

AD-A138 059

APPLICATIONS PROGRAMS TO FACILITATE USE OF A DBMS (DATA 1/3

BASE MANAGEMENT S. (U) AIR FORCE INST OF TECH

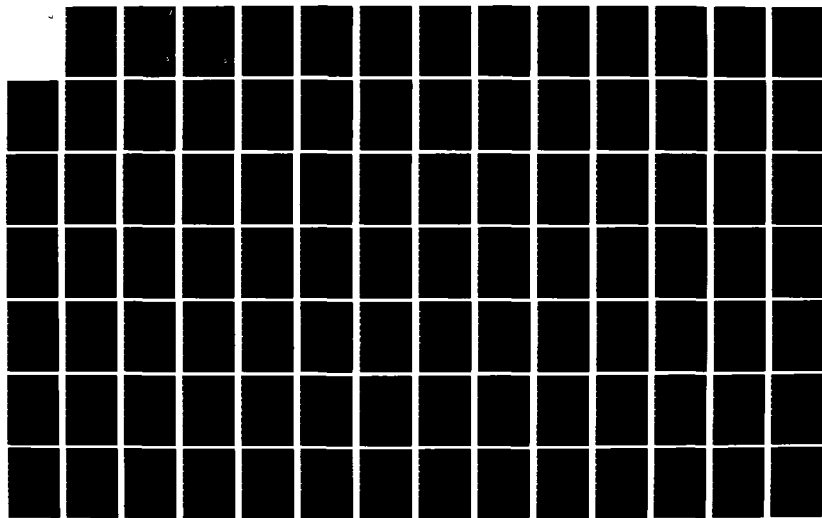
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... J D GATEWOOD

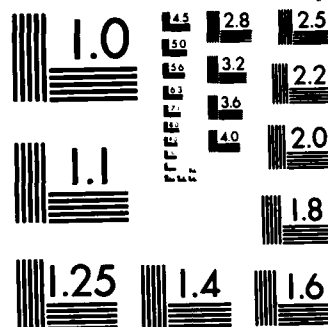
UNCLASSIFIED

DEC 83 AFIT/GCS/EE/83D-10

F/G 9/2

NL





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

AD A138059



1

APPLICATIONS PROGRAMS TO FACILITATE
USE OF A DBMS TO
STORE AND RETRIEVE GRAPHICS DISPLAYS
(INGRED II)

THESIS

AFIT/GCS/EE/83D-10 James D. Gatewood
Capt USAF

DTIC FILE COPY

DTIC
ELECTE
FEB 21 1984

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT

Approved for
Distribution

84 02 17 080

AFIT/GCS/EE/83D-10

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



APPLICATIONS PROGRAMS TO FACILITATE
USE OF A DBMS TO
STORE AND RETRIEVE GRAPHICS DISPLAYS
(INGRED II)

THESIS

AFIT/GCS/EE/83D-10 James D. Gatewood
Capt USAF

DTIC
ELECTE
S FEB 21 1984
D

Approved for public release; distribution unlimited.

APPLICATIONS PROGRAMS TO FACILITATE
USE OF A DBMS TO
STORE AND RETRIEVE GRAPHICS DISPLAYS
(INGRED II)

THESIS

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

by

James D. Gatewood, B.S.

Capt USAF

Graduate Computer Science

December 1983

Approved for public release; distribution unlimited.

Preface

Personnel in the AFIT Digital Engineering Laboratory have been working for some time to provide a graphics editing capability which can be used for software automation, graphics research, and student instruction. Prior to this thesis effort, a graphics editing capability was in place at AFIT. However, with completion of this thesis effort, the use of that graphics capability can be better used. This is because of the combination of the flexibility provided by using a DBMS and the optimization of the data structure to be stored.

I want to thank several individuals who provided valuable help and guidance to me during this thesis effort. Dr. Gary Lamont, my thesis advisor, allowed me the right amount of latitude, but helped guide me when needed. His comments on the written portion of the thesis improved it greatly. Dr. Hartrum and Major Lillie, my thesis readers, provided sound inputs for me to build on. Three AFIT students were also instrumental in the success of this thesis. Capt Mark Travis helped me over many roughspots with his knowledge of FORTRAN. Capt Jerry Owens and Captain Dale VanKirk both helped during the difficult times by verifying algorithms and validity of code. Finally, I want to thank my wife, Pat, and daughter, Chris, for all the patience and support during our AFIT experience. My gratitude ladies, you made it worthwhile.

Contents

	<u>Page</u>
Preface	ii
List of Figures	v
List of Tables	vi
Abstract	vii
I. Introduction	I-1
Overview	-1
Background	-2
Computer Graphics Overview	-4
Standards	-5
Problem	-7
Summary of Current Knowledge	-8
Scope	-10
Assumptions	-13
Approach	-13
Sequence of Presentation	-14
II. Requirements Definition	II-1
Introduction	-1
Approaches	-2
Actual Functional Requirements	-4
Notation	-8
Summary	-11
III. Design	III-1
Introduction	-1
Existing Graphics Capability	-1
DBMS Selection	-2
Notation	-4
Design Considerations	-6
RIM Capabilities	-8
Summary	-11
IV. Implementation	IV-1
Introduction	-1
Implementation Factors	-1
File Storage Versus Data	
Data Base Storage	-1
DBMS (RIM)	-4
INGRED II	-5
Implementation Details	-12
Problems	-17
Summary	-20

V. Testing	V-1
Introduction	-1
Errors	-1
Test Methods	-2
Software Lifecycle	-5
Test Results	-8
Summary	-9
VI. Conclusions/Recommendations	VI-1
Introduction	-1
Conclusions	-1
Recommendations	-3
Bibliography	BIB-1
Appendix A: Glossary	A-1
Appendix B: Data Structure	B-1
Appendix C: SADT Diagrams	C-1
Appendix D: Activity Data Dictionary	D-1
Appendix E: Data Element Data Dictionary	E-1
Appendix F: Test Plan	F-1
Appendix G: Abbreviations and Acronyms	G-1
Appendix H: Source Code	H-1
Appendix I: User's Guide	I-1
Vita	VITA-1

List of Figures

<u>Figure</u>	<u>Title</u>	<u>Page</u>
II-1	SADT Activity Box	II-10
III-1	INGRED II Top Level Design	III-7
III-2	Choice of Data Base or Data File	III-9
IV-1	INGRED II Template	IV-7
IV	Storage Procedure; Ingred vs INGRED II	IV-9
V-1	Software Life Cycle	V-6
C-1	SADT C-1: INGRED II Graphics System .	C-3
C-2	SADT C-2: INGRED II Graphics System .	C-4
C-3	SADT C-3: Perform Graphics Functions	C-5
C-4	SADT C-4: Change Graphics Figure	C-6
C-5	SADT C-6: Access Graphics Data Base .	C-7
C-6	SADT C-7: Obtain Graphics Data	C-8
C-7	SADT C-8: Store Data	C-9
C-8	SADT C-9: Request Output Desired	C-10
C-9	SADT C-10: Transfer Figure	C-11
C-10	SADT C-11: Output Information	C-12
C-11	SADT C-12: Send to Printer	C-13
I-1	INGRED II Grid Display	I-3
I-2	INGRED II TEXT Command	I-4
I-3	INGRED II Line Command	I-6
I-4	INGRED II Alternate Grid	I-7
I-5	INGRED II "Skeleton" Capabilities	I-8

List of Tables

<u>Table</u>	<u>Title</u>	<u>Page</u>
IV-1	Storage Comparison between INGRED and INGRED II	IV-16

Abstract

Applications programs were written to facilitate the use of the RIM relational DBMS to store and retrieve graphics displays generated using capabilities of a graphics editor. This DBMS storage capability is in addition to data file storage which was provided by INGRED, the graphics editor. The resulting system is called INGRED II. The addition of DBMS capabilities made the graphics editor much more flexible and more user friendly. It resulted in the capabilities to obtain a listing of displays stored in the data base, to delete displays from storage which are no longer needed, and to store pertinent information in the data base along with the display data. Neither of these capabilities was available using the original INGRED. In the study of the data structure generated by GWCORE to store display information, it was determined that some data structure information was redundant. Routines were added to optimize the data structure before storage on disk with a resultant savings of approximately 40 per cent. The time required to do the optimization proved negligible, especially considering the criticality of storage at most locations which will use INGRED II. The result of the INGRED II system is a more effective system for use in graphics research, and software design automation.

APPLICATIONS PROGRAMS TO FACILITATE
USE OF A DBMS TO
STORE AND RETRIEVE GRAPHICS DISPLAYS
(INGRED II)

I. Introduction

Overview

The potential contributions of computer graphics to several phases of the United States Air Force (USAF) mission are being explored at Wright-Patterson Air Force Base (WPAFB). The increasing use and impact of computer graphics in such areas as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), simulation, and software design and development are a few areas where current operations use computer graphics [29:40]. These significant uses emphasize the need for a system to manage the graphics data base information at several of the organizations on WPAFB.

This thesis investigation is designed to provide that data base management capability to any WPAFB unit which can benefit from it. This introductory chapter presents background information concerning the WPAFB requirements for a data base system to manage available and newly generated graphics displays. Next, the actual problem that resulted in this thesis effort is specifically introduced. The next two sections introduce a summary of current graphics data base knowledge and then the scope of the investigation.

These are followed by fundamental assumptions and the approach used. Finally, the sequence of presentation for the remaining thesis chapters is introduced.

Background

The reason for this effort is to continue the development of interactive computer graphics system software for use by the Air Force Avionics Laboratory (AFAL), Air Force Materials Laboratory (AFML), Air Force Flight Dynamics Laboratory (AFFDL), and the Air Force Institute of Technology (AFIT). The development effort is part of the interactive graphics research project which is aimed at implementing an on-line, real time graphics facility based on the Core standard. The project director is professor Gary B. Lamont. The system will be used for Computer Aided Design (CAD), software development, picture development, etc. Ultimately the facility will provide a capability compatible with the Graphical Kernel System (GKS). This effort was begun by Harold Curling [6] and continued by Kevin Rose [24]. The goal of the current investigation is to facilitate joint use of graphics capabilities at each of the above locations. Specifically, this effort will provide applications software to:

1. Allow access to the large, though dispersed, graphics data base already available.
2. Make it easier to add new graphics displays to the data base.

3. Provide a means to save displays to tape or disk and to delete displays which are no longer needed.

4. Allow easy changes to already existing displays.

To provide these capabilities, the applications programs will be used with a data base management system (DBMS).

The introduction of a DBMS with the needed utilities can solve the problems of data storage, retrieval, and modification. The system must adequately define the structure which will be used to store the desired information. The challenge is to provide a system which allows transportability between the variety of host processors and display devices used at the laboratories and at AFIT. This challenge is further complicated by the diversity in graphics requirements between the locations. The goal will be to use software engineering principles to provide a modular, device independent solution which can be easily expanded or changed as future needs dictate.

As stated earlier, this effort will provide data base applications programs to support computer graphics. Of necessity, the main thrust of the investigation will be the generation of data structures, requirements definition, design, coding, and testing as they apply to the data base. Since this investigation is concerned with data base manipulation, very little will actually be covered concerning the interesting and complex area of computer graphics. Therefore the next few paragraphs are designed as a short introduction to computer graphics. For the

interested reader, however, there are many good introductory texts on computer graphics theory, hardware, software, and algorithms. These include [2, 10, 17, 22]. In addition, [24] provides an in-depth explanation of the graphics techniques used in the particular core graphics application for which this current effort is being implemented.

Computer Graphics Overview

Computer graphics is defined as "...the input, construction, storage, retrieval, manipulation, alteration and analysis of objects and their pictorial representation" [22:319]. The bulk of the development of computer graphics has been since Dr. Ivan Sutherland's dissertation in 1963 [29:23]. This investigation showed that a computer can be employed for interactive design of line drawings using a simple cathode-ray tube display and a few auxiliary input controls.

Graphics capabilities have evolved rapidly during the past decade to the point that digital image data for maps, photographs, and drawing is being collected and processed by government, industry and the universities [5:43]. Chock further states that the two-dimensional digital data structures can be divided into two general classifications - grid and topological structures.

Grid data structures subdivide the image's area of interest into a rectangular grid. One data record is used to represent each grid cell or pixel. A value field for each cell contains a representative value for the cell.

The second type of data structure, the topological, is a family of ordered point sets, line segments, or point pairs and lists of points. In the simplest form of topological data structure, lists of points outlining features are stored with no relationship between features. The desired representation is displayed by putting together those points and segments that make up its parts.

The grid and topological approaches both have advantages under certain circumstances and the choice of which one to use must be dictated by those circumstances. However, the grid systems do appear to perform a larger number of operations per system, on the average [5:47].

Computer graphics has proved to be an effective means of human-machine interaction which allows increased use of the capabilities of computer hardware. Some computer graphics applications include: plotting, cartography, simulation and animation, art and commerce, office automation, electronic publication, CAD, and process control [22:320].

Standards

Another area of importance which relates to this investigation is that of graphics standards. Efforts to establish a graphics standard have been ongoing since 1974, and the result is two separate standards, the Core and the GKS, which are closely related [28:167]. Both are proposed standards, and are interfaces between levels of the layered architecture of a computer-graphics system. These two

standards focus most specifically on the interface between graphic utility systems and application programs. Both have developed to insure that a single application program will work with diverse input and output peripherals. To achieve device independence, the application program calls routines which drive graphics input/output (I/O) devices. The advantage of this approach is that to use a new graphics I/O device, there is only the need to have a device driver, instead of a complete new application program.

The Association for Computing Machinery's Special Interest Group on Computer Graphics (ACM SIGGRAPH) proposed standard is called the Core. The main motivation behind Core was machine and device independence to protect the software and hardware investments of users.

The European graphics standard, the Graphical Kernel System (GKS) was originally developed by Deutches Institute fur Normung, the official standards-making body of West Germany [28:167]. The International Standards Organization (ISO) is in the final stages of converting GKS from its current status as a Draft International Standard to an International Standard. The GKS at the present time is a two-dimensional system with limited functionality. Despite the fact that the Core is three dimensional, in most other respects, GKS is a superset of the Core functions [28].

The Core is the accepted, though not adopted, American standard which is being used by many U.S. commercial firms [28:169]. Because of the recent acceptance of the GKS as

the international standard, it will be replacing the Core in actual use in the U.S. too. In fact, GKS is being adopted by the American National Standards Institute (ANSI). This poses a problem for those firms which have already adopted software designed under the Core standard, but because of the similarity between the two standards, the problem should not be insurmountable. The similarity stems from the fact that early versions of GKS were patterned after the ACM SIGGRAPH Core which evolved earlier [28:169].

Much of the use of this thesis effort by the units on WPAFB will be in the area of Computer Aided Design (CAD). Most CAD systems consist of four components [8] which interface the user with the hardware/software to be used in the CAD environment. These are:

1. The language processing functions which handle the interface between the user and the system.
2. Basic graphics capabilities necessary for representing objects to be designed.
3. A data base system which stores design related data including design objects, graphical data, object properties, information on the state of the design process, and the design documentation.
4. Computational algorithms which apply to certain CAD use.

Problem

AFAL, AFML, AFFDL, and AFIT do not have a method which allows simple cataloging, storage, and retrieval of graphics display information.

More graphics displays are being used now because it is much easier to generate the displays than ever before. Therefore, a DBMS is needed which will allow people to find which graphics models are available, and also improve the speed and ease of retrieving, displaying, modifying, and deleting those graphics figures as desired.

Summary of Current Knowledge

Data base systems are gaining in popularity due to several advantages they have over normal file storage. These include:

1. Storage design independent from specific applications.
2. Explicit data definition independent of applications programs.
3. A user need not know data format or physical storage structure in order to access the information in the data base.
4. Integrity assurance independent of application programs.
5. Program recovery independent of applications programs.
6. Use of single data copy, eliminating redundancy and inconsistency.

Both data base and graphics techniques can be used effectively to improve the handling of graphics information. The DBMS selected for handling the graphics displays may be a relational, network, hierarchical, or hybrid system. [24:81], states that further work in applications development for graphics displays should include integrating

the graphics and data base capabilities that exist at AFIT. This goal is in keeping with the AFIT effort to provide a graphics editor to support the Air Force Wright Aeronautical Laboratories as discussed earlier in this chapter.

To help in this effort, there are many current sources for information concerning data bases for storing graphics and picture information. The ones used in this research project are listed in the bibliography. Some of the concepts included in these publications were significant contributors to this investigation and are discussed in more detail.

There are, as might be expected, many approaches to supporting graphics representations through data base manipulation. These include efficient, dependable systems using network, hierarchical, and relational data base applications concepts as their basis, and even some which use combinations of these concepts to optimize each specific part of the application. Many of these systems have been operating reliably for some time. The hierarchical and relational models have attracted the most attention from authors [4:14].

One hierarchical approach provides for access by using "bounding coordinates" which are established for each unit of imagery. The highest level of information identifies all of the digital image data bases for a given application. The second level of information describes a specific subset and the volumes which comprise the subset. The final level

consists of the individual digital images organized in groups which correspond to the source. This approach is akin to the top-down development of a program by which the programmer starts with a general concept and decomposes it step-by-step until an executable program results.

Recent direction [4:15] for hierarchical systems is towards storing images by decomposition of pictures into adjacent quadrants which have different resolutions. Each quadrant is one node of a tree called a Quad-tree. This approach allows easy access to stored pictorial data, flexibility in pictorial data storage, data independence, redundancy reduction in multilevel structured data, and efficient analysis of small portions of a display.

The relational approach can be especially effective when handling applications such as geographic information systems [4:14]. A typical example of this would call for analysis of a lot of data depending on regional or geographic location. The storage of data such as attributes for each geographic location lends itself to the development of a relational data base. Another advantage of this approach is that it permits algebraic use of data to answer complex questions and generate new information.

Scope

This research will include writing applications programs which allow users to easily catalog, store, and retrieve graphics display information at AFAL, AFFDL, AFML, and AFIT. [24] provided a Core standard graphics capability

[27], so this treatment will not address graphics tools or library routines, but will instead focus on manipulating the information generated from libraries and tools as outlined [24]. Furthermore, this effort does not include design of a new data base system, since an existing relational, network, or hierarchical structure will be used. This will allow the author to implement a working system which complements the capabilities of the existing software rather than just concentrating on writing a data base system which may or may not work at the end of the thesis effort. The applications programs will be written to help the user enter graphics display information in the data base, and also to use that information. The decision on the high-level language to use will be made at the completion of the design phase.

Ingrained in the goal of writing the applications programs are several other actions which must be considered when discussing the scope of this effort. Both preliminary and detailed design will require extensive research and application of software engineering methods. Psychological tests which apply to making computer applications more user-friendly will be studied and applied where applicable to insure that the resulting software will be more likely to be used. Testing, of course, is an important part of successful programming, and in this effort, the test plan will be formulated throughout each step. Both static and dynamic testing will be used. The output of the testing phase will

be a test plan, procedures to use during testing, and finally, the test report.

Another important area of consideration in data base management is that of data base security. Three things make computer security a difficult area [1:14]. First is that operating systems and utility software are becoming more and more complex. Second is that usually no one has precisely defined the security provided by the internal controls. Third, there has been little done to insure correctness of security controls that have been implemented. In the case of a data base system, this problem is compounded because the many people who access the system do not all have a need for all of the information which exists within the data base.

It is recognized that security is an important area of consideration in data base management. Because of the complexities of the implementation of this thesis effort, however, there is not time for inclusion of elaborate protection techniques. In addition, in the initial stages, no classified information will be handled in the researched data bases, and there will be no access to the system from outside the US Air Force community. Thus, the only security protection provided will be through the password protection of whatever DBMS is chosen. This does make the inclusion of at least a password protection scheme a necessary prerequisite for any DBMS chosen. There will be further discussion in Chapter 3 of the level of security provided

through the password protection capabilities of the chosen DBMS.

The total scope of this thesis investigation includes establishment of the actual user requirements, preliminary and final design of a system to satisfy those requirements, writing the applications programs from the design information, and testing the system software.

Assumptions

The following assumptions apply:

1. Personnel at the laboratories and at AFIT will cooperate in identifying problems and requirements so that a useable data base system can be built.
2. Depending on the decisions made, access to the appropriate computer and data base system will be insured.
3. Access to graphics data bases at all above locations will be allowed.
4. None of the information in the graphics display data base will be classified.

Approach

The approach used in this investigation follows:

1. Research literature to determine current trends in graphics, data base, and software development techniques.

2. Contact users to determine their needs for graphics software and how a graphics capability will be used. This information is used in the requirements definition phase.
3. Determine methods to measure how well those user needs are met.
4. Develop a data specification.
5. Decide on correct DBMS for the situation.
6. Implement the enhanced graphics editor through application programs.
7. Test the applications to see if they serve the need and operate as designed.
8. Modify as needed.
9. Thoroughly document actions throughout all phases of development.

Sequence of Presentation

Definition of user requirements is covered in the next chapter. In Chapter III, the options considered and the rationale for the design finally chosen, are presented. The details of the applications programs implementation and how the new graphics editing capability compares with that of the old system are included in Chapter IV. Chapter V includes the test objectives and details. Finally, Chapter VI includes the conclusions and recommendations concerning this thesis effort.

II. Requirements Definition

Introduction

To provide software which correctly implements capabilities for any user, it is first essential to fully define what the user needs. To do this, problems must be translated into corrective goals, the goals must be translated into solutions, and these solutions must be described in functional terms. This description describes the approach which will be used in this thesis.

In the system requirements definition phase, it is important for definitions to be clear and concise so that there will be no danger of misinterpretation. The functional specification (what the system is to do) will be expressed in terms of inputs, outputs, processes, and data structures. The definition will be as complete as possible, but it must also be understood that system requirements change, and thus, the definition must be designed so that it can handle these changes.

One important goal to establish is to control the system [34]. This means that it is important to provide capabilities to the user that are really needed and are within the capabilities of the hardware to provide, but only reasonable capabilities should be included. To successfully control the system it is necessary to define the scope of the project; break the system into small pieces to be attacked separately; require users to justify each requested feature; find similar systems to allow learning from

experience; and reject features which equipment can't support, are too complex to complete in the time allotted, or will require too many computer resources.

The approach in this chapter will be to develop a written system specification. The system requirements will be decomposed from the highest to the lowest level, and will be depicted graphically using SADT diagrams. These diagrams will insure separation of activity analysis from data analysis.

Approaches

There are three approaches available for handling the graphics display information generated from the Core graphics tools. These approaches include files, application data bases, and subject databases [20:29]. Each approach will now be discussed further.

The files approach does not use a DBMS. Separate data files are used for each application. This is the approach which Rose used in his thesis [24]. The files were designed by the programmer when the application was created. In the file approach, if the system is very large, a large number of files result, necessitating much redundancy and difficult maintenance. This is because for each use of the files, a separate version of the information must be stored on disk. This can be very costly, especially when graphics figures which require so much storage space are considered. Because data is not stored independent of the particular application, changes to applications can cause large

programming problems and thus are expensive, slow, and often resisted.

With application data bases, a DBMS is used, but separate data bases are established for separate applications. As a result, as in the file environment, there is much data redundancy and maintenance is difficult. This is because the data independence which should be built into data base systems is not realized. Instead of storing the information once, and writing the application program to access it, a copy of needed information is stored for each separate application. This approach does not achieve the major goals of data base operations.

With subject data bases on the otherhand, data are designed and stored independent of the specific function for which they are used. This allows several applications to use the same data, thus reducing redundancy. Maintenance is also much easier, because new applications or any changes in the existing application are facilitated by allowing easy access to the data. The results are faster application development and more direct user interaction with the data bases. With this approach, either the data or the applications programs can be changed without causing the other to have to be changed. Therefore old applications programs don't have to be rewritten with changes to data structure, data layout, or physical devices on which the data are stored. The data can be easily reorganized or content added to [31:9].

Comparison of the three possible approaches indicates that the subject data base is the preferred method for the software engineered approach, and the one which will be used. In addition to reducing redundancy, making maintenance easier, and speeding up application development, the subject environment has another advantage. That is, many times, data represented does not change very frequently, but the functions that use the data do change. Thus, as in the subject environment, it is best not to embed the data within the function [20:33].

Actual Functional Requirements

Determining the functional specifications for this data base approach required contacting prospective users at WPAFB as well as the AFIT users. The contacts revealed that the Aeronautical Systems Division (ASD) Computer Center had no need at this time for the data base as proposed. After the applications programs are operational, a demonstration of capabilities may change this. The AFML personnel were very interested in the concepts and provided much information concerning graphics under the CAD concept. However, the AFML personnel were using computer resources outside WPAFB to supply their graphics needs. Therefore, at the time of this requirements definition, it was decided to keep them informed on progress and capabilities, but that they would not be directly affected by the implementation of the new data base concept. By keeping them informed, however, they may be able to use some of the ideas from the

thesis in future applications at the laboratory. The Avionics Laboratory personnel were very responsive to the concepts of this thesis, especially since they were in the initial stages of implementing the INGRED graphics capabilities. Their requirements will be discussed now.

Because they are in the initial phases of using the INGRED software, Avionics personnel had not yet established a baseline of needs or problems. Therefore, most of the requirements definition focused strictly on the job being done and how the graphics data base could help them. Because of this fact, it is anticipated that more sophisticated requirements will evolve as time passes and familiarity with capabilities increases.

The Avionics Laboratory has several VAX 11/780 machines some of which have the capability to operate under either the VMS or UNIX operating systems. This fact will allow some flexibility in choosing the data base to be used, since several DBMS implementations are compatible with at least one of those operating systems. Since there is no DBMS in use at AFAL, sunk cost will not be a factor in the final decision on which DBMS to choose.

Although there are several uses of graphics within AFAL, the workcenter which will immediately be able to apply the capabilities that result from this thesis investigation is AAAT-3. This workcenter has ordered a VAX 11/780 which will operate, at least initially, using the VMS operating system. Workcenter personnel are interested in possible

acquisition of UNIX for the VAX, but a decision on this point has not been made. This operating system restriction will have to be a consideration in the final decision on which DBMS to choose.

Most displays being manipulated by AAAT-3 will be of 512 by 512 pixels. Displays of 1024 by 1024 pixels may also be needed, and would potentially require an even greater amount of storage per display. Most displays generated will need to be stored either on disk or magnetic tape, and a large majority of those displays will be called back from storage to be changed at a later date. There was a concern over the amount of disk storage space which will be available for saving the displays.

The graphicsdata base which results from this thesis effort will also be used at AFIT, especially within the Digital Equipment Laboratory (DEL). At AFIT, a VAX is available with either the VMS or UNIX operating system. In addition, outside the DEL, a VAX is available with the UNIX operating system. One important use of the system will be to support SADT, graphics research, Structure Charts, and Data Flow Diagrams. Use of these graphics capabilities can help students in several of the mathematics and engineering courses assemble and present professional, accurate requirements and design drawings.

An important consideration to the AFIT requirement is that the generation of graphics displays within the DEL will be extremely disk storage limited. As a result, any

modifications which can result in decreased storage for the displays will be considered of high priority.

One of the irritating limitations of the existing INGRED system is that there is no easy way to determine the names of existing displays, or to delete existing displays from secondary storage. In fact, the only way to determine the existence of displays, or to delete previously created displays is to have privileges which allow changing what is stored under the directory "ROSE.PIKFILE" [24:100]. Even at this, to accomplish these two necessary actions requires the user to terminate the INGRED program and then take the desired action. At the time this thesis was started, the only user with the privilege to change directory "ROSE.PIKFILE" was the system monitor. These problems must be corrected by the new system.

While this thesis document and the associated software were being written, the INGRED package was used extensively by many individuals in the development of their own thesis and course materials. Feedback from many of these individuals, and some of their instructors resulted in the requirement for the following information being stored on disk along with the display information. First there is the need for a file name which can be used to easily refer to the display for future reference. Second the category of a display should be available. This means, for instance, that if a display is being generated for an aeronautical, math, graphics, or software engineering class, that category

information can be stored along with the display. This will allow an interested student or faculty member to list existing displays by category to see if any may be applicable. Finally, the author name should be included along with the other information.

Neither AFAL nor AFIT have requirements for "split second" response to graphics queries. However, the speed with which data is manipulated is certainly a factor [9]. Therefore, timing as outlined by Foley must be provided to insure that users have a reasonable environment in which to work. In providing this environment, the application must be responsive and consistent. That is, the results of user actions must provide what the user expects.

Notation

Structured analysis is a technique used in specifying requirements and design. It is a top-down method of breaking down details into smaller and smaller detail. The objective of structured analysis is to decompose a subject into constituent parts and thus clarify each particular aspect of that subject. There are several methods which fall under structured analysis [21:148,166], but because of its versatility and scope, Structured Analysis and Design Technique (SADT) was chosen for the specification and design phases of this thesis effort. Part of the reason it was chosen was that flexibility which allows SADT to be used for the higher level required in requirements definition, and also at the lower level required in software design. In

addition, SADT communicates ideas quite effectively by showing inputs, outputs, control, and mechanisms to help express the relation of each part of the software to the whole [24:16].

SADT was designed by SofTech using structured analysis methodology. This method, [25:16] as used in this thesis, uses the activity box and data element arrows to depict the system being analyzed. The activity box represents a part of the whole system, and the data element arrows represent input, output, control, and mechanism thus expressing the relation of each part to the whole. An example of an activity box with data element labels is included on the next page.

The SADT diagrams which apply to this thesis effort are included as Appendix C. The A-0 - "Core Graphics System," A0 - "Core Graphics System," A2 - "Perform Graphics Functions," A3 - "Access Graphics Data Base," and A4 - "Output Information" diagrams were a result of the requirements analysis phase, while the remaining diagrams resulted from the design phase, and will be discussed more fully in Chapter III.

The SADT diagrams which resulted from the design phase illustrate that at this point, the thrust of this effort will be to enhance the existing graphics system by providing a means to store the graphics displays in the data base. The way the system is designed, the user can use the capabilities of the INGRED program as originally

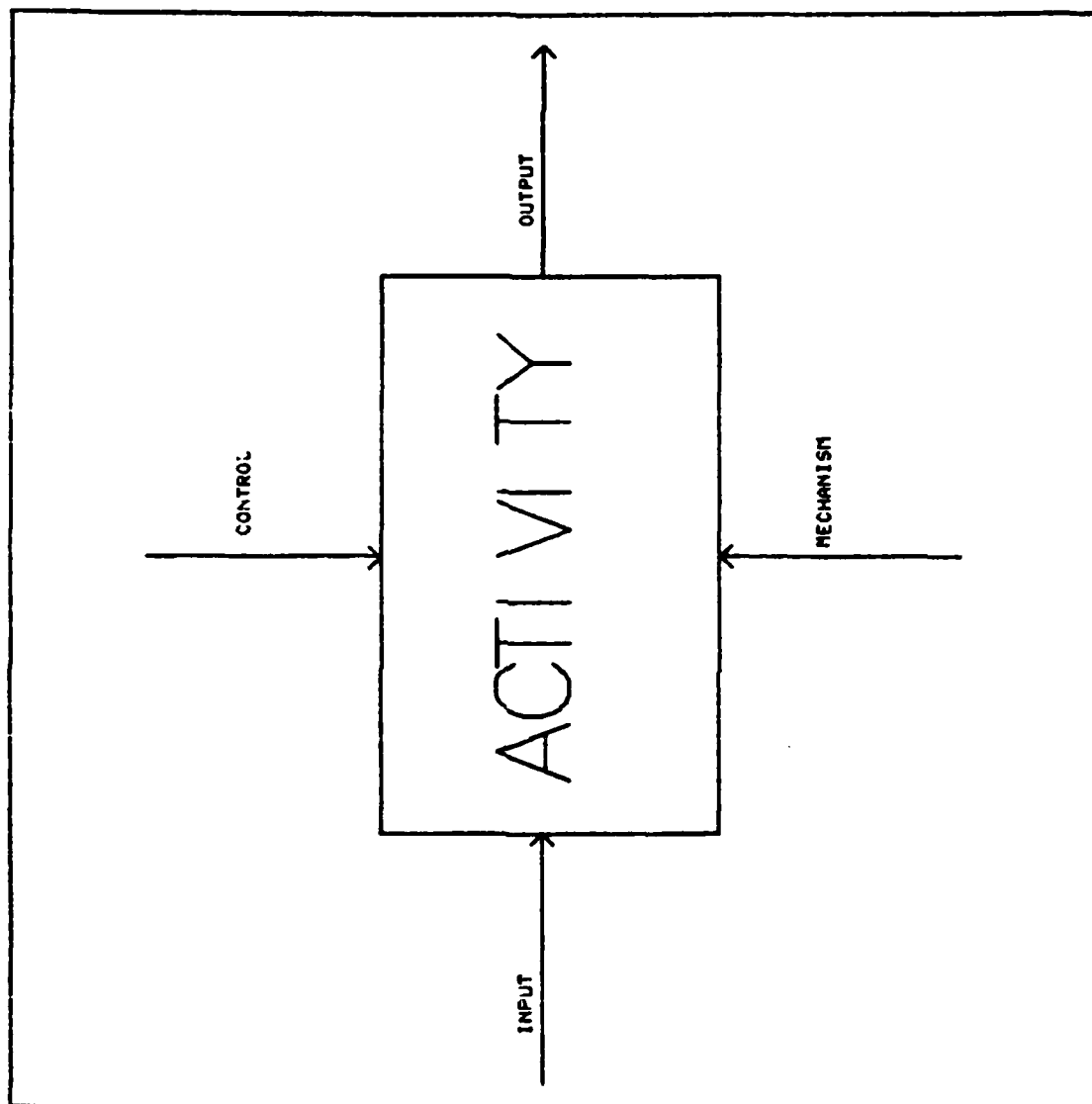


Figure II-1. SALT Activity Box

implemented, or use the additional capabilities which the DBMS provides. Both options will be available to the user based on the response to a screen prompt.

One important factor which was considered heavily throughout the requirements definition and design phases was the need to maintain the integrity of the Core. Since the Core was designed as a standard which could provide portability between applications, it is important not to jeopardize the standardization through a zeal to implement as many capabilities provided by the DBMS as possible. As a result, some things which could have been handled by the DBMS continue to be accomplished by the Core.

Summary

In conclusion, the results of the requirements definition phase are the following specifications:

- a. The graphics display system implemented must be capable of handling, as a minimum, everything that INGRED [24:100] handled.
- b. Handling time is not considered a critical factor, so the human factors were given priority. The average time between a keystroke and the result appearing on the screen when saving and restoring figures will not exceed 15 seconds.
- c. Since disk storage space was a critical resource at each location surveyed, the data base application must optimize the data structure to minimize storage space required. Therefore, saving disk storage space is

one of the primary goals for software design.

- d. The system will facilitate listing of existing displays which a user is authorized to view.
- e. The system will provide for easy deletion of displays from the secondary storage medium.
- f. Along with the display, information will be stored which indicates file name, category, and author of the display.
- g. Since each location surveyed owned a VAX, and the main user was operating under VMS, the DBMS chosen must operate on a VAX with the VMS operating system.
- h. The software must be portable enough so that users at each surveyed location can use displays (if desired) generated at other locations. The requirement for this capability is a new one since none of the users are actually sharing displays at this time.
- i. The possibility of a computer network for this graphics application, although not implemented or required at this time, should be considered in system planning and software development.
- j. The system will provide on-line storage for graphics data.
- k. To simplify retrieval of information and make the system more user friendly, it will operate on a retrieval by name basis.
- l. The system will allow quick access to and transfer of portions of the stored data.

m. The system will be thoroughly tested before delivery to the users.

The goal of this thesis effort then, will be to provide a working system which will satisfy the above requirements. The initial system will establish a baseline from which future enhancements can be incorporated. The addition of the DBMS capabilities will correct some shortcomings of the INGRED system and will also provide for a system which can evolve as the users of the system evolve. The modular design implies that any DBMS chosen should be able to support that design.

III. Design

Introduction

Since the functional requirements have been identified, the next action in this software engineered approach is to design the system which will actually satisfy the requirements. This chapter presents that design. It starts out with information concerning the graphics capabilities already available and then covers rationale for decisions on a DBMS and introduces the idea of structured analysis techniques. Finally, rationale for design of the application program is discussed.

Existing Graphics Capability

A library of graphics subroutines is available at the AFIT DEL which can be used either for applications development or as a graphics research tool [24]. It is an implementation of the George Washington University Core (GWCORE) [33] which is George Washington University's (GWU) version of the 1979 ACM Core Standard [27].

The INGRED software provides interactive graphics editing capability for preparation of two-dimensional pictures. It has the capability to store and retrieve pictures, add and delete segments, and provides for attribute control, error handling, and on-line help.

The editing capability, accessed by using INGRED includes provisions to:

- a. Draw circles

- b. Draw lines
- c. Draw arrowheads
- d. Draw boxes
- e. Erase the screen to start over
- f. Provide a grid which serves as a positioning aid
- g. Write text in five different fonts
- h. Choose color attributes (applies only to color terminals)
- i. Draw a filled box
- j. Delete segments of an existing display
- k. Retrieve an existing display from disk to do editing
- l. Enter text in different fonts
- m. Save a display to disk

The user's manual for INGRED [24:100] provides more specific information about the above capabilities.

There is also the capability with the 'META' program to send to the screen or print a display which has been stored on disk [24:100]. This feature is designed to provide the capability to access stored displays to send the files between using locations. META capabilities will not be used in this thesis since it does not allow editing of graphics figures. However, 'META' will be changed as needed to allow use of graphics figures generated by the new software.

DBMS Selection

With this understanding of the existing graphics package and the requirements, the next step was to determine the DBMS system to implement. There are many DBMS systems

[20] which can support a data base application. Availability and applicability of software of necessity became a prime consideration in the decision on which DBMS to choose for the implementation.

Applicability in this case means that AFAL, AFFDL, AFML, the ASD Computer Center, and AFIT all have access to VAX computer systems. As far as availability is concerned, three DBMS systems are compatible with the VAX and are available at AFIT for implementation and testing. These include Total, Ingres, and the Relational Information Management System (RIM). No other DBMS compatible with the VAX is owned by any of the concerned organizations. Because of the time constraints of this thesis effort, the lead-time, and the expense to acquire any other VAX-compatible DBMS, consideration was limited to these three on-hand DBMS systems.

Ingres is compatible with the UNIX operating system. It is a relational data base released in 1977 by the University of California Regents.

Total is a network data base management system which is compatible with the VMS operating System. It is the most widely used data base in existence today. Total is compatible with multiple host languages, supports multiple file types, allows concurrent access, has advanced recovery capability, supports a variety of data structures, allows multi-level privacy, and supports fixed-length records.

RIM is a relational data base. RIM-5 is the latest version available at WPAFB, and was released in 1982 [11]. It was written by Boeing Corporation under contract to the National Aeronautics and Space Administration (NASA). RIM-5 uses the VMS operating system of the VAX and can be set up as a stand-alone DBMS in which RIM-5 is used to create, update, and/or query a data base; or as an application program interface in which an application program can access a RIM-5 data base. Any language which allows FORTRAN-callable subroutines (i.e. FORTRAN 66, FORTRAN 77, Pascal, COBOL, etc.) is compatible with RIM. RIM is based on the relational algebra model, written in FORTRAN 77, uses B-tree data structure, and provides password protection for read, create, and modify actions [11:iv,3].

The VAX can operate with either the UNIX or VMS operating systems. Both operating systems are available in the AFIT DEL, but not all laboratories at WPAFB have or use UNIX.

Notation

As stated in Chapter 2, SADT diagrams were used for the requirements and design phases. These diagrams are included as Appendix C. The levels specifically applicable to design are:

- A21 - Activate "INGRED"
- A22 - Determine Next Action
- A23 - Change Graphics Figure
 - A231 - Verify File Info

- A232 - Call Change Subroutines
- A233 - Call Access Graphics Datafile
- A24 - Create Graphics Figure
- A3 - Access Graphics Data Base
 - A31 - Receive and Decode Request
 - A32 - Obtain Graphics Data
 - A321 - Receive and Decode MSG
 - A322 - Obtain Requested Data
 - A323 - Send Requested Info
 - A33 - Store Data
 - A331 - Receive and Interpret Request
 - A332 - Determine Location in Data Base
 - A333 - Place in Data Base
 - A34 - Request Output Desired
 - A341 - Request Next Action
 - A342 - Send to Screen
 - A343 - Send to Print
 - A35 - Transfer Figure
 - A351 - Determine Requested Output
 - A352 - Send to "Perform Graphics Function"
 - A353 - Send to "Output Info"
- A41 - Receive and Decode Request Msg
- A42 - Send to Terminal
- A43 - Send to Printer

The top level design of this system, which will henceforth be called INGRED II, is shown in Figure III-1 on the following page. The diagram depicts the INGRED II

application program bringing together the capabilities of the selected DBMS and the GWCORE Graphics System as implemented in INGRED.

Design Considerations

The system design provides the option to the user to use the file storage method or data base capability to store a particular display. This allows use of INGRED essentially as originally implemented if the user prefers.

There is one change which was implemented and affects the file storage technique. This resulted from examination of what was actually stored in the data structure (see Appendix B). Of the 69 opcodes provided by the GWCORE, Opcode 55 is the most often used by INGRED, and each time used requires 31 integer*4 spaces of storage. In most cases Opcode 55 does not change, so under the new software, each time an identical Opcode 55 is encountered in the information packet, an integer*4 code is substituted to allow storage in a much smaller space on disk. An integer*4 value which equates to a number greater than 'one' was chosen for the code. This insures that there will be no conflict with valid integer*4 values which must represent numbers between zero and one. When the display is called back from disk by a restore request, the Opcode 55 information is then reinserted in the appropriate locations. Thus, the GWCORE only sees the information in the information packet exactly as it expects to see it. This new software insures that actual storage requirements are

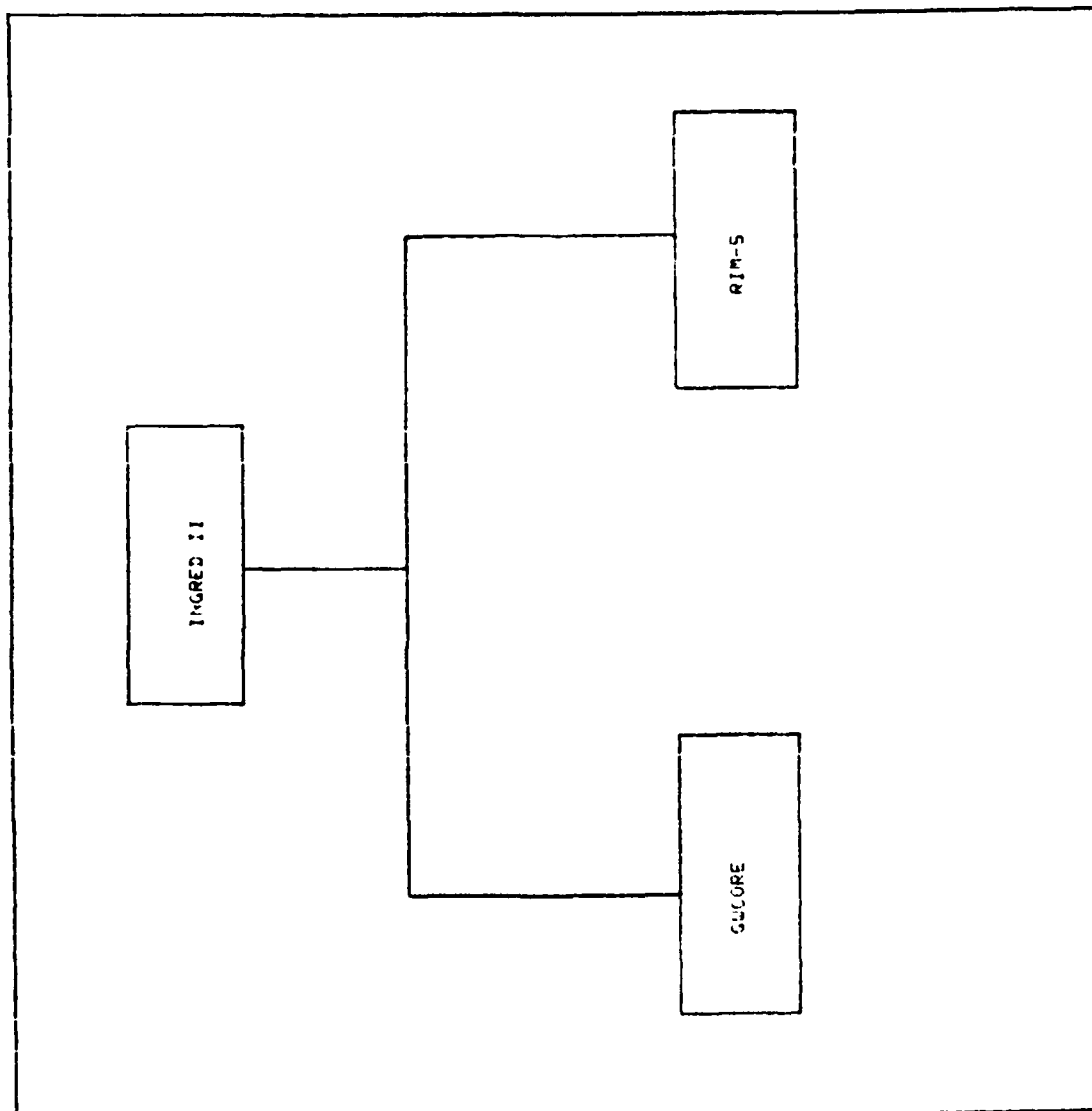


Figure III-1. INGRES II Top Level Design

reduced significantly while Core integrity is maintained. More specific information about the storage savings is included in Chapter 5.

The SADT's (Appendix C) show how the data base is brought into the storage and retrieval scheme of INGRED II. In addition, Figure III-2 shows the SADT drawing which depicts this choice point between storage in the data files or in the data base. Only when the user requests storage or retrieval through the data base will the applications programs to interface the INGRED capabilities to the DBMS be used. The DBMS will provide for storage of additional information, filename, category, and author (see Chapter II, page 7) with each display to make the stored displays more easily accessible. The inherent DBMS capabilities will also make it easy to change the type of information stored with the displays. Therefore, future enhancements to the INGRED II system will not only be possible, but also relatively easy to accomplish. These required enhancements will be identified as the users of the system become more familiar with the INGRED II capabilities and limitations.

RIM Capabilities

RIM is an evolving system, and the latest version available at WPAFB, RIM-5, contains several enhancements relative to RIM-4 [11:2]. These include:

1. Highly portable FORTRAN code.
2. Additional scientific attribute types for vectors and matrices.

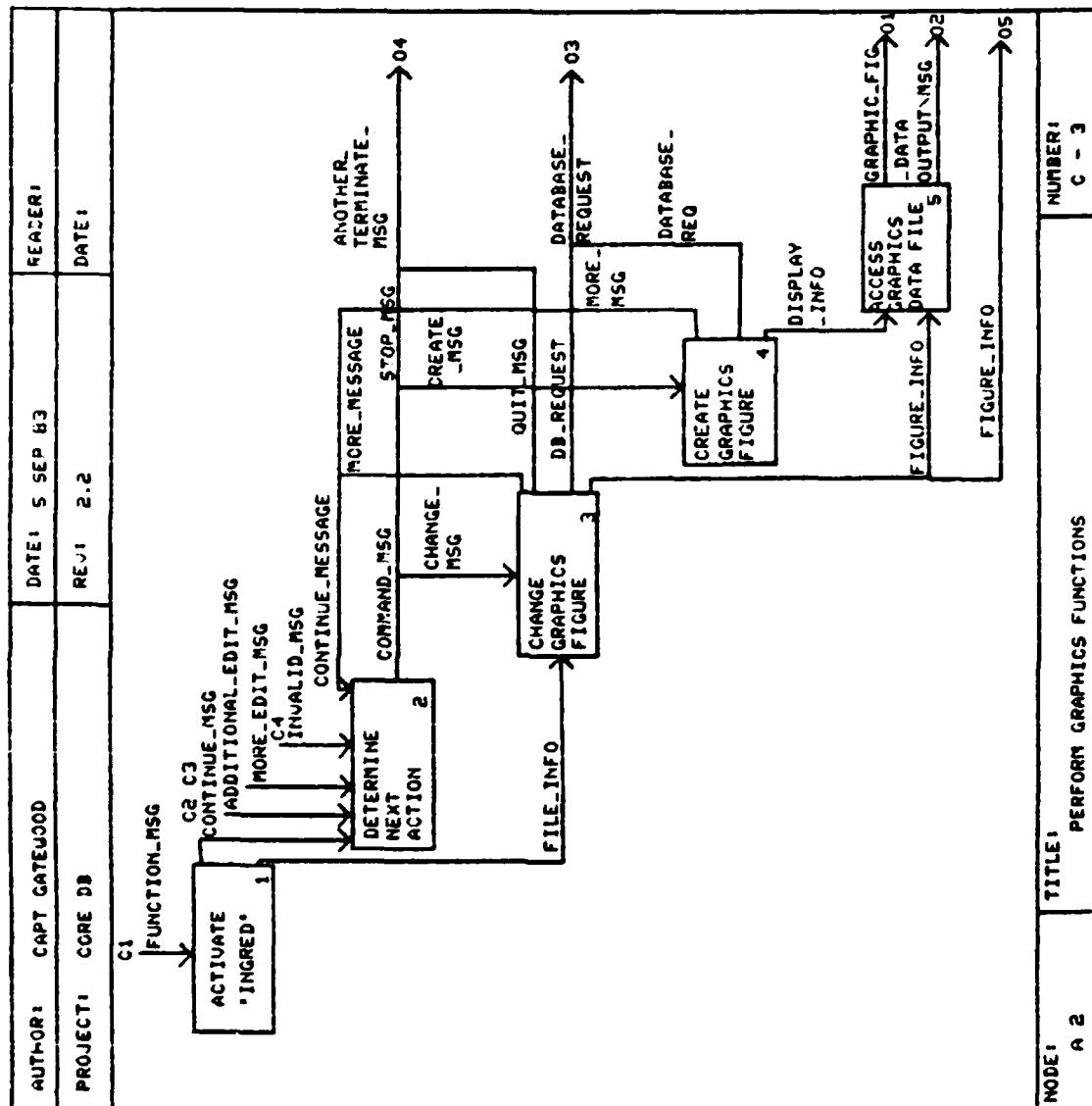


Figure III-2. Storage Procedure; INGRED vs INGRED II

3. Variable length attributes.
4. Improved sort option.
5. Improved where clause.
6. An initial set of report writing commands.
7. Introduction of a tolerance choice for floating point numbers.
8. Additional schema modification commands.
9. Enhanced FORTRAN interface.
10. RIM-to-RIM communications file.

As stated in Chapter I, a further discussion of the RIM DBMS capabilities for data base security is included now. The use of password protection under RIM is optional. Read and/or modify capabilities are possible. The only way a person can access/read a data base which is password protected is as the owner of the data base. In other words, to get into the data base the user must have the password. Relations within a data base can also be password protected. Password protection does not go below the level of the relation, so separate tuples cannot be password protected. There is the capability to change existing passwords.

The application program interface of RIM-5 provides the flexibility to use any high order language which supports FORTRAN callable functions. However, since both the GWCORE and INGRED were written in FORTRAN 77, the source that results from this thesis will also be written in FORTRAN 77.

After evaluation of the information in the RIM-5 User's Manual [11], the decision was made to implement INGRED II

through modification of the standalone subroutines rather than through the application program interface. This is desirable so that as many of the inherent RIM-5 capabilities as possible can be available to users who want to use and query the information produced under INGRED II. Also, this approach provides the flexibility to the system which allows later programmers to use the application program interface for features as desired.

The test plan is also an important part of the software design. Testing is such an important part of the software life-cycle that Chapter 5 of this document is devoted to it. In addition, the actual test plan is included as Appendix F to this document.

Summary

The information discovered in the requirements definition phase along with that discussed earlier in this chapter provide the basis for the final design decisions. These decisions follow:

- a. The INGRED II system will be implemented on the VAX 11/780. This is the logical choice, since AFIT and all of the laboratories have access to VAX equipment.
- b. Since VMS is the only operating system available to all the above users, it will be the operating system used for INGRED II.
- c. Because of the great flexibility of the RIM DBMS, which includes password protection, compatibility with several high-level languages, and variable length

capabilities, RIM was chosen as the DBMS for implementation of the system. In addition, RIM is available to the WPAFB units listed at only minimal cost since it was developed under contract to NASA.

- d. The user will have the choice in INGRED II to store display information using either data file storage of strictly the display, or data base storage which includes information in addition to the display data.
- e. INGRED II will be implemented with as many of the RIM-5 capabilities as possible. This will allow a wider use of the query capabilities of RIM-5.
- f. The only security provided with the initial delivery of the system will be the password protection provided by RIM-5.
- g. The source code will be written in FORTRAN 77 for full compatibility with RIM-5 and GWCORE.

IV. Implementation

Introduction

As stated in Chapter II, it was discovered during the requirements definition phase that an important need for the users of the INGRED capabilities within AFAL and the AFIT DEL is to optimize storage required. Therefore, this was a key area to consider during implementation. To provide a user friendly capability to generate, store, retrieve, and change graphics pictures, INGRED II was conceived. It is a synergistic combination of the GWCORE, INGRED, RIM, and the applications software which invokes needed functions of those programs. This chapter is composed of implementation factors, implementation details, problems involved in making INGRED II a working system, and a summary.

Implementation Factors

File Storage Versus Data Base Storage

One requirements decision (Chapter II, page II-4) was that a subject data base would be the best approach for this thesis effort. Through study of the information packet data structure (Appendix B), however, there was a period of time when it appeared that for this particular application, file storage might be the best approach. The final decision resulted in a subject data base for storage of the text information along with storage of the display data structure in a separate data file. A discussion of strengths and weaknesses considered in choosing this approach provides a

good insight into data base advantages as well as to the INGRED II system as implemented. The discussion considers data files first and then the DBMS approach.

Several strong advantages pointed to choosing a file storage approach rather than a DBMS. These included the following strengths:

First, as implemented in INGRED, the GWCORE handles all manipulations of the data structure. To not corrupt the integrity of the Core, it was decided this would not change. This means that any capabilities of the DBMS which could provide a streamlined manipulation of that data structure were negated. Therefore, the only way to handle the data structure and keep the Core capabilities intact was to take the information packet as passed from the GWCORE, do any manipulations, and then pass it back to the Core. Again this is a problem because it means the Core has already taken the time to handle the data structure before the DBMS gets its chance.

Second, the files being developed were not of such magnitude that searching through the whole list of displays would take any significant amount of time. This may change as more displays are generated, and more applications are found for the software, however.

Third, the data structure optimization which was accomplished in this thesis effort applied as effectively to a file storage approach as to a DBMS. In fact, the file storage approach did not require the overhead of 388 blocks

of storage space required for the DBMS routines. Thus, when considering only the the storage of display information, the data file storage technique could save the displays in less space than the DBMS storage method.

Finally, the use of the file storage approach did not require manipulating the DBMS and so, for this particular case, should not require as much time to store and retrieve information.

However, there are also many advantages to the DBMS approach, and these advantages when implemented, resulted in a system which could easily be expanded. For instance, the file storage approach did not provide a means to store pertinent information along with the display data. The DBMS made it very easy to do this. One possible application for this capability is in storing SADT information. It can be very helpful to know who generated the display, what course it supported, who the instructor was, and what date the display was made.

A second advantage of using the DBMS is that adding information later can be done easily. Thus, if the data base configuration starts out storing the display information along with a filename, later changes which require adding author, date and category can be accommodated easily. It will also be possible to delete part of the information if needed.

Another DBMS strength is that by establishing a key, such as the name of the person who generated the display,

even long lists of pictures can be searched quickly and effectively. This can be very advantageous later, as the number of displays stored increases.

Finally use of a relational DBMS on the VAX 11/780 in the DEL will allow future students of AFIT data base courses to learn more about the relational concept and to compare capabilities with the TOTAL DBMS which already exists on the DEL VAX.

DBMS (RIM)

The decision to use RIM for the INGRED II implementation was influenced heavily by information gathered during the research phase. A version of RIM-4 was available at the AFIT DEL at the time that the thesis effort started. There was also a User's Manual [11] for RIM-5 which showed some of its enhanced capabilities over RIM-4. The combined information for the two RIM versions indicated that the DBMS could do everything needed for the INGRED II implementation. However, there were some problems encountered, and they are discussed in the next section of this chapter.

After the decision was made to use RIM, it was learned that the RIM-4 version available at AFIT did not have all files required to successfully run it. After much research, and contact with various sources, the author determined that a version of the new RIM-5 was available from the AFML personnel, and that version is what was used. Note that the RIM-5 version was not obtained until late in the effort, and

it was not until this time that coded routines could be run to determine effectiveness and correctness.

The source code for RIM-5 required over 4689 blocks of storage on the VAX 11/780 disk unit. The version that was received provided source code written in FORTRAN 77. There are 221 '.for' subroutines, 11 '.com' files, 50 '.blk' files which are brought into the RIM subroutines by use of the FORTRAN "INCLUDE" statement, and 9 '.dat' files. A RIM library was then generated by compiling all the subroutines, and this library was linked with INGRED II.

The first step taken with the RIM-5 DBMS after it was successfully compiled and linked was to run it in the stand-alone mode. This approach allowed validation that the RIM-5 software was capable of performing functions as outlined in the RIM-5 User's Manual [11]. Exercising the functions of the stand-alone system insured that all subroutines which are called by the INGRED II applications program worked properly. A further discussion of RIM-5 is included as Appendix I.

INGRED II

The INGRED II changes to the grid display provided by INGRED were kept to a minimum. Additional options are provided which allow deletion of display files and the capability to get a directory of display files which are stored on disk. The grid display as it now appears under INGRED II is included as Figure V-1 on the next page. Since the INGRED package is easy to learn, it was desirable to

change the way it appears to the user as little as possible. Thus the enhancements provided under INGRED II are natural extensions to the original package. They make it more user-friendly and functional by providing directory and purge capabilities along with the ability to store information in addition the the display data structure (see Chapter III and Appendix I).

It was mentioned in Chapter 2 that the INGRED II implementation must provide at least all capabilities which INGRED provided. This was accomplished, and a display which was generated using the INGRED II capabilities is included as Figure V-2. To implement the capabilities provided by this thesis, the only changes to the source code for INGRED were in the main program and within subroutines PUTPDF, RSTOR, and COMMANDS.

In the main program, calls to PUTPDF and RSTOR were supplemented by a call to DB. The user decides which of these subroutines will be accessed by choosing the appropriate data base or data file operation as indicated on the graphics screen. By selecting 'STORE', 'RESTORE', 'DIRECTRY', or 'PURGE' the program will access subroutine DB which allows manipulation of the data base. This is how the user is provided the capability to choose either the data base storage system or the file storage system for storage and retrieval of displays. Code was also added to send an INGRED II overlay to the screen upon each first invocation of the INGRED II program. The actual code for subroutine

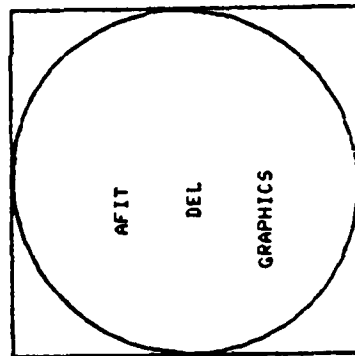
INCRED II

BY :

JIM GATEWOOD

KEVIN ROSE

DR. GARY LAMONT



IV-1. INCRED II Template

DB is included starting on page H-16 of this thesis. Calls to subroutines DBLOAD, RESTOR, DIRECT, and PURGIT are included within DB. These allow loading of the data base and data file, restoring a figure from the data file, obtaining a directory of the files in the data base, and deletion of a file from the data base and data file respectively. The capabilities of these subroutines are described in Appendix I. Neither the capability to obtain a directory nor purge a file existed under INGRED. Figure V-2 shows the INGRED capabilities with secondary storage compared with the INGRED II capabilities. Finally, in the COMMAND subroutine, changes were made to the grid display to add the commands which implemented capabilities added under INGRED II. These include: STORE, RETRIEVE, PURGE, and DIRECTORY.

In PUTPDF the only change was that a call to CMPACT was added. CMPACT is the subroutine (H-37) which checks the PDF for Opcode 55 occurrences and substitutes a flag integer*4 value when Opcode 55 is equal to the default values as shown in the GWCORE Design Document [33:36]. The rationale for this compaction is that since the work at the AFAL and AFIT is being done on Tektronix 4014 Graphics Display Terminals, and since INGRED II does not provide capability to rotate presentations, Opcode 55 values will remain unchanged in a large majority of cases. Since Opcode 55 is included in the information packet by GWCORE each time a permanent segment is created, there is much unneeded repetition of identical

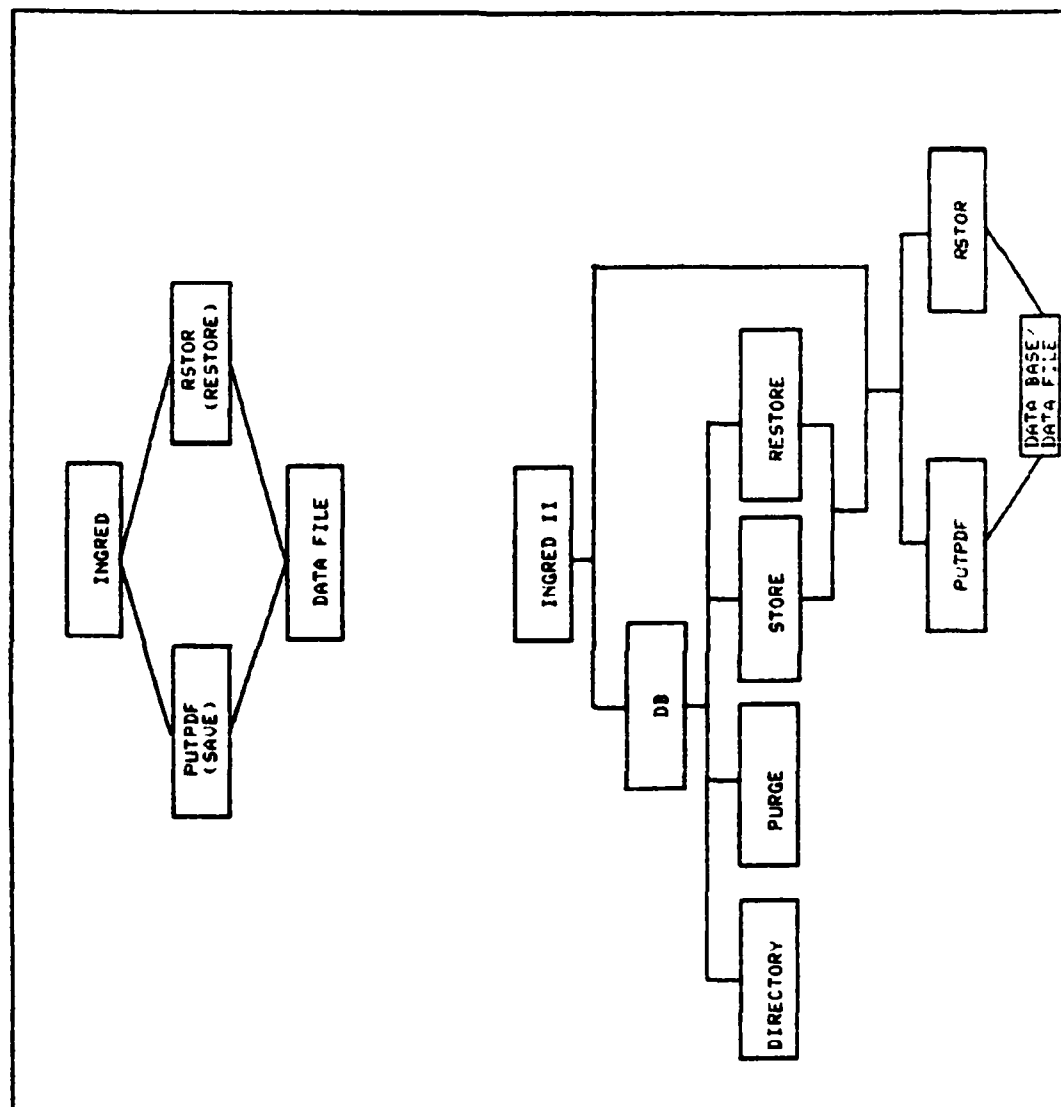


Figure IV-1. Storage Procedure; INGRES vs INGRES II

Opcode 55 entries. If any values for Opcode 55 have changed from the default values, that particular occurrence of Opcode 55 will be stored in the information packet in its complete form. CMPACT is also used when files are stored in the data base.

Just as INGRED II routine CMPACT was called from PUTPDF to compact the data structure, PUTIN is called from RSTOR to put the Opcode 55 information back into the data structure when a display is called back from secondary storage to be handled by INGRED. PUTIN finds occurrences of the integer*4 code which signifies where an Opcode 55 was deleted, and reinserts the Opcode 55 information. PUTIN is also used when files are stored in the data base.

Differences between the grid display of INGRED and INGRED II were discussed previously. These changes to the display help the user choose between the capabilities which existed with INGRED or the additional capabilities of INGRED II. The procedure for using the INGRED II data base capabilities can be explained very simply. To put information in the data base/data file instead of the data file, requires typing STORE instead of SAVE. Obtaining the display from the data base/data file requires typing RESTORE instead of RETRIEVE. The other two data base capabilities were not available with INGRED, so their invocation should also be no problem. The user can just type PURGE or DIRECTRY to use the capability required. As in the earlier discussion, the changes were made as simple

as possible in order to keep the learning process easy for users.

The data base established at this time is GRAFDB. It consists of four RIM-generated files: GRAFDB1.DAT, GRAFDB2.DAT, GRAFDB3.DAT, and SCHEMA.DAT. They contain respectively, the schema, actual data for each relation, pointers for keyed attributes, and additional schema information. The data base was generated using RIM's standalone mode, but additions, changes, and deletions under GRAFDB are made using INGRED II.

The GRAFDB data base contains one relation, "Graphics," which holds the data for all displays generated under INGRED II. The Graphics relation contains three attributes, file, category, and author. The file attribute contains the filename selected by the user, the category contains the type of application the display was generated for, and the author attribute contains the name of the author of the display. The display information is stored in the data file and is accessed by invoking the RESTORE command. When this is done, the program verifies that the file is listed in the data base and then goes out and gets the information stored in the data file. There is no limit under RIM-5 [11:D-1] to the number of tuples which can be placed in the relation.

The idea to save all display information under the same relation should prove adequate initially. However, as the number of displays increases, it may become desirable to allow storage of display information by category. This can

allow cataloging by purpose such as data base displays, math displays, aeronautical displays, graphics displays, etc. Then having a relation for each category will allow quicker sorting and make it easier for a user to find a particular stored display.

Implementation Details

It is evident from the SADT diagrams (Appendix C), the data dictionary entries (Appendices D and E), and previous discussion in this chapter, that INGRED II is designed to allow the user of the INGRED graphics capabilities to continue to generate and change pictures without any significant procedure changes. The top-level SADT, A0 (C-3) shows that all graphics functions, including storage and retrieval can be done without accessing the data base. If the decision is made to bypass the data base, all actions are taken within activity A2, "Perform Graphics Functions", which equates to the old INGRED program. The decision to use the data base rather than the data file is also made within activity A2.

The decision to use the data base for storage or retrieval of the display information is made in SADT activity A23 or A24. This choice is made when the 'NAMPAS' common block variable COMND receives the user's entry of the appropriate command at the terminal. Commands SAVE or RETRIEVE result in subroutines PUTPDF or RSTOR being called from main. STORE or RESTORE result in subroutine DB being called, and then the appropriate database manipulations are

done before PUTPDF or RSTOR are called. Within subroutine DB, when a 'SAVE' is performed, the information packet information is compacted and then stored on disk. In contrast, during a 'RESTORE' operation, the requested data is obtained from disk and then uncompactd to be sent to the screen.

Because the data base files ([GATEWOOD].PIKFILE) are separate from the GWCORE generated data files, ([ROSE.PIKFILE]) displays stored by one method cannot be restored from the files created by the other method. In other words, if a display is stored in the database, it cannot be called by requesting a 'RETRIEVE' from the data file. It must be requested from the data base.

To make INGRED II as upward compatible with INGRED as possible, subroutine PUTIN is structured so that it can uncompact a file generated by INGRED. This is because PUTIN processes the data structure information normally until the -9999899 values (see Appendix B) are encountered which signify that an Opcode 55 belongs in that location. Since there are no -99998999 values generated under INGRED, the displays are processed normally. INGRED cannot be used to restore files created by INGRED II since INGRED does not recognize the -99998999 designator.

One advantage to the way INGRED II was implemented is that the standalone mode of RIM-5 can be used to query the GRAFDB data base created by INGRED II. If, for instance, an individual wants to find out what displays are stored under the category software engineering, that individual can run

RIM (see Appendix I), formulate a query, and obtain the desired information.

To compare the amount of space required to store information with INGRED versus with the new INGRED II, a comparison case was developed using the SADT drawings in Appendix C. Close study of the data structure for one specific SADT display disclosed that Opcode 55 occurred 115 times. Opcode 55 is a GWCORE function called 'SET PRIMITIVE ATTRIBUTES' [Wenner,36], and carries information on what characteristics are desired by the GWCORE user. It consists of 31 integer*4 entries, and thus requires a significant amount of storage space. A detailed breakdown of Opcode 55 is included as Appendix B. Only two of the 115 times that Opcode 55 occurred in the studied example did any of the opcode entries change. Therefore, eliminating Opcode 55 entries in this particular case could have saved a significant amount of storage space. With this information, it was logical to try to find out if similar savings would hold with larger samples.

To provide a control group for the comparison, the SADT drawings were first stored using the INGRED file storage approach. The total disk space required to store the 12 displays was 2090 blocks. Next the displays were stored using INGRED II. The required storage space was 1136 blocks. Thus INGRED II required only 54.35 per cent of the storage space that INGRED required. A large part of the INGRED II savings resulted from optimization of the storage

based on knowledge of the actual content of the data structure.

Going back to the example of the SADT diagrams, the space savings could have been even more impressive if circles had not been used on two of the drawings. This is because circles are stored as a sequence of very short lines. This requires many entries for just one circle. It is impossible for the optimization technique to improve this situation. Thus, by using boxes instead of circles, more storage space can be saved per diagram. The actual space required for storage of all diagrams except the two with circles is 662 blocks with INGRED II as opposed to 1350 blocks for INGRED. Using these figures, INGRED II requires only 49.04 per cent as much storage as INGRED for the same displays. Table IV-1 shows the results of the storage space comparisons for INGRED and INGRED II.

Another area for comparison was the time for storing and retrieving under the new system versus under the old. Again, the SADT control group was tested using the INGRED storage versus INGRED II storage. The result of these tests was that the average restore time was 10.4 seconds using the INGRED file storage method. For retrieval using the data base as provided by INGRED II, the average time was 8.5 seconds.

It is easy to see from the above information that storage of only a few displays under INGRED II can save a substantial amount of disk space. Surprisingly, the data

<u>SADT</u> <u>NUMBER</u>	<u>INGRED</u>	<u>INGRED II</u>
C-1	52	27
C-2	412	308
C-3	214	107
C-4	132	67
C-6	181	88
C-7	109	55
C-8	108	55
C-9	228	166
C-10	123	62
C-11	109	55
C-12	89	46
C-15	179	90

Table IV-1. Storage Comparison - INGRED vs INGRED II

base approach also resulted in shorter retrieval times than the data file approach. However, overhead is created by the room required by the RIM-5 DBMS since it requires 388 blocks of storage on disk. This will be an insignificant amount of space for any user that deals with graphics displays regularly. Saving the displays from the 12 test case examples saved almost three times as much disk space as that required for RIM-5.

Problems

Four problem areas made this thesis effort difficult to complete. The first three are related to RIM and actually had the largest impact from the standpoint of time required to solve the problems. Since this was the first real experience with RIM-5 at AFIT, problems of this type were not unexpected. The fourth problem was unexpected since it had to do with the already operational INGRED software. These problems are discussed below:

The first problem was that the AFIT RIM-4 source code which the author was able to locate did not contain all files required to successfully run RIM. To determine why, required printing out all available RIM-4 source. Close examination of this source revealed that the ".BLK" files which were required for the FORTRAN 77 "INCLUDE" statements were not available either on the RIM-4 disk files or backup tape files. These '.BLK' files contained common statements which were required for the RIM routines. Without proper

documentation; it was impossible to determine what was contained in these '.BLK' files.

Since this problem seemed unsolvable without obtaining another copy of RIM-4, and since the RIM-5 User's Manual was available, the effort turned towards obtaining a copy of RIM-5 which could be used. Contact through Boeing, which originated RIM, indicated that there would possibly be problems obtaining the RIM-5 source and documentation in time to use for INGRED II. Finally, coordination with personnel from AFML disclosed that they had a working copy of RIM-5 including source code. This source was copied onto tape in the required VAX format. The only documentation available was the RIM-5 User's Manual already mentioned.

The second problem, the lack of documentation or programming experience with RIM was also significant. Because RIM-5 is so new, no one at AFIT had used it before. This meant that the author had to learn about the system strictly from the available documentation, the source code, and experimenting with the RIM executable version. Since the user's manual was not designed for the programmer, it only touched very briefly on such important topics as the RIM-5 data structure, the RIM-5 applications program interface, and actual FORTRAN 77 examples. In fact, there were no examples of actual working RIM programs included in the user's manual.

This problem was compounded by the fact that some of the information that was available in the RIM-5 User's Guide

was not accurate. A case in point is password protection. According to the manual [11:11] password protection is implemented in RIM-5. As stated earlier in this thesis, that was one of the factors which led to the selection of RIM-5 as the DBMS for INGRED II. It turned out that password protection is not implemented in RIM-5, and that the subroutine which is to check password validity always returns an affirmative value. The net result of this is that the lack of documentation or of experience with RIM-5 added significantly to the development time required for the INGRED II software.

The third problem is related to the second. This problem is that information in the RIM-5 User's Manual did not contain the level of detail required for programming a project of the magnitude of INGRED II. As a result, much time was wasted in pursuing approaches which, if better documentation had been available, would not have been tried.

Another problem, not related to RIM, was with INGRED. There was originally no plan to do any programming which dealt with the graphics portion of the INGRED II. However, use of INGRED to generate the SADT diagrams in Appendix C of this thesis revealed that there is an intermittent problem when saving displays with INGRED. The problem was that sometimes a whole display will not be stored. For example, if the display consisted of five boxes, perhaps only three boxes and a line from one of the other boxes will actually be stored properly on disk. Examination of this problem

revealed that extra values sometimes are included in the information packet. For example, Opcode 55 might include 46 integer*4 values instead of the 31 it was supposed to contain. This, of course, causes the GWCORE to do unpredictable things with the display.

The experience in this area indicates that the problem occurs when a user does several DELETE's of segments from the screen. The problem is happening within the GWCORE subroutine which marks segments for deletion, or during the ensuing "squeeze" where those segments are actually eliminated from the PDF before the save to disk. To solve this problem, the delete flag was reset after calls to the GWCORE subroutine GSQUEZ. The problem did not occur after this change was made, however, since this fix was not part of the original scope of this thesis effort, thorough testing was not done. Therefore, close observation will be required of future users to determine if the problem still exists. In the mean time, it is probably advisable to minimize the number of deletes done at the screen.

Summary

There were several decisions made during the design and implementation phases which had an impact on the overall direction of the INGRED II system.

First was the choice of the relational RIM DBMS for storage of display files. The decision was influenced heavily by a desire to implement a relational DBMS to provide an alternative to TOTAL for data base work at AFIT.

In addition, since RIM is a DOD system, it is considered public domain software for Air Force units and can be passed between locations at no cost.

Second, retaining the capability to store display information just as implemented in INGRED insures that the integrity of Core remains intact without any "corruption" by addition of DBMS capabilities. This capability, however, is inherently limited by use of data file storage.

Third, DBMS storage in addition to data file storage will give INGRED II flexibility which allows listing of existing display files, deletion of files, and storage, as needed, of pertinent information along with each display.

Fourth, making the INGRED II implementation appear to the user almost exactly as INGRED did, allows continuation of the already proven user-friendly commands and prompts.

The overall goal of the implementation phase was to establish a system which supports the requirements identified in Chapter 2. INGRED II accomplished this, and in addition, established the groundwork around which improved graphics capabilities can be built. INGRED II effectively melded the capabilities of INGRED, GWCORE, and RIM-5. It provides a useable graphics capability for use by the Air Force Wright Aeronautical Laboratories (AFWAL) and by students and instructors at AFIT.

V. Testing

Introduction

Recent research has shown that testing consumes over fifty percent of the labor required to produce a working program [3:1]. Testing and test design must be done with the idea of eliminating program errors, but must also focus on error prevention. As will be discussed later in this chapter under the section on the Software Lifecycle, it is much more desirable to prevent program errors than to correct them. Thus, the primary goal of testing must be to prevent program logical errors. This is a goal which cannot be met completely since in any new intricate program, errors will exist [3:3].

This chapter begins with a discussion of considerations which applied to the INGRED II test approach. This is followed by information concerning the test results. The final section of the chapter is a summary of the test phase.

Errors

The types of errors which can exist in a program can be broken into three general areas [22:509]:

1. An otherwise logically correct program contains one or more isolated statements or instructions that are syntactically incorrect in the programming language being used. Syntactic errors are usually recognized by the language processor and

the programmer must make the changes required to correct the errors.

2. A potentially correct algorithm may be coded in a logically incorrect way. Logical errors are also called semantic errors. If all syntactic errors are eliminated and the program does not execute as intended, logical errors exist. They are the result of improper understanding by the programmer of certain operators or mistakes in coding of an algorithm.

3. The algorithm implemented may function correctly for some but not all data values, or worse, may fail for just a few combinations of input values.

These errors create debugging problems of increasing severity and may require substantially different testing approaches.

Test Methods

Eliminating these errors requires designing tests which are thorough and effective. Test design can be approached from two different points of view[3:5]: functional testing and structural testing. In functional testing, the program, or module, is treated like a black box with inputs and outputs. The idea is that a certain input should result in the correct output, and this is what is checked in testing. Beizer further states that functional testing takes the user's point of view.

Structural testing, on the other hand, looks at the implementation details. Thus structural testing is concerned with what goes on inside the black box in achieving the output. Such things as coding details, control method, and programming style are included in structural testing.

Both structural and functional testing are important, and should be used in test design and actual testing. Functional tests, if done rigorously, can detect all errors, but it would take a large amount of time to do so. Structural tests, on the other hand, would not detect all errors, but inherently take less time to accomplish [3:5]. Thus to achieve the best possible tests, there must be a good balance between structural and functional tests.

Two other methods used to insure complete testing are static and dynamic analysis [20]. The approach in static analysis is to find errors by analyzing the program at all phases. Program code is not actually executed to find these errors. Methods by which static analysis is done include: walkthroughs, checklists, code inspections, and simulation and modeling. More information on these static analysis methods can be found in [20].

Using these methods, the test team members do:

1. Type analysis which strives to verify such things as the fact that only integers are used in variables declared as integers.

2. Unit analysis which validates, for example, that distances are in miles and speeds are in miles-per-hour.
3. Reference analysis which validates that when a variable is referenced, it contains the correct value.
4. Interface analysis which validates that arguments are passed correctly and that functions return correct values.
5. Expression analysis which searches for things such as out of bounds array references, division by zero, unreachable code, and undesired GOTO statements.

Dynamic analysis, unlike static analysis, includes actually running the program to determine what problems exist. In dynamic analysis, it is important to test that every module of code is reached and executable. This must be carried to the point that every statement is executed and every branch taken [3:46]. Beizer feels that rather than include untested code, it is better to leave it out completely since it can make other functions fail that would otherwise work.

In path testing under dynamic analysis, it is better to take many simple paths rather than a few complicated ones [3:47]. In addition, there is no problem with taking paths that will exercise portions of code more than once.

Test case design with dynamic analysis can include:

1. Error guessing in which random tests are run at areas the programmer thinks might have problems. This is traditionally one of the most used test case designs.
2. Cause-effect graphing which explores combinations such as if a then b. In this approach, decision tables are created from the cause-effect diagrams. Test cases are then written for each decision table case.
3. Boundary value analysis which actually uses maximum and/or minimum values which could be used in a program line. For example if a statement said that if $a \geq 2$, the boundary analysis test would use the values of 1, 2, and 3 to test a value in all possible cases.
4. Equivalence partitioning which takes a value from the acceptable range and a value from the unacceptable range to see that expected results are achieved in both cases.

Software Life Cycle

The Software Life Cycle illustrated in Figure V-1 is an important concept for anyone who deals with computers and the programs that make them work. The Software Life Cycle [21:13] is the current notation for the events which include identifying a problem, developing a way to solve the problem, implementing the solution, making sure the solution

SOFTWARE LIFE CYCLE

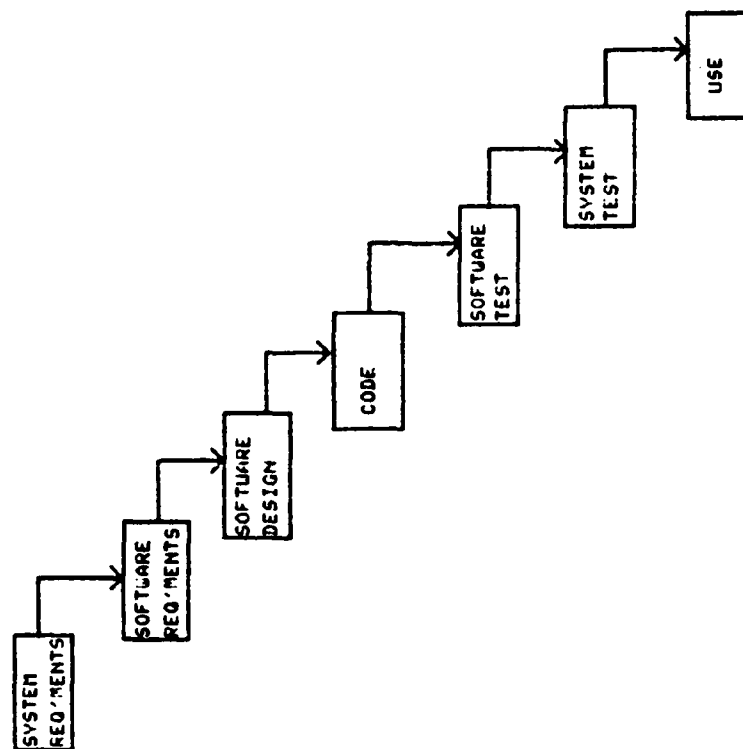


Figure V-1. Software Life Cycle

achieves what it was designed to do, and using the capabilities provided in the solution.

The Software Life Cycle only comes into effect as the result of a user need [21:13]. The steps, as illustrated in Figure V-1 include system requirements, software requirements, software design, code, software test, system test, and use. The Software Life Cycle concept emphasizes the importance of accurately accomplishing the requirements and design steps, because mistakes in those early phases result in mistakes in following phases.

In a survey by TRW [34:203], it was determined that the cost of fixing an error during coding is twice that of fixing it in the design stage. Finding the error during testing costs about ten times as much as in design. This is why software engineering is so important. The better the requirement is designed, the better the design can be done to satisfy the actual requirements. Good design, in turn, results in easier coding. In addition, attention to detail during requirements definition minimizes changes and additions later on during the Software Life Cycle [34:203]. Not until the testing phase is the designer really certain that the requirements and design phases produced the proper results. Even the testing results can be misleading if not properly done.

Test Results

Problems with testing were compounded by the fact that RIM-5 did not always work as indicated in the user's guide

[11] (see Chapter IV). This meant that some design had to be changed after initial implementation, because this was the only time that it became evident that an approach was not compatible with RIM-5. Therefore, during Confidence Testing, subroutines which had been successfully tested individually had to be changed and retested when the design was changed. This also caused problems when Integration Testing was done because some subroutines which had successfully tested individually were impacted in the Integration Testing by changes in other subroutines.

The test plan for INGRED II is included as Appendix F. In the Acceptance Test, INGRED II performed well during the System Compatibility Test (SCT). Of the eight faults introduced, the only one that caused a system problem was when a file name was entered as BOX.1. The period caused a VMS system failure which terminated the INGRED II program. This problem will be addressed in the User's Guide (Appendix I).

The Operational Test was also performed successfully, except for one problem. The user introduced the same fault as in SCT with the same result. This repetition of the problem indicates that this area should be corrected in the next upgrade of INGRED II. This can be done as part of a larger thesis effort, or as a course assignment in the graphics curriculum.

Summary

The testing approach used draws upon the preceding comments in this chapter. One strong contributor to error prevention was the strict adherence to software engineering principles followed in the functional specification, preliminary design, design, and coding of the applications programs for this thesis. Despite the structured techniques, errors were anticipated, and thus a test plan was generated (Appendix F). The actual testing consisted of module testing in which each module was subjected to selected test data, integration testing which tested groups of modules together, and systems testing which included testing the complete system. The systems testing was done only after module and integration testing were completed satisfactorily. The test plan is included as Appendix F. Close evaluation of the modules to be created indicated that several test case designs could be used to benefit. These included boundary value analysis, error guessing, and equivalence partitioning. A combination of these test case designs was used based on the the structure and function of the particular module being tested.

VI. Conclusions/Recommendations

Introduction

This chapter is composed of two main areas: the conclusions and the recommendations. In the conclusion section the advantages and disadvantages of the INGRED II are discussed along with information on the potential impact of INGRED II on WPAFB units. The recommendation section introduces ideas which the author feels can be pursued.

Conclusions

INGRED II achieved the primary goal of this thesis investigation. That goal was to satisfy the customer requirements as discussed in Chapter 2. These requirements include: saving storage; providing all capabilities present in INGRED, plus enhancements; keeping the system user-friendly; and insuring quick transfer of information between GWCORE and the data base. In addition, INGRED II is the first time the RIM DBMS was used in conjunction with the George Washington University implementation of the Core. In fact according to Rocky Krigman, graduate director of the GWCORE project, INGRED II is one of the first efforts to combine a DBMS capability with the GWCORE. INGRED II shows that the graphics and data base packages available can be used to provide a flexible, useable graphics generation system.

INGRED II as implemented provides a Core standard, user-friendly, graphics development system which employs the

advantages of a DBMS to store the graphics displays along with any additional information desired. INGRES II allows the user the flexibility to use the capabilities of the DBMS to store information along with each display, or to stay strictly with the INGRES disk storage technique of data file storage.

Since the INGRES graphics editing capabilities are so easy to learn, it was decided to provide the enhanced capabilities of INGRES II by changing the INGRES procedures as little as possible. Disk storage using combined DBMS and data file capabilities was chosen to provide an alternative to strict data file storage because of a substantial list of DBMS strengths.

INGRES II optimizes the data structure being used to store information on disk, and as a result, using either data file storage or combined data base/data file storage with INGRES II saves almost 50 per cent of the storage space required by INGRES. This optimization was especially important since a shortage of storage space for saved displays was a main concern for both potential major users of INGRES II.

There is one obvious disadvantage of the DBMS capabilities of INGRES II. This disadvantage is space. In an application for units concerned with disk storage space INGRES II requires, as a minimum, use of the 368 blocks of storage required for its executable file. It was anticipated that a second disadvantage would be that the

system works slightly slower using the data base storage than with strict data file storage. However, in the comparison testing mentioned in Chapter IV, the data base storage method proved slightly faster. This factor makes the space problem seem minor in nature. In fact, it is anticipated that as the number of displays stored to disk increases, the 368 blocks of storage required for INGRED II will become insignificant. The speed of finding files will also become even faster in comparison with sequential searching when the RIM-5 DBMS capability for keyed searching is implemented.

The advantages of INGRED II become more noticable as the amount of storage and the flexibility of the system are considered. It seems prudent to conclude that the addition of the DBMS capabilities to the inherent Core functions will insure continued use and growth of the INGRED II system.

Recommendations

At several stages of the thesis development it appeared that there could be benefit from establishment of a network between the graphics users at WPAFB. No investigation was done to determine whether similar displays are, or could be used, by different organizations. However, the potential is certainly there. The large number of VAX 11/780 systems available also lends itself to networking of more than just graphics capabilities. A good thesis effort would be to study missions and what is actually being done by certain units and then, if feasible, set up a network or

distributed data base or some other system to better use the information and capabilities available. For example, with a network set up, a display could be generated at one unit and could be accessed from the other units for either viewing or change to suit user needs.

The optimization done on the data structure of the information packet in this thesis effort was limited to Opcode 55 which was the largest and most frequently occurring opcode. Further optimization of the data structure can result in savings of even more disk space when storing graphics displays. For example when Opcodes 47 and 49 appear in sequence, of the first 11 integer*4 occurrences, 9 can be saved when storing the information on disk and then reinserted when the display is restored. A good savings can also be achieved with Opcode 3 optimization. Compaction of these two instances, combined with the Opcode 55 compaction already done will save over 50 per cent of the storage required on disk by INGRED.

It is important to note that these savings in disk storage space may cost response time, and any effort to save space should be accompanied by a study to determine trade-offs between time and space considerations. However, this author's initial studies indicate that the loss in response time will be negligible compared with the savings in space. This is especially true when considering the importance of conserving space as cited by AFIT and AFAL personnel.

It should be noted that in this thesis effort, the Opcode 55 information was compacted whenever it matched the default values set up by the GWCORE. This works well for SADT-like diagrams on the Tektroniks 4014, but if color graphics terminals are used, or if characteristics such as text font are changed often, optimization will be minimized. If the use changes to these types of requirements, the Opcode 55 optimization can be changed to meet the requirements. For example, if these things are being done, probably a better scheme would be to compare the new Opcode 55 value with the value which was used most recently in addition to comparing with the default values. This could provide a more reasonable comparison.

There are areas under the data base which can also be improved as use of INGRED II warrants. As use of the INGRED II increases there may be the requirement for an additional attribute which can inform the user whether display information requested is on the currently mounted disk, and if not, what disk it is on. This example shows one of the advantages of a DBMS. When needs change, the information in the data base can be easily changed to meet those new needs. There will certainly be other enhancements of the INGRED II which will be needed. These can be more easily recognized with increased use of the system.

As stated in the first chapters of this thesis, the only security provided was to be via password protection.

However, because the RIM-5 User's Guide [11] was incorrect, it turned out that there is not even password protection provided with INGRED II. Since there is a later version of RIM available, RIM-6, it should be determined whether this version does have password protection. If so, this version should be obtained and INGRED II should be implemented with the password protection capability.

With increased use of INGRED II, however, security can become an important area which may have to be addressed to insure use by certain units. There is much information on data base security, and adding that type capability to INGRED II could be a thesis effort for some future student.

During the research phase it became evident that there is much impetus for GKS to replace the Core. It would be a good area for AFIT to set the groundwork for an implementation of the GKS just as GWU did with the Core. This would be too much of an effort for one thesis student to implement, but could provide a good environment for a coordinated effort between a group of thesis students. In fact, the GWCORE was a collective effort of several individuals at GWU.

Although the above effort would have to be a combined endeavor, it is desirable for one thesis student to research the similarities and differences between GKS and Core, and then determine the feasibility of changing AFIT Core programs to GKS. Another possible approach would be for

a student to suggest a structure for implementing an AFITGKS by determining how to direct the thesis efforts of several AFIT students in generating the AFIT implementation of the GKS.

BIBLIOGRAPHY

1. Ames, Stanley R., Jr., Marrie Gasser, and Roger R. Schell. "Security Kernal Design and Implementation: An Introduction," Computer, 16: 14-22, July 1983.
2. Angell, Ian O. A Practical Introduction to Computer Graphics. London: The Macmilan Press Limited, 1981.
3. Beizer, Boris. Software Testing Techniques. New York: Von Nostrand Reinhold Company, 1983.
4. Chang, Shi-Kuo and Tosiyasu L. Kunii.. "Pictorial Data-Base Systems," Computer, 18: 13-21, November 1981.
5. Chock, Margaret, Alfonso F. Cardenas, and Allen Klinger. "Manipulating Data Structures in Pictorial Information Systems," Computer, 18: 43-50, November 1981.
6. Curling, Harold. Design of an Interactive Input Graphics System Based on the ACM Core Standard, MS Thesis. Wright-Patterson AFB, Ohio; Air Force Institute of Technology, December 1980.
7. Deutsch, Michael S. "Software Project Verification and Validation," Computer, 18: 54-70, April 1981.
8. Enderle, Gunter. "Graphics Metafiles A Base for Storage of Graphical Data in CAD Systems," CAD/CAM as a Basis for the Development of Technology in Developing Nations. 67-75. Amsterdam: North Holland Publishing Company, 1981.
9. Foley, James D. and Victor L. Wallace. "The Art of Natural Graphics Man-Machine Conversation," Proceedings of the IEEE, 62 (4): April 1974.
10. Giloi, Wolfgang K. Interactive Computer Graphics. Englewood Cliffs, New Jersey: Prentice-Hall Incorporated, 1978.
11. Gray, F. P. and S. O. Wahlstrom. User Guide: RIM 5.0 VAX VMS. Washington: Boeing Commercial Airplane Company, 1982.
12. Hadfield, Steven M. An Interactive and Automated Software Development Environment. MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1982.
13. Hammer, Michael and Stanley B. Zdonik, Jr. "Knowledge-Based Query Processing," Proceedings of the Sixth International Conference on Very Large Databases. IEEE, October 1980.
14. Hanlon, James. Implementation and Evaluation of Input Functions for the 1979 Core Graphics System. MS Thesis.

Washington: Electrical Engineering/Computer Science Department, George Washington University, May 1981.

15. Heidler, W., et. al. Software Testing Measures. Griffiss Air Force Base, New York: Rome Air Development Center, May 1982. (RADC-TR-82-135)

16. Howden, William E. "A Survey of Dynamic Analysis Methods," IEEE Tutorial on Software Testing and Verification Techniques. IEEE, 1978.

17. Jones, P. F. "Four Principles of Man-Computer Dialog," Tutorial: Computer Graphics, edited by Kellogg S. Booth. New York: IEEE Computer Society, 1979.

18. Marshall, George and John Rainey. An Infrared Image Processing Database for a PDP 11/10 Minicomputer System. Washington: Defense Technical Information Center, 10 December 1982. (AD B069597)

19. Marshall, Neil, et. al. Digital Image Database Storage and Retrieval. Washington: Defense Technical Information Center, 31 Aug 1982. (AD B067187)

20. Martin, James. Managing the Data-base Environment. Englewood Cliffs, New Jersey: Prentice-Hall Incorporated, 1983.

21. Peters, Lawrence J. Software Design: Methods and Techniques. New York: Youdon Press, 1981.

22. Ralston, Anthony. Encyclopedia of Computer Science and Engineering (Second Edition). New York: Van Nostrand Reinhold Company, 1983.

23. Ricks, Jeffrey S. and Robert S. Colburn. Analysis of Information Requirements and Design of the Consolidated AFIT Database and Information System (Cadis) With an AFIT/CI Implementation Design. MS Thesis. Wright-Patterson Air Force Base, Oh: School of Engineering, Air Force Institute of Technology, December 1982. (AFIT/GCS/EE/82D-11)

24. Rose, Kevin W. Development of an Interactive Computer Graphics System Library and Graphics Tools. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1982. (AFIT/GE/EE/82D-58)

25. Ross, Douglas T. "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering. Vol. SE-3, No. 1, IEEE, January 1977.

26. Shooman, Martin L. Software Engineering Design/Reliability/Management. New York: McGraw-Hill Book Company, 1983.

27. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH," Computer Graphics. New York: August 1979.
28. Straayer, David H. "Adapting Applications to the Graphical Kernal System," Computer Design, 17: 167-172, July 1983.
29. Sutherland, I. E. "SKETCHPAD," Proceedings of AFIPS 1963 SJCC, 1963.
30. Teresko, John. "CAD/CAM Goes to Work," Industry Week, 40: 40-47, 7 February 1983.
31. Ullman, Jeffrey D. Principles of Database Systems. Rockville, Maryland: Computer Science Press, 1982.
32. Vlietstra, J. "The Reality of Computer Aided Design," CAD/CAM As a Basis for the Development of Technology in Developing Nations. 5-34. Amsterdam: North Holland Publishing Company, 1981.
33. Wenner, Patricia, et. al. Design Document for the George Washington Implementation of the 1979 GSPC Core System. Washington: Electrical Engineering/Computer Science Department, George Washington University, May 1981. (GWU-IIST-80-06)
34. Zelkowitz, Marvin V. "Perspectives on Software Engineering," Computing Surveys, 10: 198-216, June 1978.

Appendix A

Glossary

The glossary contains definitions of key words which can help the reader better understand the text of this thesis. Of necessity, terms which apply to data bases, graphics, software engineering, and the military were used. Since it would be unusual for readers to be well-versed in all these areas, the glossary is included to provide some clarification.

application program interface - one of two methods by which RIM-5 can be used. the other system is the standalone. The application program interface allows programs written in languages which can call FORTRAN 77 programs to use the RIM-5 capabilities. This is how RIM-5 data bases are accessed from an application program.

attribute - identifies a column of a relation

Core - the ACM SIGGRAPH Graphic Standards Planning Committee proposed graphics standard

data base - the schema along with associated occurrences of data for a particular application

data base management system - a collection of software which allows use of a data base and presents multiple views to users

design document - the GWCORE guidelines which define the GWU implementation of the 1979 Graphic Standards Planning Committee Core System

Graphical Kernal System - GKS - the graphics standard which has been accepted in draft by the International Standards Organization

graphics - the input, construction, storage, retrieval, manipulation, alteration, and analysis of objects and their pictorial representation

GWCORE - the George Washington University implementation of the Core

information packet - used in the GWCORE for communications between the device dependent and device independent interface as well as between subroutines. It contains the opcode, sequence of integer parameters, sequence of real parameters, and the actual parameter data

INGRED - Rose's Interactive GRaphics EDitor which allows creation, editing, and storage of two-dimensional figures

INGRED II - the name given to the software which implements the capabilities of the Rose graphics tools (INGRED) and the RIM-5 DBMS

load - put information into a relation in RIM-5

META - Rose's tool which reads picture files from storage and reproduces the picture on the selected display

owner - the one who originates the particular RIM-5 data base. If password protection is implemented, the password is normally the owner name.

Parray - the array within the GWCORE which contains integer*4 information. It contains the information packet.

password - a 1-8 alphanumeric name which is required for access to password protected relations, attributes, or tuples in RIM-5

Pseudo Display File - PDF - sequential array which contains information about primitives and/or attributes

query - a request from a user of RIM-5 for information in the data base

relation - table of data in which the columns represent attributes and the rows, or tuples, represent data occurrences

relational data base - made up of relations and can be recombined to make other relations to give flexibility in use of data. RIM-5 is a relational data base.

Relational Information Management System - RIM-5 - relational DBMS which can be used on the VAX with VMS operating system

schema - definition of relations and their attributes which make up a data base

standalone - method of using RIM-5 which allows creating, updating, and/or querying a data base

Appendix B

The Data Structure

The major data structure with which this data base software is concerned is the information packet [Wenner,20]. This packet is contained in Parray which is an integer*4 array that holds up to 30000 entries. Its composition is as follows:

<u>Position Number</u>	<u>Use</u>
1	count of total entries in Parray
2 to n	opcode information
.	
.	
yn+1 to (yn+1) + x	opcode information

where each block of opcode information includes the following [Wenner,20]:

1	length of entry
2	opcode
3	integer count
4	real count
5	integer parameters
.	
.	
.	
n	real parameters
.	
.	
.	

Depending on the complexity of the particular display, the information packet could contain only a few opcodes or hundreds of them.

A specific example for one opcode will help illustrate the information contained within the information packet. Since the 'SET PRIMITIVE ATTRIBUTES' opcode is used so often, it is the one illustrated;

Function 55: SET PRIMITIVE ATTRIBUTES [Wenner, 30,37]

Integer Parameters

Linestyle	-	1 for solid line, 2 for dashed line
Pen	-	0 is default, nonzero sets Linewidth, Linestyle, Intensity
Font	-	1 for normal characters, 2 for italics
Charpath	-	1 for right, 2 for left, 3 for up, 4 for down
Charjust	-	horizontal 0-off, 1-left, 2-right, 3-center vertical 0-off, 1-left, 2-right, 3-center
Pick ID	-	set pick id
Marker	-	1-5 for point, plus, asterisk, 0, x 6-10 are device dependent symbols
Line Index	-	index value
Fill Index	-	index value
Text Index	-	index value
Polygon Interior Style	-	1-plain, 2-shaded, 3-patterned
Polygon Edge Style	-	1-solid line, 2-interior

Real Parameters

Linewidth	-	NDC coordinates (0 is thinnest line)
Charspace	-	function of width or height depending on Charpath

Line Color - color parameters C1, C2, C3
 Line Intensity - intensity
 Fill Color - color parameters C1, C2, C3
 Fill Intensity - intensity
 Text Color - color parameters C1, C2, C3
 Text Intensity - intensity

Actual Opcode 55 values (taken from the printout of the
 information packet for one of the SADT displays used in
 Appendix C) follow:

<u>Posn #</u>	<u>Integer Value</u>	<u>Meaning</u>
1	31	Number of entries in this opcode
2	55	This is function 55
3	13	There are 13 integer values
4	14	There are 14 real values
5	1	integer 1 solid line (Line style)
6	0	integer 2 default value (Pen)
7	1	integer 3 normal char (Font)
8	1	integer 4 right (Char path)
9	0	integer 5 off (Horizontal charjust)
10	0	integer 6 off (Vertical charjust)
11	1	integer 7 default (Pick ID)
12	1	integer 8 point (Marker)
13	8	integer 9 not used (Line index)
14	8	integer 10 not used (Fill index)
15	8	integer 11 not used (Text index)
16	1	integer 12 plain (Poly interior)

17	1	integer 13	solid line	(Poly edge)
18	0	real 1	thinnest line	(Linewidth)
19	49152	real 2	fraction	(Char space)
20	16512	real 3		(Line color)
21	0	real 4		(Line color)
22	0	real 5		(Line color)
23	16384	real 6		(Line intensity)
24	16512	real 7		(Fill color)
25	0	real 8		(Fill color)
26	0	real 9		(Fill color)
27	16384	real 10		(Fill intensity)
28	16512	real 11		(Text color)
29	0	real 12		(Text color)
30	0	real 13		(Text color)
31	16384	real 14		(Text intensity)

The information packet is important because it contains the information which is passed between GWCORE subroutines as well as between the device dependent and device independent parts of the GWCORE [Rose,54]. This information is just Parray combined with the count of the number of entries in Parray. This is also the information in the pseudo display file, and contains all information needed to reproduce a display. Since the information packet contains all information needed to reproduce displays, it is the information stored on disk to provide the ability to restore graphic displays. This is the only GWCORE generated information stored in the INGRED II.

Appendix C

SADT Drawings

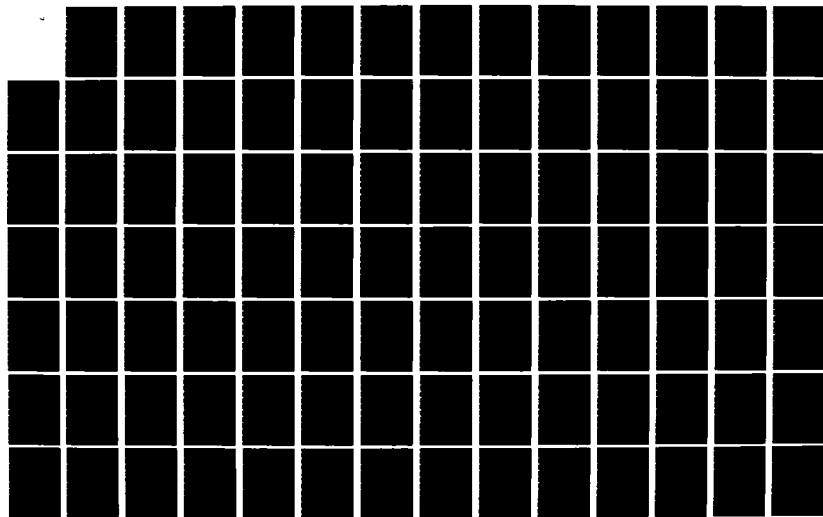
This appendix consists of two portions: the node index and the SADT drawings which resulted from the requirements definition and design of the INGRED II system. The node index lists the activity box names which were broken down further in the SADT process. The SADT diagrams themselves only show a high-level depiction of portions of the system which were already in existence before INGRED II. The portions which apply to this thesis effort are presented in more detail.

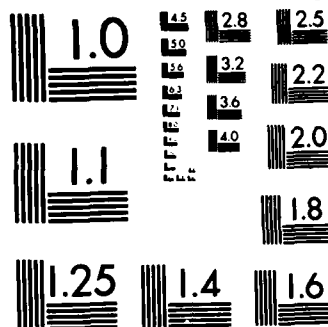
The SADT requirements diagrams were generated early in the thesis effort using the INGRED capabilities. They were then retrieved from disk and stored again using INGRED II, thus saving disk storage space. The diagrams which comprise Appendix C were then again restored from disk, this time using INGRED II, and reproduced using the Tektronix equipment in the DEL.

AD-A138 059

APPLICATIONS PROGRAMS TO FACILITATE USE OF A DBMS (DATA 2/3
BASE MANAGEMENT S. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J D GATEWOOD
DEC 83 AFIT/GCS/EE/83D-10 F/G 9/2 NL

UNCLASSIFIED





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

NODE INDEX

A-0	Core Graphics System
A0	Core Graphics System
A2	Perform Graphics Functions
A23	Change Graphics Figures
A3	Access Graphics Data Base
A32	Obtain Graphics Data
A33	Store Data
A34	Request Output Desired
A35	Transfer Figure
A4	Output Information
A43	Send to Printer

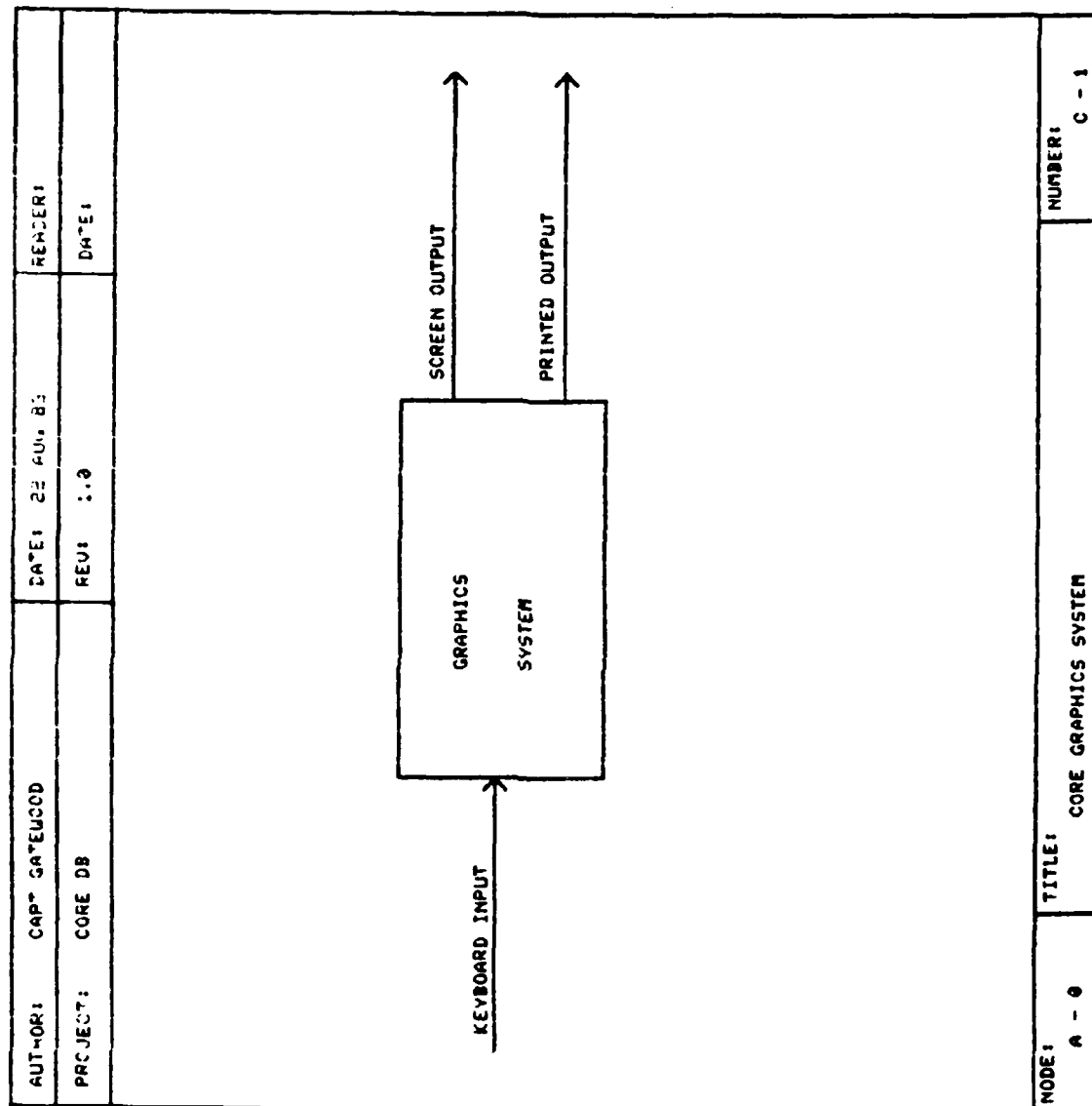


Figure C-1. SADT C-1: INGRID II Graphics System

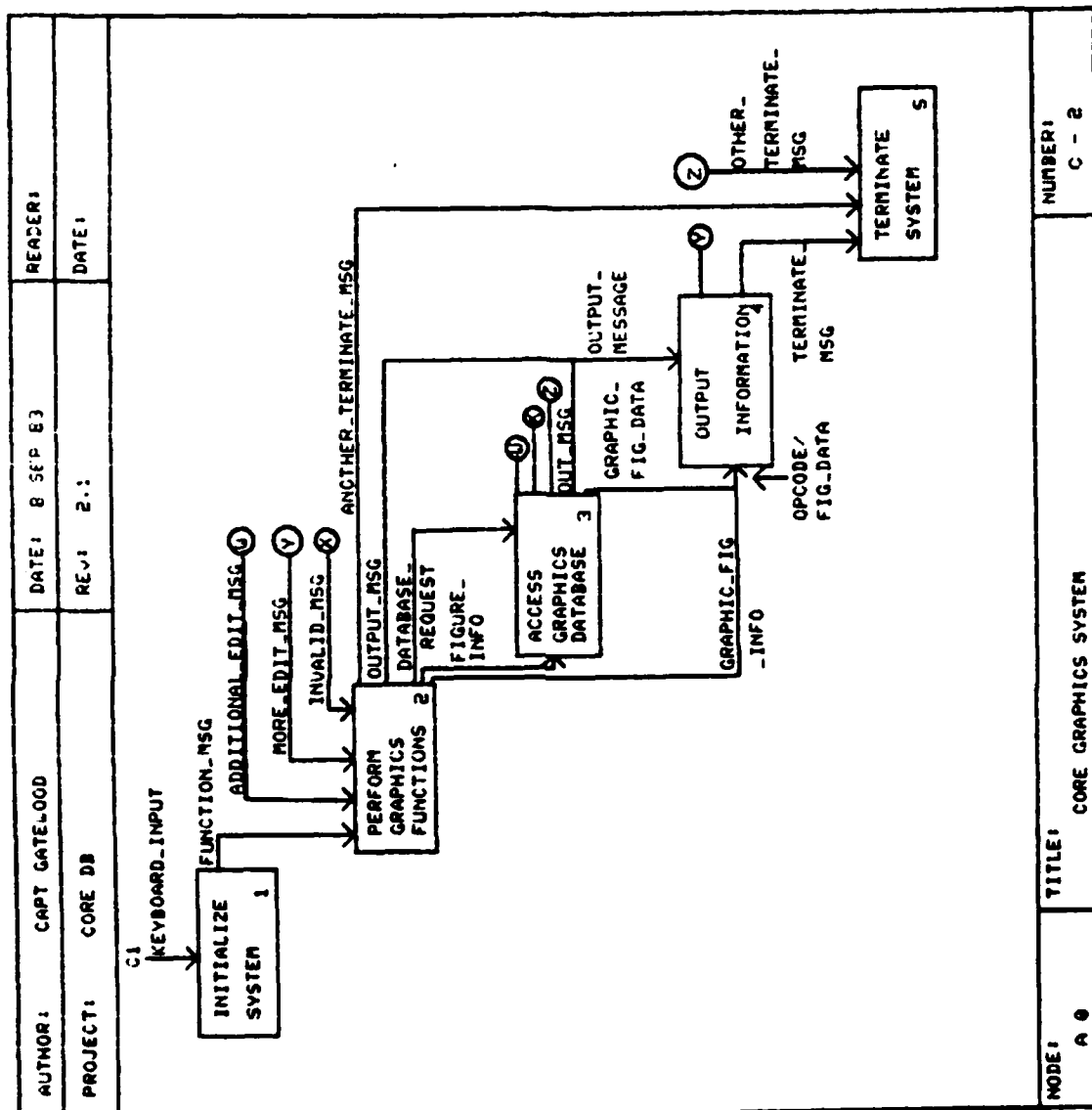


Figure C-2. SADT C-2: INGRED II Graphics System

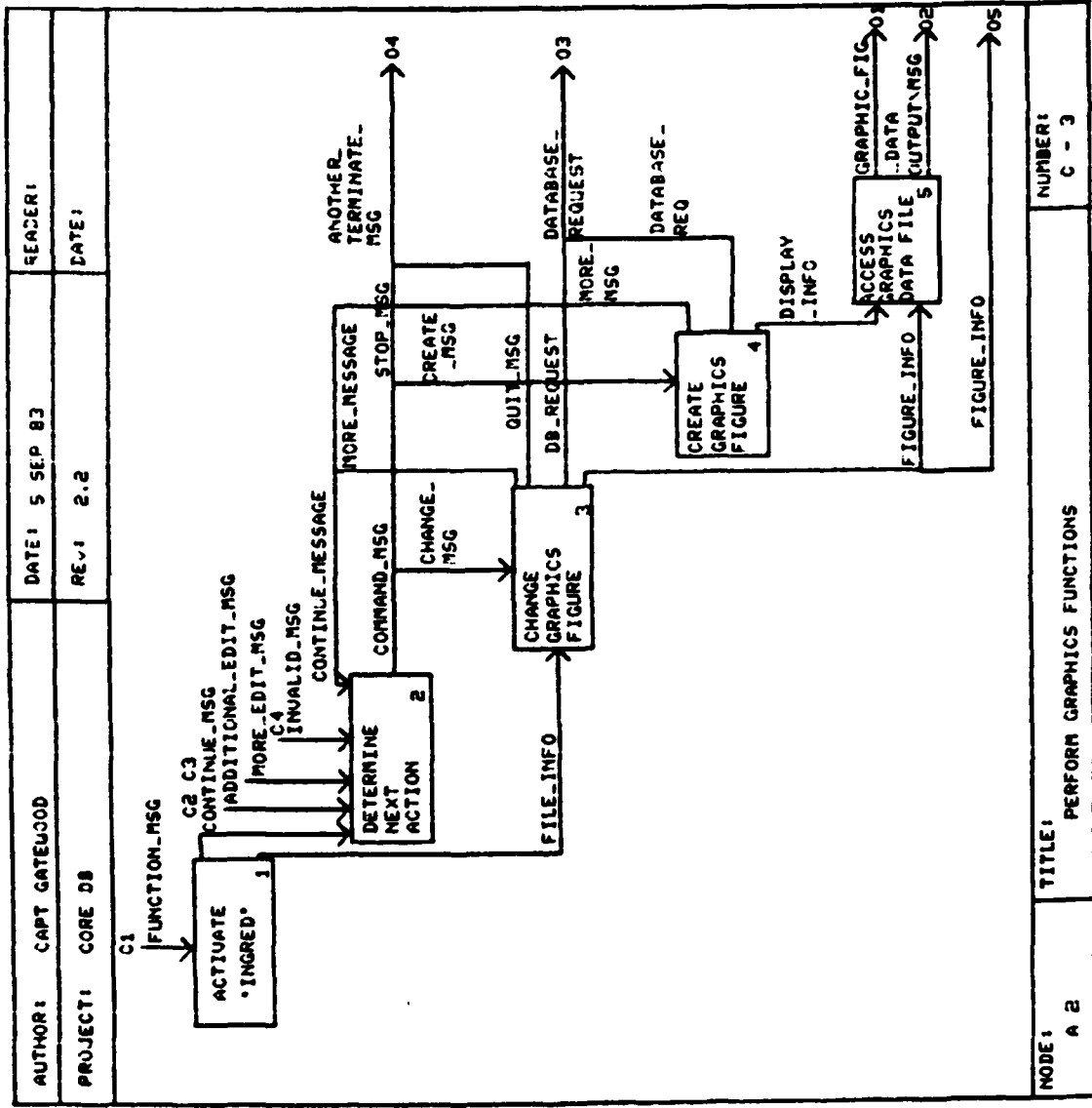


Figure C-3. SADT C-3: Perform Graphics Functions

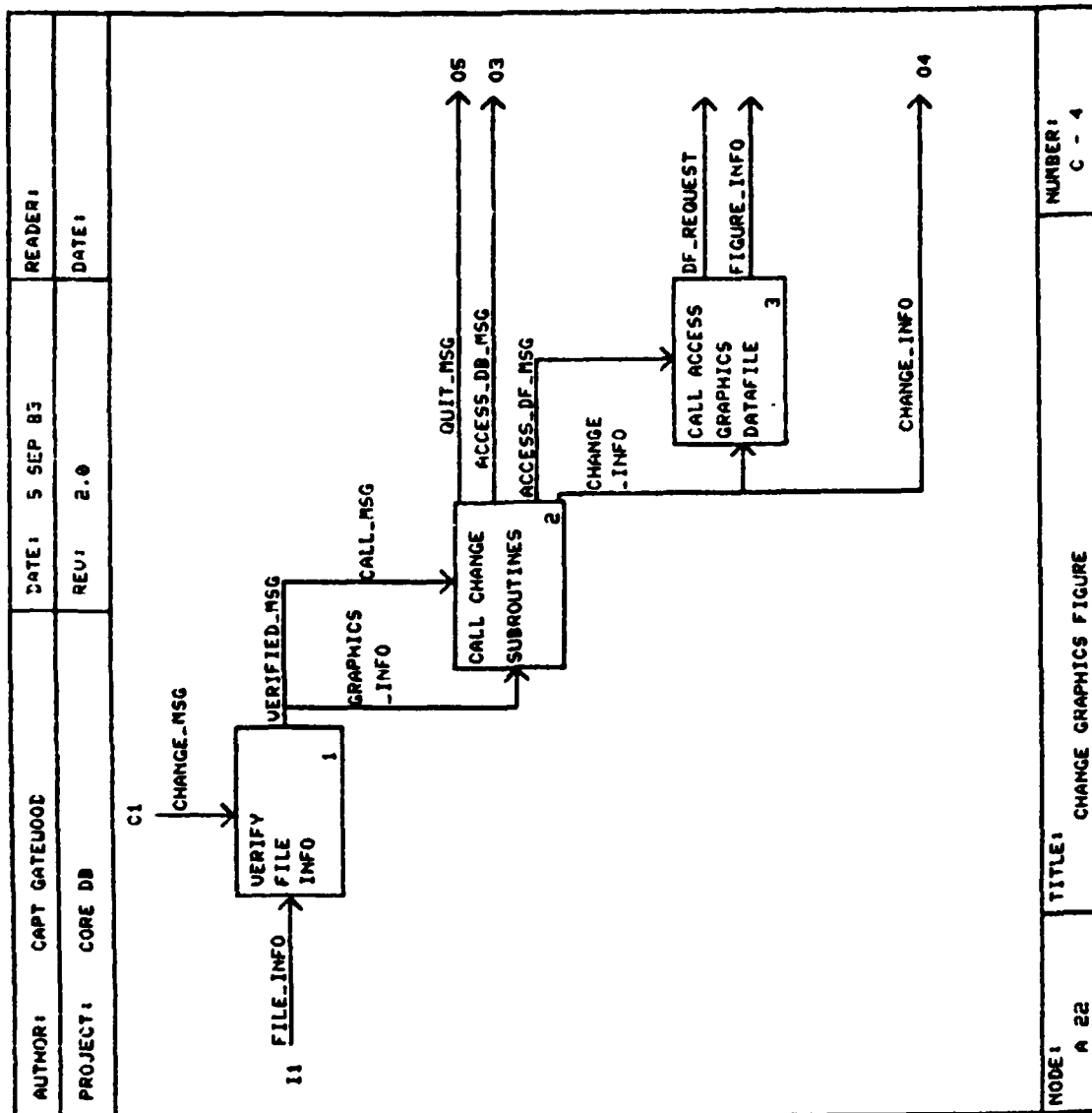


Figure C-4. SADT C-4: Change Graphics Figure

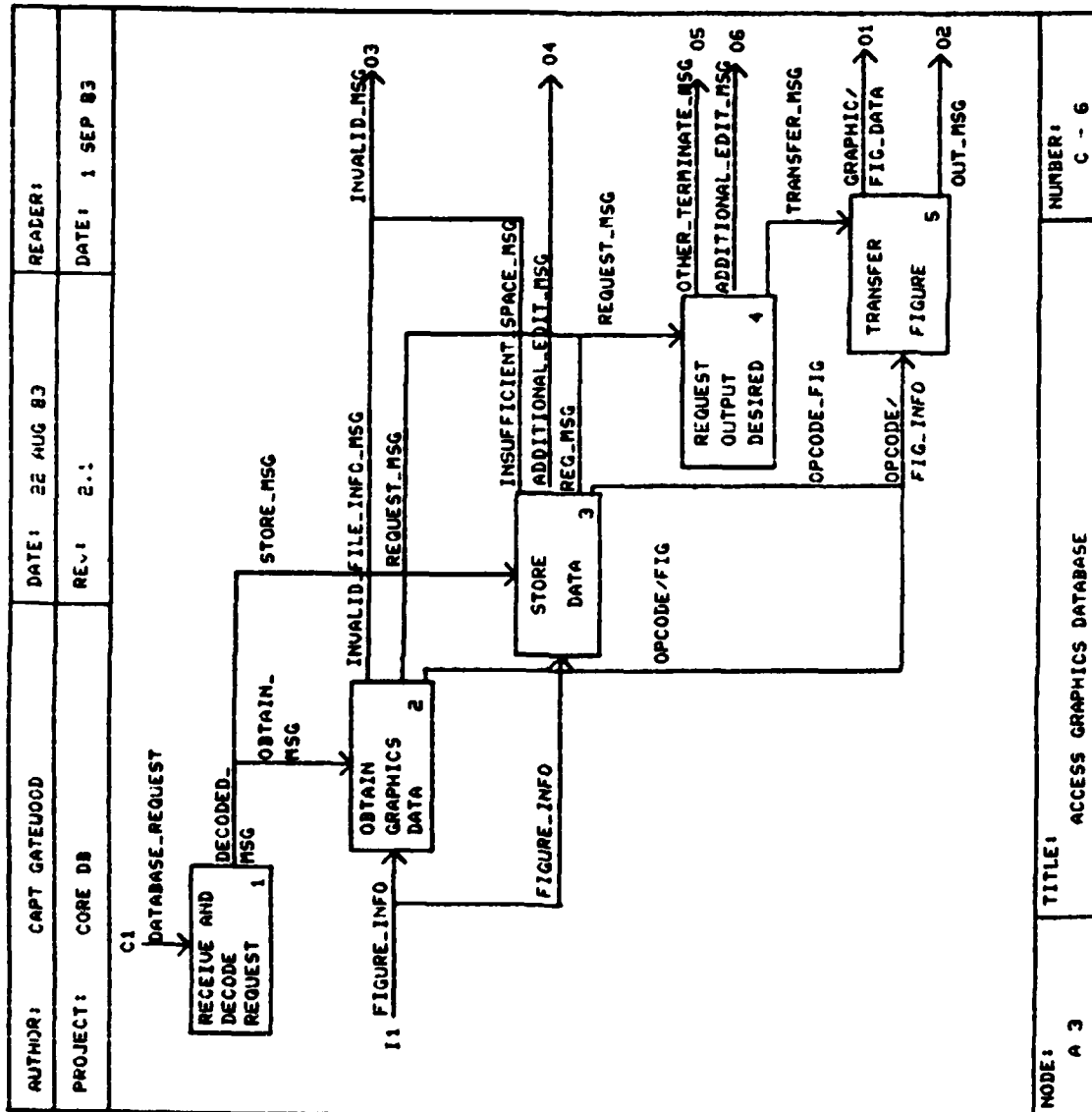


Figure C-5. SADT C-6: Access Graphics Data Base

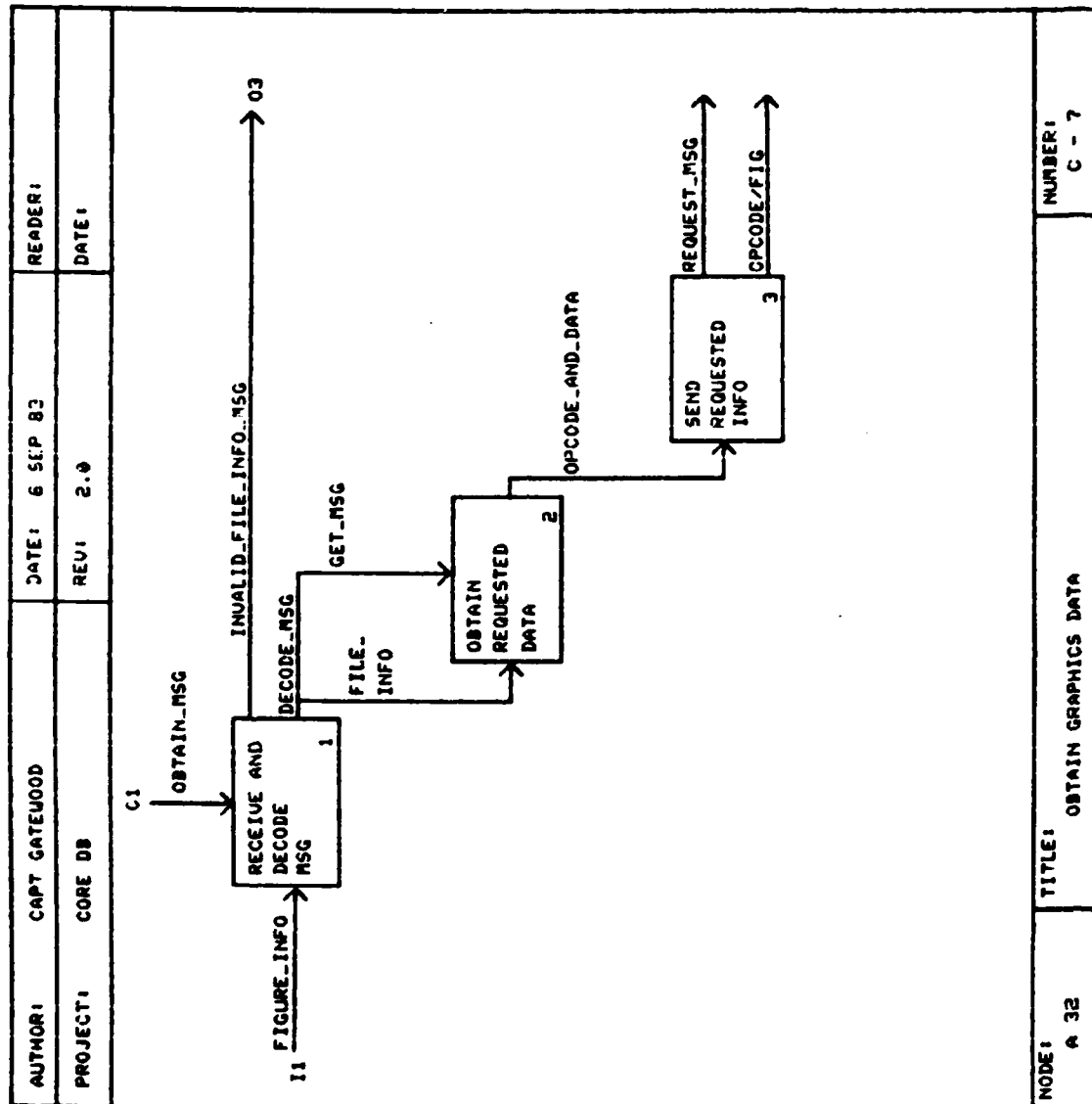


Figure C-6. SADT C-7: Obtain Graphics Data

6

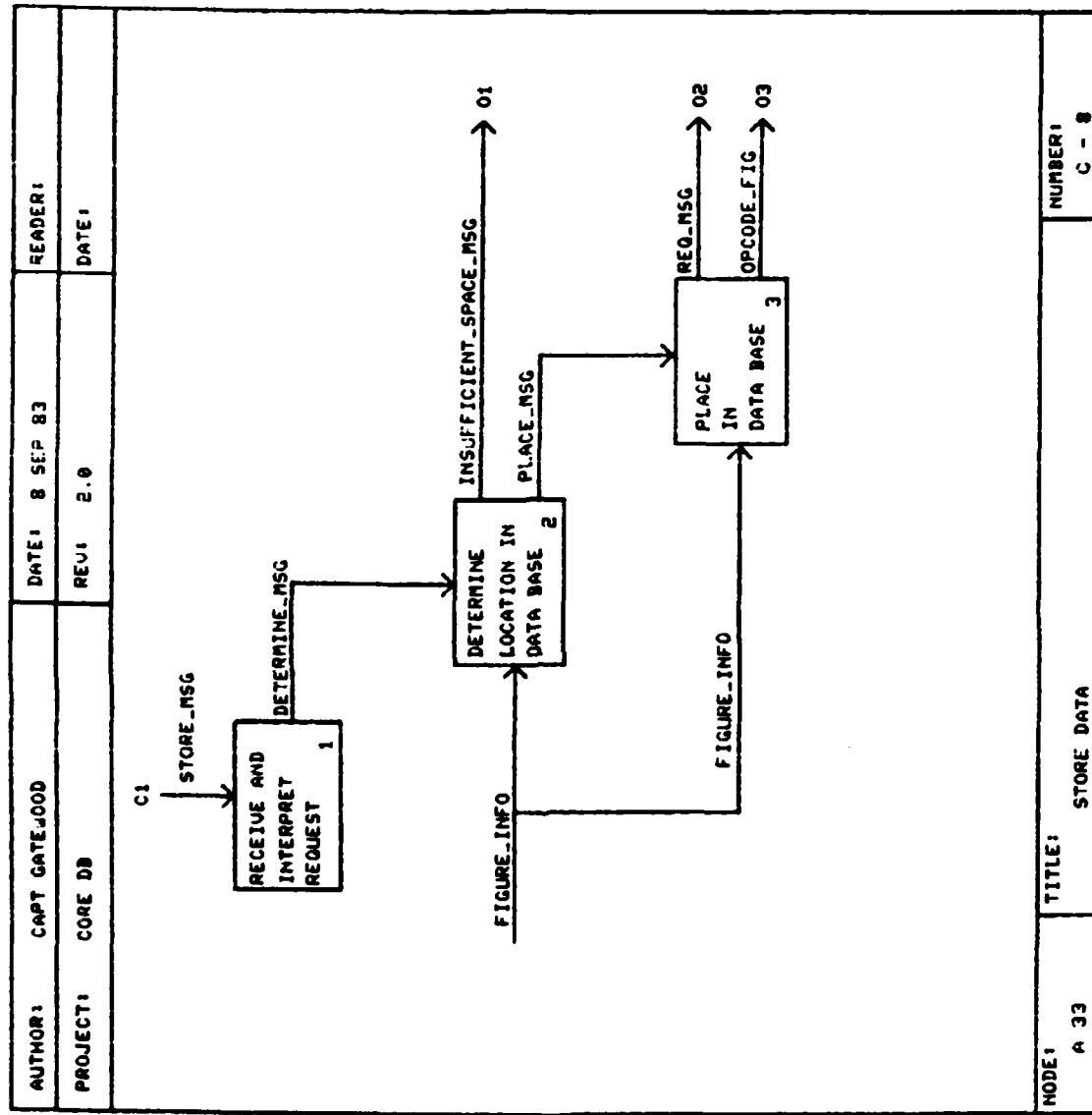


Figure C-7. SADT C-8: Store Data

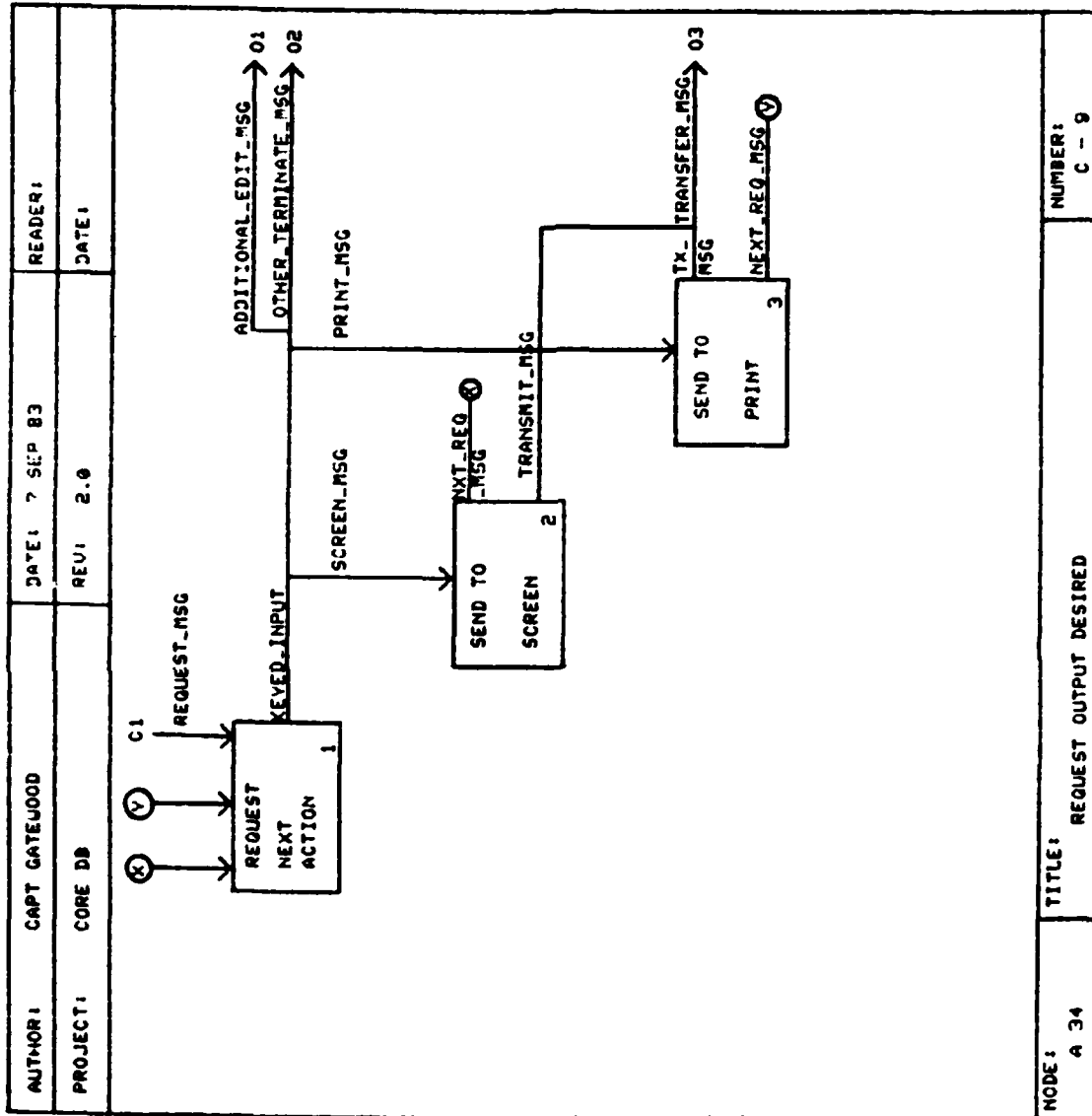


Figure C-8. SADT C-9: Request Output Desired

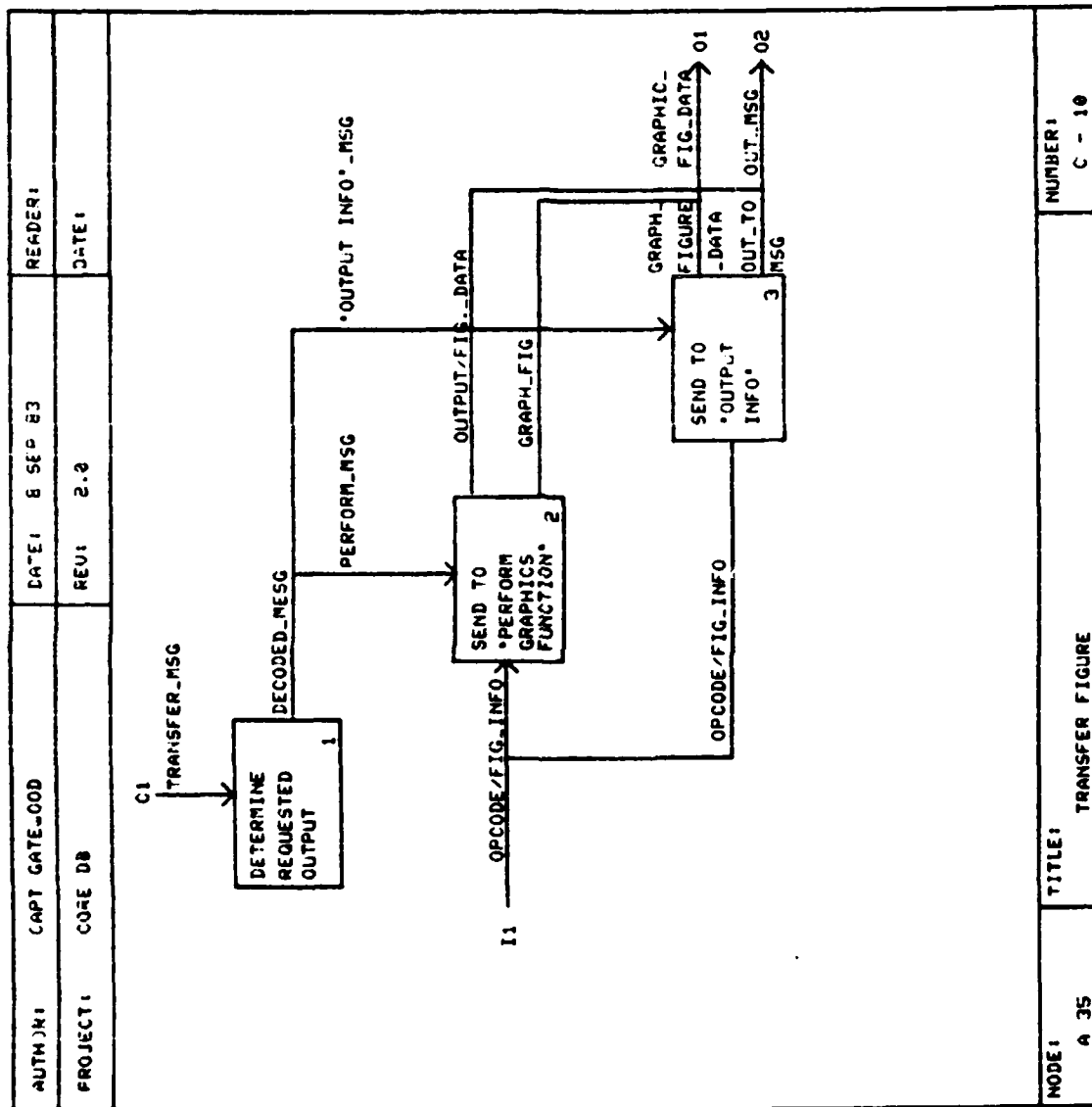


Figure C-9. SADT C-10: Transfer Figure

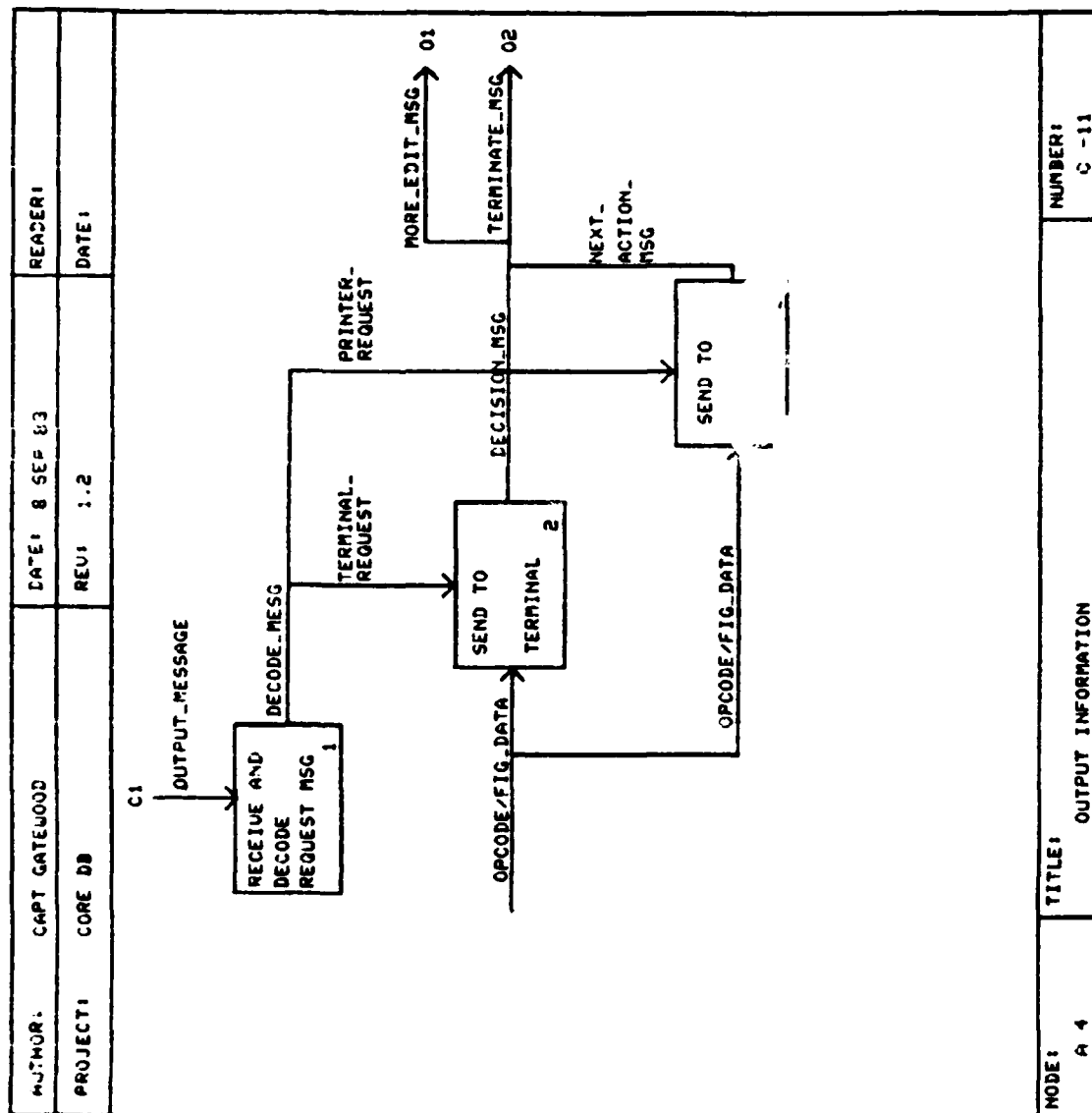


Figure C-10. SADT C-11: Output Information

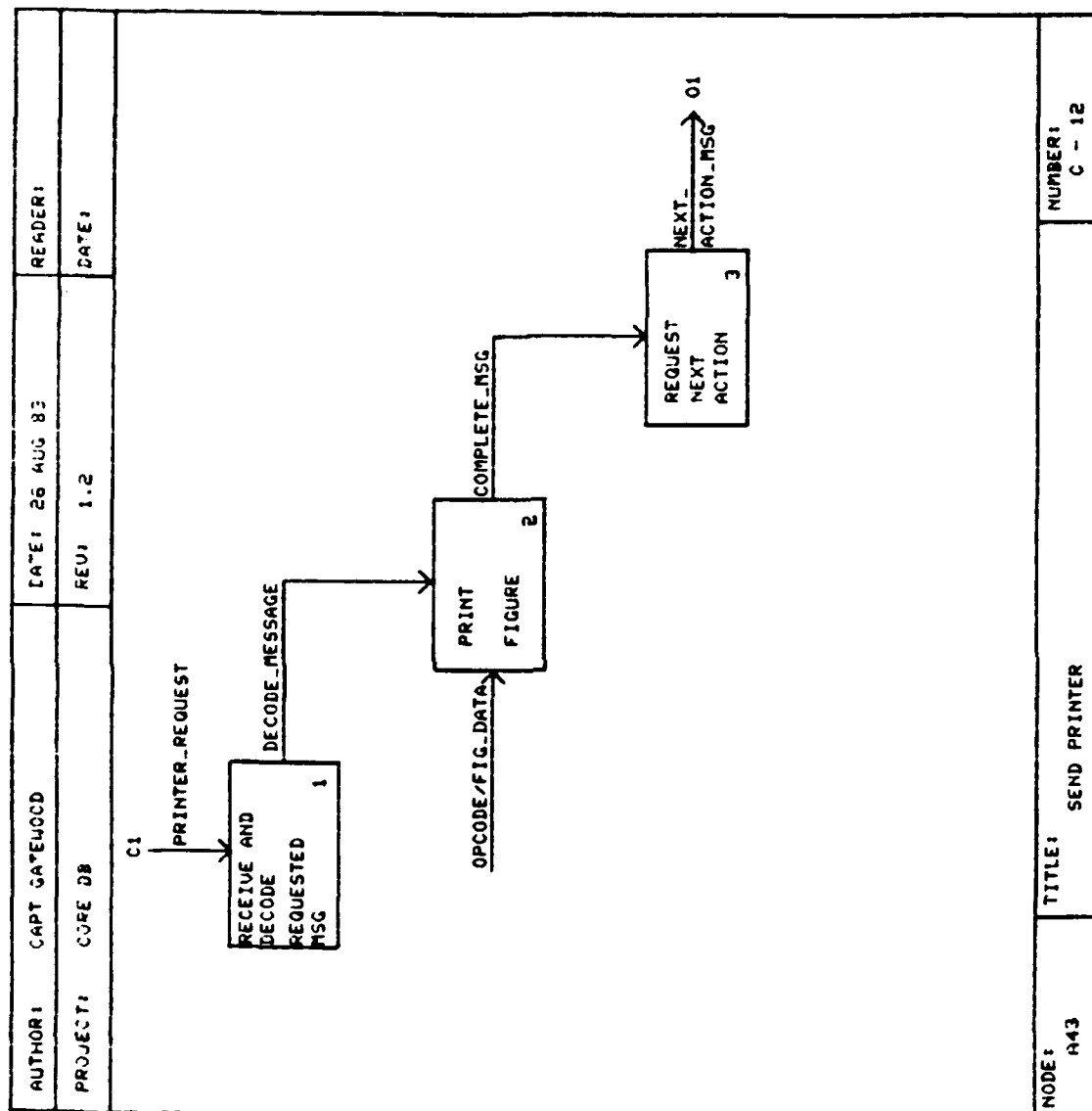


Figure C-11. SADT C-12: Send to Printer

Appendix D

Activity Data Dictionary

The activity data dictionary entries correspond to the activity boxes from the SADT diagrams of Appendix C. Each entry contains information about the particular activity which help clarify its function. This information includes the title of the activity box, its SADT number, what inputs, outputs, controls, and mechanisms are involved, a verbal description of the function of the box, any aliases used, and the date the activity was last updated.

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A24
NAME: Access Graphics Data Base
INPUTS: Figure_Info
OUTPUTS: Graphic_Fig_Data, Out_Msg
CONTROLS: Database_Request
MECHANISMS: None
DESCRIPTION: Called by A2 provides the functions by
which graphics displays are stored and
retrieved in the data base.
ALIASES: None

TYPE: ACTIVITY
DATE: 26 Aug 83
NUMBER: A25
NAME: Access Graphics Data File
INPUTS: Figure_Info
OUTPUTS: Graphic_Fig_Data, Output_Msg
CONTROLS: None
MECHANISMS: None
DESCRIPTION: This is where generated graphics displays are
stored if the old method of file storage is
chosen instead of storing in the data base.
ALIASES: None

TYPE: ACTIVITY
DATE: 23 Jul 83
NUMBER: A21
NAME: Activate "INGRED"
INPUTS: None
OUTPUTS: Continue_Msg, File_Info
CONTROLS: Function_Msg
MECHANISMS: None
DESCRIPTION: Calls either A22 or A23, whichever is appro-
priate, to do graphics processing.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A223
NAME: Call "Access Graphics Data base"
INPUTS: Change_Info
OUTPUTS: DB-Request, Figure Info
CONTROLS: Access_DB_Msg
MECHANISMS: None
DESCRIPTION: Calls A24
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A222
NAME: Call Change Subroutines
INPUTS: Graphics_Info
OUTPUTS: Access_DB_Msg, Change_Info
CONTROLS: Verified_Msg
MECHANISMS: None
DESCRIPTION: This module just calls the graphics tools
provided by Rose in his GWCORE graphics
package.
ALIASES: None

TYPE: ACTIVITY
DATE: 20 Aug 83
NUMBER: A23
NAME: Change Graphics Figure
INPUTS: File Info
OUTPUTS: DB_Request, Figure_Info, Quit_Msg, More_Message
CONTROLS: Change_Msg
MECHANISMS: None
DESCRIPTION: Changes to existing files are made here.
Calls A25 and passes information on what
graphics display is to be accessed from the
data file. Also calls A25 after changes have
been made, and sends new figure info for
storage into the data file.
ALIASES: None

TYPE: ACTIVITY
DATE: 20 Aug 83
NUMBER: A24
NAME: Create Graphics Figure
INPUTS: None
OUTPUTS: Database_Req, Display_Info, More_Msg
CONTROLS: Create_Msg
MECHANISMS: None
DESCRIPTION: Called by A22, and is used to create a new graphics display. It calls A25 to place new display in data file after creation.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A332
NAME: Determine Location in Data base
INPUTS: Figure_Info
OUTPUTS: Insufficient_Space_Msg, Place_Msg
CONTROLS: Decoded_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Store Data' which determines where in the data base the display information will be placed; sends out either an error message (i.e. no room in data base) or location message to A33.
ALIASES: None

TYPE: ACTIVITY
DATE: 26 Aug 83
NUMBER: A22
NAME: Determine Next Action
INPUTS: None
OUTPUTS: Command_Msg
CONTROLS: Continue_Msg, Additional_Edit_Msg, More_Edit_Msg, Invalid_Msg, Continue_Msg
MECHANISMS: None
DESCRIPTION: This is where it is determined which action within INGRED will be taken next.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A351
NAME: Determine Requested Output
INPUTS: None
OUTPUTS: Decoded_Mesg
CONTROLS: Transfer_Msg
MECHANISMS: None
DESCRIPTION: Takes keyboard input from A233 and the figure
data from A34 and sends to either A2 or A4.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Jul 83
NUMBER: A1
NAME: Initialize System
INPUT: None
OUTPUTS: Function Message
CONTROLS: Keyboard_Input
MECHANISMS: None
DESCRIPTION: Sets up the GWCORE for processing.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A32
NAME: Obtain Graphics Data
INPUTS: Figure_Info
OUTPUTS: Invalid_File_Info_Msg, Request_Msg, Opcode/Fig
CONTROLS: Obtain_Msg
MECHANISMS: None
DESCRIPTION: Determine what display is requested; gets that
display from storage and calls A33.
ALIASES: None

TYPE: ACTIVITY
DATE: 20 Aug 83
NUMBER: A24
NAME: Access Graphics Data Base
INPUTS: Figure_Info, Display_Info
OUTPUTS: Graphic_Fig_Data, Output Msg
CONTROLS: Database_Req
MECHANISMS: None
DESCRIPTION: Gets the stored graphics information from the
data base, or stores created display info into
the data base.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A242
NAME: Obtain Graphics Data
INPUTS: Figure_Info
OUTPUTS: Invalid_File_Info_Msg, Request_Msg, Opcode/Fig
CONTROLS: Obtain_Msg
MECHANISMS: None
DESCRIPTION: Gets display from data base for review or
change of the information.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A322
NAME: Obtain Requested Data
INPUTS: File_Data
OUTPUTS: Opcode_and_Data
CONTROLS: Get_Msg
MECHANISMS: None
DESCRIPTION: Finds data on disk as requested by user and
sends the opcode and data to A323
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A4
NAME: Output Information
INPUTS: Opcodes/Fig_Data
OUTPUTS: Terminate_Msg, More_Edit_Msg
CONTROLS: Output_Message
MECHANISMS: None
DESCRIPTION: Includes output to screen or printer.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A2
NAME: Perform Graphics Functions
INPUTS: None
OUTPUTS: Output_Mesg, Database_Request, Figure_Info
CONTROLS: Function_Msg, Additional_Edit_Msg, More_Edit_Msg,
Invalid_Msg
MECHANISMS: None
DESCRIPTION: Contains the main software for the core to do its
work.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A333
NAME: Place in Data base
INPUTS: Figure_Info
OUTPUTS: Req_Msg
CONTROLS: Place_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Store Data' which uses location
message from A332 and places Figure_Info data
from A2 in the data base; it then sends out a
completion message to A34.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A432
NAME: Print Figure
INPUTS: Opcode/Fig_data
OUTPUTS: Ccomplete_Msg
CONTROLS: Decoded_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Send to Printer' called by A431 which
uses Opcode/Fig_data from A2 or A3 to print
out display on the print device.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A321
NAME: Receive and Decode Msg
INPUTS: Figure_Info
OUTPUTS: Decode_Msg, Invalid_File_Info_Msg
CONTROLS: Obtain_Msg
MECHANISMS: None
DESCRIPTION: Reads message, checks for proper syntax,
determines what graphics information is
requested and transfers file information to
A322.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A31
NAME: Receive and Decode Request
INPUTS: None
OUTPUTS: Decoded_Msg
CONTROLS: Database_Request
MECHANISMS: None
DESCRIPTION: Takes command message from user, validates message
for syntax and sends decoded message to A32 or
A33 as requested.
ALIASES: None

TYPE: ACTIVITY
DATE: 23 Aug 83
NUMBER: A41
NAME: Receive and Decode Request Msg
INPUTS: None
OUTPUTS: Decode_Msg
CONTROLS: Output_Message
MECHANISMS: None
DESCRIPTION: Takes Output_Msg from A2 and verifies proper
syntax and determines proper routing.
ALIASES: None

TYPE: ACTIVITY
DATE: 26 Jul 83
NUMBER: A331
NAME: Receive and Decode Requested Msg
INPUTS: None
OUTPUTS: Determine_Msg
CONTROLS: Store_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Send to Printer' called by A31, or
A32 which checks the message and calls A332.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A331
NAME: Receive and Interpret Request
INPUTS: None
OUTPUTS: Decoded_Msg
CONTROLS: Store_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Store Data' which checks message
from 'Receive and Decode Request' to make sure
it is valid, then calls 'Determine Location to
Place in Data base'.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A341
NAME: Request Next Action
INPUTS: None
OUTPUTS: Keyed_Input
CONTROLS: Request_Msg, Next_Req_Msg, Nxt_Req_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Request Output Desired' which is called by 'Place in Data base' and puts out options to the screen so the user can select the next thing to do. It then calls A342, A343, A2, or A5.
ALIASES: None

TYPE: ACTIVITY
DATE: 26 Jul 83
NUMBER: A333
NAME: Request Next Action
INPUTS: Figure_Info
OUTPUTS: Req_Msg, Opcode_Fig
CONTROLS: Place_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Send to Printer' called by 'Print Figure'; it displays options on the screen so the user can stipulate what action is desired next.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A34
NAME: Request Output Desired
INPUTS: None
OUTPUTS: Transfer_Msg
CONTROLS: Request_Message
MECHANISMS: None
DESCRIPTION: Displays menu which solicits from user what action to take next with the display obtained from storage or from newly created display.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A323
NAME: Send Requested Info
INPUTS: Opcode_and_Data
OUTPUTS: Opcode/Fig, Request_Msg
CONTROLS: None
MECHANISMS: None
DESCRIPTION: Sends opcode and figure data to appropriate
module.
ALIASES: None

TYPE: ACTIVITY
DATE: 23 Aug 83
NUMBER: A32
NAME: Send to Data base
INPUTS: Opcode/Fig_Data
OUTPUTS: Decision_Msg, Opcode/Fig_Dat
CONTROLS: Database_Request
MECHANISMS: None
DESCRIPTION: Called by A31 and uses Opcode/Fig_Data from A2
in data base retrieval.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A353
NAME: Send to "Output Info"
INPUTS: Opcode/Fig_Info
OUTPUTS: Graph_Figure_Data
CONTROLS: "Output Info"_Msg
MECHANISMS: None
DESCRIPTION: Takes Opcode/Fig_Info from A32, and
"Output Info"_Msg from A351 and sends to
appropriate procedure for further processing.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A352
NAME: Send to "Perform Graphics Function"
INPUTS: Opcode/Fig_Info
OUTPUTS: Output/Fig_Data, Graph_Fig
CONTROLS: Perform_Msg
MECHANISMS: None
DESCRIPTION: Takes Opcode/Fig_Info from A32, and decoded
message on which function to perform from A35,
and sends that information to A2 for further
processing.
ALIASES: None

TYPE: ACTIVITY
DATE: 26 Aug 83
NUMBER: A343
NAME: Send to Print
INPUTS: None
OUTPUTS: Next_Req_Msg, Print_Msg
CONTROLS: Print_Msg
MECHANISMS: None
DESCRIPTION: Part of 'Request Output Desired' which is
called by A341 and uses Opcode/Fig_Info from
A33 to print on selected printing device.
ALIASES: None

TYPE: ACTIVITY
DATE: 23 Aug 83
NUMBER: A33
NAME: Send to Printer
INPUTS: Opcode/Fig_Data
OUTPUTS: Next_Action_Msg
CONTROLS: Printer_Request
MECHANISMS: None
DESCRIPTION: Called by A41 and uses Opcode/Fig_Dat in
printing out display as requested.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A342
NAME: Send to Screen
INPUTS: None
OUTPUTS: Nxt_Req_Msg, Screen_Msg
CONTROLS: Screen_Msg
MECHANISMS: None
DESCRIPTION: Part of A34 which is called by A341 and calls
A25.
ALIASES: None

TYPE: ACTIVITY
DATE: 26 Aug 83
NUMBER: A42
NAME: Send to Terminal
INPUTS: Opcode/Fig_Data
OUTPUTS: Decision_Msg, Opcode/Fig_Dat
CONTROLS: Screen_Request
MECHANISMS: None
DESCRIPTION: Sends information for display to the terminal
screen. Also determines which action the user
wants to take next.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A243
NAME: Store Changed Data
INPUTS: Display_Info
OUTPUTS: Invalid_File_Msg, Req_Msg, Opcode_Fig
CONTROLS: Store_Msg
MECHANISMS: None
DESCRIPTION: Called by A241 when request is to store data
from either A22 or A23.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A35
NAME: Transfer Figure
INPUTS: Opcode/Fig_Info
OUTPUTS: Graphic_Fig_Data, Output_Msg
CONTROLS: Transfer_Msg
MECHANISMS: None
DESCRIPTION: Moves figure from A32 or A34 to screen or
printer depending on user selection.
ALIASES: None

TYPE: ACTIVITY
DATE: 25 Aug 83
NUMBER: A5
NAME: Terminate System
INPUTS: None
OUTPUTS: None
CONTROLS: Terminate_Msg, Other_Terminate_Msg, Another_Ter-
minate_Msg
MECHANISMS: None
DESCRIPTION: Shuts down processing gracefully and closes
out all files.
ALIASES: None

TYPE: ACTIVITY
DATE: 21 Aug 83
NUMBER: A221
NAME: Verify File Info
INPUTS: File_Info
OUTPUTS: Verified_Msg
CONTROLS: Change_Msg
MECHANISMS: None
DESCRIPTION: Verifies validity of information passed from
A21.
ALIASES: None

Appendix E

Data Element Data Dictionary

The data element data dictionary is composed of entries for each data element contained in the SADT diagrams of Appendix C. This data dictionary is designed to clarify the function of each data element and show how each element contributes to the overall design. Each entry is composed of the name of the data element, the date it was last modified, a verbal description of the function of the element, where the data element was passed from, and where it is destined, what aliases apply, what the data characteristics are, and whether the data element is composed of other data elements or is part of another data element.

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Access_DB_Msg
DESCRIPTION: Contains information on whether to access the
database to store or obtain information.
Passed from A222 to A223.
SOURCES: A222
DESTINATIONS: A223
COMPOSITION: N/A
PART OF: DB_Request, Database_Request
DATA CHARACTERISTICS: Character String
ALIASES: DB_Request, Database_Request

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Additional_Edit_Msg
DESCRIPTION: Output from A341 which is control to A2 and
provides a method to get out of 'Access
Graphics Database' to do more graphics
generation.
SOURCES: A341, A34, A3
DESTINATIONS: A2, A22
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: More_Edit_Msg, Function_Msg

TYPE: DATA ELEMENT
DATE: 25 Aug 83
NAME: Another_Terminate_Msg
DESCRIPTION: Output from A2 which allows user to end ses-
sion from INGRED.
SOURCES: A2
DESTINATIONS: A5
COMPOSITION: Command_Msg, Quit_Msg
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Terminate_Msg, Other_Terminate_Msg, Quit_Msg,
Command_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Call_Msg
DESCRIPTION: Passed from A221 as control to A22.
SOURCES: A221
DESTINATIONS: A22
COMPOSITION: N/A
PART OF: Verified_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Verified_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Change_Info
DESCRIPTION: Carries information from A222 which is input
to A223 and contains graphics display information after changes have been completed.
SOURCES: A222
DESTINATIONS: A223
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Figure_Info

TYPE: DATA ELEMENT
DATE: 20 Aug 83
NAME: Change_Msg
DESCRIPTION: Request from A22 which is control to A23. It
reflects the decision to change an existing
graphics display.
SOURCES: A22
DESTINATIONS: A23
COMPOSITION: None
PART OF: Command_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Command_Msg,

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Command_Msg
DESCRIPTION: Output from A22 which acts as control to A23
or A24 to choose creating a new display or
changing an existing one, or to A5 to request
termination.
SOURCES: A22
DESTINATIONS: A23, A24, A5
COMPOSITION: Change_Msg, Create_Msg, Stop_Msg
PART OF: Another_Terminate_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Change_Msg, Anther_Terminate_Msg, Create_Msg,
Stop_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Complete_Msg
DESCRIPTION: Output from A432 which acts as control to
A433.
SOURCES: A432
DESTINATIONS: A433
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: None

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Continue_Message
DESCRIPTION: Composed of More_Msg from A24 and More_Message
from A23 and acts as control to A22.
SOURCES: A23, A24
DESTINATIONS: A22
COMPOSITION: More_Msg, More_Message
PART OF: N/A
DATA CHARACTERISTICS: Character String
ALIASES: More_Message, More_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Continue_Msg
DESCRIPTION: Output from A21 which acts as control to A22.
SOURCES: A21
DESTINATIONS: A22
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: None

TYPE: DATA ELEMENT
DATE: 20 Aug 83
NAME: Create_Msg
DESCRIPTION: Request from A22 which is control to A24. It
reflects the decision to create an new
graphics figure.
SOURCES: A22
DESTINATIONS: A24
COMPOSITION: N/A
PART OF: Command_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Command_Msg

TYPE: DATA ELEMENT
DATE: 15 Aug 83
NAME: Command_Msg
DESCRIPTION: Output from A21 which reflects the choice of
A22 or A23 as desired which thus allow for
either editing an existing figure or creating
a new figure.
SOURCES: A21
DESTINATIONS: A22, A23
COMPOSITION: Change_Msg, Create_Msg
PART OF: None
DATA CHARACTERISTICS: Character String
ALIASES: Change_Msg, Create_Msg

TYPE: DATA ELEMENT
DATE: 16 Aug 83
NAME: Database_Msg
DESCRIPTION: Portion of decoded_message from A231 which is
 control to A232.
SOURCES: A231
DESTINATIONS: A232
COMPOSITION: N/A
PART OF: Decoded Msg
DATA CHARACTERISTICS: Character String
ALIASES: Decoded_Msg

TYPE: DATA ELEMENT
DATE: 20 Aug 83
NAME: Database_Req
DESCRIPTION: Output from A24 which becomes Data-
 base_Request and acts as control to A3.
 Reflects the request to place a newly created
 display to the data base.
SOURCES: A24
DESTINATIONS: A3
COMPOSITION: None
PART OF: Database_Request
DATA CHARACTERISTICS: Flag
ALIASES: Database_Request

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Database_Request
DESCRIPTION: Calls A3 from A2 to store a newly created
 graphics display to, or obtain an existing
 graphics display from the data base.
SOURCES: A2
DESTINATIONS: A3, A31
COMPOSITION: DB_Request, Database_Req
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: DB_Request, Database_Req

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: DB_Request
DESCRIPTION: Flag from A22 that reflects whether A24 is
being called to obtain stored graphics display
information for change, or to store newly
changed information to the data base.
SOURCES: A23
DESTINATIONS: A3
COMPOSITION: None
PART OF: Database_Request
DATA CHARACTERISTICS: Flag
ALIASES: Database_Request

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Decision_Msg
DESCRIPTION: Output from A42 which allows decision by user
on whether to continue in INGRED or terminate
session.
SOURCES: A42
DESTINATIONS: A2, A5
COMPOSITION: More_Edit_Msg, Terminate_Msg
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Next_Action_Msg, Terminate_Msg, More_Edit_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Decode_Msg
DESCRIPTION: Output from A321 which is composed of Get_Msg
(control for A322) and File_Info.
SOURCES: A321
DESTINATIONS: A322
COMPOSITION: Get_Msg, File_Info
PART OF: N/A
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Get_Msg, File_Data

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Decoded_Mesg
DESCRIPTION: Output from A351 which either Perform_Msg
(control to A352) or "Output Info"_Msg (control to A353), depending on which transfer is desired.
SOURCES: A351
DESTINATIONS: A352, A353
COMPOSITION: Perform_Msg, "Output Info"_Msg
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Perform_Msg, "Output Info"_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Decoded_Mesg
DESCRIPTION: Output from A31 which divides into Store_Msg
which is control for A33 or Obtain_Msg which
is control for A32.
SOURCES: A31
DESTINATIONS: A32, A33
COMPOSITION: Obtain_Msg, Store_Msg
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Obtain_Msg, Store_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Decode_Mesg
DESCRIPTION: Output from A41 which becomes either Terminal_Request (control to A42) or Printer_Request (control to A43). Carries information on whether to send information to the terminal or printer.
SOURCES: A41
DESTINATIONS: A42, A43
COMPOSITION: Printer_Request, Terminal_Request
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Printer_Request, Terminal_Request

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Decode Message
DESCRIPTION: Output from A431 which is control to A432 and
results in printing of display figure on
printer.
SOURCES: A431
DESTINATIONS: A432
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: N/A

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Determine Msg
DESCRIPTION: Flag from A331 which is control to A332.
SOURCES: A331
DESTINATIONS: A332
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: None

TYPE: DATA ELEMENT
DATE: 20 Aug 83
NAME: Display Info
DESCRIPTION: Graphics display information which reflects
the content of the newly created figure. It
is passed from A24 to A25.
SOURCES: A24
DESTINATIONS: A5
COMPOSITION: None
PART OF: N/A
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: File_Info, Figure_Info

TYPE: DATA ELEMENT
DATE: 15 Aug 83
NAME: Fig_Data
DESCRIPTION: Figure information from A23 which will either
be used to create a graphics figure or output
the information desired.
SOURCES: A23
DESTINATIONS: A24, A3
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Character String
ALIASES: None

TYPE: DATA ELEMENT
DATE: 16 Aug 83
NAME: Fig_Output_Msg
DESCRIPTION: Request from A23 for type of output desired.
Merges with Graph_Output_Msg from A24 to
become Output_Msg which is control for A3.
SOURCES: A23
DESTINATIONS: A3
COMPOSITION: None
PART OF: Output_Msg
DATA CHARACTERISTICS: Character String
ALIASES: Output_Msg

TYPE: DATA ELEMENT
DATE: 20 Aug 83
NAME: Figure_Info
DESCRIPTION: Graphics display information passed from A2
to A3 which contains information on the
stored display which is desired, or the infor-
mation to be stored after the change process
is completed.
SOURCES: A2, A23
DESTINATIONS: A3, A32, A33, A332, A333
COMPOSITION: None
PART OF: N/A
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: None

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: File_Data
DESCRIPTION: Portion of Decode_Msg from A2421 which carries
information on what graphics data is desired
from the database.
SOURCES: A2421
DESTINATIONS: A2422
COMPOSITION: N/A
PART OF: Decode_Msg
DATA CHARACTERISTICS: Character String
ALIASES: Decode_Msg

TYPE: DATA ELEMENT
DATE: 17 Aug 83
NAME: File_Info
DESCRIPTION: Output from A321 which is input to A322. Con-
tains information on the display requested
from storage.
SOURCES: A321
DESTINATIONS: A322
COMPOSITION: N/A
PART OF: Decode_Msg
DATA CHARACTERISTICS: Array of Integer/Real
ALIASES: Figure_Info, Display_Info

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Figure_Info
DESCRIPTION: Output from A23 which passes graphics informa-
tion to A25 or A3, depending on whether desire
is to store the information in the data file
or the data base.
SOURCES: A23
DESTINATIONS: A3, A32, A25
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Display_Info, File_Info

TYPE: DATA ELEMENT
DATE: 16 Aug 83
NAME: Function_Msg
DESCRIPTION: From A1 carries information to A2 within which
it acts as control for A21.
SOURCES: A1
DESTINATIONS: A2, A21
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Character String
ALIASES: None

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Get_Msg
DESCRIPTION: Passed as control from A321 to A322. Part of
Decode_Msg.
SOURCES: A321
DESTINATIONS: A322
COMPOSITION: N/A
PART OF: Decode_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Decode_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Graph-Fig
DESCRIPTION: Output from A352 which becomes Graphic_Fig_
Data which then becomes Opcode/Fig_Data and is
input to A4.
SOURCES: A352
DESTINATIONS: A4, A42
COMPOSITION: N/A
PART OF: Graphic_Fig_Data, Opcode/Fig_Data
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Graph_Figure_Data, Graphic_Fig_Data, Opcode/Fig_Data

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Graph_Figure_Data
DESCRIPTION: Output from A353 which becomes Graphic_Fig_Data which then becomes Opcode/Fig_Data and is input to A4.
SOURCES: A353
DESTINATIONS: A4
COMPOSITION: N/A
PART OF: Graphic_Fig_Data, Opcode/Fig_Data
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Graph_Fig, Graphic_Fig_Data, Opcode/Fig_Data

TYPE: DATA ELEMENT
DATE: 16 Aug 83
NAME: Graphic_Fig_Data
DESCRIPTION: Carries graphic display information from A35 to A4 so that display can be output as desired.
SOURCES: A35, A352, A353, A3
DESTINATIONS: A4
COMPOSITION: N/A
PART OF: Opcode/Fig_Data
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Opcode/Fig_Data, Graphic_Fig_Info, Graph_Fig, Graph_Figure_Data

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Graphic_Fig_Info
DESCRIPTION: Output from A2 used as input to A4 which carries graphics display information for output.
SOURCES: A2, A25
DESTINATIONS: A4
COMPOSITION: N/A
PART OF: Opcode/Fig_Data
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Opcode/Fig_Data, Graphic_Fig_Data

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Graphics_Info
DESCRIPTION: Passed from A221 as input to A222 and contains
information on the graphics display to be
obtained from the data base.
SOURCES: A221
DESTINATIONS: A222
COMPOSITION: N/A
PART OF: Verified_Msg
DATA CHARACTERISTICS: Character String
ALIASES: None

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Invalid_File_Info_Msg
DESCRIPTION: Output from A32 which results when the infor-
mation requested is not in the database. Is
control to A2.
SOURCES: A32, A321, A3
DESTINATIONS: A2
COMPOSITION: N/A
PART OF: Invalid_Msg
DATA CHARACTERISTICS: Character String
ALIASES: Invalid_Msg, Insufficient_Space_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Invalid_Msg
DESCRIPTION: Composed of Invalid_File_Info_Msg from A32 or
Insufficient_Space_Msg from A33.
SOURCES: A32, A33
DESTINATIONS: A2, A22
COMPOSITION: Invalid_File_Info_Msg, Insufficient_Space_Msg
PART OF: N/A
DATA CHARACTERISTICS: Character String
ALIASES: Invalid_File_Info_Msg, Insufficient_Space_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Insufficient_Space_Msg
DESCRIPTION: Indicates that there is not enough space on
disk to store the display as requested.
SOURCES: A33, A332
DESTINATIONS: A2
COMPOSITION: N/A
PART OF: Invalid_Msg
DATA CHARACTERISTICS: Character String
ALIASES: Invalid_File_Info_Msg, Invalid_Msg

TYPE: DATA ELEMENT
DATE: 19 Aug 83
NAME: Keyboard_Input
DESCRIPTION: Operator uses to begin working with the
system. Input to A1.
SOURCES:
DESTINATIONS: A1
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Character String
ALIASES: None

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Keyed_Input
DESCRIPTION: Output from A341 which determines the next
action which will be taken. Options include
send message to the screen or printer, termi-
nate session, or do more editing.
SOURCES: A341
DESTINATIONS: A342, A343, A2, A5
COMPOSITION: Screen_Msg, Print_Msg, Other_Terminate_Msg,
Additional_Edit_Msg
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Additional_Edit_Msg, Other_Terminate_Msg,
Print_Msg, Screen_Msg

TYPE: DATA ELEMENT
DATE: 19 Aug 83
NAME: More_Edit_Msg
DESCRIPTION: Output from A4 which is control to A2 and
signifies that more editing is desired.
SOURCES: A4, A42, A43
DESTINATIONS: A2, A22
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Character String
ALIASES: None

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: More_Message
DESCRIPTION: Output from A23 which becomes part of
Continue_Message which is control to A22.
SOURCES: A23
DESTINATIONS: A22
COMPOSITION: N/A
PART OF: Continue_Message
DATA CHARACTERISTICS: Flag
ALIASES: More_Msg, Continue_Message

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: More_Msg
DESCRIPTION: Output from A24 which becomes Continue_Message
which acts as control to A22.
SOURCES: A24
DESTINATIONS: A22
COMPOSITION: N/A
PART OF: Continue_Message
DATA CHARACTERISTICS: Flag
ALIASES: More_Message, Continue_Message

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Next_Action_Msg
DESCRIPTION: Output from A43 which provided the user with
the ability to choose whether to continue in
INGRED or to terminate the session.
SOURCES: A43, A433
DESTINATIONS: A2, A5
COMPOSITION: Terminate_Msg, More_Edit_Msg
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Decision_Msg, More_Edit_Msg, Terminate_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Next_Req_Msg
DESCRIPTION: Output from A343 which acts as control to A341
and gives option to choose next action.
SOURCES: A343
DESTINATIONS: A341
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Nxt_Req_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Nxt_Req_Msg
DESCRIPTION: Output from A342 which is control to A341 and
provides method to choose next action after
screen display has been viewed.
SOURCES: A342
DESTINATIONS: A341
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Next_Req_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Obtain_Msg
DESCRIPTION: Passed from A31 to A32 indicating desire to
obtain rather than store display information.
SOURCES: A31
DESTINATIONS: A32, A321
COMPOSITION: N/A
PART OF: Decoded_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Decoded_Msg, Store_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Opcode_and_Data
DESCRIPTION: Output from A322 which contains graphics
information and info on what to do with it.
Input to A323.
SOURCES: A322
DESTINATIONS: A323
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Array of Integer/Real
ALIASES: None

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Opcode/Fig
DESCRIPTION: Contains graphics display information from A32
which is passed as input to A35.
SOURCES: A32, A323
DESTINATIONS: A35, A352, A353
COMPOSITION: N/A
PART OF: Opcode/Fig_Info
DATA CHARACTERISTICS: Array of Integer/Real
ALIASES: Opcode/Fig_Info, Opcode_Fig

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Opcode_Fig
DESCRIPTION: Carries graphics display information from A33
to A35 for transfer as desired.
SOURCES: A33, A333
DESTINATIONS: A35
COMPOSITION: N/A
PART OF: Opcode/Fig_Info
DATA CHARACTERISTICS: Array of Integer/Real
ALIASES: Opcode/Fig_Info, Opcode/Fig

TYPE: DATA ELEMENT
DATE: 19 Aug 83
NAME: Opcode/Fig_Data
DESCRIPTION: Carries information from A2 contained in the
graphics figures in addition to what is to be
done. It is the input to A3.
SOURCES: A2, A3
DESTINATIONS: A4, A42, A43, A432
COMPOSITION: Graphic_Fig_Info, Graphic_Fig_Data
PART OF: N/A
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Graphic_Fig_Info, Graphic_Fig_Data

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Opcode/Fig_Info
DESCRIPTION: Contains either Opcode/Fig or Opcode_Fig which
are passed as input to A35
SOURCES: A32, A33
DESTINATIONS: A35
COMPOSITION: Opcode/Fig, Opcode_Fig
PART OF: N/A
DATA CHARACTERISTICS: Array of Integers/Reals
ALIASES: Opcode/Fig, Opcode_Fig

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Other_Terminate_Msg
DESCRIPTION: Output from A34 which calls A5, and is the way
the user signifies desire to terminate session
when in the data base accessing mode.
SOURCES: A3, A34, A341
DESTINATIONS: A5
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Terminate_Msg, Another_Terminate_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Output/Fig_Data
DESCRIPTION: Output from A352 and becomes Out_Msg which
becomes Output_Message which is control to A4.
SOURCES: A352
DESTINATIONS: A4
COMPOSITION: N/A
PART OF: Out_Msg, Output_Message
DATA CHARACTERISTICS: Character String
ALIASES: Out_Msg, Output_Message

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: "Output Info"_Msg
DESCRIPTION: Output from A351 which is control to A353.
SOURCES: A351
DESTINATIONS: A353
COMPOSITION: N/A
PART OF: Decoded_Mesg
DATA CHARACTERISTICS: Flag
ALIASES: Decoded_Mesg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Output Message
DESCRIPTION: Composed of Output_Msg from A2 and Out_Msg from A3, and acts as control to A4 to signify type of output desired.
SOURCES: A2, A3
DESTINATIONS: A4, A41
COMPOSITION: Output_Msg, Out_Msg
PART OF: N/A
DATA CHARACTERISTICS: Character String
ALIASES: Output_Msg, Out_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Out_Msg
DESCRIPTION: Output from A3 which signifies type of output desired for the display.
SOURCES: A3, A35, A352, A353
DESTINATIONS: A4
COMPOSITION: N/A
PART OF: Out_Msg, Output_Message
DATA CHARACTERISTICS: Character String
ALIASES: Output_Msg, Output_Message, Output/Fig_Data, Out_To_Msg

TYPE: DATA ELEMENT
DATE: 20 Aug 83
NAME: Output_Msg
DESCRIPTION: Output from A25 which designates to A4 which type of output is desired.
SOURCES: A2, A25
DESTINATIONS: A4
COMPOSITION: N/A
PART OF: Output Message
DATA CHARACTERISTICS: Character String
ALIASES: Output_Message, Out_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Out_To_Msg
DESCRIPTION: Output from A353 which becomes Out_Msg which
becomes Output_Message which is control to A4.
SOURCES: A353
DESTINATIONS: A4
COMPOSITION: N/A
PART OF: Out_Msg, Output_Message
DATA CHARACTERISTICS: Character String
ALIASES: Output/Fig_Data, Out_Msg, Output_Message

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Perform_Msg
DESCRIPTION: Output from A35 (part of Decoded_Mesg) which
is control to A352.
SOURCES: A351
DESTINATIONS: A352
COMPOSITION: N/A
PART OF: Decoded_Mesg
DATA CHARACTERISTICS: Flag
ALIASES: Decoded_Mesg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Place_Msg
DESCRIPTION: Flag from A332 which is control to A333.
SOURCES: A332
DESTINATIONS: A333
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: None

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Printer_Request
DESCRIPTION: Output from A41 which is breakdown of
Decode_Mesg and signifies that output goes to
printer rather than the terminal.
SOURCES: A41
DESTINATIONS: A43, A431
COMPOSITION: N/A
PART OF: Decode_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Terminal_Request, Decode_Mesg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Print_Msg
DESCRIPTION: Output from A341 which is control to A343.
SOURCES: A341
DESTINATIONS: A343
COMPOSITION: N/A
PART OF: Keyed_Input
DATA CHARACTERISTICS: Flag
ALIASES: Keyed_Input, Screen_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Quit_Msg
DESCRIPTION: This provides the method from Activity A23 by
which the user can terminate the session.
SOURCES: A23
DESTINATIONS: A5
COMPOSITION: N/A
PART OF: Another_Terminate_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Another_Terminate_Msg, Terminate_Msg,
Other_Terminate_Msg, Stop_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Req_Msg
DESCRIPTION: Output from A33 which shows which type output
(screen or to printer) is desired. Control to
A34.
SOURCES: A33, A333
DESTINATIONS: A34
COMPOSITION: N/A
PART OF: Request_Message
DATA CHARACTERISTICS: Flag
ALIASES: Request_Message, Request_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Request_Message
DESCRIPTION: Contains either Request_Msg from A32 or
Req_Msg from A33.
SOURCES: A32, A33
DESTINATIONS: A34, A341
COMPOSITION: Request_Msg, Req_Msg
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Request_Msg, Req_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Request_Msg
DESCRIPTION: Output from A32 which passes to A34 which type
of output (screen or printer) is desired.
SOURCES: A32, A323
DESTINATIONS: A34
COMPOSITION: N/A
PART OF: Request_Message
DATA CHARACTERISTICS: Flag
ALIASES: Request_Message, Req_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Screen_Msg
DESCRIPTION: Output from A341 as control to A342.
SOURCES: A341
DESTINATIONS: A342
COMPOSITION: N/A
PART OF: Keyed_Input
DATA CHARACTERISTICS: Flag
ALIASES: Print_Msg, Keyed_Input

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Stop_Msg
DESCRIPTION: Flag from A22 , passed to A5 to terminate session.
SOURCES: A22
DESTINATIONS: A5
COMPOSITION: N/A
PART OF: Another_Terminate_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Terminate_Msg, Other_Terminate_Msg, Quit_Msg, Another_Terminate_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Store_Msg
DESCRIPTION: Passed from A31 and acts as control to A33, signifying that the request is to store changed data.
SOURCES: A31
DESTINATIONS: A33, A331
COMPOSITION: N/A
PART OF: Decoded_Msg
DATA CHARACTERISTICS: Flag
ALIASES: Decoded_Msg, Obtain_Msg

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Terminal_Request
DESCRIPTION: Output from A41 which is a breakdown of
Decode Mesg and signifies the desire to send
the output to the terminal rather than the
printer.
SOURCES: A41
DESTINATIONS: A42
COMPOSITION: N/A
PART OF: Decode Mesg
DATA CHARACTERISTICS: Flag
ALIASES: Decode_Mesg, Printer_Request

TYPE: DATA ELEMENT
DATE: 26 Aug 83
NAME: Terminate_Msg
DESCRIPTION: Passed from A4 as control to A5 to request
termination of the user session.
SOURCES: A4, A42, A43
DESTINATIONS: A5
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: Other_Terminate_Msg, Another_Terminate_Msg,
Decision_Msg, Next_Action_Msg

TYPE: DATA ELEMENT
DATE: 21 Aug 83
NAME: Transfer_Msg
DESCRIPTION: Passed from A34 to A35 to signify that
display can be output to the appropriate
device (screen or printer.)
SOURCES: A34, A342, A343
DESTINATIONS: A35, A351
COMPOSITION: N/A
PART OF: N/A
DATA CHARACTERISTICS: Flag
ALIASES: None

TYPE: DATA ELEMENT

DATE: 26 Aug 83

NAME: Transmit_Msg

DESCRIPTION: Output from A342 which becomes Transfer_Msg
which is control to A35.

SOURCES: A342

DESTINATIONS: A35

COMPOSITION: N/A

PART OF: Transfer_Msg

DATA CHARACTERISTICS: Character String

ALIASES: Tx_Msg, Transfer_Msg

TYPE: DATA ELEMENT

DATE: 26 Aug 83

NAME: Tx_Msg

DESCRIPTION: Output from A343 which becomes Transfer_Msg
which is control to A35.

SOURCES: A343

DESTINATIONS: A35

COMPOSITION: N/A

PART OF: Transfer_Msg

DATA CHARACTERISTICS: Character String

ALIASES: Transfer_Msg, Transmit_Msg

TYPE: DATA ELEMENT

DATE: 21 Aug 83

NAME: Verified_Msg

DESCRIPTION: Passed from A221 and contains input and control information for A222.

SOURCES: A221

DESTINATIONS: A222

COMPOSITION: Call_Msg, Graphics_Info

PART OF: N/A

DATA CHARACTERISTICS: Character String

ALIASES: Call_Msg, Graphics_Info

Appendix F

Test Plan

This test plan is designed to thoroughly test the INGRED II system. It is divided into two parts, the development test and the acceptance test.

Development Testing

The development testing for this system will consist of three stages: the walkthrough, the compilation, and the confidence test.

Walkthrough

The first walkthrough will be performed during the design phase. During this walkthrough each module will be inspected to determine the following:

1. Are interconnect calls correct?
2. Are there logic errors?
3. Is more detail needed?
4. Are all module attributes declared and used correctly?
5. Are good software engineering practices followed?
6. Are the requirements supported?

The second walkthrough will be during the coding phase. The code will be inspected for the following errors:

1. Is the code commented adequately?
2. Are there design errors?

3. Are interconnect calls correct?
4. Are there logic errors?
5. Is the code maintainable?
6. How will the performance be?
7. Is a resonable amount of storage used?

If certain modules are error prone in the judgement of the author, they will be reworked completely before attempting the next step of the development testing.

Compilation

After the errors found in the coding walkthrough have been corrected, the code will be subjected to the next phase of development testing; compilation. In this stage, errors cited by the VAX FORTRAN 77 compiler will be corrected. If any modules are considered error prone during this phase, they will be reworked completely before attempting the next phase of development testing.

Confidence Testing

In Confidence Testing (CT), each module will be tested using a combination of error guessing, equivalence partitioning, boundary analysis, and path testing. When all modules have satisfactorily passed the tests, the whole system will again be submitted to the same types of tests. During final CT tests will also be run to compare storage times and storage space required for INGRED II. The CT will consist of:

1. Create a display.
 - a. Store using data file.

- (1) Restore from data file.
 - (2) Determine space required.
- b. Store using data base.
 - (1) Restore from data base.
 - (2) Determine space required.
2. Compare storage for INGRED II and INGRED.
 - a. Read SADT's from tape to disk.
 - b. Determine amount of storage required.
 - c. Store each using INGRED II (using both data base and data file methods).
 - (1) Determine storage space required for each.
3. Compare storage time using INGRED II and INGRED.
 - a. Determine store time for simple SADT (using INGRED and INGRED II).
 - b. Determine restore time for simple SADT.
 - c. Determine store time for complicated SADT.
 - d. Determine restore time for complicated SADT.

Acceptance Test

The acceptance test is to be designed for the overall INGRED II system and will consist of two separate tests. These tests are described below.

The first test will be called the System Compatibility Test (SCT) and will test the system by introducing faults. In this case, eight faults will be introduced through program input. For each fault that fails to produce the correct response, an additional two faults will be

introduced. If the total faults introduced grows to more than 14, the SCT will be terminated and the software will have to be completely reworked before another test can be performed. If the total number of faults does not exceed 14, then success or failure of the SCT will be based on calculation of the total number of faults introduced compared with the number of faults that were handled correctly. If 70 per cent or more of the faults are handled properly, the SCT will be considered successful. All software errors found will be corrected. If less than 70 per cent of the faults are handled properly, the SCT will be considered a failure and the software must be completely reworked before another test can be performed.

The second test will be called the Operational Test and will consist of a user, with little knowledge of the INGRED II software, who will draw, store, and retrieve graphics displays with INGRED II. The user will generate one simple and one complex picture using INGRED II; store and retrieve the pictures using both the data files and the data base; display a listing of pictures stored in the data base; and delete both pictures from the disk files. The test will be considered successful if 90 per cent of the user's commands are executed correctly. All software errors found will be corrected.

Acronyms

ACM - Association for Computing Machinery
AFAL - Air Force Avionics Laboratory
AFFDL - Air Force Flight Dynamics Laboratory
AFIT - Air Force Institute of Technology
AFML - Air Force Materials Laboratory
AFWAL - Air Force Wright Aeronautical Laboratories
ANSI - American National Standards Institute
ASD - Aeronautical Systems Division
CAD - Computer Aided Design
CAM - Computer Aided Manufacturing
DBMS - Data Base Management System
DEL - Digital Equipment Laboratory
GCS - Graduate Computer Science
GKS - Graphical Kernel System
GWCORE - George Washington University Core
GWU - George Washington University
INGRED - Interactive Graphics Editor
ISO - International Standards Organization
NASA - National Aeronautics and Space Administration
RIM - Relational Information Management System
SADT - Structured Analysis and Design Technique
SCT - System Compatibility Test
SIGGRAPH - Special Interest Group for Graphics
WPAFB - Wright-Patterson Air Force Base

INGRED II SOURCE CODE

```
*****
*****
* DATE: 30 NOV 83
* VERSION: 3.1
* TITLE: INGRED II
* FILENAME: INGREDII
* OWNER: CAPT JAMES D. GATEWOOD
*       *CAPT KEVIN ROSE
*       *DR. GARY B. LAMONT
* SOFTWARE SYSTEM: DEL VAX 11/780
* OPERATING SYSTEM: VMS
* LANGUAGE: FORTRAN 77
* USE: SOFTWARE ENGINEERING, GRAPHICS RESEARCH,
*      COURSE PROJECTS
* CONTENTS: MAIN, PUTPDF, RSTOR, COMMANDS,
*           DB, RESTOR, ADDIT, DIRECT,
*           PURGIT, CHGIT, LOADRC, INTCON,
*           DBLOAD, CONVRT, CMPACT, PUTIN,
*           RMOPEN, F1OPN, INTLOD, LODREC,
*           LOADIT, RIMLOK, DELETE, LXLINE
* FUNCTION: THIS PROGRAM IS DESIGNED TO GIVE
*           THE USER THE CAPABILITY TO GENERATE,
*           STORE, AND RETRIEVE GRAPHICS
*           DISPLAYS AND ASSOCIATED INFORMATION.
*           THIS PARTICULAR VERSION HAS BEEN
*           TESTED ON THE TEKTRONIX 4014 IN THE
*           DEL.
*****
*****
```

```

*****
*****
* NAME:   INGRED II
* DATE:   30 NOV 83
* VERSION: 3.1
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED:  NAMPAS, STB, SEG, PAR,
*                      EDCTRL
* CALLING SUBROUTINE:  NONE
* SUBROUTINES CALLED:  INIT, SCORTP, STNDX, SLNDX,
*                      SFNDX, SCHSPA, NITSRF, SELSRF,
*                      PUTIN, EXINSRT, RSTRSTB, PRVW,
*                      SWINDO, SVPRT2, GRID, COMMANDS,
*                      CRTSEG, MOVA2, TEXT, CLTSEG,
*                      WKEYBD, DRWBOX, DRWFBX, DRWCIRC,
*                      DRWFCIRC, DRWLIN, DRWAH, DRWTEXT,
*                      DB, PUTPDF, RSTOR, RSTART,
*                      SETFONT, DLTSEG, SETLCOL, SETTCOL,
*                      SETFCOL, EDHELP, SETDEFR, NODEFR,
*                      NEWFRM, DELSRF, TRMSRF, TERM
* DESCRIPTION:  THIS IS THE MAIN ROUTINE FOR INGRED II
*               AND PROVIDES THE CAPABILITY TO PERFORM
*               ALL INGRED II ACTIONS.  THE SUBROUTINES
*               CALLED INCLUDE SOME FROM GWCORE, SOME
*               FROM INGRED, AND DB WHICH IS THE INGRED II
*               INTERFACE BETWEEN THE GWCORE/INGRED AND
*               RIM-5.
* HISTORY:     THIS MAIN PROGRAM IS AN ADAPTATION OF INGRED
*               WHICH WAS WRITTEN BY CAPT KEVIN ROSE.  THE
*               INGRED VERSION USED WAS 2.2 WHICH WAS DATED
*               13 SEP 82.  ANY QUESTIONS CONCERNING THIS
*               SYSTEM SHOULD BE ADDRESSED TO DR. GARY LAMONT.
*****
*****

```

```

COMMON/EDCTRL/EDFONT,DLTFLG,FIRST,NMSEQ,POSNAID
INTEGER*2 EDFONT,NMSEQ
LOGICAL DLTFLG,FIRST,POSNAID

```

```

COMMON /NAMPAS/ TUPHER,CATGRY,AUTHOR,FLNM,COMND,ABSENT
CHARACTER*3 COMND
CHARACTER*8 FLNM
CHARACTER*20 CATGRY,AUTHOR
LOGICAL TUPHER,ABSENT

```

```

COMMON /STB/ FILE,TABLE,FLPNTR,FLEND,STEND,
1 SGTMAX,MATRIX,XFMMAT,ROTATE,IMAGE,MATPTR
INTEGER*2 STEND,SGTMAX,MATPTR
INTEGER*4 FILE(30000),TABLE(5,1000),FLPNTR,FLEND
REAL MATRIX(4,4),XFMMAT(4,3,1000),ROTATE(3,1000)

```

LOGICAL IMAGE

```
COMMON /SEG/ VISIB, DETECT, HILITE, IXFORM, SEGTP, OPSEG,  
1 OPSEGX, OPTSGX, ANGLES  
LOGICAL VISIB, DETECT, HILITE, OPSEGX, OPTSGX  
REAL IXFORM(4,4), ANGLES(3)  
INTEGER*2 SEGTP, OPSEG
```

```
COMMON /PAR/ PARRAY  
INTEGER*4 PARRAY(5000)  
REAL RARRAY(5000)  
EQUIVALENCE (XTNTARY, IXTNTARY)
```

```
INTEGER SNAM, LENGTH  
LOGICAL PRIMFLG  
INTEGER*2 ANSR, J, OP, PTR  
CHARACTER*8 FILNAM  
CHARACTER*80 WORD  
PARAMETER NO=78  
INTEGER ERRCNT, CNT  
CHARACTER TCMD(10)
```

```
* INITIALIZE CORE  
  CALL INIT(3,2,2)  
  CALL SCORTP(1)  
  CALL STNDX(8)  
  CALL SLNDX(8)  
  CALL SFNDX(8)  
* INITIALIZE EDITOR  
  EDFONT=0  
  FIRST=.TRUE.  
  DLTFLG=.FALSE.  
  POSNAID=.TRUE.  
  ERRCNT=0  
  CNT=0  
* SET DEFAULT FANCY CHARACTER SPACING  
  CALL SCHSPA(-0.5)  
  NMSEQ=1  
* CREATE SEGMENT, INITIALIZE CRT, AND DRAW FIGURE  
  CALL NITSRF(10,2,2)  
  CALL SELSRF(10)  
* CALL TEMPLATE FOR INITIAL DISPLAY  
  FILNAM='HEADJDG'  
  FLNM = FILNAM  
* OPEN THE FILE  
  OPEN(UNIT=1, FILE='DMA1:[ROSE.PIKFILE]'//FILNAM//' .DAT',  
    + STATUS='UNKNOWN')  
* INSERT DELETED OPCODE 55 INFO BACK IN  
  CALL PUTIN  
* READ NUMBER OF ITEMS IN PDF  
* FLEND=FILE(1)
```

```

PTR=1
SNAM=0
PRIMFLG=.FALSE.
* READ DISK FILE INTO THE PSEUDO DISPLAY FILE
DO 11 J=1, FLEND
  PARRAY(PTR)=FILE(J)
  IF (PTR.EQ.1) THEN
    LENGTH=FILE(J)
  ELSE IF (PTR.EQ.2) THEN
    IF (FILE(J).EQ.55) THEN
      SNAM=SNAM+1
* AFTER THE FIRST SEGMENT, NEW SEGMENTS ARE DETECTED BY THE
* DEFINE ALL ATTRIBUTES OP_CODE OF 55. THIS CALL IS FOR THE PREVIOUS
* SEGMENT TO BE INSERTED IN THE EXTENT TABLE!
      IF (SNAM.NE.1) CALL EXINSRT(SNAM-1)
      NMSEQ=SNAM+1
    ENDIF
* ALL OP CODES ARE SENT TO THE SEGMENT TABLE TO BE ADDED TO LENGTH OF
* START A NEW SEGMENT
    CALL RSTRSTB(SNAM,LENGTH,J-1)
* IF OPCODE IS A PRIMITIVE: AFTER PARRAY IS FULL FIND PRIMITIVE EXTENT
    IF (FILE(J).LT.8) PRIMFLG=.TRUE.
    ELSE IF (PTR.EQ.LENGTH) THEN
      IF (PRIMFLG) CALL PKXTNT
      PRIMFLG=.FALSE.
      PTR=0
    ENDIF
    PTR=PTR+1
11 CONTINUE
* INSERT LAST SEGMENT
  CALL EXINSRT(SNAM)
* CLOSE THE FILE
  CLOSE(UNIT=1,STATUS='KEEP')
  READ(5,33)NAM
33 FORMAT(A1)
  CALL NEWFRM
  CALL DELALL
* CREATE SEGMENT, INITIALIZE CRT, AND DRAW FIGURE
*   CALL NITSRF(10,2,2)
*   CALL SELSRF(10)
* MAKE A WINDOW
  CALL SWINDO(0.0,10.0,0.0,10.0)
* SHRINK VIEWPORT TO ALLOW FOR COMMANDS AND PROMPTS
  CALL SVPRT2(0.1,1.0,0.1,1.0)
* PUT TEMPORARY SEGMENT ON SURFACE TO AID POSITIONING OBJECTS
  CALL GRID(POSNAID)
  CALL COMMANDS
50 CONTINUE
  CALL SVPRT2(0.0,1.0,0.0,1.0)
  CALL CRTSEG
  CALL MOVA2(1.0,0.8)

```

```

CALL TEXT('COMMAND: ',9)
CALL MOVA2(0.5*CNT+1.8,0.8)
CALL TEXT(' ',1)
CALL CLTSEG
* RESTORE VIEWPORT AND MOVE CURSOR TO PROMPT POSITION
CALL SVPRT2(0.1,1.0,0.1,1.0)
DO 10 K=1,10
  TCMD(K)=' '
10 CONTINUE
CALL WKEYBD(500,1,TCMD,CHS)
CNT=CNT+1
COMND=TCMD(1)//TCMD(2)//TCMD(3)
* BRANCH ON BASIS OF COMMAND
IF (COMND.EQ.'BOX') THEN
  CALL DRWBOX(NMSEQ)
  NMSEQ=NMSEQ+1
  FIRST=.FALSE.
ELSE IF (COMND.EQ.'FBO') THEN
  CALL DRWFBOX(NMSEQ)
  NMSEQ=NMSEQ+1
  FIRST=.FALSE.
ELSE IF (COMND.EQ.'CIR') THEN
  CALL DRWCIRC(NMSEQ)
  NMSEQ=NMSEQ+1
  FIRST=.FALSE.
ELSE IF (COMND.EQ.'FCI') THEN
  CALL DRWFCIRC(NMSEQ)
  NMSEQ=NMSEQ+1
  FIRST=.FALSE.
ELSE IF (COMND.EQ.'LIN') THEN
  CALL DRWLIN(NMSEQ)
  NMSEQ=NMSEQ+1
  FIRST=.FALSE.
ELSE IF (COMND.EQ.'ARR') THEN
  CALL DRWAH(NMSEQ)
  NMSEQ=NMSEQ+1
  FIRST=.FALSE.
ELSE IF (COMND.EQ.'TEX') THEN
  CALL DRWTEXT(NMSEQ)
  NMSEQ=NMSEQ+1
  FIRST=.FALSE.
ELSE IF (COMND.EQ.'QUI') THEN
  GO TO 100
ELSE IF (COMND.EQ.'STO') THEN
  CALL DB
ELSE IF (COMND.EQ.'SAV') THEN
  CALL PUTPDF
ELSE IF (COMND.EQ.'ERA') THEN
  CALL RSTART
  NMSEQ=1
ELSE IF (COMND.EQ.'RET') THEN

```

```

        CALL RSTOR
    ELSE IF (COMND.EQ.'RES') THEN
        CALL DB
    ELSE IF (COMND.EQ.'DIR') THEN
        CALL DB
    ELSE IF (COMND.EQ.'PUR') THEN
        CALL DB
    ELSE IF (COMND.EQ.'CHA') THEN
        CALL DB
    ELSE IF (COMND.EQ.'PRE') THEN
        CALL PRVW
        CNT=0
        ERRCNT=0
    ELSE IF (COMND.EQ.'FON') THEN
        CALL SETFONT
    ELSE IF (COMND.EQ.'DEL') THEN
        CALL DLTSEG
    ELSE IF (COMND.EQ.'LCO') THEN
        CALL SETLCOL
    ELSE IF (COMND.EQ.'TCO') THEN
        CALL SETTCOL
    ELSE IF (COMND.EQ.'FCO') THEN
        CALL SETFCOL
    ELSE IF (COMND.EQ.'HEL') THEN
        CALL EDHELP
    ELSE IF (COMND.EQ.'DEF') THEN
        CALL SETDEFR
    ELSE IF (COMND.EQ.'NOD') THEN
        CALL NODEFR
    ELSE IF (COMND.EQ.'GRI') THEN
        POSNAID=.NOT.POSNAID
    ELSE
        CALL SVPRT2(0.0,1.0,0.0,1.0)
        CALL CRTSEG
        CALL MOVA2(10.0,9.0-(0.5*ERRCNT+0.5))
        CALL TEXT(' NO COMMAND: ',13)
        CALL MOVA2(10.0,9.0-(0.5*ERRCNT+0.5)-0.2)
        CALL TEXT(' ',5)
        CALL TEXT(%REF(TCMD),10)
        CALL CLTSEG
        ERRCNT=ERRCNT+1
        CALL SVPRT2(0.1,1.0,0.1,1.0)
    ENDIF
    GO TO 50
* ERASE ALL BUT THE RETAINED SEGMENTS
100 CONTINUE
    CALL SVPRT2(0.0,1.0,0.0,1.0)
    CALL NEWFRM
* TERMINATE THE INTERACTIVE SESSION AND CORE
    CALL DELSRF (10)
    CALL TRMSRF (10)

```

CALL TERM
STOP
END

```

*****
*****
* NAME: PUTPDF
* DATE: 24 NOV 83
* VERSION: 2.1
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED: STB, EDCTRL, SEG, PAR
*                      NAMPAS, END
* CALLING SUBROUTINE: DB, MAIN
* SUBROUTINES CALLED: GSQUEZ, SVPRT2, CRTSEG, MOVA2,
*                      TEXT, CMPACT, CLTSEG, WKEYBD
* DESCRIPTION: IN THIS SUBROUTINE THE OPCODE 55
*              VALUES ARE CHECKED AND COMPACTED IF
*              OF THE DEFAULT VALUE, THEN THE
*              REMAINING INFORMATION IS PUT ON DISK.
* HISTORY: THIS IS AN INGRED SUBROUTINE WHICH I
*          MODIFIED FOR THIS APPLICATION.
*****
*****

```

SUBROUTINE PUTPDF

```

COMMON /STB/ FILE, TABLE, FLPNTR, FLEND, STEND,
1      SGTMAX, MATRIX, XFMMAT, ROTATE, IMAGE, MATPTR
INTEGER*2 STEND, SGTMAX, MATPTR
INTEGER*4 FILE(30000), TABLE(5,1000), FLPNTR, FLEND
REAL MATRIX(4,4), XFMMAT(4,3,1000), ROTATE(3,1000)
LOGICAL IMAGE

COMMON/EDCTRL/EDFONT,DLTFLG,FIRST,NMSEQ,POSNAID
INTEGER*2 EDFONT,NMSEQ
LOGICAL DLTFLG,FIRST,POSNAID

COMMON /SEG/ VISIB,DETECT,HILITE,IXFORM,SEGTP,OPSEG,
1      IOSEGXMIOTSGXMABGKES
LOGICAL VISIB,DETECT,HILITE,OPSEGX,SPTSGX
REAL IXFORM(4,4),ANGLES(3)
INTEGER*2 SEGTP,OPSEG

COMMON /PAR/ PARRAY
INTEGER*4 PARRAY(5000)
REAL RARRAY(5000)

COMMON /NAMPAS/ TUPHER,CATGRY,AUTHOR,FLNM,COMND,ABSENT
CHARACTER*3 COMND
CHARACTER*8 FLNM
CHARACTER*20 CATGRY,AUTHOR
LOGICAL TUPHER,ABSENT

COMMON /END/ TMPEND,TMPBUF

```

```

INTEGER*4 TMPEND,TMPBUF(30000)

INTEGER*2 J
CHARACTER*8 FILNAM
CHARACTER*80 WORD

* SQUEEZE OUT ALL THE DELETED SEGMENTS FIRST
  IF (DLTFLG) THEN
    CALL GSQUEZ
    DLTFLG=.FALSE.
  END IF
* GET THE PICTURE FILE NAME
  CALL SVPRT2(0.0,1.0,0.0,1.0)
* SEND A PROMPT
  CALL CRTSEG
  CALL MOVA2(1.0,0.65)
  CALL TEXT('FILE NAME? (8 CHARS): ',22)

* WHILE WAITING FOR RESPONSE DELETE REDUNDANT OPCODE 55'S
  CALL CMPACT

  CALL CLTSEG
  CALL SVPRT2(0.1,1.0,0.1,1.0)
  CALL WKEYBD(500,1,WORD,NCHARS)
  IF (NCHARS.GT.8) NCHARS=8
  FILNAM=WORD(1:NCHARS)
  FLNM = FILNAM
* OPEN THE FILE
  OPEN(UNIT=1,FILE='DMA1:[ROSE.PIKFILE]//FILNAM//'.DAT',
    + STATUS='UNKNOWN')
* PUT SIZE OF PDF IN FIRST ITEM IN FILE
  WRITE (1,*) FLEND
* WRITE OUT THE FILE AS A LIST FROM 1 TO FLEND
  DO 10 J=1,FLEND
    WRITE (1,*) TMPBUF(J)
10 CONTINUE
* CLOSE AND SAVE THE FILE
  CLOSE(UNIT=1,STATUS='KEEP')
  RETURN
  END

```

```

*****
*****
* NAME: RSTOR
* DATE: 16 SEP 83
* VERSION: 2.1
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED: STB, SEG, EDFONT, PAR,
* NAMPAS
* CALLING SUBROUTINE: MAIN, RESTOR
* SUBROUTINES CALLED: SVPRT2, CRTSEG, MOVA2, TEXT,
* CLTSEG, WBUTN, DELALL, WKEYBD,
* PUTIN, EXINSRT, RSTRSTB, PKXTNT
* PRVW
* DESCRIPTION: THIS SUBROUTINE RETRIEVES THE DISPLAY
* FILE FROM DISK, PUTS THE OPCODE 55
* INFORMATION BACK INTO THE APPROPRIATE
* PLACES, AND WRITES THE DATA STRUCTURE
* TO THE PDF.
* HISTORY: THIS IS AN INGRES ROUTINE WHICH I CONVERTED
* FOR USE IN THIS APPLICATION.
*****
*****

```

SUBROUTINE RSTOR

```

* PROMPT FOR A SAVED PICTURE FILE AND PUT IT IN THE DISPLAY FILE

```

```

COMMON /STB/ FILE, TABLE, FLPNTR, FLEND, STEND,
1 SGTMAX, MATRIX, XFMMAT, ROTATE, IMAGE, MATPTR
INTEGER*2 STEND, SGTMAX, MATPTR
INTEGER*4 FILE(30000), TABLE(5, 1000), FLPNTR, FLEND
REAL MATRIX(4, 4), XFMMAT(4, 3, 1000), ROTATE(3, 1000)
LOGICAL IMAGE

```

```

COMMON /SEG/ VISIB, DETECT, HILITE, IXFORM, SEGTYPE, OPSEG,
1 OPSEGX, OPTSGX, ANGLES
LOGICAL VISIB, DETECT, HILITE, OPSEGX, OPTSGX
REAL IXFORM(4, 4), ANGLES(3)
INTEGER*2 SEGTYPE, OPSEG

```

```

COMMON /EDCTRL/ EDFONT, DLTF LG, FIRST, NMSEQ, POSNAID
INTEGER*2 EDFONT, NMSEQ
LOGICAL DLTF LG, FIRST, POSNAID

```

```

COMMON /PAR/ PARRAY
INTEGER*4 PARRAY(5000)
REAL RARRAY(5000)
EQUIVALENCE (XTNTARY, IXTNTARY)

```

```

COMMON /NAMPAS/ TUPHER, CATGRY, AUTHOR, FLNM, COMND, ABSENT

```

CHARACTER*3 COMND
CHARACTER*8 FLNM
CHARACTER*20 CATGRY,AUTHOR
LOGICAL TUPHER,ABSENT

INTEGER SNAM,LENGTH
LOGICAL PRIMFLG
INTEGER*2 ANSR,J,OP,PTR
CHARACTER*8 FILNAM
CHARACTER*80 WORD
PARAMETER NO=78

```
* IF NO DELETE IS NEEDED THEN GO TO PROMPT
  IF (FIRST) GO TO 5
* PROMPT FOR CONFIRMATION TO DELETE ALL CURRENT SEGMENTS
  CALL SVPRT2(0.0,1.0,0.0,1.0)
  CALL CRTSEG
  CALL MOVA2(1.0,0.1)
  CALL TEXT('DELETE ALL CURRENT SEGMENTS? ',29)
  CALL CLTSEG
  CALL SVPRT2(0.1,1.0,0.1,1.0)
  CALL WBUTN(500,ANSR)
  IF (ANSR.EQ.NO) RETURN
* DELETE ALL RETAINED SEGMENTS PRIOR TO DISPLAYING A NEW FILE
5 CALL DELALL
* PROMPT FOR FILE NAME
  CALL SVPRT2(0.0,1.0,0.0,1.0)
  CALL CRTSEG
  CALL MOVA2(1.0,0.25)
  CALL TEXT('_INPUT FILE: ',13)
  CALL CLTSEG
  CALL SVPRT2(0.1,1.0,0.1,1.0)
  CALL WKEYBD(500,1,WORD,NCHARS)
  IF (NCHARS.GT.8) NCHARS=8
  FILNAM=WORD(1:NCHARS)
  FLNM = FILNAM
* OPEN THE FILE
  OPEN(UNIT=1,FILE='DMA1:[ROSE.PIKFILE]//FILNAM//'.DAT',
    + STATUS='UNKNOWN')
* INSERT DELETED OPCODE 55 INFO BACK IN
  CALL PUTIN
* READ NUMBER OF ITEMS IN PDF
* FLEND=FILE(1)
  PTR=1
  SNAM=0
  PRIMFLG=.FALSE.
* READ DISK FILE INTO THE PSEUDO DISPLAY FILE
  DO 10 J=1,FLEND
    PARRAY(PTR)=FILE(J)
    IF (PTR.EQ.1) THEN
      LENGTH=FILE(J)
```

```

        ELSE IF (PTR.EQ.2) THEN
            IF (FILE(J).EQ.55) THEN
                SNAM=SNAM+1
            * AFTER THE FIRST SEGMENT, NEW SEGMENTS ARE DETECTED BY THE
            * DEFINE ALL ATTRIBUTES OP_CODE OF 55. THIS CALL IS FOR THE PREVIOUS
            * SEGMENT TO BE INSERTED IN THE EXTENT TABLE!
                IF (SNAM.NE.1) CALL EXINSRT(SNAM-1)
                NMSEQ=SNAM+1
            ENDIF
            * ALL OP CODES ARE SENT TO THE SEGMENT TABLE TO BE ADDED TO LENGTH OF
            * START A NEW SEGMENT
                CALL RSTRSTB(SNAM,LENGTH,J-1)
            * IF OPCODE IS A PRIMITIVE: AFTER PARRAY IS FULL FIND PRIMITIVE EXTENT
                IF (FILE(J).LT.8) PRIMFLG=.TRUE.
                ELSE IF (PTR.EQ.LENGTH) THEN
                    IF (PRIMFLG) CALL PKXTNT
                    PRIMFLG=.FALSE.
                    PTR=0
                ENDIF
                PTR=PTR+1
10    CONTINUE
            * INSERT LAST SEGMENT
                CALL EXINSRT(SNAM)
            * CLOSE THE FILE
                CLOSE(UNIT=1,STATUS='KEEP')
                CALL PRVW
                RETURN
            END

```

```

*****
*****
* NAME:  COMMANDS
* DATE:  25 OCT 83
* VERSION:  1.3
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  NONE
* CALLING SUBROUTINE:  MAIN
* SUBROUTINES CALLED:  SVPRT2, CRTSEG, MOVA2, TEXT,
*                      LINA2, CLTSEG
* DESCRIPTION:  THIS SUBROUTINE SETS UP THE GRID
*              FOR INGRED II.
* HISTORY      THIS IS AN INGRED ROUTINE WHICH I
*              CONVERTED FOR USE IN INGRED II.
*****
*****

```

SUBROUTINE COMMANDS

```

        CALL SVPRT2(0.0,1.0,0.0,1.0)
        CALL CRTSEG
*  PRIMITIVES
        CALL MOVA2(0.0,9.85)
        CALL TEXT('FIGURES',7)
        CALL MOVA2(0.1,9.6)
        CALL TEXT('LINE',4)
        CALL MOVA2(0.1,9.3)
        CALL TEXT('ARROW',5)
        CALL MOVA2(0.1,9.0)
        CALL TEXT('BOX',3)
        CALL MOVA2(0.1,8.7)
        CALL TEXT('FBOX',4)
        CALL MOVA2(0.1,8.4)
        CALL TEXT('CIRCLE',6)
        CALL MOVA2(0.1,8.1)
        CALL TEXT('FCIRCLE',7)
        CALL MOVA2(0.1,7.8)
        CALL TEXT('TEXT',4)
*  BOX FOR PRIMITIVES
        CALL MOVA2(0.0,7.6)
        CALL LINA2(0.95,7.6)
        CALL LINA2(0.95,9.8)
        CALL LINA2(0.0,9.8)
        CALL LINA2(0.0,7.6)
*  ATTRIBUTES
        CALL MOVA2(0.0,7.3)
        CALL TEXT('ATTRBUTS',8)
        CALL MOVA2(0.1,7.0)
        CALL TEXT('FONT',4)
        CALL MOVA2(0.1,6.7)

```

```

CALL TEXT('LCOLOR',6)
CALL MOVA2(0.1,6.4)
CALL TEXT('TCOLOR',6)
CALL MOVA2(0.1,6.1)
CALL TEXT('FCOLOR',6)
* BOX FOR ATTRIBUTES
CALL MOVA2(0.0,5.9)
CALL LINA2(0.95,5.9)
CALL LINA2(0.95,7.25)
CALL LINA2(0.0,7.25)
CALL LINA2(0.0,5.9)
* EDITOR COMMANDS
CALL MOVA2(0.0,5.55)
CALL TEXT('CONTROL',7)
CALL MOVA2(0.1,5.3)
CALL TEXT('PREVIEW',7)
CALL MOVA2(0.1,5.0)
CALL TEXT('SAVE',4)
CALL MOVA2(0.1,4.7)
CALL TEXT('RETRIEVE',8)
CALL MOVA2(0.1,4.4)
CALL TEXT('ERASE',5)
CALL MOVA2(0.1,4.1)
CALL TEXT('QUIT',4)
CALL MOVA2(0.1,3.8)
CALL TEXT('DELETE',6)
CALL MOVA2(0.1,3.5)
CALL TEXT('GRID',4)
* BOX FOR CONTROL
CALL MOVA2(0.0,3.4)
CALL LINA2(0.95,3.4)
CALL LINA2(0.95,5.5)
CALL LINA2(0.0,5.5)
CALL LINA2(0.0,3.4)
* DATA BASE COMMANDS
CALL MOVA2(0.0,3.1)
CALL TEXT('DATA BASE',9)
CALL MOVA2(0.1,2.8)
CALL TEXT('STORE',5)
CALL MOVA2(0.1,2.5)
CALL TEXT('RESTORE',7)
CALL MOVA2(0.1,2.2)
CALL TEXT('DIRECTRY',8)
CALL MOVA2(0.1,1.9)
CALL TEXT('PURGE',5)
* BOX FOR DATA BASE
CALL MOVA2(0.0,3.0)
CALL LINA2(0.95,3.0)
CALL LINA2(0.95,1.5)
CALL LINA2(0.0,1.5)
CALL LINA2(0.0,3.0)

```

```
*  HELP
    CALL MOVA2(0.1,1.0)
    CALL TEXT('HELP',4)
    CALL CLTSEG
    CALL SVPRT2(0.1,1.0,0.1,1.0)
    RETURN
    END
```

```

*****
*****
* NAME: DB
* DATE: 23 SEP 83
* VERSION: 2.1
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED: RIMCOM, END, NAMPAS, CONST4,
*                      CONST8, RMKEYW, CDCDBS, FLAGS,
*                      FILES
* CALLING SUBROUTINE: MAIN
* SUBROUTINES CALLED: LXCONS, RMSTRT, SETIN, SETOUT,
*                      LXSET, TEXT, RMCLOS, CMPACT,
*                      GSQUEZ, SVPRT2, CRTSEG, MOVA2,
*                      CLTSEG, CONVRT, RMDATE, RMTIME,
*                      INTCON, DBLOAD
* DESCRIPTION: INITIALIZES RIM, ACCEPTS INPUT FOR
*              FILENAME REQUESTED, AND CALLS THE RIM
*              ROUTINES NEEDED TO DO REQUESTED
*              MANIPULATIONS
*****
*****

```

SUBROUTINE DB

```

COMMON /RIMCOM/ RMSTAT
INTEGER RMSTAT

```

```

COMMON /END/ TMPEND,TMPBUF
INTEGER*4 TMPEND,TMPBUF(30000)

```

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

INCLUDE 'CONST4.BLK'
INCLUDE 'CONST8.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CDCDBS.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'DCLAR6.BLK'

```

```

LOGICAL TTY
INTEGER VER,UDXX,MACH(2)
DATA UDXX /4HUD21/
REAL*8 CATGRY
INTEGER NUMATT,NUMROW,INTOPT

```

```

CKFIND = 0
NUMOPN = 0
BATCH = .FALSE.
ABSENT = .FALSE.
K = 0
IF(.NOT.TTY(K)) BATCH=.TRUE.

```

* INITIALIZE

```

NINT = 5
NOUT = 6
NOUTR = 6
CALL LXCONS
CALL RMSTRT
CALL SETIN(K8IN)
CALL SETOUT(K8OUT)
INTOPT = 0
NEXTOP = K8BEGI
ECHO = .FALSE.
CALL LXSET(KWECHO,K4ON)
CALL RMDATE(IDAY)
CALL RMTIME(ETIME)

```

```

IF (COMND.EQ.'DIR') THEN
  CALL DIRECT
  GO TO 75
END IF

```

```

IF (COMND.EQ.'STO') THEN
  CALL PUTPDF

```

* TAKE OUT THE REDUNDANT OPCODE 55'S

```

  CALL CONVRT
  INTOPT=0
  CALL INTCON(INTOPT)
  IF (ABSENT) GO TO 999
  CALL DBLOAD
  ELSE IF (COMND.EQ.'RES') THEN
    CALL RESTOR
  ELSE IF (COMND.EQ.'PUR') THEN
    CALL PURGIT
  ELSE IF (COMND.EQ.'CHA') THEN
    CALL CHGIT
  END IF

```

```

75  CALL RMCLOS

```

```

999  CONTINUE
      RETURN
      END

```

```

*****
*****
* NAME:  RESTOR
* DATE:  23 NOV 83
* VERSION:  2.2
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  END, NAMPAS, BUFFER, RIMCOM,
*                      VARDAT, RMATTS, RMKEYW, CONST4,
*                      CONST8, RIMPTR, WHCOM, TUPLEA,
*                      TUPLER, FLAGS, FILES, MISC,
*                      STRCOM
* CALLING SUBROUTINE:  DB
* SUBROUTINES CALLED:  INTCON, LXLREC, LXSREC, LXITEM,
*                      LFIND, LOCREL, WHERE, RIMLOK,
*                      PUTIN, ADDIT, RMCLOS, RSTOR
* DESCRIPTION:  CALLS GWCORE SUBROUTINE RSTOR WHICH
*               RETRIEVES A DISPLAY FROM THE DATA FILE.
* HISTORY:  THIS ROUTINE ORIGINALLY WAS OF A DIFFERENT
*           SCOPE WHEN THE IDEA WAS TO STORE THE DISPLAY
*           INFORMATION IN THE DATA BASE. THE COMMENTED
*           OUT CODE WAS USED THEN AND WORKED. IT IS
*           IN IN CASE THE READER DESIRES TO PURSUE THE
*           APPROACH AT SOME LATER DATE. THIS COMMENTED
*           OUT CODE INCLUDES PARTS FROM RIM-5 SUBROUTINES
*           RMMAIN, INTCON, RIM AND QUERY.
*****
*****

```

SUBROUTINE RESTOR

```

COMMON /END/ TMPEND, TMPBUF
INTEGER*4 TMPEND, TMPBUF(30000)

```

```

COMMON /NAMPAS/ RELATE, ATTRIB, FLNM, COMND, ABSENT
REAL*8 RELATE, ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

INCLUDE 'BUFFER.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'VARDAT.BLK'
INCLUDE 'RMATTS.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'CONST8.BLK'
INCLUDE 'RIMPTR.BLK'
INCLUDE 'WHCOM.BLK'
INCLUDE 'TUPLEA.BLK'
INCLUDE 'TUPLER.BLK'

```

```

INCLUDE 'FLAGS.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'SRTCOM.BLK'

```

```

LOGICAL SADR
INCLUDE 'DCLAR4.BLK'

```

```

CALL RSTOR
RETURN

```

```

****

```

```

* THE REST OF THIS WAS HERE WHEN I PLANNED TO STORE THE GWCORE
* DATA STRUCTURE IN THE RIM DATA BASE TOO. I LEFT IT AS AN
* EXAMPLE OF ONE WAY TO ACCOMPLISH THE STORAGE. LESSONS LEARNED
* DICTATE THAT STORAGE USING THE USER APPLICATION INTERFACE IS
* MORE DESIRABLE THAN THIS WAY BECAUSE OF RIM RESTRICTIONS ON
* NUMBER OF COMMANDS AND NUMBER OF CHARACTERS PER RECORD.
* HOWEVER, IF THE PROPER ARRAYS ARE RESET WITH LARGE ENOUGH
* VALUES, THIS APPROACH COULD WORK TOO.

```

```

* READ(5,9)FLNM
* 9 FORMAT(A8)
* INTOPT = 0
* CALL INTCON(INTOPT)
* IF (ABSENT) GO TO 999
* OPEN(UNIT=1,FILE='DMA1:[GATEWOOD]RENN97.JDG',STATUS='UNKNOWN')
* WRITE(1,*)'SELECT DISPLAY FROM GRAPHICS WHERE FILE EQ ',FLNM
* CLOSE(UNIT=1,STATUS='KEEP')
* NEXTOP=K8READ
* NUNEOF=0
* LENREC=0
* NIN=17
* OPEN(UNIT=17,FILE='DMA1:[GATEWOOD]RENN97.JDG',STATUS='UNKNOWN')
* CALL LXLREC(DUM,LENREC,DUM)
* CLOSE(UNIT=17,STATUS='DELETE')
* NIN=5
* ITEMS=LXITEM(DUM)
* ISAVE=LSTCMD
* CALL LXSREC(1,1,3,LSTCMD,1)
* ITEMS = LXITEM(IDUMMY)
* NS=0
* FIND THE FROM IN THE QUERY
* J=LFIND(1,ITEMS,KWFROM,4)
* FIND THE WHERE IN THE QUERY
* JW=LFIND(1,ITEMS,KWWHER,5)
* RNAME=BLANK
* CALL LXSREC((J+1),1,8,RNAME,1)
* FIND THE RELATION IN THE RELATION TABLE
* I=LOCREL(RNAME)
* FIND THE WHERE IN THE QUERY
* K=LFIND(1,ITEMS,KWWHER,5)

```

```

*      NBOO=0
*      LIMTU=ALL9S
*  PROCESS THE WHERE CLAUSE
*      CALL WHERE(K)
*  SEE IF ANY TUPLES SATISFY THE WHERE CLAUSE
*      CALL RIMLOK(IDUMMY,1,1,LENGTH)
*  OPEN THE FILE
*      OPEN(UNIT=1,FILE='DMA1:[GATEWOOD]RELN99.JDG',STATUS='UNKNOWN')
*  INSERT DELETED OPCODE 55 INFO BACK INTO FILE
*      CALL PUTIN
*      CALL ADDIT
*      CALL RMCLOS
*  999 CONTINUE
*      RETURN
*      END

```

```

*****
*****
* NAME:  ADDIT
* DATE:  17 NOV 83
* VERSION:  1.2
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  STB, SEG, EDCTRL, PAR,
*                      NAMPAS
* CALLING SUBROUTINE:  RESTOR, DIRECT
* SUBROUTINES CALLED:  RSTRSTB, PKXTNT, EXINSRT, PRVW
* DESCRIPTION:  THIS SUBROUTINE READS THE FILE FROM
*              DISK INTO THE PSEUDO DISPLAY FILE.
*****
*****

```

SUBROUTINE ADDIT

```

COMMON /STB/  FILE, TABLE, FLPNTR, FLEND, STEND,
+            SGTMAX, MATRIX, XFMMAT, ROTATE, IMAGE, MATPTR
INTEGER*2  STEND, SGTMAX, MATPTR
INTEGER*4  FILE(30000), TABLE(5,1000), FLPNTR, FLEND
REAL  MATRIX(4,4), XFMMAT(4,3,1000), ROTATE(3,1000)
LOGICAL  IMAGE

```

```

COMMON /SEG/  VISIB, DETECT, HILITE, IXFORM, SEGTP, OPSEG,
+            OPSEGX, OPTSGX, ANGLES
LOGICAL  VISIB, DETECT, HILITE, OPSEGX, OPTSGX
REAL  IXFORM(4,4), ANGLES(3)
INTEGER*2  SEGTP, OPSEG

```

```

COMMON /EDCTRL/  EDFONT, DLTFLG, FIRST, NMSEQ, POSNAID
INTEGER*2  EDFONT, NMSEQ
LOGICAL  DLTFLG, FIRST, POSNAID

```

```

COMMON /PAR/  PARRAY
INTEGER*4  PARRAY(5000)

```

```

COMMON /NAMPAS/  RELATE, ATTRIB, FLNM, COMND, ABSENT
REAL*8  RELATE, ATTRIB
CHARACTER*3  COMND
CHARACTER*8  FLNM
LOGICAL  ABSENT

```

```

REAL  RARRAY(5000)
EQUIVALENCE(XTNTARY, IXTNTARY)
INTEGER  SNAM, LENGTH
LOGICAL  PRIMFLG
INTEGER*2  ANSR, J, OP, PTR
CHARACTER*8  FILNAM
CHARACTER*8  WORD

```

PARAMETER NO=78

```
PTR=1
SNAM=0
PRIMFLG=.FALSE.
* READ DISK FILE INTO THE PSEUDO DISPLAY FILE
  DO 10 J=1,FLEND
    PARRAY(PTR)=FILE(J)
    IF (PTR.EQ.1) THEN
      LENGTH=FILE(J)
    ELSE IF (PTR.EQ.2) THEN
      IF (FILE(J).EQ.55) THEN
        SNAM=SNAM+1
* AFTER THE FIRST SEGMENT, NEW SEGMENTS ARE DETECTED BY THE
* DEFINE ALL ATTRIBUTES OP_CODE OF 55. THIS CALL IS FOR THE PREVIOUS
* SEGMENT TO BE INSERTED IN THE EXTENT TABLE!
        IF (SNAM.NE.1) CALL EXINSRT(SNAM-1)
        NMSEQ=SNAM+1
      ENDIF
* ALL OP CODES ARE SENT TO THE SEGMENT TABLE TO BE ADDED TO LENGTH O
* START A NEW SEGMENT
      CALL RSTRSTB(SNAM,LENGTH,J-1)
* IF OPCODE IS A PRIMITIVE: AFTER PARRAY IS FULL FIND PRIMITIVE EXTE
      IF (FILE(J).LT.8) PRIMFLG=.TRUE.
      ELSE IF (PTR.EQ.LENGTH) THEN
        IF (PRIMFLG) CALL PKXTNT
        PRIMFLG=.FALSE.
        PTR=0
      ENDIF
      PTR=PTR+1
10 CONTINUE
* INSERT LAST SEGMENT
  CALL EXINSRT(SNAM)
* GET RID OF THE TEMPORARY DISK FILE
  CLOSE(UNIT=1,STATUS='DELETE')
  IF (COMND.EQ.'RES') THEN
    CALL PRVW
  END IF
  RETURN
END
```

```

*****
*****
* NAME: DIRECT
* DATE: 13 NOV 83
* VERSION: 1.4
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED: BUFFER, RIMCOM, VARDAT, RMATTS,
* RMKEYW, CONST4, CONST8, RIMPTR,
* WHCOM, TUPLEA, TUPLER, FLAGS,
* FILES, MISC, SRTCOM, ATTBLE,
* END, NAMPAS, EDCTRL
* CALLING SUBROUTINE: DB
* SUBROUTINES CALLED: SVPRT2, CRTSEG, MOVA2, TEXT,
* CLTSEG, WBUTN, DELALL, LXLREC,
* LXID, LXWREC, LXLENC, LXSREC,
* RMDBGT, RMOPEN, LXITEM, LOCREL,
* SELECT, ADDIT, COMMANDS, GRID
* DESCRIPTION: THIS SUBROUTINE IS CALLED SO THE USER
* GET A DIRECTORY OF THE DISPLAYS THAT
* ARE CURRENTLY STORED IN THE DATA BASE.
* HISTORY: DIRECT INCLUDES PORTIONS OF RIM-5 SUBROUTINES
* RMAIN, INTCON, RIM, QUERY.
*****
*****

```

SUBROUTINE DIRECT

```

INCLUDE 'BUFFER.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'VARDAT.BLK'
INCLUDE 'RMATTS.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'CONST8.BLK'
INCLUDE 'RIMPTR.BLK'
INCLUDE 'WHCOM.BLK'
INCLUDE 'TUPLEA.BLK'
INCLUDE 'TUPLER.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'SRTCOM.BLK'
INCLUDE 'ATTBLE.BLK'

```

```

COMMON /END/ TMPEND, TMPBUF
INTEGER*4 TMPEND, TMPBUF(30000)

```

```

COMMON /NAMPAS/ RELATE, ATTRIB, FLNM, COMND, ABSENT
REAL*8 RELATE, ATTRIB
CHARACTER*3 COMND

```

CHARACTER*8 FLNM
LOGICAL ABSENT

COMMON /EDCTRL/ EDFONT,DLTFLG,FIRST,NMSEQ,POSNAID
INTEGER*2 EDFONT,NMSEQ
LOGICAL DLTFLG,FIRST,POSNAID
PARAMETER NO=78

LOGICAL SADR
INCLUDE 'DCLAR1.BLK'
INCLUDE 'DCLAR2.BLK'
INCLUDE 'DCLAR4.BLK'
INCLUDE 'DCLAR6.BLK'
INTEGER DBSTAT,VAL

* RESETS THE SEGMENT TABLE
INTEGER*2 ANSR

DBNAME=6HGRAFD
NAMDB=DBNAME
VAL = 3

* IF NO DELETE IS NEEDED, THEN GO TO PROMPT
IF (FIRST) GO TO 5

* PROMPT FOR VERIFICATION TO DELETE ALL CURRENT SEGMENTS
CALL SVPRT2(0.0,1.0,0.0,1.0)

CALL CRTSEG
CALL MOVA2(1.0,0.1)
CALL TEXT('DELETE ALL CURRENT SEGMENTS? ',29)
CALL CLTSEG
CALL SVPRT2(0.1,1.0,0.1,1.0)
CALL WBUTN(500,ANSR)
IF (ANSR.EQ.NO) RETURN

* DELETE ALL RETAINED SEGMENTS PRIOR TO SHOWING DIRECTORY

5 CALL DELALL
INTOPT = 0

* SHOW THAT I WANT TO QUERY THE DATA BASE

CALL LXLREC(VAL,1,LXERR)
IXID1 = LXID(1)
IXREC1 = 0
IF (IXID1.EQ.KZINT) IXREC1 = LXIREC(1)
IDBT = IXREC1
INTOPT = K4COM

* GET THE DATA BASE NAME

CALL LXLREC(NAMDB,6,LXERR)
IXID1 = LXID(1)
IXREC1 = LXWREC(1,1)
IXLEN = LXLENC(1)
NAMDB = BLANK
CALL LXSREC(1,1,8,NAMDB,1)

* GET THE DATA BASE

CALL RMDBGT(NAMDB,DBSTAT)
CALL RMOPEN(NAMDB)

```

IF (ABSENT) GO TO 999
NEXTOP = K8READ
OPEN(UNIT=1,FILE='DMA1:[GATEWOOD]RENN97.JDG',STATUS='UNKNOWN')
WRITE(1,*)'SELECT FILE FROM GRAPHICS'
CLOSE(UNIT=1,STATUS='KEEP')
NEXTOP=K8READ
NUMEOF=0
LENREC=0
NIN=21
OPEN(UNIT=21,FILE='DMA1:[GATEWOOD]RENN97.JDG',STATUS='UNKNOWN')
CALL LXLREC(DUM,LENREC,DUM)
CLOSE(UNIT=21,STATUS='DELETE')
NIN=5
ITEMS=LXITEM(DUM)
ISAVE=LSTCMD
CALL LXSREC(1,1,3,LSTCMD,1)
ITEMS = LXITEM(IDUMMY)
NS=0
* FIND THE FROM IN THE QUERY
J=LFIND(1,ITEMS,KWFROM,4)
NBOO = 0
LIMTU = ALL9S
RNAME=BLANK
CALL LXSREC((J+1),1,8,RNAME,1)
* FIND THE RELATION IN THE RELATION TABLE
I=LOCREL(RNAME)
NBOO=0
LIMTU=ALL9S
CALL SELECT
CALL ADDIT
* PROMPT TO CONTINUE AFTER VIEWING DIRECTORY
CALL SVPRT2(0.0,1.0,0.0,1.0)
CALL CRTSEG
CALL MOVA2(1.0,0.1)
CALL TEXT('PRESS ANY KEY TO CONTINUE: ',27)
CALL CLTSEG
CALL SVPRT2(0.1,1.0,0.1,1.0)
CALL WBUTN(500,ANSR)
CALL DELALL
CALL COMMANDS
CALL GRID(POSNAID)
999 CONTINUE
RETURN
END

```

```

*****
*****
* NAME: PURGIT
* DATE: 18 NOV 83
* VERSION: 2.1
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED: NAMPAS, END, EDCTRL, RMATTS,
* RMKEYW, CONST4, CONST8, FLAGS,
* FILES, RIMCOM, MISC, BUFFER,
* TUPLEA, TUPLER, ATTBLE
* CALLING SUBROUTINE: DB
* SUBROUTINES CALLED: SVPRT2, CRTSEG, MOVA2, TEXT,
* CLTSEG, WBUTN, DELALL, LXLREC,
* LXID, LXWREC, LXLENC, LXSREC,
* RMDBGT, RMOPEN, LOADRC, LFINDD,
* LOCREL, BLKDEF, BLKLOC, DELETE,
* CLKCLR, RMCLOS
* DESCRIPTION: THIS ROUTINE IS USED TO PURGE THE TUPLE
* AND THE DISPLAY FILE FOR THE FILE NAME
* THAT THE USER REQUESTS.
* HISTORY: THIS ROUTINE USES PORTIONS OF RIM-5 ROUTINES
* INTCON, MODIFY, AND LODREC.
*****
*****

```

SUBROUTINE PURGIT

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

COMMON /END/ TMPEND,TMPBUF
INTEGER*4 TMPEND,TMPBUF(30000)

```

```

COMMON /EDCTRL/ EDFONT,DLTFLG,FIRST,NMSEQ,POSNAID
INTEGER*2 EDFONT,NMSEQ
LOGICAL DLTFLG,FIRST,POSNAID
PARAMETER NO = 78

```

```

INCLUDE 'RMATTS.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'CONST8.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'BUFFER.BLK'

```

```

INCLUDE 'TUPLEA.BLK'
INCLUDE 'TUPLER.BLK'
INCLUDE 'ATTBLE.BLK'

```

```

LOGICAL NE,EQ
LOGICAL EQKEYW
INTEGER DBSTAT,VAL
INCLUDE 'DCLAR4.BLK'
INCLUDE 'DCLAR1.BLK'
INCLUDE 'DCLAR2.BLK'
INCLUDE 'DCLAR6.BLK'

```

```

DBNAME=6HGRAFDDB
NAMDB=DBNAME
VAL = 4

```

```

* IF NO DELETE IS NEEDED, THEN GO TO PROMPT
  IF (FIRST) GO TO 5
* PROMPT FOR VERIFICATION TO DELETE ALL CURRENT SEGMENTS
  CALL SVPRT2(0.0,1.0,0.0,1.0)
  CALL CRTSEG
  CALL MOVA2(1.0,0.1)
  CALL TEXT('DELETE ALL CURRENT SEGMENTS? ',29)
  CALL CLTSEG
  CALL SVPRT2(0.1,1.0,0.1,1.0)
  CALL WBUTN(500,ANSR)
  IF (ANSR.EQ.NO) RETURN
* DELETE ALL RETAINED SEGMENTS PRIOR TO SHOWING DIRECTORY
5  CALL DELALL
* GET THE FILE NAME
  CALL SVPRT2(0.0,1.0,0.0,1.0)
* SEND THE PROMPT
  CALL CRTSEG
  CALL MOVA2(1.0,0.65)
  CALL TEXT('FILE NAME? (8 CHARS): ',22)
  CALL CLTSEG
  CALL SVPRT2(0.1,1.0,0.1,1.0)
* GET RESPONSE
  READ(5,9) FLNM
9  FORMAT(A8)
* SHOW THAT I WANT TO DELETE FROM DATA BASE
  CALL LXLREC(VAL,1,LXERR)
  IXID1 = LXID(1)
  IXREC1 = 0
  IF (IXID1.EQ.KZINT) IXREC1 = LXIREC(1)
  IDBT = IXREC1
  INTOPT = K4COM
* GET THE DATA BASE NAME
  CALL LXLREC(NAMDB,6,LXERR)
  IXID1 = LXID(1)
  IXREC1 = LXWREC(1,1)
  IXLEN = LXLENC(1)

```

```

        NAMDB = BLANK
        CALL LXSREC(1,1,8,NAMDB,1)
* GET THE DATA BASE
        CALL RMDBGT(NAMDB,DBSTAT)
        CALL RMOPEN(NAMDB)
        IF (ABSENT) GO TO 999
        NEXTOP = K8READ
        OPEN(UNIT=1,FILE='DMA1:[GATEWOOD]RELL98.JDG',
+STATUS='UNKNOWN')
        WRITE(1,*) 'DELETE ROW FROM GRAPHICS WHERE FILE EQ ',FLNM
        CLOSE(UNIT=1,STATUS='KEEP')
* DELETE THE GWCORE DATA STRUCTURE
        OPEN(UNIT=1,FILE='DMA1:[ROSE.PIKFILE] '//FLNM//' .DAT',
+STATUS='UNKNOWN')
        CLOSE(UNIT=1,STATUS='DELETE')
        CALL LOADRC
        NEXTOP = K8READ
        ITEMS = LXITEM(IDUMMY)
        NS = 0
        J = LFINDD(1,ITEMS,KWFROM,4)
* FIND THE RELATION NAME IN RELATION TABLE
        RNAME = BLANK
        CALL LXSREC((J+1),1,8,RNAME,1)
        I = LOCREL(RNAME)
* CALL DELETE TO FINISH PROCESSING
        CALL BLKDEF(10,MAXCOL,1)
        KQ1 = BLKLOC(10)
        CALL DELETE(BUFFER(KQ1))
        CALL BLKCLR(10)
        CALL RMCLOS
999  CONTINUE
        RETURN
        END

```

```

*****
*****
* NAME:  CHGIT
* DATE:  10 NOV 83
* VERSION:  1.0
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  NONE
* CALLING SUBROUTINE:  DB
* SUBROUTINES CALLED:  NONE
* DESCRIPTION:  THIS SUBROUTINE IS A DUMMY AT THIS TIME.
*               IT CAN BE IMPLEMENTED TO CHANGE ATTRIBUTES
*               WITHIN A PARTICULAR TUPLE IF THAT IS
*               DETERMINED DESIRABLE.
*****
*****

```

SUBROUTINE CHGIT

```

PRINT*, 'YOU ARE IN CHANGE'
PRINT*, 'LEAVING CHANGE'

```

```

RETURN
END

```

```

*****
*****
* NAME:  LOADRC
* DATE:  25 NOV 83
* VERSION:  1.3
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  NAMPAS, END, RMATTS, RMKEYW,
*                      CONST4, CONST8, FLAGS, FILES,
*                      RIMCOM, MISC
* CALLING SUBROUTINE:  PURGIT
* SUBROUTINES CALLED:  SETIN, LXITEM, LXSREC
* DESCRIPTION:  THIS ROUTINE OPENS THE DATA FILE OF
*              THE FILE NAME WHICH WAS SUPPLIED IN
*              PURGIT BY THE USER.  LOADRC THEN
*              CLOSSES THAT FILE WITH STATUS DELETE.
*****
*****

```

SUBROUTINE LOADRC

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

COMMON /END/ TMPEND,TMPBUF
INTEGER*4 TMPEND,TMPBUF(30000)

```

```

INCLUDE 'RMATTS.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'CONST8.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'MISC.BLK'

```

```

LOGICAL EQKEYW
INCLUDE 'DCLAR4.BLK'

```

```

NUMEOF=0
1  CONTINUE
   LENREC = 0
   NIN = 21
   OPEN(UNIT=21,FILE='DMA1:[GATEWOOD]RELL98.JDG',STATUS='UNKNOWN')
   CALL LXLREC(DUM,LENREC,DUM)
   CLOSE(UNIT=21,STATUS='DELETE')
   NIN = 5
   IF (LXID(1).NE.K4EOF) GO TO 100

```

NUMEOF = NUMEOF +1
IF (CONNI) GO TO 1
CALL SETIN(K8IN)
GO TO 1
100 CONTINUE
ITEMS = LXITEM(DUM)
ISAVE = LSTCMD
CALL LXSREC(1,1,3,LSTCMD,1)
IF (ITEMS.GT.2) GO TO 1000
GO TO 1000
1000 CONTINUE
RETURN
END

```

*****
*****
* NAME: INTCON
* DATE: 22 OCT 83
* VERSION: 1.4
* CALLING SUBROUTINE: DB
* INPUTS: INTOPT
* OUTPUTS: INTOPT
* COMMON BLOCKS USED: RMATTS, RMKEYW, CONST4, FLAGS,
* FILES, RIMCOM, MISC, NAMPAS
* SUBROUTINES CALLED: LXLREC, LXID, LXWREC, LXLENC,
* LXSREC, RMOPEN, INTLOD
* DESCRIPTION: THIS IS WHERE IT IS DETERMINED WHICH
* RIM DATA BASE IS REQUIRED, THE DATA BASE
* IS THEN OPENED, AND INTLOD IS CALLED.
* EXECUTION OPTIONS (IDBT) ARE:
* IDBT=1 CREATE A NEW DATA BASE
* IDBT=2 UPDATE AN EXISTING DATA BASE
* IDBT=3 QUERY A DATA BASE
* IDBT=4 ENTER THE COMMAND MODE
* IDBT=5 EXIT
* HISTORY: THIS IS A RIM SUBROUTINE WHICH I MODIFIED
* THIS APPLICATION. IT NOW ONLY INCLUDES THE
* OPTION TO LOAD A DATA BASE.
*****
*****

```

SUBROUTINE INTCON(INTOPT)

```

INCLUDE 'RMATTS.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'MISC.BLK'

```

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

INTEGER DBSTAT,VAL
INCLUDE 'DCLAR2.BLK'

```

```

DBNAME=6HGRAFDB
NAMDB=DBNAME
VAL=2

```

```

* DECIDE TO LOAD OR QUERY THE DATA BASE

```

```

        IF (COMND.EQ.'RES') VAL=3
        CALL LXLREC(VAL,1,LXERR)
        IXID1 = LXID(1)
        IXREC1 = 0
        IF(IXID1.EQ.KZINT) IXREC1 = LXIREC(1)
        IDBT = IXREC1
* GET THE DATA BASE NAME
        CALL LXLREC(NAMDB,6,LXERR)
        IXID1 = LXID(1)
        IXREC1 = LXWREC(1,1)
        IXLEN = LXLENC(1)
        IF((IXID1.EQ.KZTEXT).AND.(IXLEN.LE.6)) GO TO 140
140    NAMDB = BLANK
        CALL LXSREC(1,1,8,NAMDB,1)
        IF(IDBT.NE.1) GO TO 180
*
* OPEN THE RIM FILE
*
180    CONTINUE
        CALL RMOPEN(NAMDB)
        IF (ABSENT) GO TO 999

* WANT TO LOAD ADDITIONAL DATA

        CALL LXLREC(INTOPT,1,LXERR)
        IXID1 = LXID(1)
        IXREC1 = 0
        IF(IXID1.EQ.KZINT) IXREC1 = LXIREC(1)
        IF(IXID1.EQ.KZTEXT) IXREC1 = LXWREC(1,1)

* LOAD DATA

        INTOPT = 0
        CALL INTLOD(INTOPT)
999    CONTINUE
        RETURN
        END

```

```

*****
*****
* NAME:  DBLOAD
* DATE:  29 OCT 83
* VERSION:  2.2
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  CONST8, RMKEYW, CONST4, TUPLER,
*                      RULCOM, FILES, BUFFER, MISC,
*                      FLAGS, RIMCOM, STB, STEND,
*                      END, NAMPAS
* CALLING SUBROUTINE:  DB
* SUBROUTINES CALLED:  LXSREC, LOCREL, RELGET, BLKDEF,
*                      LOADIT, RMDATE, RELPUT, BLKCLR
* DESCRIPTION:  DETERMINES AND LOCATES THE RELATION TO BE
*               LOADED, CALLS THE ROUTINES TO DETERMINE
*               NEXT ATTRIBUTE TO LOAD, AND LOADS IT.
* HISTORY:  THIS IS A RIM SUBROUTINE WHICH I MODIFIED
*           FOR THIS APPLICATION.
*****
*****

```

SUBROUTINE DBLOAD

```

COMMON /STB/ FILE, TABLE, FLPNTR, FLEND, STEND, SGTMAX,
1          MATRIX, SFFMAT, ROTATE, IMAGE, MATPTR
INTEGER*2 STEND, SGTMAX, MATPTR
INTEGER*4 FILE(30000), TABLE(5,1000), FLPNTR, FLEND
REAL MATRIX(4,4), XFMMAT(4,3,1000), ROTATE(3,1000)
LOGICAL IMAGE

```

```

COMMON /END/ TMPEND, TMPBUF
INTEGER*4 TMPEND, TMPBUF(30000)

```

```

COMMON /NAMPAS/ RELATE, ATTRIB, FLNM, COMND, ABSENT
REAL*8 RELATE, ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

INCLUDE 'CONST8.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'TUPLER.BLK'
INCLUDE 'RULCOM.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'BUFFER.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'RIMCOM.BLK'

```

```
INCLUDE 'DCLAR1.BLK'  
RELATE=8HGRAPHICS  
RNAME=RELATE  
MOD=4HNONE
```

```
* RELATION NAME SPECIFIED.
```

```
*
```

```
400 CONTINUE  
    RNAME=BLANK  
    CALL LXSREC(2,1,8,RNAME,1)  
    I=LOCREL(RNAME)  
600 CONTINUE  
    CALL RELGET(ISTAT)
```

```
* READ THE INFO FROM THE FILE
```

```
700 CONTINUE  
    CALL BLKDEF(10,1,MAXCOL)  
    KQ1 = BLKLOC(10)
```

```
* CHANGE SYSINPUT TO RELN96.JDG  
    CALL LOADIT(BUFFER(KQ1))
```

```
* UPDATE THE DATE OF LAST MODIFICATION.
```

```
    CALL RMDATE(RDATE)  
    CALL RELPUT  
    CALL BLKCLR(10)
```

```
* END OF LOADING.
```

```
RETURN  
END
```

```

*****
*****
* NAME:  CONVRT
* DATE:  16 SEP 83
* VERSION:  1.3
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  END, NAMPAS
* CALLING SUBROUTINE:  DB
* SUBROUTINES CALLED:  NONE
* DESCRIPTION:  CONVERTS THE INFORMATION ENTERED FROM THE
*               SCREEN AND PUTS INTO COMMON BLOCK FOR RIM-5.
*****
*****

```

SUBROUTINE CONVRT

```

COMMON /END/ TMPEND,TMPBUF
INTEGER*4 TMPEND,TMPBUF(30000)

```

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

INTEGER*4 NEWEND
MAXVAL = 500
TMPVAL = 0
I = 1
J = 1
NEWEND = TMPEND

```

```

* OPEN THE FILE
  OPEN(UNIT=1,FILE='DMA1:[GATEWOOD]RELN96.JDG',STATUS='UNKNOWN')
* PUT THE FILE NAME IN
  WRITE(1,*)FLNM
  CLOSE(UNIT=1,STATUS='KEEP')
  RETURN
  END

```

```

*****
*****
* NAME:  CMPACT
* DATE:  16 AUG 83
* VERSION:  2.3
* INPUTS:  NONE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  STB, EDCTRL, END
* CALLING SUBROUTINE:  PUTPDF, DB
* SUBROUTINES CALLED:  NONE
* DESCRIPTION:  CHECKS GWCORE GENERATED INFORMATION PACKET
*               FOR OCCURRENCES OF OPCODE 55, SUBSTITUTES
*               FLAG VALUE AND COMPACTS OUT THE REDUNDANT
*               OPCODE 55 INFORMATION TO SAVE STORAGE SPACE
*               REQUIRED.
*****
*****

```

SUBROUTINE CMPACT

```

COMMON /STB/ FILE, TABLE, FLPNTR, FLEND, STEND,
1  SGTMAX, MATRIX, SFMMAT, ROTATE, IMAGE, MATPTR
INTEGER*2 STEND, SGTMAX, MATPTR
INTEGER*4 FILE(30000), TABLE(5, 1000), FLPNTR, FLEND
REAL MATRIX(4, 4), XFMMAT(4, 3, 1000), ROTATE(3, 1000)
LOGICAL IMAGE

```

```

COMMON /EDCTRL/ EDFONT, DLTF LG, FIRST, NMSEG, POSNAID
INTEGER*2 EDFONT, NMSEQ
LOGICAL DLTF LG, FIRST, POSNAID

```

```

COMMON /END/ TMPEND, TMPBUF
INTEGER*4 TMPEND, TMPBUF(30000)

```

```

INTEGER*4 STRING(1:27)
INTEGER*4 BUF(1:27)
INTEGER I, J, K, L, M, BEGIN, START, CMPAR1, CMPAR2
INTEGER*4 INSERT, FILL
LOGICAL BADFLG

```

```

CMPAR1 = 31
CMPAR2 = 55
INSERT = -99998999
FILL = -99998998
CTR = 0
INCLUDE 'STRING.BLK'

```

```

DO 1 I=1, FLEND
  BADFLG = .FALSE.

```

```

* LOOK FOR LENGTH 31, OPCODE 55

```

```

      IF ((FILE(I).EQ.CMPAR1).AND.(FILE(I+1).EQ.CMPAR2)) THEN
        START = I+4
        DO J=1,27
          L=START+(J-1)
          BUF(J) = FILE(L)
        END DO
      * SEE IF THIS IS SAME AS 'STRING' WHICH HAS DEFAULT OPCODE 55
      * INFORMATION IN IT
        DO J=1,27
          K=J
          IF (BUF(J).NE.STRING(K)) BADFLG=.TRUE.
        END DO
        IF (.NOT.BADFLG) THEN
      * IF SAME THEN PUT IN INTEGER TO SHOW WHERE INFO WAS TAKEN OUT
          FILE(I) = INSERT
          FILE(I+1) = INSERT
          DO 4 K=START-2, START+26
      * NOW PUT IN INTEGER TO SHOW WHAT TO BLANK OUT
            FILE(K) = FILL
          4 CONTINUE
        END IF
      END IF
      1 CONTINUE
      * NOW CHECK FOR SPACES IN THE ARRAY WHICH CAN BE DELETED
        TMPEND = FLEND
        J=1
        I=1
        DO WHILE (I.LE.FLEND)
          IF (FILE(I).NE.FILL) THEN
            TMPBUF(J)=FILE(I)
            J=J+1
            I=I+1
          ELSE IF (FILE(I+1).EQ.FILL) THEN
            DO K=1,29
              TMPEND=TMPEND-1
            END DO
            I=I+29
          ELSE
            TMPBUF(J)=FILE(I)
            J=J+1
          END IF
        END DO

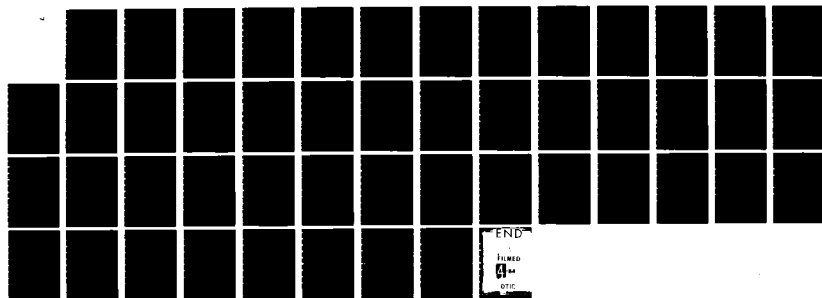
        FLEND=TMPEND
        RETURN
      END

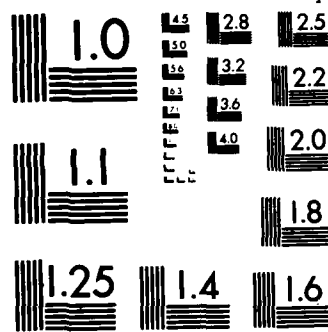
```

AD-A138 059

APPLICATIONS PROGRAMS TO FACILITATE USE OF A DBMS (DATA 3/3
BASE MANAGEMENT S. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... J D GATEWOOD
DEC 83 AFIT/GCS/EE/83D-10 F/G 9/2 NL

UNCLASSIFIED





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

```

*****
*****
* NAME: PUTIN
* DATE: 18 AUG 83
* VERSION: 2.1
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED: STB, SEG, EDCTRL
* CALLING SUBROUTINE: RSTOR, DB
* SUBROUTINES CALLED: NONE
* DESCRIPTION: CHECKS THE STORED DISPLAY DATA FOR FLAG
*               VALUES WHICH DESIGNATE THAT OPCODE 55
*               INFORMATION BELONGS AND INSERTS AT THAT
*               POINT.
*****
*****

```

SUBROUTINE PUTIN

```

COMMON /STB/ FILE, TABLE, FLPNTR, FLEND, STEND,
1      SGTMAX, MATRIX, SFMMAT, ROTATE, IMAGE, MATPTR
INTEGER*2 STEND, SGTMAX, MATPTR
INTEGER*4 FILE(30000), TABLE(5,1000), FLPNTR, FLEND
REAL MATRIX(4,4), XFMMAT(4,3,1000), ROTATE(3,1000)
LOGICAL IMAGE

```

```

COMMON /SEG/ VISIB, DETECT, HILITE, IXFORM, SEGTP, OPSEG,
1      OPSEGX, OPTSGX, ANGLES
LOGICAL VISIB, DETECT, HILITE, OPSEGX, OPTSGX
REAL IXFORM(4,4), ANGLES(3)
INTEGER*2 SEGTP, OPSEG

```

```

COMMON /EDCTRL/ EDFONT, DLTF LG, FIRST, NMSEG, POSNAID
INTEGER*2 EDFONT, NMSEQ
LOGICAL DLTF LG, FIRST, POSNAID

```

```

INTEGER I, J, K, L, START, CMPAR1
INTEGER*4 STRING(1:27), INSERT, NEWEND, TMPEND

```

```

CMPAR1 = 31
INSERT = -99998999
INCLUDE 'STRING.BLK'

```

```

READ (1,*) FLEND
TMPEND=FLEND
NEWEND=0
DO J=1,FLEND
  READ(1,*) FILE(J)
END DO
I=1
DO WHILE (I.LE.TMPEND)

```

```

IF ((FILE(I).EQ.INSERT).AND.(FILE(I+1).EQ.INSERT)) THEN
  START=I
  NEWEND=TMPEND
  TMPEND=TMPEND+29
  DO WHILE (NEWEND.NE.START+1)
    FILE(NEWEND+29)=FILE(NEWEND)
    NEWEND=NEWEND-1
  END DO
  FILE(START)=31
  FILE(START+1)=55
  FILE(START+2)=13
  FILE(START+3)=14
  K=0
  DO L=START+4,START+30
    K=K+1
    FILE(L) = STRING(K)
  END DO
END IF
I=I+1
END DO
FLEND = TMPEND
RETURN
END

```

```

*****
*****
* NAME:  RMOPEN
* DATE:  22 OCT 83
* VERSION:  1.2
* INPUTS:  IFILE
* OUTPUTS:  NONE
* COMMON BLOCKS USED:  CONST4, RIMCOM, FLAGS, ATTBLE
* CALLING SUBROUTINE:  INTCON
* SUBROUTINES CALLED:  RMCLOS, RMSTART, RMDATE, RMTIME,
*                      GETT, STRMOV, PUTT, F1OPN, F2OPN,
*                      F3OPN
* DESCRIPTION:  OPENS THE REQUESTED RIM DATA BASE SO THAT
*               IT CAN BE MANIPULATED
*   PARAMETERS:
*       IFILE---NAME OF THE DATABASE
* HISTORY:  THIS IS A RIM-5 SUBROUTINE WHICH I CONVERTED
*           FOR USE IN THIS APPLICATION
*****
*****

```

SUBROUTINE RMOPEN(IFILE)

```

INCLUDE 'CONST4.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'ATTBLE.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'DCLAR4.BLK'
DATA ICALLS /0/

```

```

IF(ICALLS.EQ.0) DFLAG = .FALSE.
ICALLS = ICALLS + 1
RMSTAT = 0

```

```

* CLOSE ANY EXISTING DATABASES AND INITIALIZE

```

```

IF(DFLAG) CALL RMCLOS
CALL RMSTRT

```

```

* SET THE NEW DATABASE NAME, DATE, AND TIME
  CALL RMDATE(DBDATE)
  CALL RMTIME(DBTIME)

```

```

* FIND THE LAST NON-BLANK CHARACTER.

```

```

      DO 100 I=1,7
      CALL GETT(IFILE,I,IT)
      IF (IT.EQ.IBLANK) GO TO 200
100 CONTINUE
      I = 7

```

200 CONTINUE
* FIX UP THE FILE NAMES.

FILE = BLANK
CALL STIRMOV(IFILE,1,I,FILE,1)
RIMDB1 = FILE
CALL PUTT(RIMDB1,I,K41)
RIMDB2 = FILE
CALL PUTT(RIMDB2,I,K42)
RIMDB3 = FILE
CALL PUTT(RIMDB3,I,K43)

* OPEN FILE 1.

CALL F1OPN(RIMDB1)
IF (RMSTAT.NE.0) GO TO 900

* OPEN FILE 2.

CALL F2OPN(RIMDB2)
IF (RMSTAT.NE.0) GO TO 900

* OPEN FILE 3.

CALL F3OPN(RIMDB3)
IF (RMSTAT.NE.0) GO TO 900

* IF THIS IS A NEW DATABASE WE NEED TO SET UP THE FIRST BTREE.

IF (DFLAG) THEN
CALL RMDATE(DBDATE)
GO TO 999
END IF

900 CALL SVPRT2(0.0,1.0,0.0,1.0)
* SEND THE PROMPT
CALL CRTSEG
CALL MOVA2(1.0,0.45)
CALL TEXT('THERE IS NO GRAFDB DATA BASE. CREATE IT NOW.',44)
CALL CLTSEG
CALL SVPRT2(0.1,1.0,0.0,1.0)
CALL WBUTN(500,KNOB)
ABSENT = .TRUE.

999 CONTINUE
RETURN
END

```

*****
*****
* NAME: F1OPN
* DATE: 15 OCT 83
* VERSION: 1.1
* INPUTS: IFILE
* OUTPUTS: NONE
* COMMON BLOCKS USED: CONST8, F1COM, RIMCOM, ATTBLE,
* RELTBL, FLAGS
* CALLING SUBROUTINE: RMOPEN
* SUBROUTINES CALLED: RIOOPN, RIOIN, BLKMOV, ZEROIT,
* RIOOUT
* DESCRIPTION: SETS UP FILE 1 OF THE RIM DIRECTORY FILE
* WHICH CONTAINS THE DATA BASE DEFINITION
* PARAMETERS:
* FILE----NAME OF THE FILE TO USE FOR FILE1
* HISTORY: THIS IS A RIM-5 SUBROUTINE WHICH I CONVERTED
* FOR USE IN THIS APPLICATION.
*****
*****

```

SUBROUTINE F1OPN(FILE)

```

INCLUDE 'CONST8.BLK'
INCLUDE 'F1COM.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'ATTBLE.BLK'
INCLUDE 'RELTBL.BLK'
INCLUDE 'FLAGS.BLK'
LOGICAL EQ
INCLUDE 'DCLAR4.BLK'

```

DBNAME=6HGRAFD8

```

* OPEN THE DIRECTORY FILE.

CALL RIOOPN(FILE,FILE1,LENBF1,IOS)
IF(IOS.NE.0) RMSTAT = 2100 + IOS

* READ IN THE FIRST RECORD FROM THIS FILE.

CALL RIOIN(FILE1,1,RELBUF,LENBF1,IOS)
IF(IOS.NE.0) GO TO 500
CRREC = 0

* MOVE THE CONTROL DATA TO WHERE IT IS NEEDED.

IF(EQ(RELBUF(3),K8RMDT)) GO TO 100
RMSTAT = 10
GO TO 1000
100 CONTINUE
IF(EQ(RELBUF(1),DBNAME)) GO TO 200

```

```

        RMSTAT = 11
        GO TO 1000
200  CONTINUE
        CALL BLKMOV(OWNER,RELBUF(5),2)
        CALL BLKMOV(DBDATE,RELBUF(7),2)
        CALL BLKMOV(DBTIME,RELBUF(9),2)
        LF1REC = RELBUF(11)
        NRROW = RELBUF(12)
        NAROW = RELBUF(13)

*  SUCCESSFUL OPEN.

        DFLAG = .TRUE.
        RMSTAT = 0
        GO TO 9999

*  EMPTY FILE 1 - WRITE THE FIRST RECORD ON IT.

500  CONTINUE
        CALL ZEROIT(RELBUF,LENBF1)
        CALL RIOOUT(FILE1,1,RELBUF,LENBF1,IOS)
        LF1REC = 1
        CAREC = 0
        CRREC = 0
        NRROW = 74
        NAROW = 227
        RMSTAT = 15
        GO TO 1000

*  UNABLE TO OPEN FILE 1.

1000 CONTINUE
        DFLAG = .FALSE.
9999 CONTINUE
        RETURN
        END

```

```

*****
*****
* NAME: INTLOD
* DATE: 8 NOV 83
* VERSION: 2.2
* INPUTS: INTOPT
* OUTPUTS: INTOPT
* COMMON BLOCKS USED: FILES, RMATTS, CONST4, CONST8,
*                     RMKEYW, MISC, RIMCOM, FLAGS,
*                     TUPLER, TUPLEA
* CALLING SUBROUTINE: INTCON
* SUBROUTINES CALLED: LOCREL, RELGET, LXLREC, LXID,
*                     LXSREC, LOCATT, ATTGET, STRMOV
* DESCRIPTION: DETERMINE WHAT RELATIONS ARE TO BE LOADED.
* HISTORY: THIS IS A RIM-5 SUBROUTINE WHICH I CONVERTED
*          FOR USE IN THIS APPLICATION.
*****
*****

```

SUBROUTINE INTLOD(INTOPT)

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

INCLUDE 'FILES.BLK'
INCLUDE 'RMATTS.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'CONST8.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'TUPLER.BLK'
INCLUDE 'TUPLEA.BLK'
INCLUDE 'DCLAR1.BLK'
INCLUDE 'DCLAR3.BLK'

```

```

INTEGER STATUS
LOGICAL EQ,NE
REAL*8 MOD

```

```

RELATE=8HGRAPHICS
MOD=4HNONE
UNO=1

```

* LOCATE THE REQUESTED SELECTION

```

I = LOCREL(BLANK)

```

```

        IF(I.EQ.0) GO TO 200

*   DISPLAY AVAILABLE RELATIONS

200 CONTINUE

*   GET THE USERS SELECTION

250 CONTINUE

*   THE FIRST 1 MEANS GET THE GRAPHICS RELATION, THE SECOND ONE JUST
*   MEANS THE FIRST ENTRY IS ONE CHARACTER IN LENGTH

        CALL LXLREC(1,1,LXERR)
        IDX = LXID(1)
        IRECX = LXIREC(1)
        IF((IRECX.GE.1).AND.(IRECX.LE.K)) GO TO 260

*   LOCATE THE REQUESTED SELECTION

260 CONTINUE
        I = LOCREL(BLANK)
        K=0
270 CALL RELGET(STATUS)

*   CHECK PERMISSION TO MODIFY THE RELATION

300 CONTINUE
        IF(EQ(MPW,NONE)) GO TO 360
        IF(EQ(MPW,USERID)) GO TO 360
        IF(EQ(USERID,OWNER)) GO TO 360
        CALL LXLREC(MOD,4,LXERR)
        MPW1 = NONE
        IDX = LXID(1)
        IF((IDX.EQ.KZTEXT).AND.(LXLENC(1).LE.8)) GO TO 340
        GO TO 300

*   CHECK THE PASSWORD

340 CONTINUE
        MPW1 = BLANK
        CALL LXSREC(1,1,4,MPW1,1)
350 CONTINUE
        IF(EQ(MPW1,MPW)) GO TO 355
        IF(EQ(MPW1,OWNER)) GO TO 355

*   GET THE ATTRIBUTES FOR THIS RELATION

355 CONTINUE
        USERID = MPW1
360 CONTINUE

```

```

      I = LOCATT(BLANK,NAME)
      NUM = 0
400  CALL ATTGET(STATUS)
      IF(STATUS.NE.0) GO TO 450
      NUM = NUM + 1
      NAMES(NUM) = ATTNAM
      IF(NUM.LT.8) GO TO 400
      WRITE(NOUT,410) (NAMES(J),J=1,7)
410  FORMAT(7(1X,A8),2X,1H+)
      NUM = 1
      NAMES(1) = NAMES(8)
      GO TO 400

*   PRINT LAST LINE OF ATTRIBUTES

450  WRITE(NOUT,460) (NAMES(J),J=1,NUM)
460  FORMAT(7(1X,A8))

*   GO GET THE DATA - CALL DBLOAD

      NAMES(1) = BLANK
      NAMES(2) = BLANK
      CALL STRMOV(KWLOAD,1,4,NAMES,1)
      CALL STRMOV(NAME,1,8,NAMES,6)
      CALL LXLREC(NAMES,16,LXERR)
      INTOPT = K4LOD
      GO TO 999

*   QUIT

997  CONTINUE
      INTOPT = K4QUIT
      GO TO 999

*   EXIT

998  CONTINUE
      INTOPT = K4EXIT
      GO TO 999

999  CONTINUE
      RETURN
      END

```

```

*****
*****
* NAME:   LODREC
* DATE:   11 NOV 83
* VERSION: 1.1
* INPUTS: NONE
* OUTPUTS: NONE
* COMMON BLOCKS USED:  NAMPAS, END, RMATTS, RMKEYW,
*                      CONST4, CONST8, FLAGS, FILES,
*                      RIMCOM, MISC
* CALLING SUBROUTINE:  LOADIT
* SUBROUTINES CALLED:  LXLREC, SETIN, LXITEM, LXSREC
*                      LXSET, SETOUT, RMCLOS
* DESCRIPTION:  THIS IS WHERE THE INFORMATION FROM THE
*               RELN96.JDG FILE IS ACTUALLY LOADED INTO
*               THE DATA BASE.
* HISTORY:  THIS IS A RIM-5 SUBROUTINE WHICH I CONVERTED
*           FOR USE IN THIS APPLICATION.
*****
*****

```

SUBROUTINE LODREC

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

COMMON /END/ TMPEND,TMPBUF
INTEGER*4 TMPEND,TMPBUF(30000)

```

```

INCLUDE 'RMATTS.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'CONST8.BLK'
INCLUDE 'FLAGS.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'MISC.BLK'

```

```

LOGICAL EQKEYW
INCLUDE 'DCLAR4.BLK'

```

```

NUMEOF = 0
1 CONTINUE
LENREC = 0
NIN=20
OPEN(UNIT=20,FILE='DMA1:[GATEWOOD]RELN96.JDG',STATUS='UNKNOWN')
CALL LXLREC(DUM,LENREC,DUM)
CLOSE(UNIT=20,STATUS='DELETE')

```

```

NIN=5
IF(LXID(1).NE.K4EOF) GO TO 100
NUMEOF = NUMEOF + 1
IF(BATCH) GO TO 200
IF(CONNI) GO TO 1
CALL SETIN(K8IN)
GO TO 1
100 CONTINUE
ITEMS = LXITEM(DUM)
ISAVE = LSTCMD
CALL LXSREC(1,1,3,LSTCMD,1)
IF(ITEMS.GT.3) GO TO 1000
IF(EQKEYW(1,KWHELP,4)) GO TO 200
GO TO 1000
200 CONTINUE
CALL RMCLOS
999 CONTINUE
STOP
1000 CONTINUE
RETURN
END

```

```

*****
*****
* NAME:  LOADIT
* DATE:  11 NOV 83
* VERSION:  1.1
* INPUTS:  NONE
* OUTPUTS:  MAT
* COMMON BLOCKS USED:  CONST4, RMKEYW, FILES, RULCOM,
*                      START, TUPLEA, TUPLER, MISC,
*                      FLAGS
* CALLING SUBROUTINE:  DBLOAD
* SUBROUTINES CALLED:  ZEROIT, LOCATT, ATTGET, PARVAL,
*                      CHKTUP, PRULE, ADDDAT, RELPUT,
*                      BTADD
* DESCRIPTION:  HANDLES LOADING OF DATA VALUES INTO THE
*              RIM DATA BASE.
*              PARAMETERS:
*              MAT-----SCRATCH ARRAY FOR BUILDING TUPLES
* HISTORY:  THIS IS A RIM-5 SUBROUTINE WHICH I CONVERTED
*          FOR USE IN THIS APPLICATION.
*****
*****

```

SUBROUTINE LOADIT(MAT)

```

INCLUDE 'CONST4.BLK'
INCLUDE 'RMKEYW.BLK'
INCLUDE 'FILES.BLK'
INCLUDE 'RULCOM.BLK'
INCLUDE 'START.BLK'
INCLUDE 'TUPLEA.BLK'
INCLUDE 'TUPLER.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'FLAGS.BLK'

```

* DIMENSION STATEMENTS.

```

INTEGER COLUMN
LOGICAL EQKEYW
DOUBLE PRECISION DTEMPRM
REAL TEMP(2)
INTEGER ITEMP(2)
EQUIVALENCE (DTEMP,TEMP(1))
EQUIVALENCE (TEMP(1),ITEMP(1))
INTEGER ENDCOL
INTEGER MAT(1)

```

* READ A CARD.

```

100 CONTINUE
CALL LODREC

```

```

LSTCMD = K4LOA
ITEMS = LXITEM(IDUMMY)
IF(ITEMS.GT.2) GO TO 160
IF(EQKEYW(1,KWLOAD,4)) GO TO 5000
IF(ITEMS.GT.1) GO TO 160
IF(EQKEYW(1,KWEND,3)) GO TO 5000
160 CONTINUE

* ASSUME THIS IS A DATA CARD.

* ZERO OUT THE TUPLE.

CALL ZEROIT(MAT,MAXCOL)

* CHECK EACH ATTRIBUTE AND MOVE IT TO THE TUPLE FROM INPUT.

NUMKEY = 0
I = LOCATT(BLANK,NAME)
IF(I.NE.0) GO TO 5000
J = 1
ENDCOL = NCOL + 1
DO 1000 I=1,NATT
CALL ATTGET(ISTAT)
COLUMN = ATTCOL
IF(ATTKEY.NE.0) NUMKEY = NUMKEY + 1

* CALL PARVAL TO CRACK VALUE STRING

IF(ATTWDS.EQ.0) GO TO 200

* FIXED ATTRIBUTE

CALL PARVAL(J,MAT(COLUMN),ATYPE,ATTWDS,ATTCHA,0,IERR)
IF(IERR.NE.0) GO TO 100
GO TO 1000
200 CONTINUE

* VARIABLE ATTRIBUTE

MAT(COLUMN) = ENDCOL
NCOLT = ENDCOL + 1
CALL PARVAL(J,MAT(ENDCOL+2),ATYPE,ATTWDS,ATTCHA,NCOLT,IERR)
IF(IERR.NE.0) GO TO 100
MAT(ENDCOL) = ATTWDS
MAT(ENDCOL+1) = ATTCHA
ENDCOL = ENDCOL + ATTWDS + 2
1000 CONTINUE
ENDCOL = ENDCOL - 1

* SEE IF ALL APPLICABLE RULES ARE SATISFIED.

```

```

        IF(.NOT.RUCK) GO TO 1100
        IF(.NOT.RULES) GO TO 1100
        CALL CHKTUP(MAT,ISTAT)
        IF(ISTAT.EQ.0) GO TO 1100
        IF(ISTAT.LT.0) GO TO 1050
        WRITE(NOUT,1010)
1010  FORMAT(53H -ERROR- THE DATA FAILS TO SATISFY THE FOLLOWING RULE
        CALL PRULE(ISTAT)
        GO TO 100
1050  CONTINUE
        ISTAT = -ISTAT
        WRITE(NOUT,1060) ISTAT
1060  FORMAT(32H -ERROR- UNABLE TO PROCESS RULE ,I4)
        GO TO 100
1100  CONTINUE
        NTUPLE = NTUPLE + 1
        CALL ADDDAT(1,REND,MAT,ENDCOL)
        IF(RSTART.EQ.0) RSTART = RENDD
        CALL RELPUT

*  ALL DONE.

5000  CONTINUE
        RETURN
        END

```

```

*****
*****
* NAME: RIMLOK
* DATE: 23 NOV 83
* VERSION: 1.1
* INPUTS: MAT, INDEX, LENGTH
* OUTPUTS: MAT, IFLAG
* COMMON BLOCKS USED: RMATTS, CONST4, MISC, RIMCOM
*                      RIMPTR, WHCOM, START, BUFFER
*                      FLAGS
* CALLING SUBROUTINE: RESTOR
* SUBROUTINES CALLED: MOTSCN, GETDAT, ITOH, GETDAT, KOMPXX
* DESCRIPTION: LOCATES THE NEXT DESIRED TUPLE FROM THE
*              RELATION.
*      PARAMETERS:
*      MAT-----ARRAY TO HOLD ONE TUPLE
*                  IF(IFLAG.NE.0) MAT IS POINTER TO TUPLE
*                  IN INPUT BUFFER.
*      INDEX---PAGE BUFFER TO USE
*      IFLAG---0 IFF TUPLE IS RETURNED
*                  ELSE POINTER TO TUPLE IS RETURNED IN MAT
*      LENGTH--LENGTH OF TUPLE IN WORDS
* HISTORY: THIS IS AN ADAPTION OF RIM-5 SUBROUTINE RMLOOK.
*****
*****

```

```

SUBROUTINE RIMLOK(MAT, INDEX, IFLAG, LENGTH)

```

```

INCLUDE 'RMATTS.BLK'
INCLUDE 'CONST4.BLK'
INCLUDE 'MISC.BLK'
INCLUDE 'RIMCOM.BLK'
INCLUDE 'RIMPTR.BLK'
INCLUDE 'WHCOM.BLK'
INCLUDE 'START.BLK'
INCLUDE 'BUFFER.BLK'
INCLUDE 'FLAGS.BLK'

```

```

DIMENSION MAT(1)
LOGICAL QUAL, OK, BTEST
LOGICAL EQTEST

```

```

* SCAN MAT.

```

```

    RMSTAT = 0
    1 CONTINUE

```

```

* SEE IF WE ARE USING A KEY VALUE.

```

```

    IF(NS.EQ.0) GO TO 30

```

IF(NS.EQ.3) GO TO 10

* FIRST TIME THROUGH. USE BTLOOK TO FIND THE TUPLES.

```
START = KSTRT
NBOOX = IABS(NBOO)
NUMP = KOMPOS(NBOOX)
IF(KATTY(NBOOX).EQ.KZINT ) CALL BTLKI(WHRVAL(NUMP),NID,MID)
IF(KATTY(NBOOX).EQ.KZREAL) CALL BTLKR(WHRVAL(NUMP),NID,MID)
IF(KATTY(NBOOX).EQ.KZDOUB) CALL BTLKR(WHRVAL(NUMP),NID,MID)
IF(KATTY(NBOOX).EQ.KZTEXT) CALL BTLKT(WHRVAL(NUMP),NID,MID)
NS = 3
IF(NID.NE.0) GO TO 20
10 CONTINUE
IF(MID.EQ.0) GO TO 1300
CALL MOTSCN(MID,NID)
IF(NID.NE.0) GO TO 20
GO TO 10
20 CONTINUE
CID = NID
CALL GETDAT(INDEX,NID,ITUP,LENGTH)
GO TO 40
30 CONTINUE
IF(NID.EQ.0) GO TO 1300
CALL ITOH(N1,N2,NID)
IF(N2.EQ.0) GO TO 1300
CID = NID
CALL GETDAT(INDEX,NID,ITUP,LENGTH)
IF(NID.LT.0) GO TO 1300
```

* SCAN THROUGH EACH BOOLEAN CONDITION OF THE WHERE CLAUSE.

```
40 CONTINUE
IVAL = IVAL + 1
IF(NBOO.LE.0) GO TO 1200
IF(IVAL.GT.MAXTU) GO TO 1300
QUAL = .TRUE.
DO 1000 J=1,NBOO
ITYPE = KATTY(J)
IF(ITYPE.EQ.0)ITYPE = KZINT
OK = .FALSE.
CALL ITOH(NR,LEN,KATTL(J))
NUM = KOMLEN(J)
NK = KOMTYP(J)
NUMP = KOMPOS(J)
IP = ITUP + KATTP(J) - 1
IF(KATTP(J).NE.0) GO TO 100
```

* TUPLE NUMBERS

OK = .TRUE.

```

      IF(NK.EQ.2) OK = .FALSE.
      DO 80 JJ=1,NUM
      BTEST = .FALSE.
      CALL KOMPXX(IVAL,WHRVAL(JJ+NUMP-1),1,NK,BTEST,ITYPE)
      IF(NK.EQ.2) OK = OK .OR. BTEST
      IF(NK.NE.2) OK = OK .AND. BTEST
80    CONTINUE
      GO TO 900
100   CONTINUE
      IF(NK.LT.10) GO TO 300

```

* ATTRIBUTE - ATTRIBUTE COMPARISON

```

      KP = ITUP + NUMP - 1

```

* DUMMY TOLERANCE FOR ATTRIBUTE TO ATTRIBUTE

```

      TTOL = TOL
      TOL = 0.
      IF(LEN.NE.0) GO TO 120

```

* SET POINTER FOR VARIABLE ATTRIBUTES

```

      IP = BUFFER(IP) + ITUP - 1
      KP = BUFFER(KP) + ITUP - 1
      IF(NK.EQ.13) OK = .TRUE.
      LEN = BUFFER(IP)
      IF(BUFFER(KP).NE.BUFFER(IP)) GO TO 900
      IF(BUFFER(KP+1).NE.BUFFER(IP+1)) GO TO 900
      OK = .FALSE.
      IP = IP + 2
      KP = KP + 2
120   CONTINUE
      NK = NK - 10
      CALL KOMPXX(BUFFER(IP),BUFFER(KP),LEN,NK,OK,ITYPE)
      TOL = TTOL
      GO TO 900
300   CONTINUE
      IF(LEN.NE.0) GO TO 320

```

* SET POINTER FOR VARIABLE ATTRIBUTE

```

      IP = BUFFER(IP) + ITUP - 1
      LEN = BUFFER(IP)
      NR = BUFFER(IP+1)
      IP = IP + 2
320   CONTINUE

```

* REGULAR ATTRIBUTE

```

      NPOS = KOMPOS(J)

```

```

NPOT = KOMPOT(J)
OK = .TRUE.
EQTEST = .FALSE.
IF((NK.EQ.2).OR.(NK.EQ.9)) EQTEST = .TRUE.
IF(EQTEST) OK = .FALSE.
DO 400 JJ=1,NUM
BTEST = .FALSE.
CALL ITOH(NNR,NW,WHRLN(NPOT))
IF(NK.LE.1) GO TO 350
IF(BUFFER(IP).EQ.NULL) GO TO 350
IF((LEN.EQ.NW).AND.(NR.EQ.NNR)) GO TO 350

*   COMPARE OF DIFFERENT LENGTHS

IF(NK.EQ.9) GO TO 350
IF(NK.NE.3) GO TO 375
OK = .TRUE.
GO TO 900
350 CONTINUE
IF(NK.NE.9)CALL KOMPXX(BUFFER(IP),WHRVAL(NPOS),NW,NK,BTEST,ITYP)
IF(NK.NE.9) GO TO 375

*   CONTAINS

M1 = LSTRNG(BUFFER(IP),1,NR,WHRVAL(NPOS),1,NNR)
IF(M1.GT.0) BTEST = .TRUE.
375 CONTINUE
IF(EQTEST) OK = OK.OR.BTEST
IF(.NOT.EQTEST) OK = OK.AND.BTEST
IF(OK.AND.EQTEST) GO TO 900
NPOS = NPOS + NW
NPOT = NPOT + 1
400 CONTINUE
900 CONTINUE
IF(BOO(J).EQ.K4AND) QUAL = QUAL .AND. OK
IF(BOO(J).EQ.K4OR ) QUAL = QUAL .OR. OK
1000 CONTINUE
IF(.NOT.QUAL) GO TO 1

*   FOUND IT.

1200 CONTINUE
LIMVAL = LIMVAL + 1
IF(LIMVAL.GT.LIMTU) GO TO 1300
MAT(1) = ITUP
IP = ITUP
*   OPEN THE FILE
OPEN(UNIT=1,FILE='DMA1:[GATEWOOD]RELN99.JDG',
+STATUS='UNKNOWN')
I = 1
*   THE FIRST FIVE VALUES IN THE FILE ARE FOR RIM USE,

```

```
* AND CAN'T BEINTERPRETED BY GWCORE, SO GET RID OF THEM
  IP = IP+5
  DO WHILE (I.LE.LENGTH-5)
    WRITE(1,*) BUFFER(IP)
    IP = IP+1
    I = I+1
  END DO
  CLOSE(UNIT=1,STATUS='KEEP')
1250 CONTINUE
  RETURN
```

```
* END OF DATA.
```

```
1300 CONTINUE
  NS = 0
  RMSTAT = -1
  RETURN
END
```

```

*****
*****
* NAME:  LXLINE
* DATE:  15 NOV 83
* VERSION:  1.1
* INPUTS:  RECORD, LENREC, NUML, LEN, LOC
* OUTPUTS:  RECORD, LINE
* COMMON BLOCKS USED:  NAMPAS, END, LXCARD, PROM,
*                      LXCON
* CALLING SUBROUTINE:  LXLREC
* SUBROUTINES CALLED:  GETT
* DESCRIPTION:  THIS IS THE ROUTINE WHICH GETS THE NEXT
*               LINE FOR LXLREC TO PARSE.  IF LENREC IS
*               NIN IS READ.
* HISTORY:  THIS IS A RIM-5 SUBROUTINE WHICH I CONVERTED
*           FOR USE IN THIS APPLICATION
*****
*****

```

```

SUBROUTINE LXLINE(RECORD,LENREC,NUML,LINE,LEN,LOC)

```

```

COMMON /NAMPAS/ RELATE,ATTRIB,FLNM,COMND,ABSENT
REAL*8 RELATE,ATTRIB
CHARACTER*3 COMND
CHARACTER*8 FLNM
LOGICAL ABSENT

```

```

COMMON /END/ TMPEND, TMPBUF
INTEGER*4 TMPEND, TMPBUF(30000)

```

```

INCLUDE 'LXCARD.BLK'
INCLUDE 'PROM.BLK'
INCLUDE 'LXCON.BLK'
DIMENSION LINE(80)
INTEGER RECORD(1)

```

```

IF (COMND.EQ.'STO') THEN
  NIN = 20
ELSE IF (COMND.EQ.'RES') THEN
  NIN = 17
ELSE
  NIN = 21
END IF
8 IF(LOC.NE.0) GO TO 200
  NUML = NUML + 1
  IF(LENREC.NE.0) GO TO 100

```

```

* FROM FILE NIN

```

```

LEN = 80

```

```

      READ (NIN,10,END=13) LINE
10  FORMAT(80A1)
      LXEOF = .FALSE.
      GO TO 14
13  CONTINUE
      LXEOF = .TRUE.
14  CONTINUE

      IF(LXEOF) GO TO 1000
      IF(NOUT.EQ.0) GO TO 1000
      IF(.NOT.ECHO) GO TO 1000
      GO TO 1000
100 CONTINUE

*   GET LINE FROM RECORD

      LEN = 0
      I1 = 80*(NUML-1) + 1
      I2 = 80*NUML
      IF(I1.GT.LENREC) GO TO 1000
      IF(I2.GT.LENREC) I2 = LENREC
      DO 150 I=I1,I2
      LEN = LEN + 1
      CALL GETT(RECORD,I,LINE(LEN))
150 CONTINUE
      GO TO 1000
200 CONTINUE
      NEWLEN = LEN - LOC
      IF(NEWLEN.LE.0) GO TO 230
      DO 220 I=1,NEWLEN
      LOC = LOC + 1
      LINE(I) = LINE(LOC)
220 CONTINUE
230 CONTINUE
      LEN = NEWLEN
      LOC = 0
1000 CONTINUE
      IF(LEN.LE.0) RETURN

*   IGNORE TRAILING BLANKS

      ICHECK = LEN + 1
      DO 1100 I=1,LEN
      ICHECK = ICHECK - 1
      IF(LINE(ICHECK).NE.BLANKS) GO TO 1200
1100 CONTINUE
      ICHECK = 1
1200 CONTINUE
      LEN = ICHECK
      RETURN
      END

```

Appendix I

User's Guide

Introduction

This user's guide is designed to make the use of INGRED II as easy as possible. Section I of the guide is aimed at the user who is unfamiliar with the INGRED II. The commands are listed separately starting on page I-10, and can be detached from this document for use at the graphics terminal. Section II is a condensed RIM-5 User's Manual. This section was considered necessary because this is the first time that RIM-5 has been used at AFIT. Section III is designed to help the programmer use INGRED II, the Relational Information Management System (RIM), and the George Washington University Core (GWCORE) more effectively.

Section I

Background

INGRED II brings together the capabilities of two completely unrelated software systems, the GWCORE and the RIM DBMS. GWCORE provides a standardized method of dealing with graphics displays. RIM, on the other hand, is a relational DBMS which provides a means of storing and manipulating data. The capabilities of these two systems complement each other in INGRED II.

To provide a user interface to GWCORE, the interactive graphics editor (INGRED) was written. INGRED gave AFIT and other WPAFB units the capability to create, change, store, and retrieve graphics displays. A user's manual exists for INGRED, and much of the information in this INGRED II User's Manual is taken from that source.

Editing capabilities under INGRED II ARE EXACTLY AS THEY WERE UNDER INGRED. The enhancement provided by INGRED II is in the area of saving or restoring a display. In addition, INGRED II also adds the capability to easily purge existing files from disk, or obtain a listing of all displays currently stored on disk.

INGRED II

INGRED II contains an interactive graphics editor, INGRED, which allows creation and editing of two-dimensional

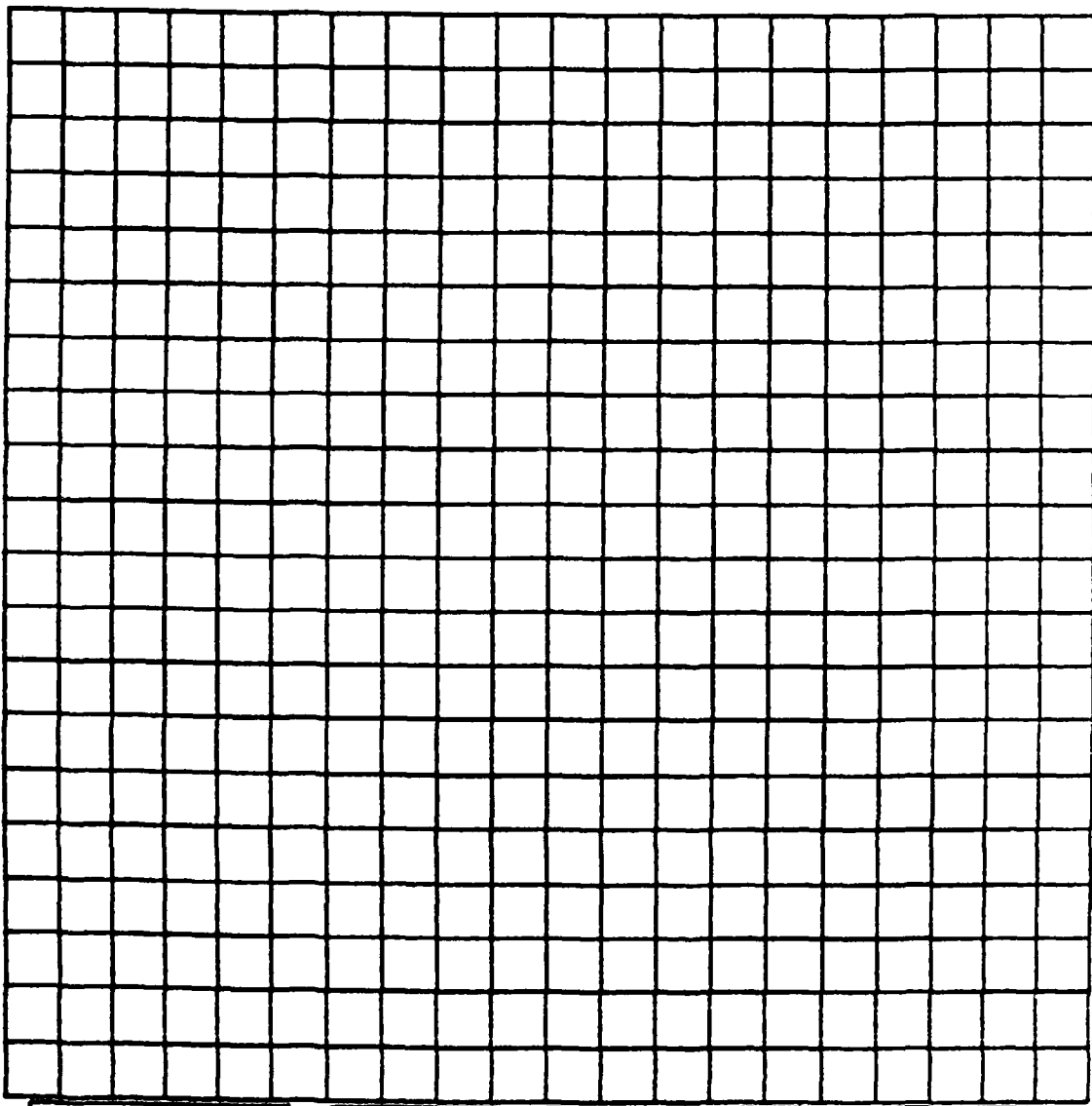
figures. The system allows creation of the pictures on the Tektronix 4014 for display on any graphics device that has a suitable driver for the CORE graphics software library. The display capabilities are not limited to the 4014. For example, color may be specified, and can be displayed on the ISC 8001 color monitor.

INGRED II is resident on the AFIT VAX 11/780 AFITUSER disk pack. To run INGRED II, set the default directory to DMA1:[GATEWOOD], then "RUN INGREDII." If the text characters on the Tektronix 4014 are larger than desired, push the local/line button (in the top left corner) to local, then press the escape button, then type semi-colon (;), and then return. This will provide the alternate small character font of the Tektronix 4014. INGRED II uses [GATEWOOD.PIKFILE] for storing and retrieving displays in the data file. It uses files GRAFDB1.DAT;1, GRAFDB2.DAT;1, GRAFDB3.DAT;1, and SCHEMA.DAT to store and retrieve additional information about each display in the data base.

Using the Graphics Editing Capabilities

The following discussion describes inputs and responses when using INGRED II. The responses illustrated are of actual hardcopies taken from the Tektronix display. When the program is running (RUN INGREDII), a grid is provided which serves as a positioning aid. Options available to the user are also illustrated along the left side of the screen (see Figure I-1). These options provide the capability to draw lines, arrows, boxes, circles and to write text; to choose the font, line color, text color, and polygon fill color; to preview, save, retrieve, or erase the display; to list files, store, restore, or purge a display; to delete a segment, change the grid appearance, get help, or quit the session. The 'COMMAND:' prompt appears below the grid along with a flashing cursor. The cursor signifies that INGRED II is waiting for the user to type in a command.

Figure I-2 shows that two commands have been typed, namely, 'FONT' and 'TEX'. The first command, 'FONT', results in the secondary prompt 'FONT TYPE? (1-4):'. The user must respond with a single integer in the range indicated. For commands, the command is entered by pressing the return key (this allows correction of mistakes prior to return by using the rub out key), while input for single integers like the font type, require only pressing the key to return to the command level. The second command, 'TEX', is the abbreviated form of the 'TEXT' command. Any command will be accepted by the system if the first three letters are provided. Entering 'TEX' produces the graphics crosshair cursor. The user positions the crosshairs with the thumbwheel controls on the Tektronix keyboard. Once the crosshairs are positioned, pressing any key will return the



FIGURES
LINE
ARROW
BOX
FBOX
CIRCLE
FCIRCLE
TEXT

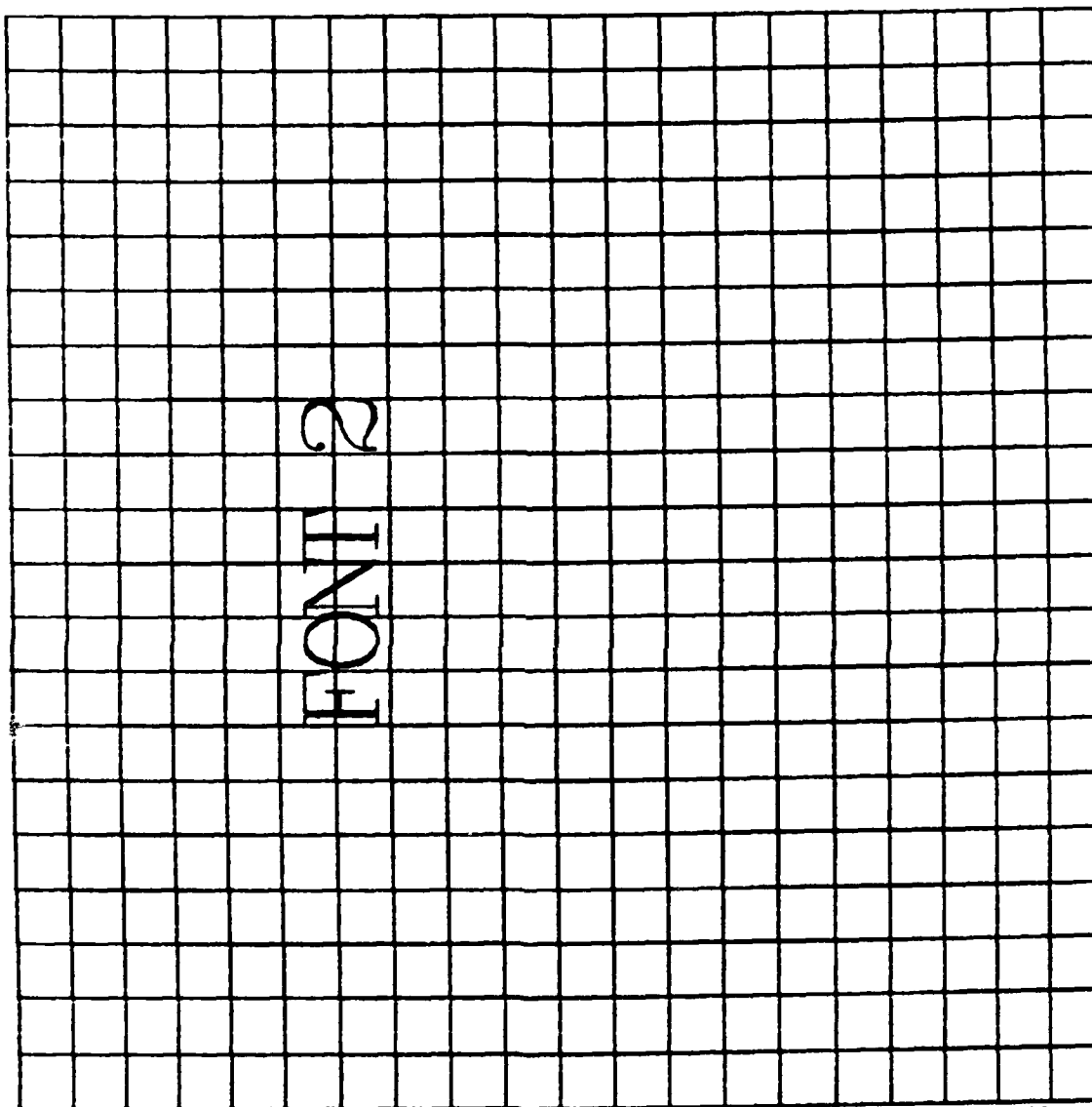
ATTRIBUTES
FONT
LCOLOR
TCOLOR
FCOLOR

CONTROL
PREVIEW
SAVE
RETRIEVE
ERASE
QUIT
DELETE
GRID

DATA BASE
STORE
RESTORE
DIRECTRY
PURGE

HELP

CONTINUED



FIGURES

- LINE
- ARROW
- BOX
- FBOX
- CIRCLE
- FCIRCLE
- TEXT

ATTRIBUTES

- FONT
- LCOLOR
- TCOLOR
- FCOLOR

CONTROL

- PREVIEW
- SAVE
- RESTORE
- ERASE
- QUIT
- DELETE
- GRID

DATA BASE

- DIRECTRY
- PURGE

HELP

COMMAND: FONT TEX

block cursor at the point previously marked by the graphics crosshairs. The user then inputs the text string desired. The text is displayed in the Tektronix font style. The RUBOUT key is used to delete erroneous input. The string is transmitted with the return key. The string is then redrawn on the screen with the current font as selected. The fonts available are illustrated in Figure I-2.

Figure I-3 illustrates the use of several options. 'LCOL', the line-color command is ignored by the Tektronix 4014 since it does not have capability for color presentation. However, if the picture is saved and redrawn using a color terminal, the line appears in the same location as it did on the Tektronix, but with the correct color. The second command in this example is 'LINE'. This command is used to draw a line on the screen. With 'LINE', the crosshairs must be positioned twice (once for each end of the line). The third command, 'BOX' produces a box when the location of the opposite corners of the box are selected by positioning and selecting the crosshairs. The final command shown is 'GRID' which has no immediate feedback. However, if 'PREVIEW', 'DELETE', or 'ERASE' is entered, when the new grid is drawn, the grid appears as a collection of small crosses as shown in Figure I-4. Preview redraws the current picture without the grid, prompts, or error messages. 'PREVIEW' should be entered periodically to "clean up" the screen.

Figure I-4 shows the alternate grid style. This figure also illustrates an effective way to draw lines to link figures in pictures generated under INGRED II. Since the 'ARROW' command as currently implemented only provides the capability to have the arrows point north, south, east, or west, it is best to have connecting lines exactly vertical or exactly horizontal. To do this most efficiently, make the lines appear as they do in Figure 4. One characteristic of the editor helps do this. When the first portion of the line has been completed, and the second 'LINE' command is entered, the cursor will reappear exactly at the ending point of the first line. This means that you do not have to meticulously place the cursor to have the two portions of the lines meet. This characteristic also helps place the arrows exactly where they belong at the end of lines. The 'CIRCLE' command produces the crosshairs, and the center and any point on the circumference of the desired circle can be selected. One caution on systems which have storage space problems. The circle is made up of many small individual lines, and as such, requires many internal opcodes to draw, thus requiring a significant amount of space to store. Therefore, it is best to use boxes whenever possible instead of circles.

Figure I-5 illustrates another capability which can help improve the speed with which pictures are generated by

FIGURES

LINE
ARROW
BOX
FBOX
CIRCLE
FCIRCLE
TEXT

ATTRIBUTES

FONT
LCOLOR
TCOLOR
FCOLOR

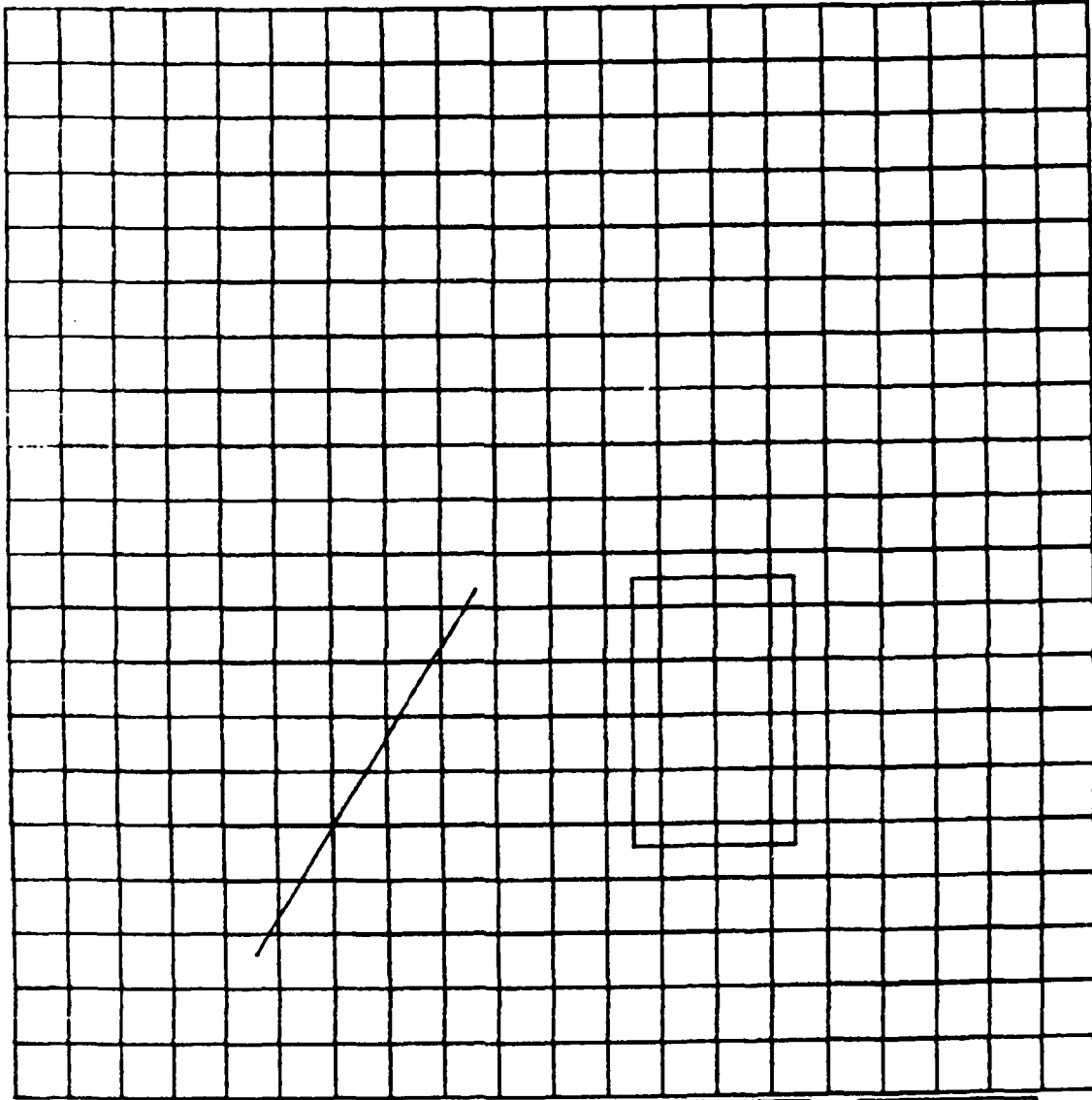
CONTROL

PREVIEW
SAVE
RESTORE
ERASE
QUIT
DELETE
GRID

DATA BASE

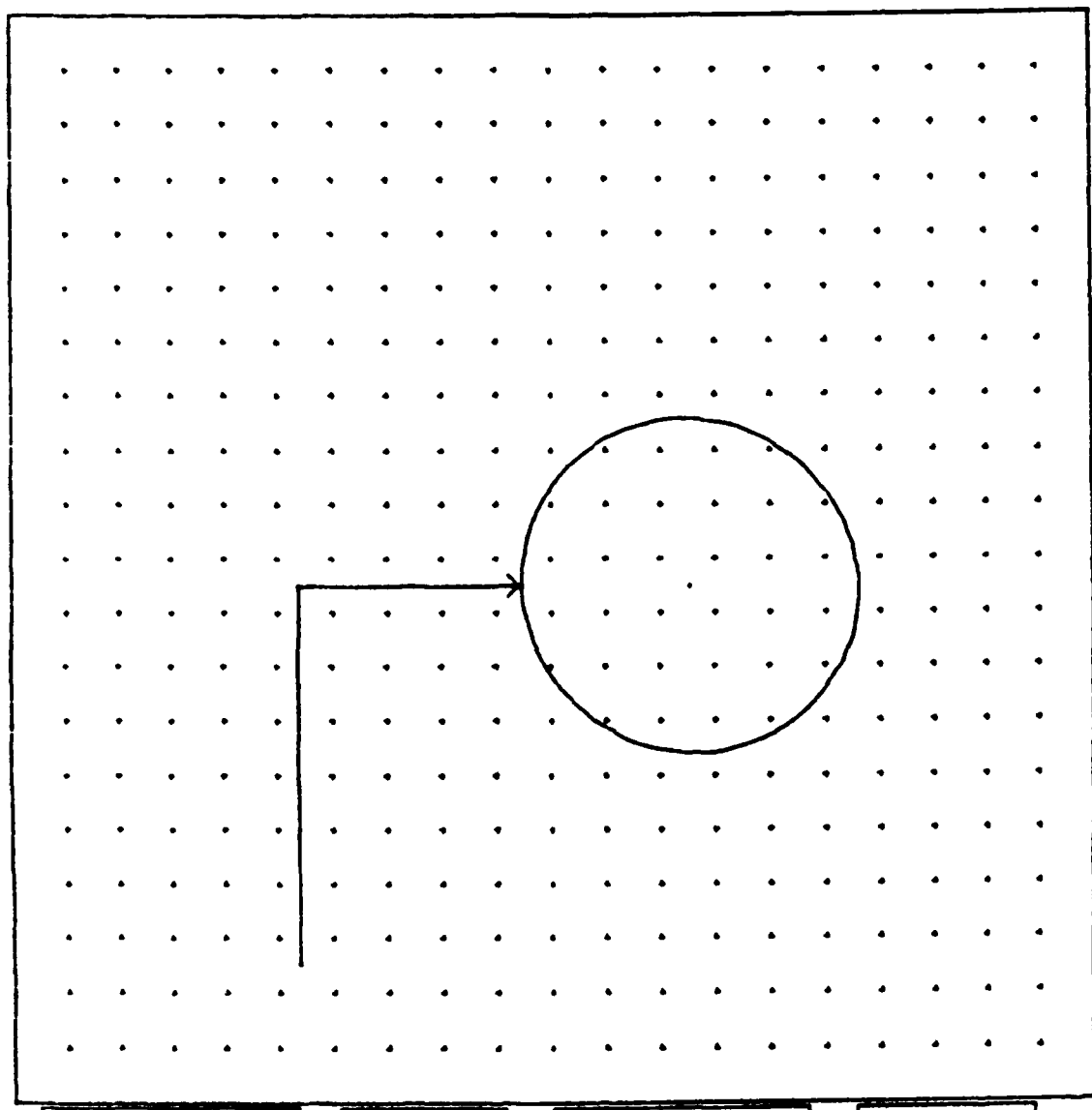
DIRECTRY
PURGE

HELP



COMMANDS: LCOL LINE BOX GRID

LINE INDEX? COLOR: 1..8: 6



FIGURES

- LINE
- ARROW
- BOX
- FBON
- CIRCLE
- FCIRCLE
- TEXT

ATTRIBS

- FONT
- LCOLOR
- TCOLOR
- FCOLOR

CONTROL

- PREVIEW
- SAVE
- RESTORE
- ERASE
- QUIT
- DELETE
- GRID

DATA BASE

- DIRECTRY
- PURGE

HELP

COMMAND: LINE LINE ARROWCIR

DIRECTION? (N..S..E..U): S

AUTHOR:	DATE:	READER:
PROJECT:	REV:	DATE:
NODE:	TITLE:	NUMBER:

INGRED II. "Skeletons" of pictures can be stored and by using the 'RESTORE' or 'RETRIEVE' command the "skeletons" can be returned to the screen to preclude having to regenerate a portion of the picture each time it is used. This approach can work very well for SADT and Structure Chart type diagramming.

All INGRES II commands are discussed in the following pages which were designed to be detached and used for quick reference at the graphics terminal.

INGRED II COMMANDS

The commands are discussed in the order they are grouped, by function, when seen on the graphics display. That is, first FIGURES will be covered, then ATTRIBUTES, CONTROL, DATA BASE, and finally, HELP. Any command can be shortened to the first three letters, and INGRED II will recognize the command.

FIGURES

FIGURES represent the available objects or segments that can be drawn on the graphics display under user control. Any FIGURE command can be aborted when the crosshairs are on the screen by typing an 'A' instead of using the space bar to select an endpoint.

LINES

Use the LINE command to produce a line anywhere within the grid. Invocation of the command produces a crosshair which can be moved by use of the thumbwheel controls on the right hand side of the Tektronix 4014 keyboard. Move the crosshair to the location where you want one end of the line to start and press the return key. The crosshairs will then reappear and you can choose the other line endpoint in the same way as the first. The line will then be drawn on the screen. NOTE: Lines of any slope or length (within the limits of the grid) can be drawn.

BOX

The BOX command is used to produce a square or rectangle. A crosshair will appear, and should be moved to the location where you want one corner of the box. Select that point by pressing the return key and a small cross will denote that corner. Next move the cursor to where the opposite corner of the box will be and press return. The box will now be drawn.

CIRCLE

The CIRCLE command is used to produce a circle of any radius (up to the limits of the grid) on the screen. To produce a circle, move the crosshair to the point where the center of the circle will be

and press the return key. A small cross will appear at that point. The crosshair will then reappear and can be moved to any point on the circumference of the circle. With the crosshair at this point, press the return key and the circle will be drawn. NOTE: Drawing a circle takes many internal commands, and thus requires a lot of space when stored to disk. Thus when a figure will be saved, substitute boxes for circles when possible.

TEXT

The TEXT command is invoked to print letters, words, and sentences on the screen. To properly locate the text on the screen, move the crosshair to the location where you want the first letter to be printed. Select the point by pressing the return key and then type in the text as desired. The RUBOUT key can be used to delete mistyped letters. Type in the correct letter and, although it will appear as an overstrike, the correction holds. To see the way the screen will really appear after a text correction just PREVIEW the screen, and the text will be redrawn. If the font attribute is other than the Tektronix default, the echoed text will be different from the final display text (see FONT under ATTRIBUTES).

ATTRIBUTES

The attributes are used to control the appearance and quality of the figures displayed on the screen. All attributes may be input by using the first three letters of the desired command. HELP is available for each attribute so you can be prompted from the screen on what the capabilities are.

The color attribute is an integer value which applies to LCOLOR, FCOLOR, and TCOLOR. The default value for all ATTRIBUTES is white. The values for the color indices are as follows:

<u>ATTRIBUTE</u>	<u>VALUE</u>
BLACK	1
RED	2
GREEN	3
BLUE	4
CYAN	5
YELLOW	6
MAGENTA	7
WHITE	8

LCOLOR

LCOLOR sets the color attribute for lines. Based on the integer values listed on the previous page, the color of lines produced can be specified.

FCOLOR

FCOLOR sets the color attribute for polygons. Based on the integer values listed on the previous page, the color of polygons produced can be specified.

TCOLOR

TCOLOR sets the color attribute for text. Based on the integer values listed on the previous page, the color of the text produced can be specified.

FONT

The FONT command is used to change the size and type of letters produced on the screen. Only the Tektronix hardware font is proportionally spaced. In response to the FONT command the prompt 'FONT TYPE? (INTEGER 0-4):' is displayed. Your response will be based on the following:

- 0 Tektronix hardware font (default)
- 1 CORE software font - MODERN
- 2 CORE software font - ROMAN
- 3 CORE software font - ROMAN ITALICS
- 4 CORE software font - BLOCK

CONTROL

The CONTROL commands provide the user with the means to manipulate the pictures generated by INGRED II. If you stay completely with capabilities within the CONTROL environment and avoid using the DATA BASE capabilities of INGRED II, you will have the capabilities as originally implemented under the INGRED editor. CONTROL command options include: PREVIEW, SAVE, RETRIEVE, ERASE, QUIT, DELETE, and GRID. DEFER and NODEFER are not implemented at this time. All control commands may be abbreviated to the first three letters. HELP information is available on individual CONTROL commands.

PREVIEW

The PREVIEW command provides a method to display the current picture without the positioning grid or command prompts. It can also be used to clear the screen of extraneous input and error messages. Invoke the PREVIEW command before copying pictures from the screen to the Tektronix copier. The space bar will return the user to the editing mode from PREVIEW.

SAVE

The SAVE command allows you to place the current picture on disk (in the data file) for future use. With SAVE, a prompt will ask for the file name you want to store the display under (up to eight characters can be entered). If saved in the data file, you will be using the capabilities as originally implemented by INGRED. This method is fast and effective, but file information and purge of disk files is difficult. The capabilities when the display is stored in the data base will be discussed later under DATA BASE.

RETRIEVE

RETRIEVE allows you to obtain a previously saved picture file from disk for further manipulation. The RETRIEVE command deletes the current picture information from the display file before reading in the new picture. Only files saved using the data files (SAVE) can be restored using the RETRIEVE command.

ERASE

The ERASE command allows you to erase the picture from the screen. When this is done, all information about that picture is lost.

DELETE

The DELETE command provides a method to eliminate individual segments from the display. To do this, move the crosshair within the figure to be deleted and then press the return key. In the case of TEXT, place the crosshair in the middle of the first character. After the return key is pressed, the editor will prompt by:

1. Redrawing the selected figure.
2. Prompting with 'DELETE THIS SEGMENT?'.
3. Then waiting for YES/NO confirmation. If 'N' is selected, the crosshair will reappear and may be repositioned. If 'Y' is selected, the segment will be deleted and the figure will be redrawn.

DEFER and NODEFER (not presently implemented)

DEFER and NODEFER control the updating of the screen in response to deletions. If many deletions are to be accomplished, then the DEFER mode prevents the screen from erasing and redrawing the segments until the last deletion occurs.

QUIT

Quit terminates the interactive editor session. The information in the display file is lost unless previously saved.

GRID

The GRID command changes the positioning grid on the display surface to a set of points to mark the intersections of the default grid. You may find it easier to generate pictures with the GRID configured in this manner. To change from one grid type to the other just type 'GRID'. The most recently selected grid will not appear until a DELETE, ERASE, or a PREVIEW command. The default is the rectilinear grid.

DATA BASE

The DATA BASE instructions allow you to manipulate the picture file which was generated under the interactive editor. The options available include: DIRECTORY, PURGE, STORE, and RESTORE. Any of the DATA BASE commands can be activated by typing in only the first three letters.

STORE

The STORE command allows you to place the current picture on disk for future use. With STORE, a prompt will ask you for the desired file name (up

to eight letters). The choice to store in the data base will give you flexibility in purging unneeded displays from the disk file and also in obtaining a listing of the displays available on disk (see commands above).

RESTORE

RESTORE allows you to retrieve a previously saved picture file from disk for further manipulation. The RESTORE command deletes the current picture information from the display file before reading in the new picture.

DIRECTORY

The DIRECTORY command provides a means for you to obtain a listing of displays stored on disk in the data base. Listings of displays in the data file are not available.

PURGE

The PURGE command allows you to eliminate a display which is stored in the data base. You will be prompted for the name of the display to be deleted. If you do not remember the name of the display to be purged, you can invoke the DIRECTORY command to receive a listing of all display files stored in the data base. There is no purge capability for displays stored in the data file.

Section II

RIM-5

Background

The Relational Information Management System Version 5 (RIM-5) relational DBMS was published in February 1982. It was written by Boeing Commercial Airplane Company under contract to the US government [NASA contract NAS1-14700 (IPAD)]. Most of the information in this user's guide was taken from USER GUIDE: RIM 5.0 VAX VMS published by Boeing. For clarification of instructions, reference will be made to pages in that document.

RIM-5 is written entirely in FORTRAN 77, but is accessible by any language which can make calls to FORTRAN 77 subroutines. It can be run on the VAX 11/780 using the VMS operating system. RIM-5 is based on the relational algebra model for data management. It can be accessed via either the standalone mode or the application program interface; both will be discussed below.

Standalone Mode

The standalone mode of RIM-5 is how the everyday user will access and use the data base. It provides the capability for an interactive session to create, update, and query a data base. The standalone system can be operated using either the command or menu modes. The same functions can be performed using either mode, but the menu approach provides prompts to help the user process the data base information. It is more valuable to the new user, and will probably give way to use of the command mode as the user becomes more familiar with RIM-5.

To access the standalone capability of RIM-5, simply enter RUN RIM. Make sure you are in the proper directory when this command is entered. If you want to query a data base that already exists, make sure you are logged into the directory which contains that data base. For instance, if you have the RIM-5 executable file (RIM.EXE) on user directory [GATEWOOD.CORE] and the data base you want to query is on directory [GATEWOOD], one way to properly run the program is:

```
SET DEF DMA1:[GATEWOOD]
```

```
RUN [GATEWOOD.CORE]RIM
```

This way RIM will look for the data base in the logged-in directory and will make any updates to that directory. If you check the directory, you can see that each data base is made up of four files. If the name of the data base created is GRAFDB, then the three files will be called GRAFDB1.DAT, GRAFDB2.DAT, and GRAFDB3.DAT. These contain respectively, the data base definition (schema), the actual

data for each relation, and the keyed data. SCHEMA.DAT contains additional data base definition information.

Upon executing RIM-5 you will automatically be placed in the Command mode. You will see:

RIM COMMAND MODE

ENTER "MENU" FOR MENU MODE

Procedures for the menu mode will be discussed here since this will be the most effective mode for the new user. In response to your entry of MENU, RIM will prompt you with the following:

SELECT THE EXECUTION OPTION DESIRED

- 1) CREATE A NEW DATA BASE
- 2) UPDATE AN EXISTING DATA BASE
- 3) QUERY AN EXISTING DATA BASE
- 4) ENTER COMMAND MODE
- 5) EXIT

Any action which can be performed by RIM (including exiting the program) can be executed by selecting one of these five options. If you enter a 5 at the terminal, your RIM-5 session will be terminated, and you will receive the \$ prompt back from the VMS operating system and can go from there.

From this point on, computer supplied information will appear in capital letters and user response in lower case letters. Page numbers to reference in the RIM user's guide are set off to the right within square brackets

To build a new data base, enter a 1. The following is an example of how the session might go from there:

```
ENTER THE NAME OF THE DATABASE
R> grafdb [44]
ENTER THE NAME OF THE DATABASE OWNER
R> Lamont [44]
ENTER THE NAME ASSIGNED TO THIS RELATION
R> Graphics [44]
ENTER THE READ PASSWORD FOR THIS RELATION
R> ee793 [44]
ENTER THE MODIFY PASSWORD FOR THIS RELATION
R> ee793 [44]
ENTER THE ATTRIBUTES OF THIS RELATION
ENTER END WHEN COMPLETE
      NAME TYPE LENGTH(IF>1)      "KEY"(IF KEY)
R> File Text 8 [44]
R> Age Integer 3
R> Display imat 1,var
R> end
DO YOU HAVE ADDITIONAL RELATIONS TO DEFINE - Y OR N
R> n [45]
BEGIN RIM SCHEMA COMPILATION
RIM SCHEMA COMPILATION FOR GRAFDB IS COMPLETE
DO YOU WANT TO LOAD THE DATABASE - Y OR N
```

```

R> y [46]
SELECT THE RELATION TO BE LOADED
    1) GRAPHICS
R> 1 [46]
ENTER THE MODIFY PASSWORD FOR THIS RELATION
R> ee793 [46]
ENTER THE ATTRIBUTE VALUES IN THE SPECIFIED SEQUENCE
ENTER END WHEN COMPLETE
BEGIN DATA LOADING
FILE AGE DISPLAY
R> picture 31 ((3) (21) (23456) (10)) [46,13]
R> sky 3 ((1) (311))
R> end
END RIM DATA LOADING
DO YOU HAVE ADDITIONAL RELATIONS TO LOAD - Y OR N
R> n [46]
THE GRAFDB DATABASE HAS BEEN CREATED/UPDATED
DO YOU WANT TO QUERY THE DATABASE AT THIS TIME - Y OR N
R> y [46]
RIM COMMAND MODE
listrel all [21]
      RELATION : GRAPHICS
LAST MOD : 83/11/28      READ PASSWORD : NONE
SCHEMA : GRAFDB         MODIFY PASSWORD : NONE
      NAME      TYPE      LENGTH      KEY
      FILE      TEXT      8 CHARACTERS
      AGE       INTEGER    3
      DISPLAY   IMAT       1,VARIABLE

R> exit
SELECT THE EXECUTION OPTION DESIRED
    1) CREATE A NEW DATABASE
    2) UPDATE AN EXISTING DATABASE
    3) QUERY AN EXISTING DATABASE
    4) ENTER COMMAND MODE
    5) EXIT
5

```

The 5 takes you out of RIM and back to the VMS operating system.

After this first session, if you want to update the existing data base, select option 2 and you will then be prompted for the name of the data base. Then RIM-5 will provide the following prompt:

```

SELECT THE UPDATE OPTION DESIRED
    1) DEFINE ADDITIONAL RELATIONS
    2) LOAD ADDITIONAL DATA

```

Entering a 1 or a 2 will allow you to define relations or load data into the relations just as depicted in the initial example.

To query a data base select a 3 from the menu and you will again be prompted for the name of the data base. The query option provides you many ways to manipulate the data base information. These are covered on pages 14 - 22 of the RIM-5 user's guide. Several of the options are discussed below:

SELECT

Select is used to display or print data from a relation. To print a listing of all file names from the graphics relation which was created earlier do:

select file from graphics

The response is:

PICTURE

SKY

"Where" conditions can also be used with the select option. To print all tuple information where age is greater than 10 do:

select all from graphics where age gt 10

The response is:

PICTURE 31 ((3) (21) (23456) (10))

Application Program Interface

Using the Application Program Interface

RIMLIB.OLB (contained on the AFITUSER disk under directory [GATEWOOD.CORE], or on the RIM-5 tape number HD31320125124) the application program interface. It is also the executable file accessed when the RIM standalone capability is used. A RIM data base can be modified and accessed by any programming language which can call FORTRAN 77 subroutines (i.e. Pascal, C, FORTRAN IV, etc.) The procedures for doing this are contained in Chapter 3 of the RIM-5 User's Guide, pages 47-60. Further explanation is contained in Appendices A, B, C, and D of the guide. A short explanation follows.

If you decide to use the application program interface, you will have to manage the data base files. File 1 contains the data base definition (schema) data, file 2 contains the actual data for each relation, and file 3 contains the pointers for the "keyed" attributes [2,3]. These are files GRAFDB1, GRAFDB2, and GRAFDB3 if you use GRAFDB as the data base name. You must save the data base files after execution of your program if you want to retain changes that were made.

Variable RMSTAT contains the status of the current data base activity. This variable is declared in the common block RIMCOM. A zero indicates the operation was successful, and any other number [2,48] indicates a specific error. RMSTAT should be checked after each operation since if an operation is not successful, subsequent operations may not work as expected.

Several routines are accessed when using the application program interface. They are covered in detail in reference 2 page 47 - 60, but will be discussed here also.

RMOPEN

This routine must be called to access the required data base. It initializes the internal tables used by RIM and opens the appropriate data base. RMOPEN will be the first routine called whenever you are dealing with the RIM data base via the application program interface.

RMCLOSE

This routine must be called if you want to save information in a data base which has been modified. It is also required before you can access another data base. It closes the current data base.

RMLREL

This routine sets an implicit pointer to the first relation in the data base that your password allows access to.

RMGREL

RMGREL returns the data about the current relation, and increments the pointer to point to the next relation you have permission to access. Thus RMLREL must be called before RMGREL.

RMLATT

This routine sets the implied pointer to the first attribute of the specified relation. A call to RMLREL and/or RMGREL must precede the call to RMLATT.

RMGATT

RMGATT returns the data about the current attribute and increments the attribute pointer to point to

the next attribute. RMLATT must be called before RMGATT. It is important to note that RMSTAT is set to -1 when all attributes of the relation have been found. This can be used in listing all attributes of a relation for instance.

RMFIND

This routine establishes an implied pointer chain to all rows of a relation in the internal order of the rows. You can have up to 6 pointers at one time. They are numbered 0-5. A new pointer is established each time RMFIND is called.

RMWHER

RMWHER qualifies a set of rows for retrieval. It is called to perform the functions of a normal "where clause" in data base manipulation. A call to RMWHER must be preceded by a call to RMFIND.

RMSORT

This routine sorts data prior to retrieval. Calls to RMSORT must be preceded by a call to RMFIND.

RMGET

This routine gets a row of data from the specified relation and advances the pointer to the next row. Which row is next is determined by RMFIND or RMWHER, or RMSORT, or a combination of the three. Again, RMSTAT is set to -1 when all qualified rows have been retrieved.

RMLOAD

This routine loads a row of data into the specified relation. Calls to RMLOAD must be preceded by a call to RMFIND.

RMPUT

RMPUT modifies a row of data in the specified relation. It must be preceded by a call to RMGET.

RMDEL

RMDEL deletes a row of data in the specified relation. It must be preceded by a call to RMGET.

Section III

Programmer's Manual

Background

This manual is designed for individuals with knowledge in data base and graphics techniques who want to improve the INGRED II system through software enhancement. To make changes to INGRED II will require knowledge in both graphics and data base techniques. there are several important documents which will help:

1. Design Document for the George Washington University Implementation of the 1979 GSPC Core System (Dr. Lamont has a copy.)
2. User Guide: RIM 5.0 VAX VMS (all data base instructors have a copy.)
3. GWCORE source code documentation (located in the red binder labelled GWCORE in the storage cabinet near the DEL VAX.)
4. RIM-5 source code (located in the yellow binder in the storage cabinet near the DEL VAX.)
5. Appendix C of this thesis (discusses GWCORE and RIM data structures.)
6. This Programmer's guide.

GWCORE

The GWCORE design document reflects that the heart of the GWCORE is the 68 Opcodes which perform all Core operations. Each opcode performs a specific function, for example, opcode 3 is used to draw a line, opcode 7 for writing text, opcode 14 for deleting segments, and opcode 31 for setting line, fill, and text colors. The opcodes are covered in detail in pages 23-40 of the design document.

All opcodes used in creating a particular display are stored in the pseudo display file (PDF). This PDF plus an integer which reflects the number of PDF entries make up the information packet which is stored on disk.

Detailed information on all capabilities of the GWCORE is contained in the documentation included with the source code. As stated above, this documentation is included in the red binder labelled GWCORE in the storage cabinet near the DEL VAX. All source code is also included in the cabinet.

RIM-5

RIM-5 can be used in the standalone mode or by access through the applications program interface. The standalone mode, as delivered, requires that the user process interactively and load parameters via the keyboard. This is not feasible with most graphics applications. The capability for INGRED II to store display information in the data base was, however, an adaptation of the standalone capability of RIM-5.

Pages 57 and 58 of the RIM-5 User's Guide explain the RIM data structure. Figure 3.5.1 on page 59 of the guide shows graphically what is explained on pages 57 and 58. Understanding this data structure is the most important element of successfully programming using the RIM-5 application program interface.

To change the INGRED II capabilities will require access to the following source code and libraries:

```
RIMLIB.OLB      (in dir [GATEWOOD.CORE])
GWCORELIB.OLB   (in dir [GATEWOOD.CORE])
INGRED II       (in dir [GATEWOOD])
STRING.BLK      (in dir [GATEWOOD.CORE])
    -this file contains the values used for the
      default comparison of Opcode 55
all RIM '.BLK' files (mag tape HD31320125124)
    -the tape is labelled RIM VERSION 5 Oct 83
    -the '.BLK' files contain common block
      declarations used by RIM along with some data
      initializations
all RIM Source Code (mag tape HD31320125124)
    -the tape is labelled RIM VERSION 5 Oct 83
    -in addition to the '.BLK' files, the tape
      contains the main RIM-5 program and all
      subroutines. It does not contain the executable
      version of RIM, so must be compiled and linked
      before it can be used. There is a help file to
      assist in this.
```

INGRED II

The details of the INGRED II application program are discussed in Chapter IV of this thesis. A short recap at this time may prove beneficial.

Data base GRAFDB which was used for INGRED II was created using the RIM-5 standalone capability. This was an easy method to create the data base quickly and accurately, and know that it could be accessed by the application interface program. It is also possible to create the data base using your application program.

The INGRED II routines which load the data base are variations of the RIM standalone program. Using this approach will force you to learn a lot about the RIM methodology in a short time. It also means that all RIM capabilities are available unless you consciously decide to omit them. This allows a lot of flexibility in the use of any applications programs. Again, this loading can be done strictly using your application programs along with the RIM application program interface if you prefer.

INGRED II brings together the capabilities of the GWCORE and RIM systems through manipulation of the information packet data structure. Thus, if changes are made to either GWCORE or RIM-5, as long as the data structures required remain the same, INGRED II will still operate as effectively as with the original systems.

Vita

James Duane Gatewood was born October 25, 1947 in Lyons, Nebraska. He graduated from Sergeant Bluff High School, Sergeant Bluff, Iowa in 1966. He then studied mathematics at Dakota Wesleyan University, Mitchell, South Dakota, from which he graduated in 1970. He entered the USAF in 1970 and was commissioned in 1974. While enlisted, he served as a 205X0, Electronic Emission Monitor Analysis Specialist. Upon commissioning he was trained as a 30XX Communications Officer. From September 1974 through September 1975 he was the Plans Officer at the 1974th Communications Group, Udorn Royal Thai Air Force Base Thailand. He then served from October 1975 through May 1977 as a Programs Officer at European Communications Area, Kapaun Air Station, Germany. Following this he was appointed Commander, Detachment 6, 1989th Communications Group, Inoges Radio Relay Location, Spain where he served from June 1977 through June 1979. In July 1979 he was assigned to 1974th Communications Group, Scott Air Force Base, Illinois where he served as Chief of Record Communications and then as Chief, C-E Operations. He entered the Air Force Institute of Technology's Masters degree program in June 1982 as an OCAY, Information Processing, Student. While assigned to AFIT he was selected as an Outstanding Young Man of America.

Permanent Address: 516 West 2nd Avenue

Mitchell, South Dakota 57301

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			Approved for public release; distribution unlimited.	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/EE/83D-10			5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433			7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code)			10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See Box 19			PROGRAM ELEMENT NO.	PROJECT NO.
			TASK NO.	WORK UNIT NO.
12. PERSONAL AUTHOR(S) Gatewood, James D. B.S., Capt, USAF				
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1983 December	
15. PAGE COUNT 240				
16. SUPPLEMENTARY NOTATION				
APPROVED FOR RELEASE: 1107 APR 1997 7 Feb 84 Do not Release and Professional Development Wright-Patterson AFB OH 45433 (AIC)				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary)	
FIELD	GROUP	SUB. GR.		
09	02		Data base, Graphics, Relational Data base	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
Title: APPLICATIONS PROGRAMS TO FACILITATE USE OF A DBMS TO STORE AND RETRIEVE GRAPHICS DISPLAYS (INGRED II)				
Thesis Chairman: Dr. Gary B. Lamont				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS ☐			21. ABSTRACT SECURITY CLASSIFICATION	
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Gary B. Lamont			22b. TELEPHONE NUMBER (Include Area Code) 513-255-3576	22c. OFFICE SYMBOL AFIT/ENG

AFIT/GCS/EE/83D-10

Abstract

Applications programs were written to facilitate the use of the RIM relational DBMS to store and retrieve graphics displays generated using capabilities of a graphics editor. This DBMS storage capability is in addition to data file storage which was provided by INGRED, the graphics editor. The resulting system is called INGRED II. The addition of DBMS capabilities made the graphics editor much more flexible and more user-friendly. It resulted in capabilities to obtain a listing of displays stored in the data base, to delete displays from storage which are no longer needed, and to store pertinent information in the data base along with the display data. Neither of these capabilities was available using the original INGRED. In the study of the data structure generated by GWCORE to store display information, it was determined that some data structure was redundant. Routines were added to optimize the data structure before storage on disk, with a resultant savings of over 45 per cent. The time required to do the optimization proved negligible, especially considering the criticality of storage at most locations which will use Ingred II. The result of the INGRED II system is a more effective system for use in graphics research and software design automation.



