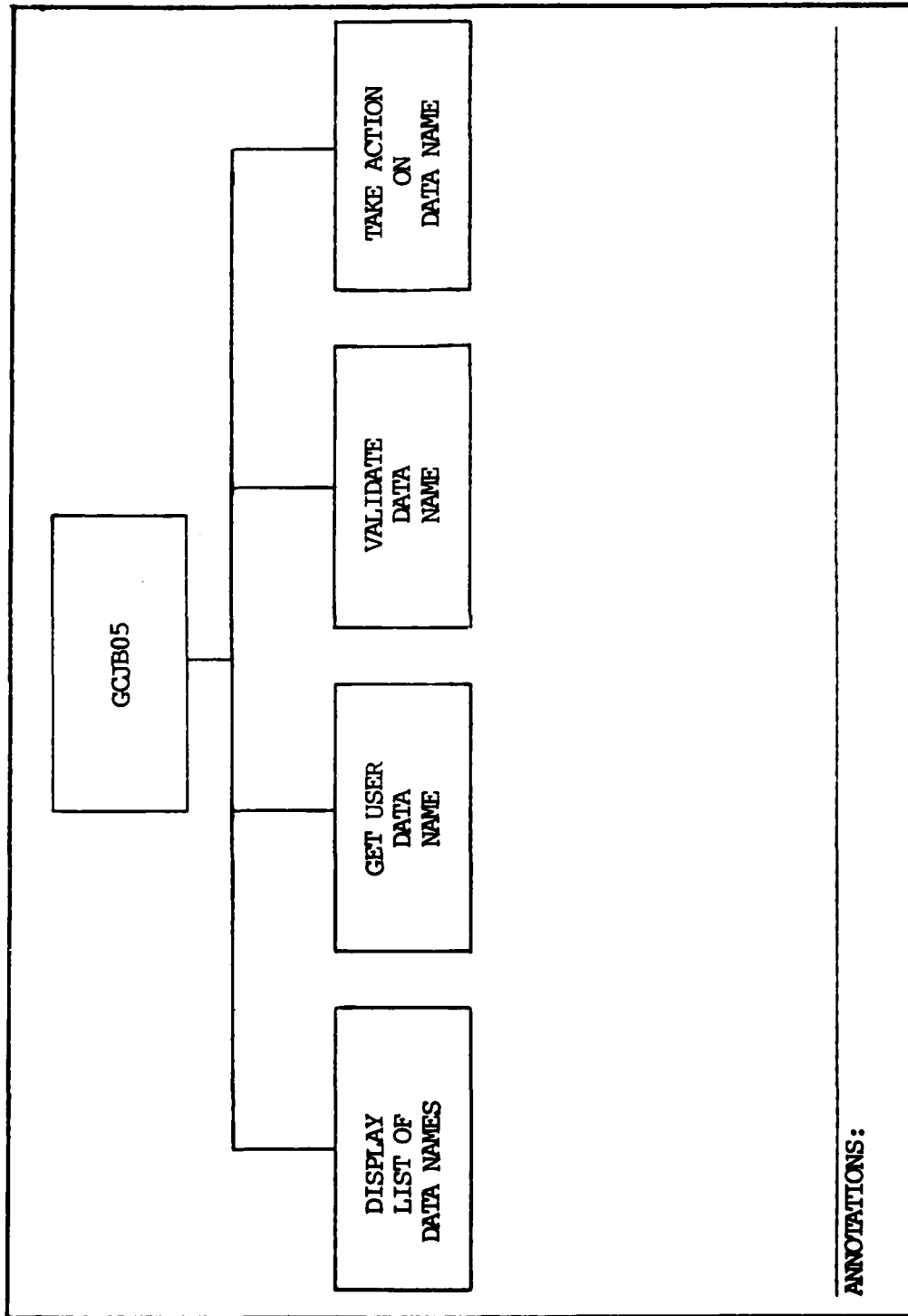


Figure E-19. Structure Diagram - GCJB05

APPENDIX F
DATA COLLECTION ORGANIZATIONAL STRUCTURES

The structure charts on the following pages represent the design levels completed on the Data Collection Module. Note that detailed design is only complete for job/process data collection. All other forms of data collection must be addressed as part of another effort.

DATA COLLECTION ORGANIZATIONAL STRUCTURES

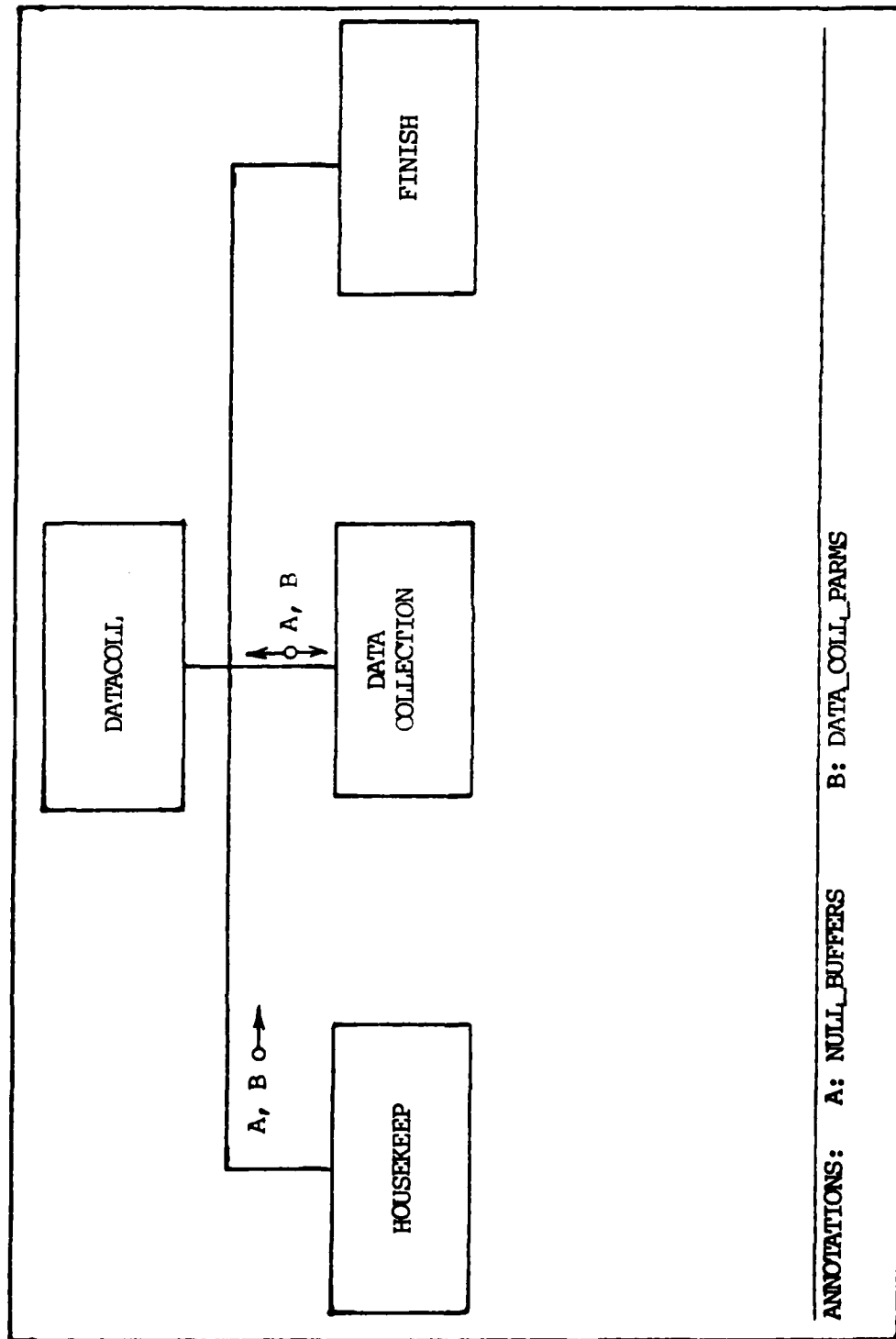


Figure F-1. Data Collection structure chart 1.0 (DATA_COLL).

DATA COLLECTION ORGANIZATIONAL STRUCTURES

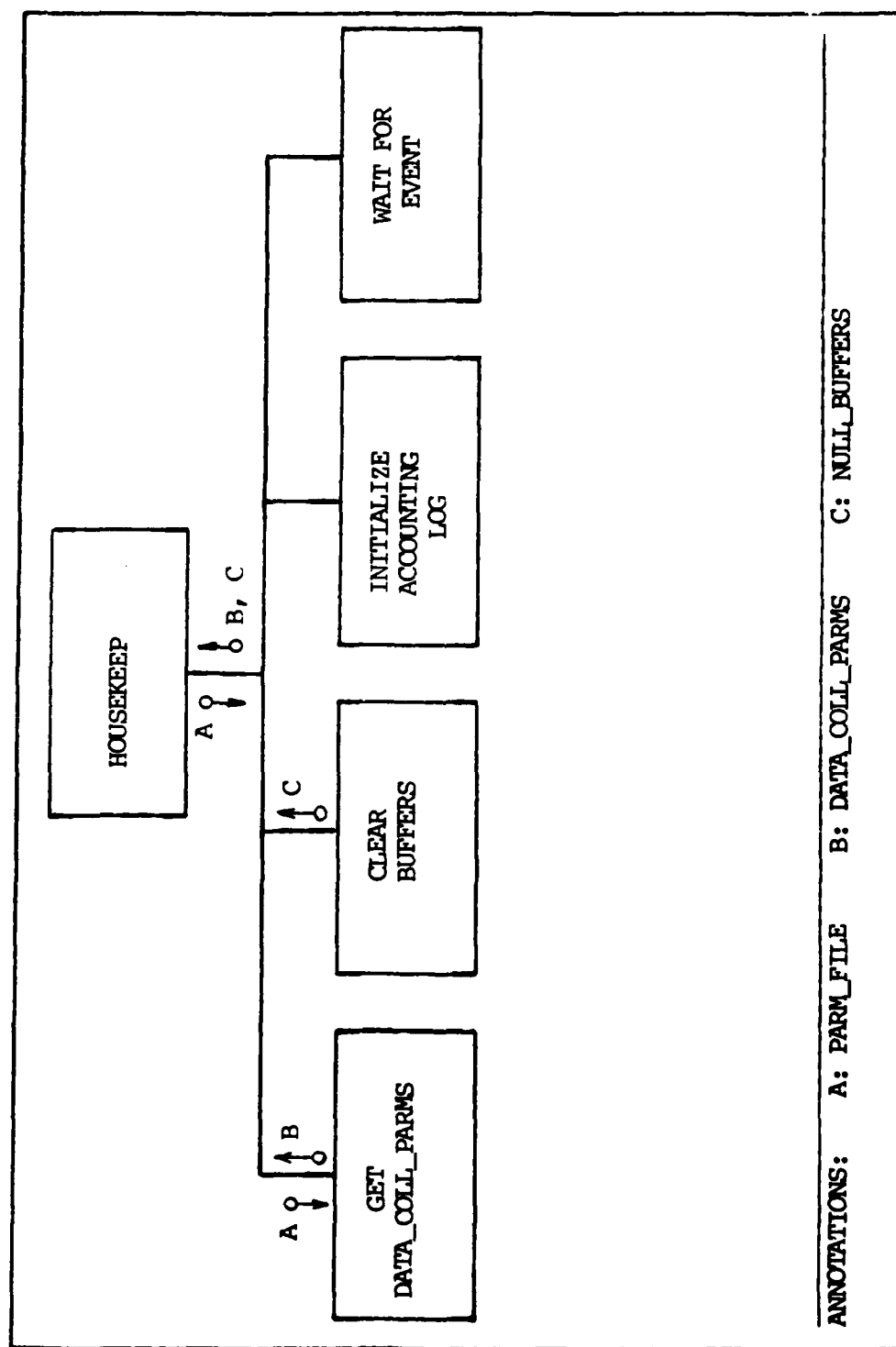


Figure F-2. Data Collection Structure Chart 1.1 (HOUSEKEEP).

DATA COLLECTION ORGANIZATIONAL STRUCTURES

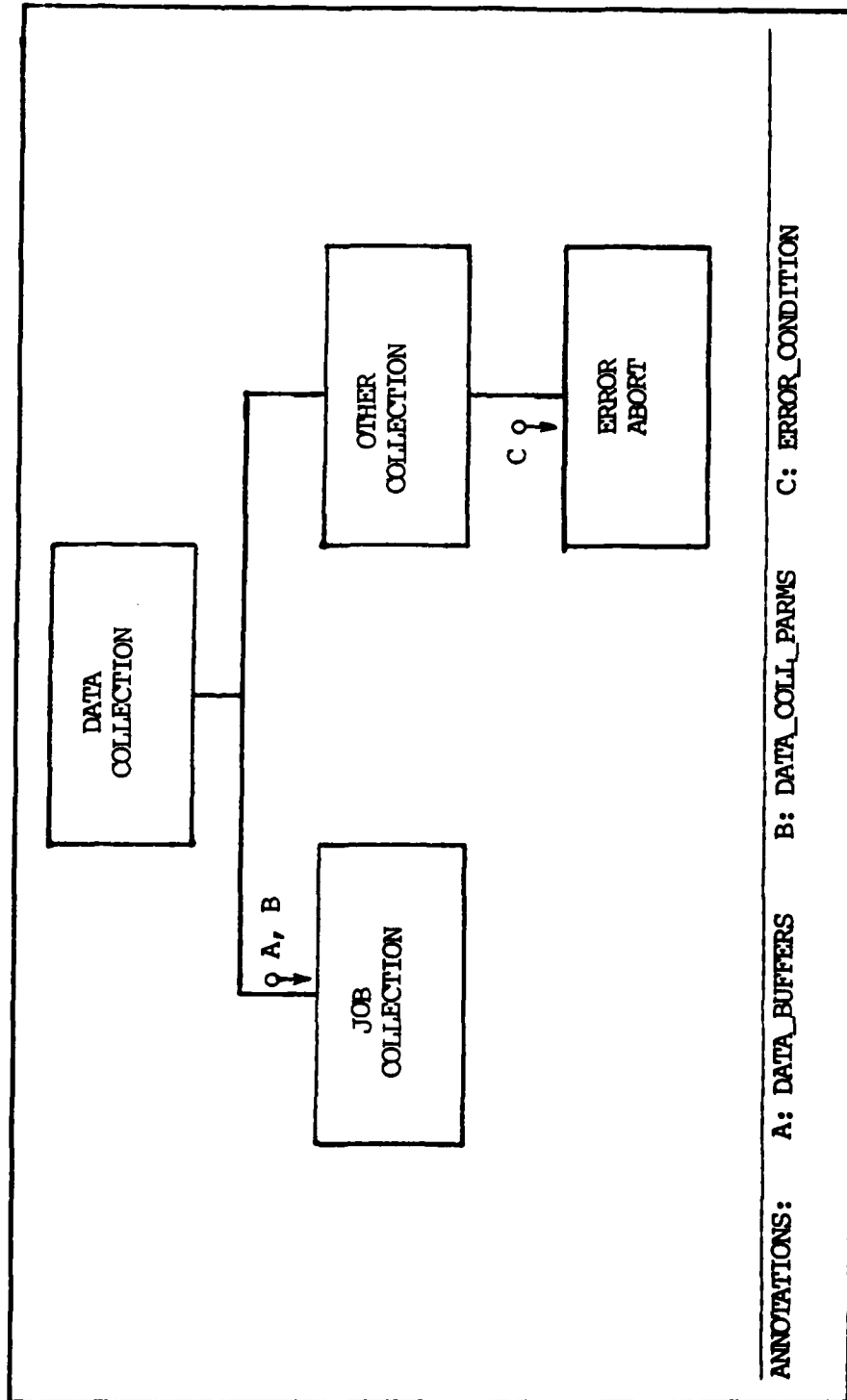


Figure F-3. Data Collection Structure Chart 1.2 (DATA_COLLECTION).

DATA COLLECTION ORGANIZATIONAL STRUCTURES

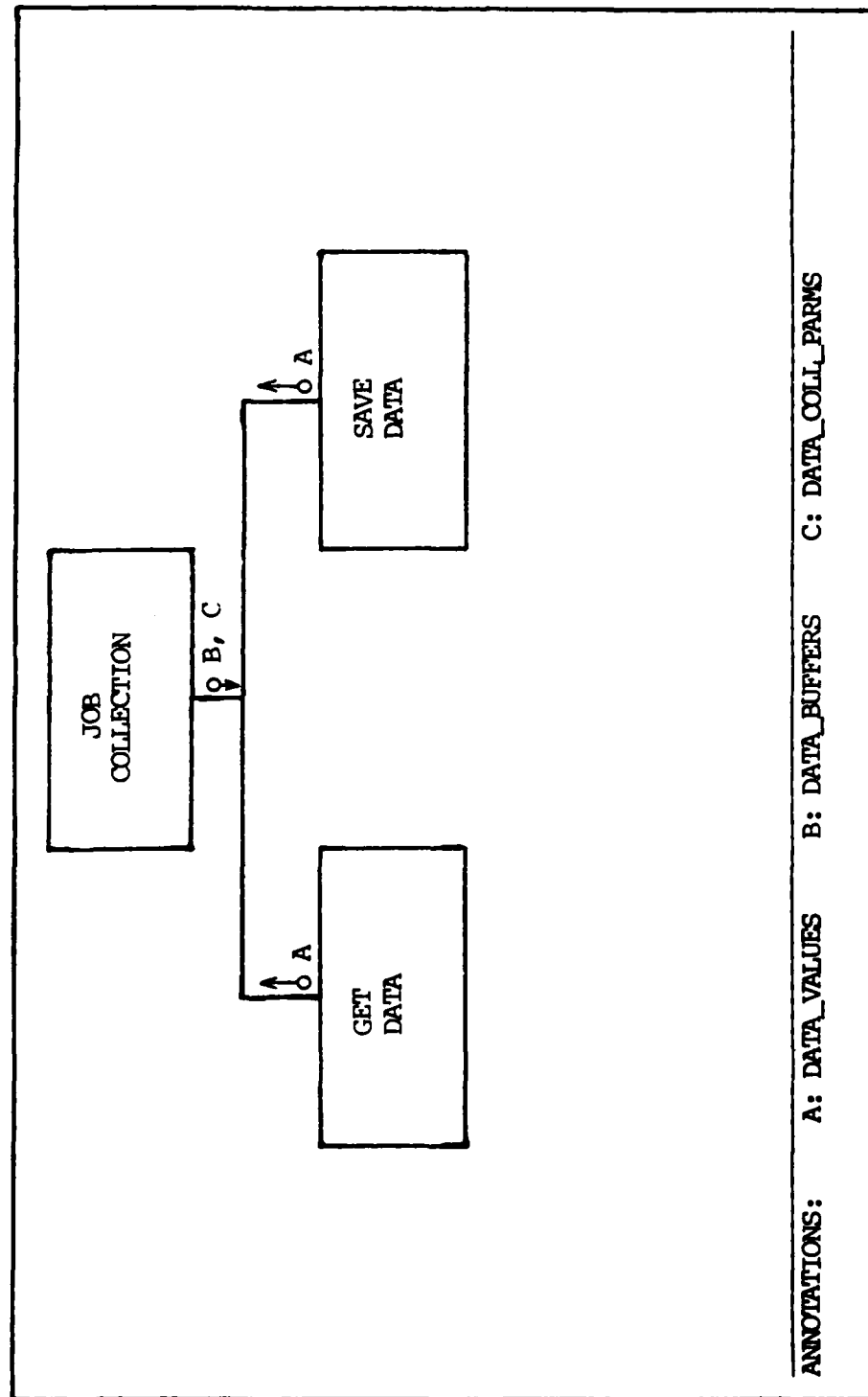


Figure F-4. Data Collection Structure Chart 1.4 (JOB_COLLECTION).

DATA COLLECTION ORGANIZATIONAL STRUCTURES

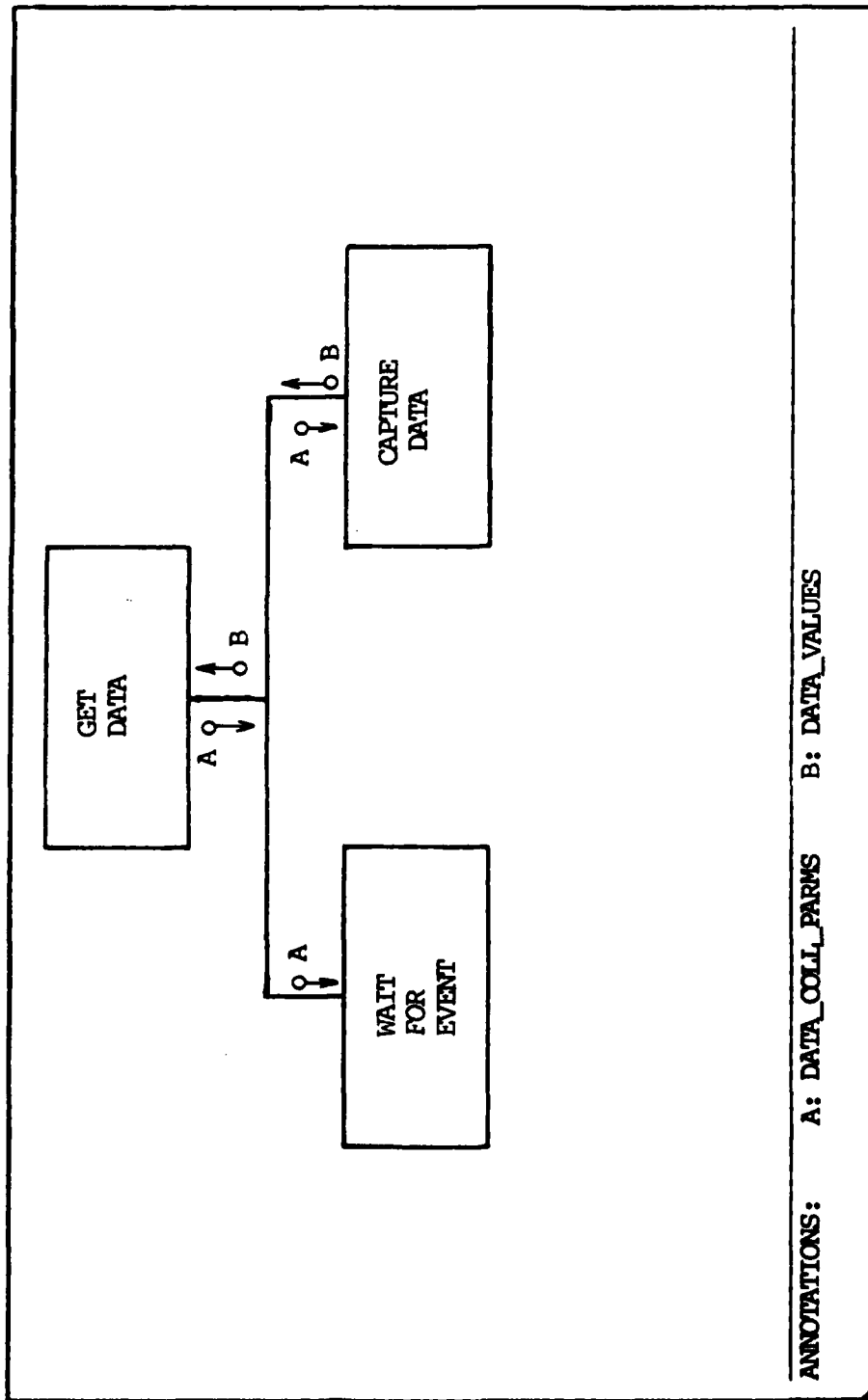


Figure F-5. Data Collection Structure Chart 1 (GET_DATA).

DATA COLLECTION ORGANIZATIONAL STRUCTURES

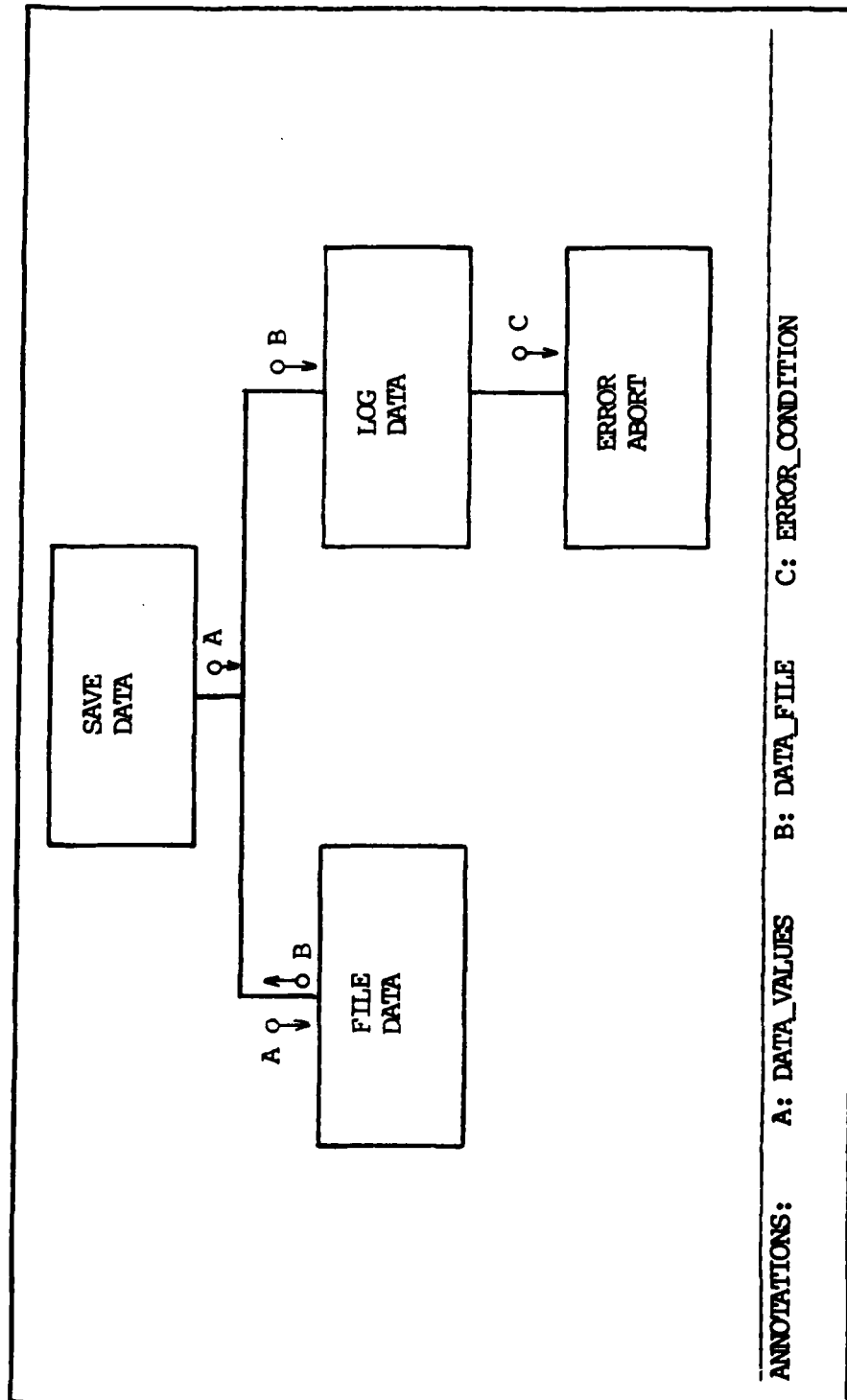


Figure F-6. Data Collection Structure Chart 1.5 (SAVE_DATA).

DATA COLLECTION ORGANIZATIONAL STRUCTURES

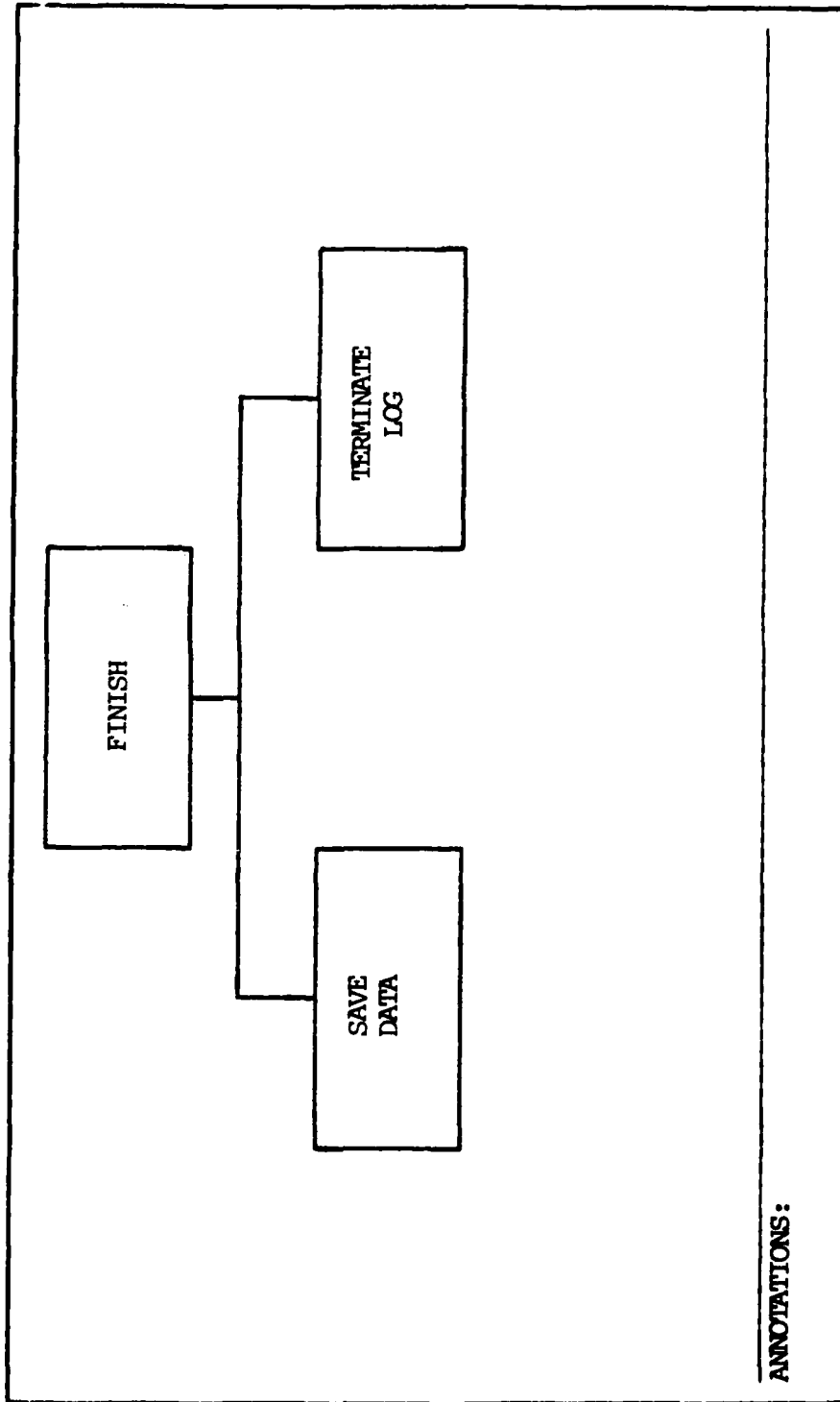


Figure F-7. Data Collection Structure Chart 1.6 (FINISH).

INDEX

Balance set, A-2, A-4
Base priority, A-5

Compatability mode, 1-2
Current priority, A-5

Data gathering, 2-2
Dialogue frame, 5-1
Display, 1-3

Files
 log, 1-4
Framedit., 5-6
Framefig, 5-6
Frameitemlist, 5-3
Free page list, A-3

Hibernation, 3-7

Job, A-2
Job mix, 1-3

Log files
 activity, 1-4
 error, 1-4
 operator, 1-4

Mailbox, 3-4
Memory
 management, 1-2, A-1
 balance set, A-2
 first-in-first-out, A-2
 free page list, A-3
 memory pages, A-2
 modified page list, A-3
 page fault, A-2
 page swapping, A-2
 resident set, A-2
 shared page cache, A-3
 virtual memory, A-1
 working set, A-2
 pages, A-2
Modified page list, A-3

Native mode, 1-2

 modified, A-3
Page swapping, A-2
Pdp-11, 1-2
Performance
 measurement tools
 accl, 1-4
 display, 1-3
 logs, 1-3
 syl, 1-4
 measures, 2-1
Permttool requirements, 2-14
Permttool., 1-1
Primary memory, A-2
Priority
 base, A-5
 current, A-5
 processing, A-1
Process, A-2
 definition, A-4
 scheduling, A-1, A-4
Process priorities, A-5
 base priority, A-5
 categories, A-5
 current priority, A-5
 legal range, A-5
Process scheduling
 balance set, A-4
 process swapping, A-5
 residency quantum, A-5

Real memory, 1-2
Residency quantum, A-5
Resident set, A-2

Shared page cache, A-3
System degradation, 1-5
System log files, 1-4

Utilization
 disk, 1-2

Vax, 1-2, 1-4 to 1-6, 1-8
Vax-11/780, 1-1, 1-5
Vax/vms, 1-5
Vms, 1-2

INDEX

Page cache
 shared, A-3
Page fault, A-2
Page list
 free, A-3

Working set, A-2

Yourdon, 1-4

VITA

Eugene C. Gilpin, Jr. was born in Grants Pass, Oregon, on 5 September, 1946. He graduated from high school in Grants Pass, and attended Oregon State University, Southern Oregon College, and the University of Nebraska. He received the degree of Bachelor of General Studies (Biology) in December, 1972 from the University of Nebraska. After graduation, he attended the Officer's Training School, from which he was commissioned a Second Lieutenant in June, 1973. He has served as a computer programmer and systems analyst with the Strategic Air Command (SAC) at Westover AFB, Massachusetts, and Offutt AFB, Nebraska. In addition, he has served with the Defense Communications Agency in Washington D. C., and the Air Force Manpower and Personnel Center at Randolph AFB, Texas. He entered the Air Force Institute of Technology in June, 1981.

Permanent Address: 12090 Galice Road

Merlin, Oregon 97532

END

FILMED

3-83

DTIC