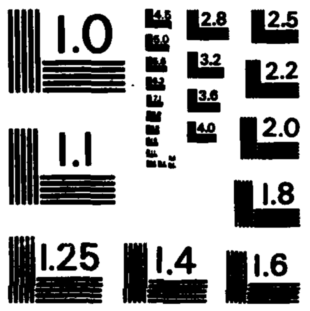
AD-A126 434   INTEGRATED APPLICATION SOFTWARE SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA   J C WATERS DEC 82   1/2

UNCLASSIFIED   F/G 9/2   NL


1.0
4.5
5.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTEGRATED APPLICATION SOFTWARE SYSTEM

by

John Christopher Waters

December 1982

Thesis Advisor: Dusan Zdenek Badal

Approved for public release, distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A126434	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Integrated Application Software System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Christopher Waters		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December, 1982
		13. NUMBER OF PAGES 172
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Interactive Application Software System
Relational Database Management System

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

As increasing data processing power becomes available at decreasing cost, greater numbers of nontechnical personnel are gaining access to automated systems that enhance their productivity. However, the sharp distinction between each

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) of the support packages, and the requirement for the user to become familiar with different models, concepts and vocabularies is a barrier to reaching higher effectiveness. The premise is that these common support systems have equivalent functions and a large intersection of operations that can be integrated. It is the purpose of this thesis to study a possible Integrated Application Software System (IASS) that will combine the needed capabilities into a functional system and present the user with a single data model and vocabulary set. The data model which is proposed for use by the IASS is the relational data model, since it is universally understandable, and has a robust theoretical foundation.

Integrated Application Software System

by

John Christopher Waters
Lieutenant, United States Navy
B.S., Rensselaer Polytechnic Institute, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1982

AUTHOR: [illegible]

APPROVED BY: [illegible]
Thesis Advisor

[illegible]
Co-Advisor/Second Reader

[illegible]
Chairman, Department of Computer Science

W M Woods
Dean of Information and Policy Sciences

ABSTRACT

As increasing data processing power becomes available at decreasing cost, greater numbers of nontechnical personnel are gaining access to automated systems that enhance their productivity. However, the sharp distinction between each of the support packages, and the requirement for the user to become familiar with different models, concepts and vocabularies is a barrier to reaching higher effectiveness. The premise is that these common support systems have equivalent functions and a large intersection of operations that can be integrated. It is the purpose of this thesis to study a possible Integrated Application Software System (IASS) that will combine the needed capabilities into a functional system and present the user with a single data model and vocabulary set. The data model which is proposed for use by the IASS is the relational data model, since it is universally understandable, and has a robust theoretical foundation.

TABLE OF CONTENTS

I. INTRODUCTION

The utilization of computers in many areas, such as personal computing or office and manufacturing automation, is rapidly expanding. No longer is their use being relegated to support personnel, but is spreading into the ranks of lower and middle level management. The majority of such users are non-computer professionals who are coming to depend on the computer to provide a support capability in the accomplishment of their primary responsibilities.

Over the past years, numerous software packages have been made available to support a broad spectrum of users in varying environments. Capabilities such as word processing, database management, modeling, form generation, and electronic mail have become essential. The point to be made is that the original purpose of introducing the computer was to increase the effectiveness and efficiency of the organization. While the present performance of each support package is satisfactory, the manner in which they are presented to the user is not. As illustrated in Figure 1.1, each support system is typically disjoint from all others, and the user is presented with differing models, command vocabularies, and operating instructions. This non-integrated combination of application software requires a great deal of effort on the part of the user to become familiar with a new system and remember it along with the other systems that are used.

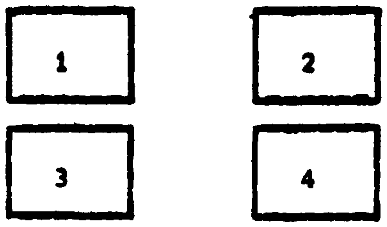


Figure 1.1 - Disjoint Support Systems.

What is needed to increase productivity is an integrated system that combines the capabilities of the support packages into a system which presents the user with a single, yet easy, conceptual data model and vocabulary set. It is such a system that is called an Integrated Application Software System (IASS), and the purpose of this thesis is to develop a design for its implementation. It is important to emphasize that the IASS is not built around the already existing application programs, but the reverse. Given an IASS's common user interface and conceptual level, the designer will evaluate each application and design a new application program to capitalize on the IASS capabilities.

The design objectives for the IASS are:

(1) Ensure a high degree of user friendliness and emphasize simplicity.

(2) Minimize the initial and acquired user skill level necessary to gain major functional use of the system.

(3) Minimize the learning time required to gain functional use of the system.

(4) Present a logical interface between each of the IASS's capabilities, but minimize the explicit user navigation between them.

(5) Determine the largest intersection of functional capabilities for the individual application programs and integrate them into a common conceptual level.

(6) Develop as small a command vocabulary as possible at the user interface. Ensure that these commands form an intersection of the application program commands, and are consistent between each of the applications.

(7) Eliminate the dependence on user programming in order to use the system.

(8) Embody the notion of software adaptivity whereby a user, already familiar with at least one application in the IASS, can learn a new application by studying only the small increment of new commands and functions that are specific to the new application.

(9) New applications, not originally considered in the original IASS, are implemented by adding a small increment of functions and commands to the IASS.

(10) Allow for the interaction of the included applications in support of each other.

While the IASS cannot be expected to completely integrate the separate features of each support package, it can strive to maximize the intersection between them. Figure 1.2 shows a simple illustration, in Venn Diagram form, of an IASS.

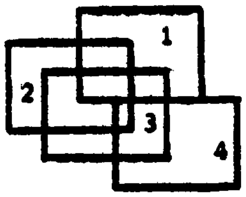


Figure 1.2 - IASS Support Package Intersection.

In the following chapters the basis of an IASS design will be discussed. Chapter 2 will describe a selected group of currently successful application support programs that will be considered for integration into an IASS. Chapters 3 and 4 will cover the conceptual level of the IASS which is invisible to the user. Chapter 3 discusses a common data object for the IASS, and Chapter 4 the conceptual level operations allowed on it. Chapter 5 covers the use of the conceptual level by the included application programs. Chapter 6 covers how the user will interface to the IASS. Finally, Chapter 7 presents the conclusions that can be made from this limited study of an IASS.

II. DESCRIPTION OF SUPPORT SYSTEMS

In order to demonstrate the applicability of an Integrated Application Software System (IASS) and its conceptual level integration approach, six common software applications were selected. The applications were chosen based on their perceived effect in supporting an office based user.

a. Text Editor
b. Word Processor
c. Relational Database Management System
d. Electronic Spread Sheet
e. Forms Generator
f. Electronic Mail

As a non-integrated collection of application software, each of these packages is implemented to accomplish a set of operations on a specific file type, and the set of included operators is tailored for that type. Commands are not usually transferable between applications and the application vocabulary is very "baroque" in that most of the operators exist as a matter of convenience to the user. Too often it is a very small percentage of the overall vocabulary that is used over ninety percent of the time. The majority of users only learn a subset of the vocabulary necessary to accomplish the essential functions of the application package, and disregard the rest.

Representative commercially available software packages corresponding to the six selected application types were reviewed to determine the nature of the pertinent file types and the essential functions required of the application. The following sections will present the results of this review as they apply to each application package.

A. TEXT EDITOR

The purpose of a text editor is to prepare a text file in an appropriate form for use by a subsequent process. "WORD STAR", and the "VI" and "EDIT" systems for UNIX were analyzed. These systems are described in Appendices (A), (B), and (C).

The text editor can be divided into five major functional categories of commands which are supported; <Create>, <Insert>, <Modify>, <Delete>, <Move>, and <Retrieve>.

1. Creation

A facility used to define an empty file into which the text will be entered. This involves a directory entry of some type, the allocation of storage space, and the creation of a buffer area.

2. Insertion

Done an object at a time in relation to some referenced point in the text file, such as the cursor position. It is possible to insert any object at a specified point. Insertion is non-destructive in that the object is inserted between existing contents, without overwriting. The object to be inserted may be entered by the user, come from a buffer, or come from another file.

3. Modification

In relation to some referenced point in the text file the current contents are altered . Modification is destructive in that the current contents are destroyed by writing over them. Global modification is possible.

4. Deletion

In relation to some referenced point in the text file an object of any granularity level is removed. Global deletion is possible.

5. Movement

The current point of reference in the text file is changed to meet the desires of the user. Control of movement is possible at any object level. The user should have the capability to move to a predefined location or to a location that meets some condition.

6. Retrieval

While not directly responsive to user retrieval commands, the text editor supports the user by displaying the local area around the point of reference. In a general sense this is a desired retrieval of text from the file. Retrieval is automatically done for the user when the point of reference changes to ensure the user can establish the context of the point of reference.

B. WORD PROCESSOR

Files prepared by a text editor can be processed by a Word Processor (WP). "WORD STAR", an on-line WP, and "NROFF -ME", a WP for the UNIX operating system, were analyzed. These two systems are described in Appendices (A) and (D).

Currently there are two approaches to word processing:

(1) Off-Line Formatting. In addition to the actual text in the edited file, a combination of special characters, and characters strings, can be embedded in the text file for use by the WP. These special embedded character strings are commands used by the word processor to produce the desired printed format of the text file. This requires a two step procedure by the user. The user first visualizes the desired format of the output and then translates it into a combination of the actual text and the embedded WP commands. The text file is then processed by the WP. This is a level of indirection that delays feedback to the user as to the effect of a command.

(2) On-Line Formatting. In this case, while most of the WP commands are still embedded in the text file, they are immediately executed. As the user inputs the text, it is displayed on the screen in the desired format. The user thereby receives immediate feedback as to the effect of formatting commands. However, the problem of the display

format being bigger than the display dimensions is a problem, if only a minor one.

The WP, unlike the other application packages, does not directly manipulate a data object in the course of its operation. Instead, it intercepts a stream of data from the data object and alters the display format prior to output. For this reason is seems logical to consider the WP to be a part of the editor - the application package that manipulates the data object upon which the WP depends.

C. DATABASE MANAGEMENT SYSTEM

Data is simply symbols which are stored. In and of itself, a datum has no significance. However, when coupled to an entity, the class of a datum becomes an attribute of the entity and the value of the datum can be used as information to describe an instance of the entity. When data is stored in a computer it is known as a database. To transform the raw data into an abstraction suitable for a person to use and/or modify is the major function of a Database Management System (DBMS). In a sense a DBMS acts as an interpreter between the user and the computer. It interprets user statements describing what is to be done with the database into the lower level algorithms necessary to perform the operation on the conceptual and eventually the physical representation of the data in the computer. From the perspective of the computer, the DBMS translates the physical implementation of the data through the conceptual representation to the appropriate user view. In this way the DBMS provides two levels of data independence. Data independence implies that modification can be made to the implementation of the data without affecting the logic of the application programs. Data independence between the conceptual schema and its physical implementation allows changes to the physical implementation of the conceptual schema while permitting application programs to run as if no change had occurred. Similarly, data independence between the conceptual schema and a user view allows changes to be made to the conceptual schema while permitting application programs to run as if no change had occurred.

In addition to the data management function, a DBMS also provides functions to ensure system integrity. Towards this end a DBMS enforces database security constraints. The security facility ensures that unauthorized access to data is not allowed. A DBMS typically ensures that the required properties of data are guaranteed. These properties can be either syntactic, that is structural, or semantic, for instance contained within a specified domain. A DBMS typically provides a mechanism to protect the database from a system crash by regularly making back-up copies of the database. In the event of a system crash, a DBMS typically provides a facility to restore the database to a previously consistent state. Finally, in a multi-user environment, a

DBMS typically provides a synchronization mechanism to protect the database from inconsistencies which might result from simultaneous access to a database, especially if one access entails a change to a shared data item.

"DBASE II" and "SEQUITUR" were reviewed as representative relational data base models and are described in Appendices (E) and (F). A relational DBMS was selected as an IASS conceptual database model due to its familiar and universally understood data object, the relational table. The basic organizational unit in the relational table is the named and domained field. A record of arity "n" in the relational table contains n fields, each containing a value from its domain. A relational table is the next higher level of organizational abstraction. The overall schema of the relational table is defined by physical properties of the fields and embodies the relationship which is defined by the field set.

A DBMS can be logically divided into three functional parts - data definition, data manipulation, and query retrieval. These parts can be further refined into the functional operators <Create>, <Insert>, <Modify>, <Delete>, <Move>, and <Retrieve>.

1. <u>Creation</u>

The existence of a relational table is typically due to a need perceived by the user to organize data. Creation is therefore the process by which the relational table is defined in a database by its identity and composition. The existence of the table is noted in some form of a database table directory. The composition of the table defines the schema of the relational table. Modification of a user defined table can be viewed as a special case of creation or re-creation. A table name can either be changed or the schema redefined. In addition to these explicit methods of creating relational tables, implicit methods also exist. As a result of the relational operation JOIN, a new relational table can be created. The method of naming the new relational table is implementation specific. The composition of the table, however, is derived from the schema of the operand relational tables.

2. <u>Insertion</u>

Insert is a component of the set of data manipulation operators. The action of an insert is to place a record into the relational table. The origin of the record to be inserted is irrelevant to the operation. The effect of the operation is that a new relational table is derived from the old relational table, order not being significant.

3. <u>Modification</u>

Modify is a component of the set of data manipulation operators. The action of a modify is to change the data in a field. DBMS's typically do not restrict the origin of the new data to what the user supplies at a terminal but can be as a result of evaluation of expressions or derived from other relations in the database.

4. Deletion

Delete is a component of the set of data manipulation operators and is in fact the inverse operation of insert. The action of delete is to remove a record from the relational table, the final disposition of the deleted record is immaterial to the operation. A delete operation is typically a two step process. A record is first marked for deletion and then explicitly removed from the table.

5. Movement

The movement operator can be viewed either as a passive data manipulation operator or a query retrieval operator. Movement encompasses the process of changing the current point of reference in the database. The ultimate destination is determined from manipulating data in the database or as a direct result of a query on the database. The point of reference can be of any organizational abstraction from an entire relation to an individual character in a field. This range in movement implies that this operator subsumes the theoretical relational algebra operators PROJECTION and SELECTION. Movement is a required operator in order to scan and extract information from, or in conjunction with, the performance of any of the other operations on the database.

6. Retrieval

Retrieve is exclusively a query response operator. The condition of a query specifies the information to be extracted or derived from a database. A query can be in many forms. Traditionally a query facility is embodied in a specialized language which the user employs to extract information from the database. In this simplest form, a query is equivalent to the relational algebra PROJECTION operator possibly following the SELECTION operation on the referenced relation. Queries can exist in subtler forms. Movement through a database can actually be the result of an underlying, implicit retrieve operation. Some uses of forms embody a retrieval operation as they extract information from a database to derive its contents. Reports also embody the retrieve operator in the same way as a form. From the database, information is retrieved and displayed in a user specified format.

D. ELECTRONIC SPREAD-SHEET

An Electronic Spread-Sheet package provides an important numerical modeling capability to the user. This application provides the user with a piece of "electronic" scratchpaper for doing fairly complicated numerical problems, and models that are of a recurring nature. Instead of reaching for pencil, paper, and calculator each time, the user will call the electronic spread sheet and by entering the needed values cause the spread-sheet to complete the calculation/model. It compliments the inclusion of the word processor and database management system in the IASS. The commercially available "VISICALC" system was reviewed, and is described in Appendix (G).

Spread-sheets are commonly divided into addressable (row,column) entry positions, similar to a checkerboard, and are used to graphically display numerical data in a tabular format. A small portion of the spread-sheet is usually visible on the screen at any one time and the user must use window and screen commands to move across the entire sheet. Each entry position is an independent entity and can contain any of the data types - character, numeric, or function. The contents of an entry position can be expressed in relation to the value of a previous entry position.

System operations consist of <Create>, <Insert>, <Modify>, <Move>, <Delete>, and <Retrieve>.

1. Create

The user initializes a data storage structure for a new spread-sheet. The dimensions for the new spread-sheet are initialized and all entry positions are set to null values.

2. Insertion

Given an already existing spread-sheet, the user adds a new column or row to the spread-sheet at an indicated location. This enlarges one of the spread-sheet dimensions by one.

3. <u>Modification</u>

Change the current value in an existing entry position to a new value or function.

4. <u>Deletion</u>

Given an already existing spread-sheet, the user deletes an entire column or row from the spread-sheet at an indicated location. This will reduce one of the spread-sheet dimensions by one.

5. <u>Movement</u>

Allows the user to view the contents of the entire spread-sheet through the limited dimensions of the screen display by permitting the user to maneuver the screen across the spread-sheet.

6. <u>Retrieval</u>

The tabular display of the spread-sheet on the user's screen is the result of a predefined retrieval from the stored representation. As changes are made to entry positions and they, in turn, effect other entry positions, the tabular display is kept updated by automatic retrievals. Additionally as the user moves, or alters, the window into the spread sheet new information must be retrieved to meet the changed request. The user cannot specifically ask for information from the system, but instead accepts the single retrieval the spread-sheet package was designed to automatically produce.

E. FORMS GENERATOR

By definition, a "form" is a printed document with blanks to be filled in, and "format" is the arrangement, or plan, of a presentation. Traditionally, a document is assumed to be a piece of paper, and the input device used to place values onto the document is the human.

In the Electronic Data Processing (EDP) environment, these notions are generalized to where a document can also be derived from, or stored into, a database or data file. Regardless of the semantic generalizations introduced by EDP, the logical view of a form, as well as its function, remain the same. A form is used as a template to display information and/or collect a set of data. A form is distinguishable from a report in that a form represents only one instance, or a coalescence, of a set of data elements. The report contains the form as a special case, but repeats it for each instance in the set of data elements.

A Form Generator is a utility to assist the user in designing a displayable form at "design-time" and employing it at "use-time". Since creation and use times are different, the design-time display must represent the use-time display of the form as closely as possible. From the design the Form Generator must translate the visual specifications into the appropriate representation for use by the display function at use-time. In addition to the physical layout of the displayed form, the design and internal representation must contain information regarding the value association or derivation at use-time. The DBASE II form generation facility, and the separate "ZIP" screen oriented form generator were evaluated and are discussed in Appendix (H).

The design-time environment includes both initial form design and design modification. Form design is done by use of an editor and an on-screen capability is essential to achieve design-time and use-time visual proximity. The editor could be an integral part of the form generator or separate. Value association is not done by the form generator directly. The user states the value association of a "block" in terms of the use-time function which must evaluate the "block" values.

The process of form generation entails describing both the display features of a "block" in the form and the use-time association. The functional list of operators to support a form generator are <Create>, <Insert>, <Modify>, <Delete>, <Move>, and <Retrieve>.

1. Creation

Creating a new form entails naming the form and making it known to the rest of the system for use. Only the empty structure is created and will require the user to enter information into it.

2. Insertion

The user adds a new "block" to the form by specifying its characteristics. Characteristics may be - position, prompt, input/output, type, and processing information. A grouping of these "blocks" will make a form. Actual "block" specification and addition is done through a level of indirection where the user draws the "block(s)" on the screen and the system determines the parameters necessary to make the form.

3. Modification

The user changes one or more of the characteristics of an already existing "block".

4. Deletion

The user removes an entire "block" from the form.

5. Movement

The user has a point of reference within the given form. At any given time some "block" is the point of reference, and commands are available for the user to move this point of reference.

6. Retrieval

The user desires to see the format in which the form will be displayed both at design-time and run-time. The retrieval operation is automatic and translates the information stored in the form's structure into the appropriate display. Actual design and modification of a form is done on this display and the form generator determines the additional information it will need to recreate the finished form on demand.

F. ELECTRONIC MAIL

Electronic Mail is a facility for sending messages from one user to another. The "MAIL" system used by the UNIX operating system was reviewed and a description is given in Appendix (I).

An Electronic Mail system uses a predefined message form which contains information, such as destination, subject, and main body. Once created, the message is sent to the destination where it is placed in a message file, called a "mailbox", for reading. The major functional operations in a mail system are <Create>, <Insert>, <Modify>, <Delete>, <Movement>, and <Retrieval>.

1. Creation

The system generates an empty message form which the user fills in.

2. Insertion

Messages are inserted into the various mailboxes that the mail system maintains. A message is sent to another user by storing it in the system mailbox.

3. Modification

Messages are initially created with no values in the message form. Composing a message therefore, entails making modifications to the null parts of the message.

Modifications can also be made to a message any time during composition, before sending it to its destination. Finally, fields in a message may be modified by the recipient, in order to retransmit the message.

4. Deletion

By reviewing messages from the system mailbox, they are deleted from the system mailbox and placed into a local area. The user may delete messages from the system or local mailboxes at any time.

5. Movement

All the previous commands are performed in relation to a point of reference. The point of reference in a mailbox can be changed by the user in order to browse through the messages, or edit them.

6. Retrieval

Reading a message is done by retrieving the contents of the message fields and displaying them to the user.

In review, Chapter 2 has shown that a general commonality exists between the functions of the given applications. This commonality has been presented as the set of six command categories - <Create>, <Insert>, <Modify>, <Delete>, <Move>, and <Retrieve>. The following chapters will lead to an integration based on this commonality.

III. THE COMMON DATA OBJECT

The key to achieving an integrated system which can support formatted and unformatted data is to map the logical file types associated with the applications, into one conceptual data object. This conceptual data object is then part of a model of the applications and their use. The functional intersection of operations on the files can be implemented by a single set of primitive conceptual operations on the common data object.

The IASS must represent each of the logical file types associated with the included applications in such a way as to support the essential functions of each application. The data object chosen for this IASS design is the table. The table is a natural method of organizing data and therefore should be understandable, even by naive users. A table is a two dimensional array containing rows and columns. The IASS uses the table to represent a "real-world" entity. Each column represents one attribute of that entity and each row represents a unique occurrence of an entity. A table is almost equivalent to a relational database relation, except that a table implies that rows and columns have an order in the table which can be used in a positional addressing scheme. Since addressing is associative in a relation, the table must include columns which represent key values to uniquely identify each row. With this slight modification, any datum in a table can be accessed by specifying the name of the table, the value of the key, and the name of the attribute containing the datum. Hereafter, the common data object will be referred to as a relational table. Rows of such tables are usually referred to as "tuples" and columns are referred to as "attributes". The assumptions to be made concerning the relational table in this thesis are: (1) Row or column order is not significant, (2) All columns are named and must be unique within the table, and (3) Each row is uniquely identifiable by a key value.

In the following sections each logical file type will be described as a relational table. The attributes of each table type were selected based upon its perceived primary use. As such, the set of attributes associated with each table was determined in order to provide the information necessary to support that primary application. These tables are merely special cases of a relational database table. Based on their predefinition, their use can be bounded within the primary application and therefore can be "typed". To be of a certain type, it is sufficient that the table contains the minimum set of attributes necessary for that specific type as a subset of its total set of attributes. (e.g. A given system table might have five attributes. Three of those are the required attributes for a type-1 application table. The remaining two attributes could be the required attributes for a type-2 table.) This implies that, as an implementation issue, a single table could be considered to have multiple types, but for simplicity let us assume that a table will have only one type. As applications are added to the IASS, the accompanying minimal set of attributes must be defined to represent the new table type. There are many structural organizations which could represent a logical file type. The final decision on the organization of the table must be made to maximize the use of the conceptual level operations that are available to manipulate the data in the table. These conceptual level operations will be covered in Chapter 4. It is important to note that the table is a structural organization used as a model and therefore problems may arise in expressing the actual applications by the table model.

A. TEXT

Text, as data in a text file, is a "continuous" string of individual characters from some character set (e.g. ASCII). The use of text as data is by character, where each character is a unit of data used in an application.

Objects such as words, sentences, lines, or paragraphs are logical abstractions, hidden in the text, that are useful as information only to a human user. Any IASS manipulation that may alter this hidden logical abstraction will directly effect the ability of the IASS to transform the data back into information. (e.g. deleting every other word.) This will require the imposition of limitations on the use of table operators on the text table in order to protect this logical abstraction. (i.e. operations incapable of taking into account the logical abstraction of text will not be used.)

The only naturally occurring data elements in a text file is the single character, and the entire character stream. Their domain is all the elements in the applicable character set. These two problems, the continuous nature of text and its discreteness being limited to a single character or the entire file, make the text file the most difficult file type to model as a relational table. The table must quantize the continuous text stream into column units, thus destroying the continuity of the text. Additionally, while the relational table operators recognize the column as being an object, in fact it has no natural occurrence in the text file. Any definition of a column which represents text objects between the single character and the entire text file, is an arbitrary quantization of the text stream. Figure 3.1 illustrates the problem by arbitrarily choosing a column size equal to ten characters (the character "@" represents a carriage return and line feed). This division of the text stream into units, for use as tuples in a table, has no corresponding unit in the original text file, and has imposed structural limitations by the column boundaries.

(a) Text Stream

MR. JOHN SMITH@1349 WILMINGTON DR.@CARSON, CA

(b) Tuple Representation

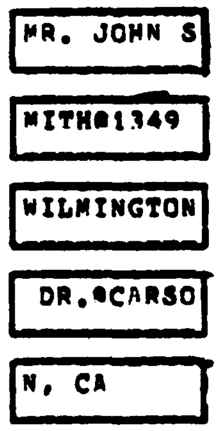


Figure 3.1 - Text Representation Problem

This problem of using a "discrete" representation will have to be acknowledged and steps taken at the application level to ensure the limitations imposed by the problem are not violated. In determining the size of a text tuple, neither the single character nor the entire file are acceptable units to be used in the relational table model since they would require a large amount of processing by the application level to transform them into usable units. (The argument is similar to memory management questions of paging versus segmentation and how large each unit should be.) Some arbitrary size, between the two extremes, will have to be chosen during implementation. For now it is assumed that the size limit exists.

The text file can be conceptually viewed as a text table, as in Figure 3.2. The text stream is represented by the set of rows in the table. Each "contents" column is densely packed in that no unused space is left in any row, except the very last row in the table. The text table does not match in any way the perceived "display" of the text file, as shown in Figure 3.1. The display structure (line-oriented, screen-oriented, or whatever) is considered an application level issue and will be covered there. Each row in the text table has a unique sequence number, "id", to mark the relative position of its contents in the text stream.

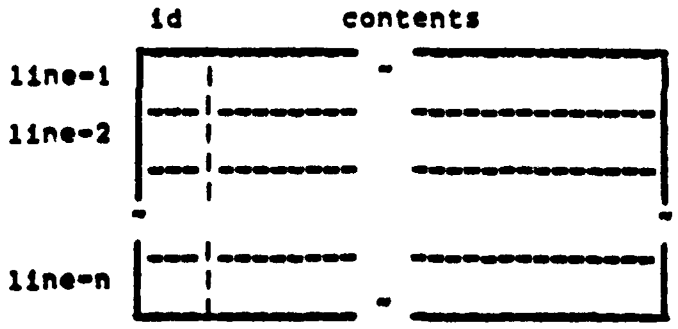


Figure 3.2 - Text Table

B. DATABASE

A relation in a relational database is described as a table, as shown in Figure 3.3. Relational database tuples are represented as the rows of a table and the attributes as the columns. The description of an attribute is defined by the user, and the set of attributes define the modifiable structure of the relation table. Chapter 1, Section (C) covers the concepts behind the relational DBMS in greater detail.

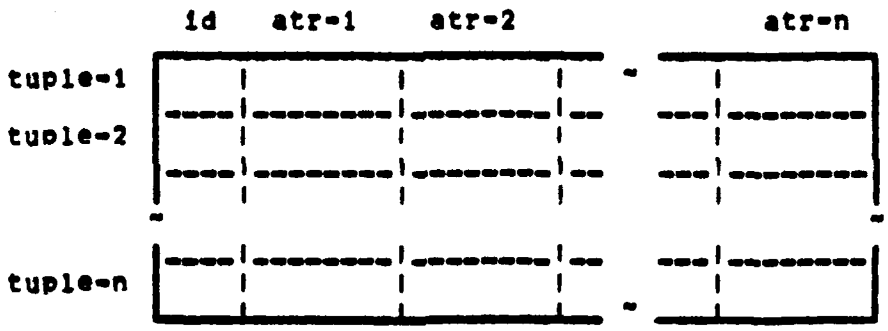


Figure 3.3 - Relation Table

C. SPREAD-SHEET

A spread-sheet is a database used as a numerical model in a predefined tabular display format. A spread-sheet can be represented as a collection of entry position tuples in a table, as shown in Figure 3.4. Each row in the table represents a single entry position in the spread-sheet. The table columns represent the predetermined elements necessary to describe the entry position, such as the location, value, and function.

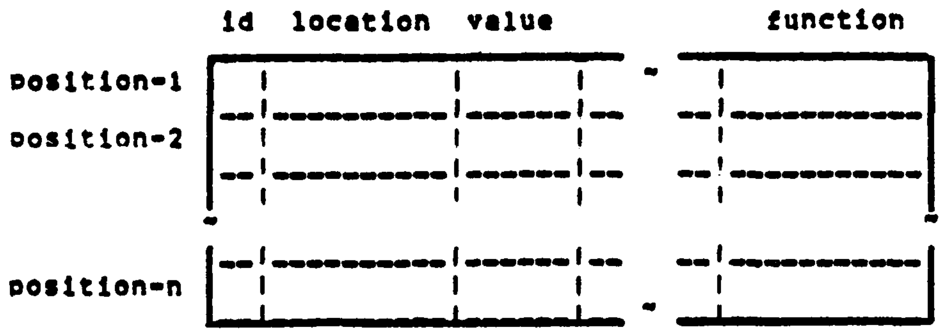


Figure 3.4 - Spread Sheet Table

D. FORM

A form is a template through which input and output values are transmitted. The information stored in a form database is used to prepare the visual image of the template in a user specified format.

The basic subunit of the form is called a "block", and it represents a basic unit of data for the form. The easiest way to visualize a "block" is to consider the Internal Revenue Service 1040 Tax Form. It is used as an input form, and each entry has a corresponding number to identify it as a separate entity, or "block". Each of these "blocks" has an a position on the form, an identifying number, a prompt, and space for an entry of some type. This means that the form table must include positional data for the block as well as data to determine how the block is to be used for specific applications.

A form can be represented as a collection of tuples contained in a table, as shown in Figure 3.5. Each row in the table represents a single "block" on the form and contains a description of the "block". The columns of the table are the predefined attributes of a block - unique ID, screen location, prompt string, input or output identifier, and the functional use of the block. Each table column represents an element of that description.

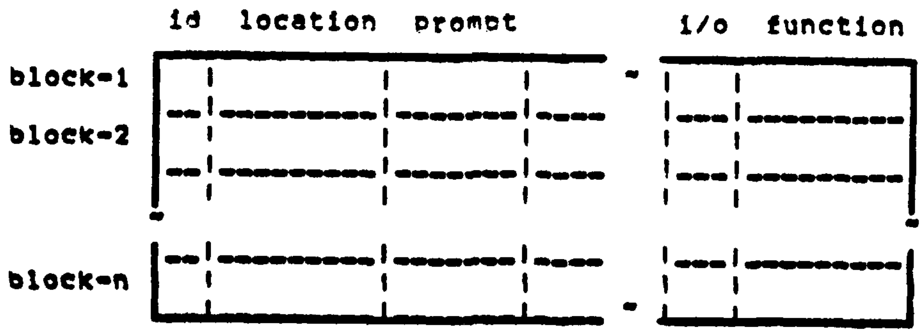


Figure 3.5 - Form Table

The "id" is a unique identifier for the block and would be system controlled. "Location" specifies the starting position for the "block" on the form. If required, a "prompt" string could be included to indicate the purpose of the entry position on the form (e.g. Name, Address, Number of exemptions, etc.). The "i/o" field will tell the Form Generator how to interpret the "function" field. The exact use of the "i/o" field is implementation dependent, but some obvious entries are "input", "output", and "text". Lastly, the "function" field will contain a command string for the block. If it is an input block, then the "function" field might contain the location where the user entered value is to be stored. If it is an output block, it might contain a query to be made on a database. If it is a text block, it might contain the name of the text file that is to be inserted in the form. As can be clearly seen, the actual use of the form table will be very implementation dependent and such issues will not be directly addressed in this thesis.

E. ELECTRONIC MAIL

Electronic Mail is a preformatted message sent to another user. Data contained in a message can be used as information to determine addressee and subject associated with a message. The data in a message is manipulated typically in the course of an editing session or by an application program to output a message to be read by the recipient. A "mailbox" can contain any number of messages. Each message contains a unique ID number, heading, and body. The ID attribute is a unique identifier of a message in a mailbox. The domain of the ID attribute is all unique identifiers as defined in the system. The ID could be local to a mailbox, or be of a global nature. The body of the message is textual and its domain is a continuous stream of characters or a reference to a text file. Heading consists of an originator, recipient, date, and subject. The originator, recipient, and subject are character strings of some maximum length. The date is some allowable value as determined by the date convention used by the system. Each message tuple is a complete message being routed from a sender to a receiver(s).

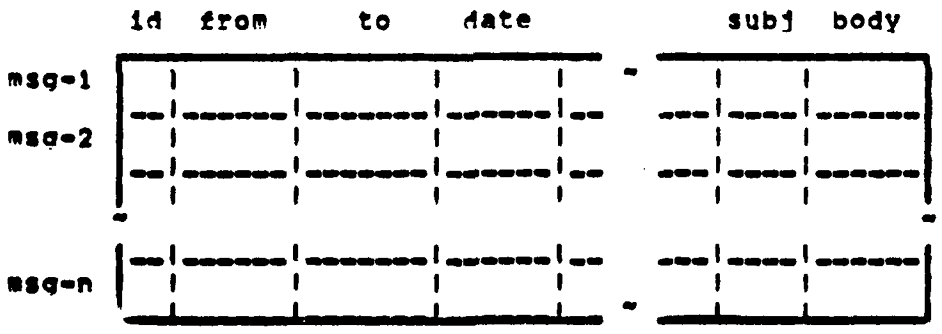


Figure 3.6 - Mailbox Table

The mail file can be represented as a collection of mail tuples contained in a table, as shown in Figure 3.6. A "mailbox" is a table with the rows representing individual messages and the columns representing the predetermined format of the message, such as from, to, date, subject, and main body.

This will end the discussion of representing the logical file types as a single conceptual data object in the form of a relational table. This chapter has shown that each of the five logical file types can be mapped into a table format with varying degrees of success. A "secret" that will be possessed by the application level is just how successful this mapping was. Of the presently included file types only the text type has shown signs of major problems. However, similar situations could occur as new applications are added to the IASS. The solution to the problem is to accomplish at the application level what cannot be done at the conceptual level due to the modeling limitations. The next chapter, Chapter 4, will cover the conceptual level operations that are available to manipulate the common data object tables.

IV. OPERATIONAL INTERSECTION

A major concept behind the Integrated Application Software System is the existence of a common "conceptual level" that is used by all the included application programs. It is important to note that it is the application programs, and not the user, that will interface with the conceptual level.

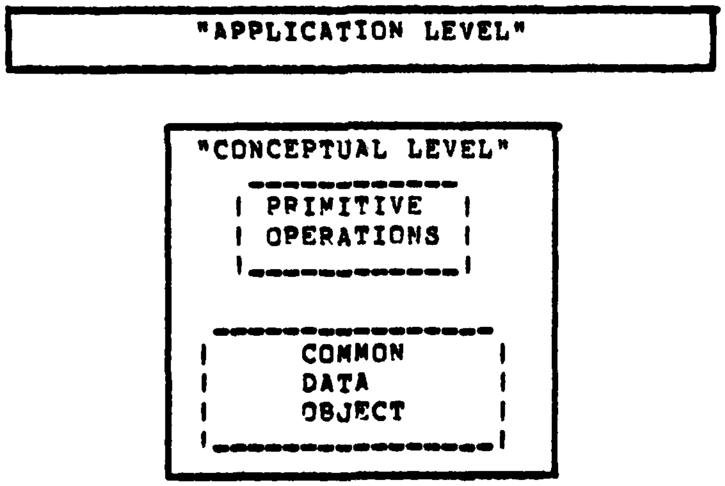


Figure 4.1 - IASS Conceptual Level.

This conceptual level will manage the data in the common data object, described in Chapter 3. A set of primitive, or basic, operations designed to manipulate the common data object and enforce integrity constraints will be included at this level. Figure 4.1 illustrates the conceptual level.

The application packages will call these operations to perform desired manipulations at the conceptual level in support of the user. The specific application determines the combination of primitive operations necessary to retrieve data from the data tables in conformance with the use of the table as a model of the application. Only those operations that cannot be accomplished at the conceptual level, due to modeling failures, need be included in the application level.

A. BASIC OPERATIONS

The set of table primitive operations is the source of a major IASS operational intersection. All applications attached to the IASS can use these commands in order to perform their function. However, as modeling problems will exist, each application area may have limits that make a certain operation meaningless.

Since the common data object is a relational table, the natural set of primitive operations are the basic table manipulation operations inherent from relational database theory. The operations are named: INSERTION, MODIFICATION, DELETION, PROJECTION, SELECTION, UNION, SET DIFFERENCE, CARTESIAN PRODUCT, INTERSECTION, QUOTIENT, JOIN, and NATURAL JOIN. These operations are set theoretic in that their operands are tables (sets of tuples) and their results are tables. This feature of the relational operators eliminates the need for any application to be concerned with iteration control. These operators are divided into two groups, Unary and Binary, based on the number of operands required.

1. Unary Table Operations

The first five operators are unary operators in that they use a single table operand. The operators are:

a. Insertion

Given a relation R, INSERTION adds a new tuple to R at a specified, or default, location.

b. Modification

Given a relation R, MODIFICATION changes one, or more, of the components of a tuple, or tuples, in the relation R.

c. Deletion

Given a relation R, DELETION removes a tuple, or tuples, from the relation R.

d. Projection

Given a relation R of arity "a", a PROJECTION of R is a relation formed by removing some of the components of R and/or rearranging some of the remaining components.

e. Selection

Given a relation R, a SELECTION on R is the set of tuples in R that make true some conditional statement based on the components of R. The operands of the conditional statement are constants or the components of R.

The operations of the conditional statement are the arithmetic comparison operators - less than, equal to, greater than, less than or equal to, greater than or equal to, and not equal to - and the logical operators - AND, OR, and NOT.

2. Binary Table Operators

The seven binary operators will use two tables as operands. A description of the seven operators follows and for help in understanding them, some examples will be given. For that purpose two "representative relations" are given in Figure 4.1 for use in each of the descriptions and examples.

A	B	C
a	b	c
d	a	f
c	b	d

Relation "R"

D	E	H
b	g	a
d	a	f

Relation "S"

Figure 4.1 - Initial relational tables.

a. Union

Given two relations, R and S, the UNION of R and S are those tuples that are in R, or S, or both. The UNION operation is denoted by $(R \cup S)$, and Figure 4.2 shows the results. Both tables must be of the same arity, and an attribute in the first table must be matched by the same attribute in the second table. (i.e. in this case D = A, E = B, and F = C.)

A	B	C
a	b	c
d	a	f
c	b	d
b	g	a

Figure 4.2 - Result of (R ∪ S).

b. Set Difference

Given two relations, R and S, the SET DIFFERENCE of R and S is the set of tuples that are in R, but not in S. SET DIFFERENCE is denoted (R - S), and Figure 4.3 shows the results. Both tables must be of the same arity, and an attribute in the first table must be matched by the same attribute in the second table. (i.e. in this case D = A, E = B, and F = C.)

A	B	C
a	b	c
c	b	d

Figure 4.3 - Result of (R - S).

c. Cartesian Product

Given two relations, R of arity "a1" and S of arity "a2", the CARTESIAN PRODUCT of R and S is the set of tuples of arity "(a1 + a2)" whose first a1 components form a tuple in R and whose last a2 components form a tuple in S. CARTESIAN PRODUCT is denoted by (R X S), and Figure 4.4 shows the results. Each of the resulting attributes of the CARTESIAN PRODUCT operation must be unique.

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

Figure 4.4 - Result of (R X S).

d. Intersection

Given two relations, R and S, the INTERSECTION of R and S is the set of tuples that are in both R and S, not those that only occur in one relation. INTERSECTION is a shorthand for R - (R - S), is denoted by (R ∩ S), and Figure 4.5 shows the results. Both tables must be of the same arity, and an attribute in the first table must be matched by the same attribute in the second table. (i.e. in this case D = A, E = B, and F = C.)

A	B	C
d	a	f

Figure 4.5 - Result of $(R \cap S)$.

e. Quotient

Given two relations, X of arity "a1" and Y of arity "a2" where a1 is greater than a2 and there is at least one tuple in S, the QUOTIENT of X and Y is the relation of arity (a1 - a2) composed by: First take the PROJECTION of X over the first (k1-k2) components and call the resulting relation T; Second, take the CARTESIAN PRODUCT of T and Y and call the resulting relation U. Lastly, determine the SET DIFFERENCE between U and X.

A	B	C	D
a	b	c	d
a	b	e	f
b	c	e	f
e	d	c	d
e	d	e	f
a	b	d	e

Relation X

C	D
c	d
e	f

Relation Y

A	B
a	b
e	d

$(X \div Y)$

Figure 4.6 - Result of $(X \div Y)$.

QUOTIENT is denoted by (X ÷ Y), and Figure 4.6 gives sample X and Y relations, and the result of (X ÷ Y).

f. Join

Given two relations, R of arity "a1" and Z of arity "a2", the result of a JOIN would be a relation of arity (a1 + a2) containing those tuples in the CARTESIAN PRODUCT of R and Z where a component in R stands in some relation to a component in Z. A JOIN is denoted by $R \underset{\theta}{\bowtie} Z$, and Figure 4.7 shows a sample relation Z and the results of $(R \underset{\theta}{\bowtie} Z)$.

D	E
b	m
d	n

Relation Z

A	B	C	D	E
a	b	c	b	m
c	b	d	b	m

$(R \underset{B=D}{\bowtie} Z)$

Figure 4.7 - Result of $(R \underset{B=D}{\bowtie} Z)$.

g. Natural Join

Given two relations, R of arity "a1" and U of arity "a2" where R and U have "c1" attribute names in common, the result of a NATURAL JOIN is a relation of arity (a1 + a2 - c1) formed by taking the CARTESIAN PRODUCT of R and U, then performing a SELECTION based on the equality of the common attributes in the tuples, and lastly performing a

PROJECTION with all possible attributes listed once. NATURAL JOIN is denoted by (R ⋈ U), and Figure 4.8 shows a sample relation U and the results of (R ⋈ U).

B	C	E
b	c	l
a	f	r

Relation U

A	B	C	E
a	b	c	l
d	a	f	r

(R ⋈ U)

Figure 4.8 - Results of (R ⋈ U).

B. SUITABILITY OF OPERATIONS

Looking at the conceptual data object as the relational table, and given the list of operations from section (A), above, it should be obvious that any operation, or series of operations, performed on a table will produce a theoretically useful relational table for some application. It will have a valid table structure and therefore can be manipulated by any operator. There are an infinite number of manipulation possibilities which can result in a endless speculation of applications. The conceptual view of the table and its operators only takes on significance when bounded by some application. It is the application that gives meaning to the usefulness or unsuitability of some operation or series of operations. Therefore, the intention of this section is to measure the usefulness of the basic relational operations within the functional scope implied by the selected set of applications data types described in Chapter 3.

Before describing each of the operations it is important to define some of the descriptive words that will be used for the operations:

Table Structure - the number of attributes, or fields, the table contains and the characteristics of each (name, type, size).

Syntactically Correct - the results of the operation is within the bounds of the operation definitions presented in Section (A) above. There are two subsets of this descriptor:

Meaningful Result - The result of the operation will be a table with an identified set of attributes and in all probability, at least one tuple. The resulting table will have the same structure as the original table, or one of the original tables, and will therefore be of that identifiable type. The application will be able to successfully use the resulting table.

Meaningless Result - The result of the operation will be a table with an identified set of attributes and in all probability, at least one tuple. The result will have the same structure as the original table, or one of the original tables, and therefore will be of that identifiable type. However, due to modeling or inefficiency, the resulting table will create difficulties for the application.

Syntactically Incorrect - The operation violates one or more of the bounds stated in the definitions presented in Section (A) above.

In the next two subsections the effects of the various operations will be discussed. Subsection (1) will cover the results when the operators are used on tables of the same type. Subsection (2) will cover operators used on tables of differing types.

1. Intra-Type Operations

This section will cover the effects of twelve basic relational operations when the operands are the same type of tables - text, database, spread-sheet, form, or mail. This section does not cover the results of mixed type operations as they will be covered in Subsection (2). At the conclusion of this Subsection, Table 4.1 will summarize the findings.

The very simple operations such as INSERT, MODIFY, and DELETE will not be discussed in the context of each table type since they are the minimum operations necessary to manipulate data in any table type and therefore meaningful for all table types.

a. Text Table Type Operations

The incompatibility between a text file and its representation as distinct units are revealed when attempting to apply the relational operators to the text table. What is in a tuple of text is merely a substring from the original text stream. As such, the situations where a tuple can stand alone as data for operations other than text processing are limited. Since the domain of the "contents" field is all character strings from the character set, there is no canonical ordering between the character strings. Whereas equality between contents fields can be established, there is no other comparison operator which will have applicability.

(1) Projection

The PROJECTION operation is meaningful, since it is necessary to retrieve either the "contents" or the "id" field from the text table.

(2) Selection

The SELECTION operation is meaningful but there are restrictions. Tuples would be selected from the table by performing the SELECTION condition on the "id" field. The "contents" field presents difficulties when used as a basis of the SELECTION condition since it can only be compared for equality, and that requires an exact specification of the contents in the condition.

(3) Union

The UNION operation is meaningful on text files and results in "appending" the second file to the end of the first, but there are modeling problems. The resulting text table could have more that one tuple with the same "id" field. For this reason the UNION operation should be considered with care.

(4) <u>Set Difference</u>

The SET DIFFERENCE operation is meaningful, but there are modeling problems. This operation must be used keeping in mind its exactness. Only tuples from the first table exactly matching tuples in the second will be removed. It cannot be guaranteed to remove duplicate strings from the text file since the text table model cannot accurately represent the text file.

(5) <u>Cartesian Product</u>

The CARTESIAN PRODUCT is incorrect since the resulting table structure will have duplicate attributes.

(6) <u>Intersection</u>

The INTERSECTION operation is meaningful, but there are modeling problems. The result would be the removal of all tuples from the first text table that were not also in the second text table. It cannot be guaranteed to find the common string(s) in two text files since the text table model cannot accurately represent the text file.

(7) <u>Quotient</u>

The QUOTIENT operation is incorrect since both text tables are of the same arity.

(8) Join

The JOIN operation is incorrect since the resulting table would have a structure with duplicate attributes.

(9) Natural Join

The NATURAL JOIN operation is meaningful, but since it duplicates the effect of the INTERSECTION operation in a less efficient manner it should be considered meaningless.

b. Spread-Sheet Table Type Operations

(1) Projection

The PROJECTION operation is meaningful for such operations as retrieving the information contained in a specific column, or columns, of the table.

(2) Selection

The SELECTION operation is meaningful for removing a tuple, or tuples, from the table for processing.

(3) Union

The result of the UNION operation on spread-sheet tables is meaningful, but there are modeling problems. The resulting table could now have more than one tuple attempting to represent the same entry position, tuples no longer representing the proper entry position, and/or entry positions no longer relating to their proper preceding entry positions. It is almost certain that such problems will occur and for that reason the UNION operator should be considered with care.

(4) Set Difference

The SET DIFFERENCE operation is meaningful, but there are modeling problems. The result of the operation is basicly those entry positions that are unique to the first spread sheet and not to the second. To ensure usability the implementation must include positional and value information in the tuple. The tuple cannot depend on order in the table for position, or functions relating to other tuples for value, since these other tuples may have been removed by the SET DIFFERENCE operation.

(5) Cartesian Product

The CARTESIAN PRODUCT operation is incorrect since the resulting table structure would have duplicate attributes.

(6) Intersection

The INTERSECTION operation on spread-sheet tables is meaningful, but there are modeling problems. The reasons are the same as those given for SET DIFFERENCE above.

(7) Quotient

The QUOTIENT operation is incorrect since both spread-sheet tables are of the same arity.

(8) Join

The JOIN operation is incorrect since the resulting table structure would have duplicate attributes.

(9) Natural Join

The NATURAL JOIN operation is meaningful, but duplicates the effect of the INTERSECTION operation in a less efficient manner and should therefore be considered meaningless.

c. Form Table Type Operations

(1) Projection

The PROJECTION operation is meaningful, since it can be used for retrieving parts of the block description used in the application.

(2) Selection

The SELECTION operation is meaningful, since it can be used for retrieving the block descriptions used by the application.

(3) Union

The UNION operation is meaningful, but there are modeling problems. The resulting table could contain tuples that are competing for the same position on the display. For this reason the UNION operation should be considered with care.

(4) Set Difference

The SET DIFFERENCE is meaningful, but there could be modeling problems. It would be used in finding those blocks on a form that are not in common with those on another form. Modeling constraints require that the block's location information be stored in the tuple and not depend on order in the table.

(5) Cartesian Product

The CARTESIAN PRODUCT operation is incorrect since the resulting table structure would contain duplicate attributes.

(6) Intersection

The INTERSECTION operation is meaningful, but there could be modeling problems. It would be used in finding those blocks on a form that are common with those of another form. Modeling constraints require that the block's location information be stored in the tuple and not depend on order in the table.

(7) Quotient

The QUOTIENT operation is incorrect since both form tables will have the same arity.

(8) Join

The JOIN operation is incorrect since the resulting table structure will have duplicate attributes.

(9) Natural Join

The NATURAL JOIN operation is meaningful, but duplicates the effect of the INTERSECTION operation in a less efficient manner and should therefore be considered meaningless.

d. Mail Table Type Operations

(1) Projection

The PROJECTION operation is meaningful in retrieving the contents of message fields for use in the application.

(2) Selection

The SELECTION operation is meaningful in retrieving a message for use in the application.

(3) Union

The UNION operation is meaningful in adding new messages to the message table by appending mailboxes together, but there are modeling problems. The resulting mailbox could have more than one message with the same "id" field. For this reason the UNION operation should be considered with care.

(4) Set Difference

The SET DIFFERENCE operation is meaningful in finding those messages in one mailbox that are not in another.

(5) Cartesian Product

The CARTESIAN PRODUCT operation is incorrect since the resulting table structure will have duplicate attributes.

(6) Intersection

The INTERSECTION operation is meaningful in finding those messages that are common to two mailboxes.

(7) Quotient

The QUOTIENT operation is incorrect since both mail tables are of the same arity.

(8) Join

The JOIN operation is incorrect since the resulting table structure will have duplicate attributes.

(9) Natural Join

The NATURAL JOIN operation is meaningful, but produces the same effect as the INTERSECTION operation with less efficiency, therefore it should be considered meaningless.

This section has described the operational effects of the basic operations when used on one or two tables of the same type. Figure 4.1, on the next page, summarizes the findings of this Subsection.

2. Inter-Type Operations

The previous section covered the effects of the five binary operators when conducted on tables of the same type. This section will cover these operators when used only on tables of differing types. Table 4.2 will summarize the findings of this Subsection.

a. Union

Since the UNION operation can only produce a usable output table when the structure of the two tables are identical, this binary operator could only be meaningful when one of the tables was a database type that happened to match the structure of the other table. In this special case the result would be meaningful, and in all other cases the UNION operation is incorrect.

Table 4.1 - Intra-Type Operations.

OPERATION	TEXT	DATA BASE	SPREAD SHEET	FORM GEN.	ELECT. MAIL
1. Insert	M	M	M	M	M
2. Modify	M	M	M	M	M
3. Delete	M	M	M	M	M
4. Projection	M	M	M	M	M
5. Selection	[M]	M	M	M	M
6. Union	[M]	M	[M]	[M]	[M]
7. Set Difference	[M]	M	[M]	[M]	M
8. Cartesian Product	-	M	-	-	-
9. Intersection	[M]	M	[M]	[M]	M
10. Quotient	-	M	-	-	-
11. Join	-	M	-	-	-
12. Natural Join	*	M	*	*	*

M	=	Operation is meaningful.
[M]	=	Operation is meaningful, but there are modeling problems.
-	=	Operation is incorrect.
*	=	Operation meaningless due to duplication.

b. Set Difference

Since the SET DIFFERENCE operator can only produce a usable output table when the structure of the two tables are identical, this binary operation could only be meaningful when one of the tables was a database type that happened to match the structure of the other table. In this special case the result would be meaningful, and in all other cases the SET DIFFERENCE operation is incorrect.

c. Cartesian Product

The CARTESIAN PRODUCT operator can produce a meaningful table structure for all combinations of table types that will not result in a table with duplicate attributes. The presence of duplicate attributes in the resulting table would make the CARTESIAN PRODUCT operation incorrect.

d. Intersection

Since the INTERSECTION operator can only produce a valid output table when the structure of the two tables are identical, the operation would only be useful when one of the tables was a database type that happened to match the structure of the other. In this special case the result would be meaningful, and in all other cases the INTERSECTION operation is incorrect.

e. Quotient

The QUOTIENT operator can only produce a meaningful output table when the arity of the second table is smaller than the first, and all its attributes are also found in the first table. This would then limit the type of the second table to database, and then require the proper table structure for the QUOTIENT operation.

f. Join

Given the fact that the actual structure of each table type is an implementation issue and therefore variable, it is conceivable that all table types could have at least one column structure in common with another table type and the JOIN operation would produce a meaningful table.

g. Natural Join

For the same reasons as stated for the JOIN operation, it is conceivable the the NATURAL JOIN operation would produce a valid and potentially useful table.

Table 4.2 - Inter Type Table Operations

		TABLE-2				
OPERATION	TABLE-1	TEXT (TX)	DATA BASE (DB)	SPREAD SHEET (SS)	FORM (FM)	MAIL (ML)
UNION	TX	+	?	-	-	-
	DB	?	+	?	?	?
	SS	-	?	+	-	-
	FM	-	?	-	+	-
	ML	-	?	-	-	+
SET DIFFERENCE	TX	+	?	-	-	-
	DB	?	+	?	?	?
	SS	-	?	+	-	-
	FM	-	?	-	+	-
	ML	-	?	-	-	+

Table 4.2 - (cont.)

OPERATION	TABLE-1	TABLE-2: TEXT (TX)	DATA BASE (DB)	SPREAD SHEET (SS)	FORM (FM)	MAIL (ML)
CARTESIAN PRODUCT	TX	+	M	M	M	M
	DB	M	+	M	M	M
	SS	M	M	+	M	M
	FM	M	M	M	+	M
	ML	M	M	M	M	+
INTERSECTION	TX	+	?	-	-	-
	DB	?	+	?	?	?
	SS	-	?	+	-	-
	FM	-	?	-	+	-
	ML	-	?	-	-	+
QUOTIENT	TX	+	?	-	-	-
	DB	?	+	?	?	?
	SS	-	?	+	-	-
	FM	-	?	-	+	-
	ML	-	?	-	-	+
JOIN	TX	+	?	-	-	-
	DB	?	+	?	?	?
	SS	-	?	+	-	-
	FM	-	?	-	+	-
	ML	-	?	-	-	+
NATURAL JOIN	TX	+	?	-	-	-
	DB	?	+	?	?	?
	SS	-	?	+	-	-
	FM	-	?	-	+	-
	ML	-	?	-	-	+

M	=	Operation is potentially meaningful.
?	=	Operation could produce a meaningful result but depends on the database table structure.
-	=	Operation is incorrect.
+	=	Intersection of same command, effects covered in section 4.B.1 and Table 4.1.

V. APPLICATION LEVEL INTERFACE

In the preceding chapters the structure of the conceptual level of the IASS has been covered. Chapter 3 discussed the table as a common data object, and Chapter 4 introduced the primitive operations allowed on the table. This chapter will describe how each of the application level packages interfaces to the conceptual level. All application packages in the IASS must make use of the common data object as an important part of their modeling effort. If the common data object can closely model a given application, then maximum use can be made of the conceptual level in order to accomplish the application's functions. However, if the common data object is a poor model of the application, then the application will have to provide more of its own service needs and therefore will create a large application specific series of operations.

In Chapter 4 the twelve basic primitive operations of the conceptual level were discussed, and they are listed again in Table 5.1. These operations will be discussed as system level operations where they are invisible to the IASS user. Those IASS operations that are visible to the user will be discussed in Chapter 6.

In Chapter 2 the basic user visible functions of each application were grouped into six command categories;

<Create>, <Insert>, <Modify>, <Delete>, <Move>, and <Retrieve>. When issued by the user they will cause the application to perform one or more operations in support of the user. While the use of the conceptual level by the application is generally invisible to the user, the sample list of included operations can be viewed as to how they will support the six visible command categories.

Table 5.1 - Conceptual Level Primitive Operations.

OPERATION	ABBREVIATION
1. Insertion	I
2. Modification	M
3. Deletion	D
4. Projection	P
5. Selection	S
6. Union	UN
7. Set Difference	SD
8. Cartesian Product	CP
9. Intersection	IS
10. Quotient	QT
11. Join	JN
12. Natural Join	NJ

In the following sections each of the five included application packages will be covered as to use of the Conceptual Level and their own "Workspace". What is meant by the application's "Workspace" is that part of the application program where the operationally specific functions of the application occur. This would include variables, constants, program logic, buffers, and whatever other implementation specific items are necessary. The Workspace is what makes each application unique to the user and is the part that must be inserted when a new application is added to the IASS. It is not the intention of this chapter to focus on the Workspace, so its coverage will be general and brief. The primary point of interest will be how each application can make use of the conceptual level.

In discussing use of the conceptual level, application specific operations will be described where each requires the use of one, or more, conceptual level operations. If one application operation can be defined in terms of a previously defined application operation, the previous operation will appear in brackets, "<>".

A. EDITOR/WORD PROCESSOR

As discussed in Chapter 3, Section (A), the Editor/Word Processor (E/WP) presents the greatest modeling problem for the conceptual level. This means that the E/WP will perform the majority of its operation in the Workspace and not at the conceptual level.

1. <u>E/WP Workspace Operations</u>

A large number of the operations necessary to model the E/WP will have to be located in the Workspace area since the data-table is a poor model of text. Some of the operations necessary at the Workspace level are; reassembling text tuples into a continuous text stream, keeping track of the proper ordering of text tuples, performing string searches, block moves, and character replacements. All operations for formatting and display will be conducted here.

2. E/WP Conceptual Level Operations

Although there are modeling problems with text, it does not mean that the E/WP cannot make use of the conceptual level. The following operations use the conceptual level but do very little direct manipulation of the text, since that is performed in the Workspace. The operations themselves were chosen based on a perceived minimum application need and the ability to use the conceptual level. This is not a complete listing of possible E/WP operations since that is a very implementation dependent question.

a. Insert Text Tuple

As the Workspace finishes with enough characters to constitute the "contents" field of a tuple, it will determine the proper "id" field sequence for the new tuple and then issue an INSERT operation to place the tuple in the table.

b. Get Text Tuple

The E/WP must determine the "id" of the next tuple it needs. A SELECTION is performed, based on that "id" field. The resulting tuple is then DELETE'ed from the original table and the "contents" field of the result is PROJECT'ed out and placed in the Workspace.

c. Append Text Files

The Workspace will <Get> the last tuple from file-1 and then proceed to SELECT each tuple from file-2, in order, PROJECT'ing out the "contents" field, and place it in the Workspace. As the Workspace gets enough characters to make a complete tuple, it will <Insert> the tuple into the end of file-1.

d. Insert A Text File

The Workspace will <Get> tuples from table-1 until it finds the correct insertion point. Then all tuples will be SELECT'ed from table-2, one at a time, in "id" order. The "contents" field of each will be PROJECT'ed out and placed in the Workspace. As the Workspace gets enough characters to form a complete tuple they will be <Inserted> into table-1 with the proper "id" field. When all tuples from table-2 have been copied into table-1, the Workspace will <Get> the remaining original tuples from table-1 and <Insert> them back into table-1.

e. Delete To A Buffer

The Workspace will <Get> tuples from the referenced table and <Insert> them back into the table until it finds the point at which the deletion is to begin. From that point it will continue to <Get> tuples from the table until it finds the point at which the deletion is to stop. As the Workspace collects enough characters to form a tuple, it will assign a proper "id" and INSERT the tuple in the buffer table. After the stop point, the Workspace will continue to <Get> tuples from the referenced table, and will <Insert> completed tuples back into it until the end is reached.

f. Copy To A Buffer

The Workspace will <Get> tuples from the referenced table and <Insert> them back into the table until it finds the point at which the copying is to begin. From that point it will continue to <Get> tuples from the table until it finds the point at which the copying is to stop. As the Workspace collects enough characters to form a tuple it will <Insert> them, with their original "id", back into the table. Simultaneously it will INSERT the same tuples, with new "id" fields, into the buffer table. After the stop point, the Workspace will continue to <Get> tuples from the referenced table, and will <Insert> completed tables back into it until the end is reached.

The use of the conceptual level by the E/WP is summarized in Table 5.2.

B. DATABASE MANAGEMENT SYSTEM

1. Database Workspace

Since the DBMS package is a relational database system, the user will be permitted direct access to the conceptual level primitive operations without constraint. The user accepts complete responsibility for the validity and usefulness of all actions. This means there is little need for a Workspace since the user does just about everything.

Table 5.2 - Editor/Word Processor Interface.

APPLICATION OPERATION:	PRIMITIVE OPERATIONS											
	I	M	D	P	S	UN	SD	CP	IS	QT	JN	NJ
a. Insert Tuple	X	-	-	-	-	-	-	-	-	-	-	-
b. Get Tuple	-	-	X	X	X	-	-	-	-	-	-	-
c. Append File	X	-	X	X	X	-	-	-	-	-	-	-
d. Insert File	X	-	X	X	X	-	-	-	-	-	-	-
e. Delete To Buffer	X	-	X	X	X	-	-	-	-	-	-	-
f. Copy To Buffer	X	-	X	X	X	-	-	-	-	-	-	-
USED	X	-	X	X	X	-	-	-	-	-	-	-

X = Primitive operation is used.
- = Primitive operation is not used.

2. <u>Database/Conceptual Level Operations</u>

As stated above, the user is permitted direct access to all the conceptual level primitive operators. There are no limits placed on the user in structuring these operators to produce a desired result. However, it is obvious that in implementation some issues will be encountered that will place limits on the user.

C. SPREAD-SHEET

The spread-sheet is very similar to a database in that it stores the facts related to a user defined "real-world" situation, i.e. it is a model. The major difference is that the user is limited to the predefined retrievals and displays provided by the spread-sheet. The spread-sheet has control and responsibility for the operation, while the user has responsibility for the content.

1. Spread-Sheet Workspace Operations

The Workspace is responsible for use of the spread-sheet data table since the user does not see or manipulate it directly. It contains the logic necessary to interface with the user and control the display.

2. Spread-Sheet Conceptual Level Operations

As the user issues application specific commands the Workspace translates them into a series of application and conceptual level operations. The list of included operations cannot be claimed to be definite or complete because that is an implementation issue and really without bounds. However, the list is considered to be a workable set of operations for a representative spread-sheet application.

a. Update Entry Positions

The Workspace must know in which order entry positions are to be updated, "row" or "column" order. Each entry position is SELECT'ed in turn based on its "location" field, and its "function" field is PROJECT'ed out. The Workspace evaluates the contents of the "function" field, and resolves references to other entry positions by SELECT'ing them and PROJECT'ing out the "value" field. When the new value is finally computed, a MODIFY operation is conducted to change the "value" field. The Workspace continues until all entry positions are updated.

b. Make An Entry In A Entry Position

The Workspace must know which column and row entry position is being referenced, and the value or function to be entered. A MODIFY operator will be used, based on a condition statement, to find the tuple with the proper "location" entry and then change its "function" and "value" field. If the Spread-Sheet is in automatic recalculation mode, then related entry positions will have to be <Updated>.

c. Add A New Column Or Row

The workspace must know the column or row on the spread-sheet that is being referenced and where the new column/row is to be placed relative to it. The MODIFY operator will be used to find those entry positions that must be moved, and change their "location" and "function" fields to take into account the shift in position. New tuples, with "location" fields corresponding to the added row/column will be INSERTED. Lastly, all entry positions will be <Updated>.

d. Delete A Column Or Row

The Workspace must know the column or row on the spread-sheet that is being referenced. A DELETION operator is issued with a condition statement corresponding to the proper "location". Next, a MODIFY operation is conducted on the "location" field of the proper entry positions necessary to close the resulting gap. Lastlv, all entry positions will be <Updated>.

e. Append Spread-Sheets

The Workspace must know whether sheet-2 is to be appended to the side or bottom of sheet-1. Given that information, a SELECTION is done on sheet-1 to find the maximum "location" field and it is PROJECT'ed out and saved in the Workspace. A MODIFY operation is next conducted on all tuples in sheet-2 to add the proper, row or column, value saved above to all entry position references in the "location" and "function" fields of sheet-2. Sheet-2 is then UNION'ed to sheet-1, and the resulting sheet is <Updated>.

f. Spread-Sheet Intersection

Given that you want to display the common entry positions of sheet-1 and sheet-2: Perform the INTERSECTION operation between sheet-1 and sheet-2. Then <Update> the resulting table.

g. Spread Sheet SET DIFFERENCE

Given that you want to disply those entry positions that are unique to sheet-1 and not found in sheet-2: Perform the SET DIFFERENCE between sheet-1 and sheet-2. Then <Update> the resulting table.

Table 5.3 - Spread-Sheet Interface.

APPLICATION OPERATION:	PRIMITIVE OPERATIONS											
	I	M	D	P	S	UN	SD	CP	IS	OT	JN	NJ
a. Update	-	X	-	X	X	-	-	-	-	-	-	-
b. Make An Entry	-	X	-	-	-	-	-	-	-	-	-	-
c. Add Row Or Column	X	X	-	X	X	-	-	-	-	-	-	-
d. Delete Row Or Column	-	X	X	X	X	-	-	-	-	-	-	-
e. Append Sheets	-	X	-	X	X	X	-	-	-	-	-	-
f. Inter-section	-	X	-	X	X	-	-	-	X	-	-	-
g. Set Difference	-	X	-	X	X	-	X	-	-	-	-	-
USED	X	X	X	X	X	X	X	-	X	-	-	-

X = Primitive operation is used.
- = Primitive operation is not used.

The use of the conceptual level by the Spread-Sheet application is summarized in Table 5.3.

D. FORM GENERATOR

It is the purpose of the Form Generator to create a table that will be used at a later time in support of other application packages or the user directly. The Form table is probably the most complex table of the five included in the IASS since it will be called on to do so much. The table reads like a set of step by step instructions on how to input or output the provided data. As this is a heavily implementation dependent application, not much emphasis will be placed on specific uses.

1. Form Generator Workspace Operations

The Workspace in the Form Generator must be fairly intelligent since it has two modes of operation. The first is "design-time" when it must interpret the user commands into a series of block entries in the form table, The second is "use-time" when it must use the information in the table to create the desired output form. This requires that the application logic, its ability to interface to the other applications, and any needed structures be contained in the Workspace.

2. Form Generator Conceptual Level Operations

The Form Generator does little more than build the table at "design" time, and read the instructions in the table at "use" time. It therefore seems that it can make fairly extensive use of the conceptual level operators. However, a complete list of all possible operations is impossible since the Form Generator application seems to be the most implementation dependent application of all. The list of operations that follows is intended as a representative group of basic operations and is not definitive.

a. Clear Workspace

If the Workspace is empty, then do nothing. However, if there are entries in the Workspace then issue a MODIFY operation, based on "location", to change those fields that have entries. If, no block was found, then issue an INSERT operation to place the block in the table. Lastly, erase the Workarea.

b. Find Block

First, <Clear> the workspace. Use a SELECTION operation to find the new block being referenced. PROJECT out the "location" field, and any other fields that are needed. If no block was found, then wait for next user command.

c. Add A New Block

The Workspace must start blank since it cannot have found a referenced block where a new block is being added. The user enters the proper information into the Workspace and when the user is finished, the Workspace will be <Cleared>.

d. Edit A Block

When the user edits an already existing block then it will have been found by the "Find Referenced Block" operation described above. The Workspace will wait until the user is finished editing, and then <Clear> the Workspace.

e. Delete A Block

<Clear> the workspace. Issue a DELETE operation based on the user generated condition.

f. Append Forms Together

<Clear> the Workspace. Given that form-2 is to be appended to the bottom of form-1: Use the SELECT operation to find the block with the highest row number and lowest column number in form-1. PROJECT out, and save in the Workspace, the "row" field. Issue a MODIFY operation on all blocks in form-2, and add the saved "row" number from form-1 to the "row" field in form-2. Then UNION form-2 to form-1.

g. Add A Blank Line To The Form

The Workspace must know the referenced row number on the form. Clear the Workspace. Issue a MODIFY operation on all blocks, on or below the referenced row, to update their "location" field.

h. Form Intersection

Given that the desired display is those blocks that are found both in form-1 and form-2, first <Clear> the Workspace.

If position on the form is important: Perform the INTERSECTION operation on form-1 and form-2. Pass the resulting table to the Workspace.

If position on the form is not important: PROJECT out the "prompt", "i/o", and "function" fields of form-1 and form-2. Do an INTERSECTION operation on the new tables and then NATURAL JOIN the result to the original table-1. Pass the resulting table to the Workspace.

i. Form Set Difference

Given that the desired display is those blocks in form-1 that are not found in form-2, first <Clear> the Workspace.

If position on the form is important: Perform the SET DIFFERENCE between form-1 and form-2. Pass the resulting table to the Workspace.

If position on the form is not important: PROJECT out the "prompt", "i/o", and "function" fields of form-1 and form-2. Perform the SET DIFFERENCE between these resulting tables. Take this result and NATURAL JOIN it to the original table-1. Pass the resulting table to the Workspace.

The use of the conceptual level by the Form Generator application is summarized in Table 5.4.

Table 5.4 - Form Generator Interface.

APPLICATION OPERATION:	PRIMITIVE OPERATIONS											
	I	M	D	P	S	UN	SD	CP	IS	GT	JN	NJ
a. Clear Workspace	X	X	-	-	-	-	-	-	-	-	-	-
b. Find Block	X	X	-	X	X	-	-	-	-	-	-	-
c. Add Block	X	-	-	-	-	-	-	-	-	-	-	-
d. Edit Block	-	X	-	-	-	-	-	-	-	-	-	-
e. Delete Block	-	-	X	-	-	-	-	-	-	-	-	-
f. Append Forms	X	X	-	X	X	X	-	-	-	-	-	-
g. Blank Line	X	X	-	-	-	-	-	-	-	-	-	-
h. Inter-section	X	X	-	X	-	-	-	-	X	-	-	X
i. Set Difference	X	X	-	X	-	-	X	-	-	-	-	X
USED	X	X	X	X	X	X	X	-	X	-	-	X

X = Primitive operation is used.
- = Primitive operation is not used.

E. ELECTRONIC MAIL

The purpose of the Mail application is to enable the user to leave messages for other users who are not presently available. Again this is a very implementation dependent application in determining exactly what services you wish to provide. As before, implementation issues will be avoided as much as possible.

1. Electronic Mail Workspace Operations

The Workspace is responsible for translating user commands into application operations necessary to create and read messages. It contains the logic necessary to use the Mail table, interpret user commands, and control the display.

2. Electronic Mail Conceptual Level Operations

A fairly wide range of application operations can be accomplished by using the conceptual level operations. While the following list of operations cannot be considered complete or definite, it is representative of an Electronic Mail application.

a. Pickup Mail

Upon entering the IASS, the MAIL system is automatically directed to pickup any mail for the user. The MAIL system generates a SELECTION operation on the system mailbox with the condition that the message(s) is addressed to the user. The resulting table is SET DIFFERENC'ed with the system mailbox and then UNION'ed with the user's mailbox.

b. Read Mail

The Workspace must have an "id" of the desired message. A SELECTION operation is performed on the user's mailbox based on the "id" field. The subparts of the message are PROJECT'ed out and placed in the Workspace.

c. Find Mail

Given a user entered condition statement the Workarea will generate a SELECTION operation based on that condition. The proper field(s) of the messages will be PROJECT'ed out and placed in the Workspace to support an appropriate display.

d. Edit A Message

The Workspace will know the "id" of the message being edited. When the user is completed, a MODIFY operation will be issued based on that "id" to change any fields that were edited. If no message with that "id" was found by the MODIFY operation then it must be a newly created message and the Workspace will INSERT it into the user mailbox.

e. Delete Mail

The Workspace will know the "id" of the message(s) or be given a user defined condition statement. Based on these, a DELETION operation will be performed on the user's mailbox.

f. Multi-hat

Given that the Workspace has a single message with a multi-hat destination: PROJECT out the contents of the "to" field in the message, and place it in the Workspace. The Workspace will find a database "alias" table with that name which has "id" and "to" fields. The tuples in this table correspond to the actual names in the multi-hat name. Taking the original message, PROJECT out the "from", "subj", "date", and "body" fields. Take the result and perform a CARTESIAN PRODUCT with the alias table. Now UNION the results with the system mailbox.

g. Send Mail

Each time the user leaves the Mail application, any outgoing mail is automatically sent. The Workspace generates a SELECTION based on the condition to find all messages not addressed to the user. The resulting "outgoing" table is SET DIFFERENC'ed with the user's mailbox. A SELECTION is then performed on the outgoing table to find any multi-hat destinations. The resulting multi-hat table is SET DIFFERENC'ed with the outgoing table, and the remaining outgoing messages are UNION'ed with the system mailbox. The messages in the multi-hat table are then SELECT'ed one at a time, DELET'ed from the multi-hat table, and then processed by the <Multi-Hat> operation.

h. Mail Synopsis

PROJECT out the "from", "to", and "subject" fields of all the messages in the user's mailbox. The Workarea will use this new table by SELECT'ing each message in "id" order, PROJECT'ing out the three fields, and using the results to create the display.

The use of the conceptual level by the Mail application is summarized in Table 5.5.

Table 5.5 - Mail Intersection.

APPLICATION OPERATION:	I	M	D	P	S	UN	SD	CP	IS	QT	JN	NJ
a. Pickup	-	-	-	-	X	X	X	-	-	-	-	-
b. Read	-	-	-	X	X	-	-	-	-	-	-	-
c. Find	-	-	-	X	X	-	-	-	-	-	-	-
d. Edit	X	X	-	-	-	-	-	-	-	-	-	-
e. Delete	-	-	X	-	-	-	-	-	-	-	-	-
f. Multi-Hat	-	-	-	X	-	X	-	X	-	-	-	-
g. Send	-	-	X	X	X	X	X	X	-	-	-	-
h. Synopsis	-	-	-	X	X	-	-	-	-	-	-	-
USED	X	X	X	X	X	X	X	X	-	-	-	-

X = Primitive operation is used.
- = Primitive operation is not used.

The results of the preceding five sections are summarized in Table 5.6. It shows that the application packages can make extensive use of the majority of the primitive operations found at the conceptual level. This chapter has not tried to show all the possible application operations and their use of the conceptual level. Instead a fairly representative and basic set of operations was discussed. The actual list of operations included in each IASS application will be a very implementation dependent issue.

Table 5.6 - Application Intersection Overview.

	PRIMITIVE OPERATIONS											
APPLICATION	I	M	D	P	S	UN	SD	CP	IS	QT	JN	NJ
1. ED & WP	X	-	-	X	X	-	X	-	-	-	-	-
2. DBMS	X	X	X	X	X	X	X	X	X	X	X	X
3. Spread Sheet	X	X	X	X	X	X	X	-	X	-	-	-
4. Form Gen.	X	X	X	X	X	X	X	-	X	-	-	X
5. Mail	X	X	X	X	X	X	X	X	-	-	-	-
TOTAL	5	4	4	5	5	4	5	2	3	1	1	2

X = Primitive operation is used.
- = Primitive operation is not used.

VI. USER INTERFACE

The Integrated Application Software System (IASS) combines the capabilities of the five software application packages described in Chapter 2. Each of these applications performs a set of functions on an associated logical file type. Integration of these distinct systems is brought about by determining the set of common functions performed by each system on the associated logical file type, defining a common data object to represent the logical file types, and finally defining a system of primitive operations on the common data object.

The previous chapters have covered the integration approach at the system level, where it is invisible to the user. Another integration level is vital to the IASS design and it must occur where the user interfaces to the IASS's applications. The result of integration at the user interface will be a system-wide, common set of user operations and associated commands, and a regular display organization of the logically distinct file types at the user interface. This is probably the most important command interface since it is the one the user can actually perceive and evaluate.

Implementation of the IASS requires designing for one conceptual display model using a single physical organizational schema. The IASS will therefore ensure minimal system complexity at both the user and system interface. The basic IASS hierarchy is depicted in Figure 6.1.

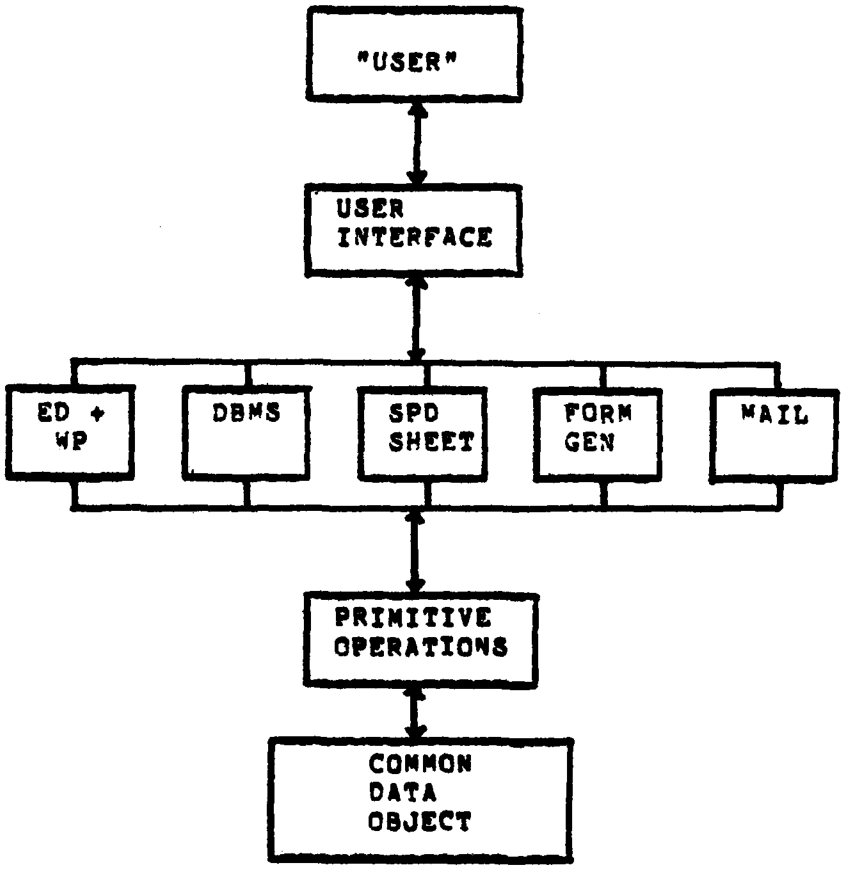


Figure 6.1 - IASS Hierarchy

A. USER INTERFACE MODULE

The interface between the user and the system couples the user to the applications of the IASS, and allows control of the operation of the system. The User Interface Module (UIM) is an abstraction to contain the features of the IASS which the user can assume are present in any context of system use. The UIM encompasses both the environmental and operational assumptions of the system.

The UIM is not a stand-alone entity. It depends heavily on each application program to interpret the user input, and to determine the appropriate display structure. It is the application program that translates the common UIM commands into a series of application operations that may, or may not, make use of the conceptual level.

1. UIM Display Format

The point of observation of the system is the terminal screen, which is a finitely dimensioned "window" into the logical file or data object in reference. For all applications the display size is not limited to the size of the screen window. If the display is large, the user will be able to use the UIM to maneuver the window over the display to accomplish the desired tasks. The UIM does not control the type of display presented to the user, since this is an application dependent requirement. However, the UIM does give the user a consistent method of interacting with the display as the applications change.

2. UIM Command Line

The UIM uses a common display organization which is augmented by the specific application program in use. In any context of use, the top line of the screen is dedicated for presentation of the UIM command line. The next line will always be used by the application program for its command line. In this manner, the top two lines of the screen will always contain system information and prompts for the user.

3. UIM Editing

From the results of the requirements analysis phase of this study, the function which is common to each utility is that of making changes to a referenced file, i.e. editing. All the UIM really provides the user with is screen editing features. It will be the application program that interprets the UIM commands into more complex operations. Editing should be done on-screen, with immediate feedback available to the user. For user protection, all editing will be done on a copy of the designated file and only committed as a permanent modification when explicitly directed by the user. The common set of editing commands provided by the UIM may be augmented by functions from the underlying application program to ensure that special editing actions, which are specifically associated with a logical file type, are executable.

4. UIM Commands

The operational assumptions of the IASS UIM compromised a set of user commands. These commands can be assumed to produce a similar effect when executed in any context of system use. It is the purpose of each application to provide the appropriate translation between the UIM commands, the display of the data object, and its conceptual representation.

Based on the categories of commands defined in Chapter 2, there are three major categories of commands in the UIM. The first is "Movement" commands that control the position of the cursor on the screen, and the location of the screen in reference to the application display. The second category, is "Editing" commands that make additions, deletions, and/or changes to the data in the display. The third category is "System" commands that are not used by the application, but by the IASS to perform its "housekeeping" functions. There are many different methods for attempting to present these categories to the user, and they are all implementation dependent. For the purpose of this thesis a possible listing of standard UIM commands is given in Table 6.1.

All of the basic categories presented in Chapter 2, with the exception of the <Modify> command, are directly implemented by UIM commands. There did not seem to be any simple way, short of accepting a complex command syntax, to implement one UIM command per category.

Table 6.1 - UIM Commands

UIM CATEGORY		UIM COMMAND	CHAPTER 2 CATEGORY
Movement	1.	Move cursor - right - left - up - down - top of screen - bottom of screen - start of file - end of file	Move
	2.	Scroll Screen - right - left - up - down	Move
Editing	3.	Find Object	Move & Retrieval
	4.	Insert Object	Insert
	5.	Copy Object	Insert
	6.	Add Object	Insert
	7.	Move Object	Delete & Insert
	8.	Delete Object	Delete
System	9.	Enter Application	Create
	10.	Output	Retrieval
	11.	Save Changes	-
	12.	Quit Application	-

The <Modify> category did not need a command of its own because it became apparent that the application itself will be able to determine when modification is necessary based on use of the UIM. In Section (B) which follows, each of the five applications will be covered with the intention of showing how they use the standard UIM commands to implement the command categories of Chapter 2.

B. APPLICATION PROGRAM LEVEL

The Application Program Level (APL) consists of the actual application programs that interface to the user through the UIM, and operate on the common data object by the primitive operations available at the conceptual level. Each application program has a specific user support function and although it uses the common UIM it has certain features that make it unique. Each of the IASS application programs will be discussed in the following sections.

Four areas of each application will be discussed. First, the display format used by each application to take advantage of the features of the UIM. Since the screen will be the common input/output medium for the IASS it is important that the perceived use of the display be similar between applications. Second, the editing features of the application. Again, as this is part of the common interface medium for the IASS it is important that the user perceive a similarity between editing features in each application. Third, specialized "functions" of the application. These are the unique features of each application that will differentiate them from other included applications.

Fourth, use of the standard UIM commands by the application. This will show how the application will use a standardized command to perform an application specific operation.

1. IASS Editor/Word Processor

The IASS text table is not limited by design to supporting any one type of editor and word processor arrangement. However, as was pointed out previously, the on-screen editor with on-line formatting is rapidly becoming the accepted standard and any regression to line oriented editors, or the like, would meet with serious user resistance. For this reason an on-screen editor with on-line formatter is assumed.

a. E/WP Display Format

The Editor/Word Processor (E/WP) has a single display called "page" format, and Figure 6.2 provides an illustration. The screen represents a window over the sheet of paper on which the text is being written. What the user sees through the window is the format that will actually be output. The default Word Processor (WP) settings will match the size of the E/WP display to the size of the screen (i.e. no part of the text will be hidden off either side of the screen). However, the user is free to issue WP commands that will result in a display larger than the screen. In this case the user will have to use cursor movement commands to bring the off-screen portions into view.

```
 (user entered command)  R## C##  (Textfile Name)
...*....2....*....3....*....4....*....5....*....6.

   If you're not using the standard library, or if
will have to construct calls to other programs usi


through temporary files. Here it is natural to mak
```

Figure 6.2 - "Page" Display Format for Editor/Word Processor.
(Display size is larger than screen size.)

b. E/WP Editing

All text files make use of the Word Processor (WP), and it is always on-line. At entry the WP is set to certain default output values for page length, line size, line spacing, left margin, right margin, and tab size. The user can accept these default settings or can issue commands to the E/WP to change them. The E/WP continuously scans the text stream, recognizes the commands, and immediately changes the output format.

Editing is accomplished by positioning the cursor at any point on the screen and typing in an entry. The entry is inserted between the present contents of the location. Deletion and modification are accomplished in a similar on-screen manner. Only commands that make use of large objects, like blocks or other files, need resort to a command-line format. As the E/WP formatter is always on-line, the user does not have to worry about starting new lines, indenting, line spacing, new page, and other similar format considerations. The E/WP automatically does it for the user, but the user can still take manual control if required.

To assist the user in certain editing situations the E/WP will provide a certain number of buffers for the user to treat as temporary storage areas. Actually they are temporary text files created by the E/WP and are destroyed when the session is ended.

The E/WP actually operates on a copy of the original file, so any changes made during a session only become permanent when the user issues a "save" command. If the user quits without saving, then it is as if the session never occurred.

c. E/WP Functions

(1) Formatting Commands

A series of specialized Word Processing commands must be included in the E/WP application to allow the user to modify the format of the text. These commands must be structured in such a way so that they are readily distinguished in the text stream and will not be mistaken for text.

d. E/WP Commands

(1) Find

Used to locate character or string sized objects specified by the user. The located object is displayed by scrolling the window to its location, and the user is notified if another match exists. This operation will use a command-line format.

(2) Insert

Used to place any valid object at a point referenced by the cursor. The present contents are not overwritten, merely pushed aside to make room for the new object. This operation will use an on-screen format. It is possible to insert large objects, such as buffers or other text files, into the file being edited, but this will require the use of a command-line format.

(3) Move

Used to change the location of line or block size objects specified by the user. The object is deleted from its present location and then inserted in the new location. This operation will require the use of a command-line format.

(4) Delete

Used to remove any object (character, line, block) specified by the user. The object is normally not saved, but if the user specifies it can be saved in a buffer. This operation will require the use of both on-screen and command-line formats.

(5) <u>Copy</u>

Used to duplicate an object and then insert the duplicate at a specified cursor location or into a buffer. This operation will require the use of a command-line format.

(6) <u>Add</u>

Used to create a blank line(s) on the display, above or below the cursor position. This operation will require the use of an on-screen format.

(7) <u>Output</u>

Used to send the file to some designated output device. The output format of the file will exactly match the format seen on the display during editing.

2. IASS Database Management System

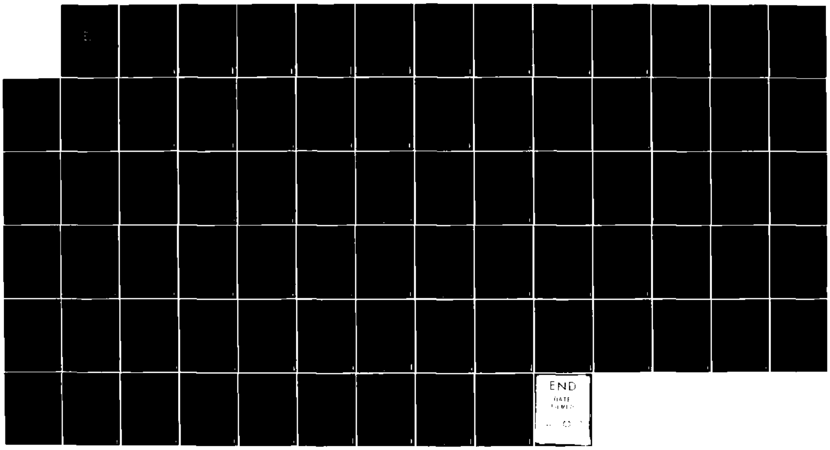
The main intention of the Database Management System (DBMS) is to emphasize simplicity. It does not make sense to follow the lead of many currently available DBMS packages that add large amounts of complexity to gain minor and seldom used capabilities.

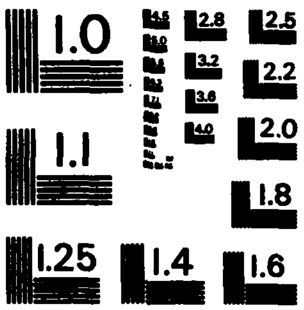
a. DBMS Display Format

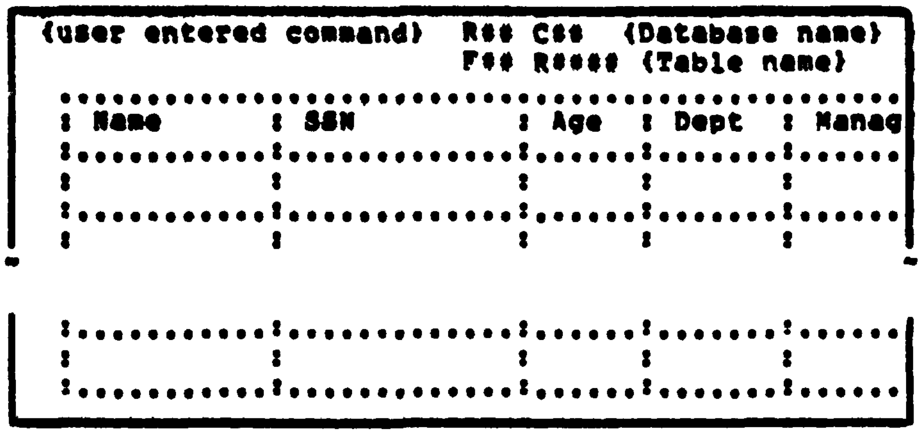
There will be two formats in which database tables will be displayed. The first, and default format is called "table". The table will be displayed in a tabular, row and column, format for the user to edit. Figure 6.3 illustrates the "table" display format.

AD-A126 434   INTEGRATED APPLICATION SOFTWARE SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA   J C WATERS DEC 82   2/2

UNCLASSIFIED   F/G 9/2   NL







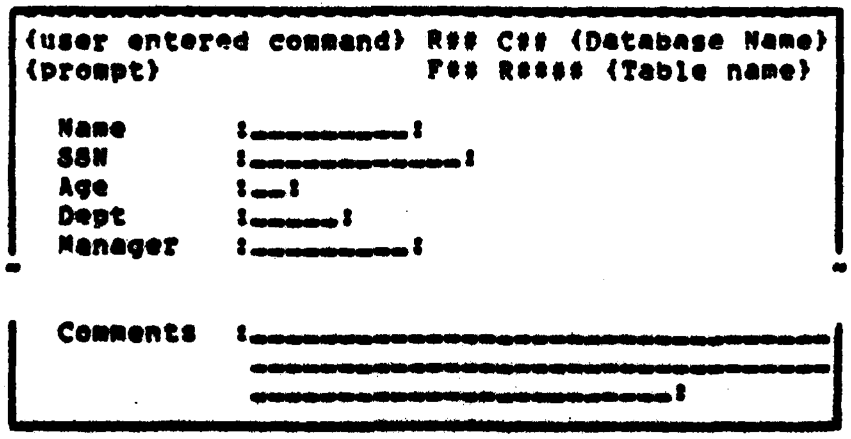
```
(user entered command)  R## C##  (Database name)
                        F## R#### (Table name)

 ..............................................
 : Name      : SSN        : Age  : Dept  : Manag
 :...........:............:......:.......:......
 :           :            :      :       :
 :...........:............:......:.......:......
 :           :            :      :       :


 :...........:............:......:.......:......
 :           :            :      :       :
 :...........:............:......:.......:......
```

Figure 6.3 - "Table" Display Format for DBMS-Table.

The second format will be "page", and it displays the records in a vertical format with no portion of the table lying off-screen to the sides.



```
(user entered command) R## C## (Database Name)
(prompt)               F## R#### (Table name)

  Name        :__________:
  SSN         :_____________:
  Age         :__:
  Dept        :______:
  Manager     :__________:


  Comments    :________________________________
               ________________________________
               _________________________:
```

Figure 6.4 - "Page" Display Format for DBMS-table.

Instead, each field has at least one line on which to display its contents, and can have more than one line if needed. The user will scroll in a vertical direction to view the entries in the table. Figure 6.4 illustrates the "page" format.

b. DBMS Editing

The DBMS makes use of the UIM described in section (A), above. However there are some constraints. The user is not free to write anywhere on the screen, but is limited to writing within field positions. Field positions are the bounded locations at column and row intersections for the "table" display, and the locations between the delimiters ":" for the "page" display. Entries cannot be made to pass through field boundaries, and the user will be prompted if it is attempted. To cross such boundary requires the use of a cursor motion command.

The UIM commands will be used to edit the displayed table. Cursor motion keys will be used to position the cursor in a valid field position. Text will be inserted, or deleted, inside the field position. The user will be allowed to scroll across the screen in all four directions, up to the limit of the size of the table.

Records are added to the table by editing a "blank" tuple. Blank tuples are found automatically at the end of the table display, or created by using the "add" command to insert them above, or below, the present cursor position.

The IASS provides a degree of protection to the user in that none of the editing changes (record insertions, modifications, or deletions) are made until the user isses a "save" command. At that time the original table is removed and the copy on which the editing was done takes its place. If the user quits without saving, then none of the editing changes are made.

c. DBMS Functions

(1) <u>Relational Operators</u>

The following relational operations described in Chapter 4, Section (A) are available to the user in the DBMS. The operators are: MODIFICATION, DELETION, PROJECTION, SELECTION, UNION, SET DIFFERENCE, CARTESIAN PRODUCT, INTERSECTION, QUOTIENT, JOIN, and NATURAL JOIN. The INSERTION operation is not included since it is already performed by the UIM, and the MODIFICATION and DELETION operands are for mass operations where using the UIM modification and deletion capabilities would be too difficult. All these operations would require the use of a command-line format. The unary operations (MODIFICATION, DELETION, PROJECTION, and SELECTION) must be used with a DBMS table already in reference. The other binary operations will have to specify the DBMS tables involved as part of their command-line format.

(2) Arithmetic

In support of conditional statements and query processing the DBMS will have to include a basic arithmetic capability. The following operations are needed: Addition, Subtraction, Multiplication, Division, Equals, Greater-Than, Less-Than, Greater-Than-Or-Equal-To, and Less-Than-Or-Equal-To. These are not intended as stand alone operations, but are necessary to the successful performance of other operations.

(3) Aggregate

In support of conditional statements and query processing the DBMS will have to have a basic aggregate function set. The following functions need to be included: Total, Count, Max, Min, and Average. These are not intended as stand alone operations, but are necessary to the successful performance of other operations.

(4) Query Processing

In support of user defined questions across multiple DBMS tables, there must be a capability similar to the "Find" command, only not limited to a single table. The result of a query would be a new table containing the desired information, but no change would have been made to the tables that provided the information. A command-line format would be required for this operation.

d. DBMS Commands

(1) Find

Used to bring a tuple, from the single table in reference, that meets a specified condition onto the screen by scrolling. Additionally, the user is notified if another tuple meets the same condition. This operation will require the use of a command-line format.

(2) Insert

New tuples are added to a table by editing "blank" tuples. Blank tuples are automatically found at the end of the table, or are created by using the "add" command to insert them above, or below, the present cursor position. This operation is performed in an on-screen format.

(3) Delete

By using the UIM editor features a single tuple can be deleted, in an on-screen format. However, for multiple tuple deletions a command-line format would be required.

(4) Add

As an aid to the user, the add command will place a "blank" tuple either above or below the tuple referenced by the cursor. This operation is performed in an on-screen format. If no entries are made in the blank tuple, it is ignored by the DBMS application.

(5) Output

Used to send the contents of a table, or tables, to some designated output device. The output format always defaults to the "table" display. If more sophisticated output formats are desired then a Form Generator table can be specified to control the output.

3. IASS Spread-Sheet

The spread-sheet application is a fairly well defined and conceptually simple program. Its functions are well defined and it is not expected to react to unexpected demands.

a. Spread-Sheet Display Format

Spread-sheet has a single display format and it is called "table". Figure 6.5 illustrates the "table" display. Only the value of each entry position is displayed, not its functional content. The spread-sheet "table" display does not have a direct correspondence to the spread-sheet data table described in Chapter 3, Section (C) since there is no need for the user to be concerned with how the application program must use it.

b. Spread-Sheet Editing

The spread-sheet makes use of the UIM described in Section (A), above. However, this application makes use of two cursors which are related to each other. The "position" cursor is the lower one, and it moves across the columns and rows of the "table" display. Its function is to indicate which entry position is currently being referenced by the application.

```
{user entered command} R## C## {spread-sheet name}
{prompt}               R## C##
.................................................
:  :  A  :  B  :  C  :  D  :  E  :  F  :  G  :  H
:..:.....:.....:.....:.....:.....:.....:.....:....
: 1:     :     :     :     :     :     :     :
:..:.....:.....:.....:.....:.....:.....:.....:....
: 2:     :     :     :     :     :     :     :


:..:.....:.....:.....:.....:.....:.....:.....:....
: 9:     :     :     :     :     :     :     :
:..:.....:.....:.....:.....:.....:.....:.....:....
```

Figure 6.5 - "Table" Display Format for Spread-Sheet.

The UIM cursor motion commands control the "position" cursor and allow it to roam over the entire spread sheet. The "command" cursor is always in the command line area of the screen, and is used to write into the entry position marked by the "position" cursor. It is on the command line that the functional contents of the referenced entry position are displayed.

c. Spread-Sheet Functions

(1) <u>Arithmetic</u>

Since the Spread-Sheet is a numerical modeling tool it will need a substantial arithmetic capability. The following operations are needed; Addition, Subtraction, Multiplication, Division, Exponentiation, Absolute Value, Truncation, Rounding, Logarithms, and Trigonometric Functions. These operations must be capable of stand alone operations similar to those in a calculator, and be capable of inclusion in other operations and conditions.

(2) Aggregate

Since the Spread-Sheet is a numerical modeling tool it will need a substantial aggregate modeling capability. The following operations are needed: Total, Count, Maximum, Minimum, Average, and Net-Present-Value. These operations will not have a stand alone capability since they are intended for inclusion in other operations.

d. *Spread-Sheet Commands*

(1) Find

Used to find those entry positions, in the referenced spread-sheet, that meet some specified condition. The cursor will be placed on the first such entry position and a prompt generated to show if there is another. This operation will require the use of a command-line format.

(2) Insert

Used to place the contents of another spread-sheet alongside the current spread-sheet at the indicated edge. This operation will require the use of a command-line format.

(3) Move

Used to change the current position of an entire row or column on the spread-sheet. This operation will require the use of a command-line format.

(4) Delete

In reference to a specific entry position, it sets the value to null. For rows or columns it totally removes them and moves the surrounding rows and columns to fill the gap. This operation will require the use of a command-line format.

(5) Copy

Used to duplicate a row, column, or specific entry position at another referenced location. This operation will require the use of a command-line format.

(6) Add

Used to place a blank row or column in a location referenced by the present cursor position. This operation will require the use of a command-line format.

(7) Output

Used to send the contents of the spread-sheet to some indicated output device. The user can indicate whether to send the spread-sheet display, which only contains the entry position values, or the contents of the actual spread-sheet table, which contains both the value and the function, to the output device. Subparts of the whole spread-sheet may be indicated for output.

4. <u>IASS Form Generator</u>

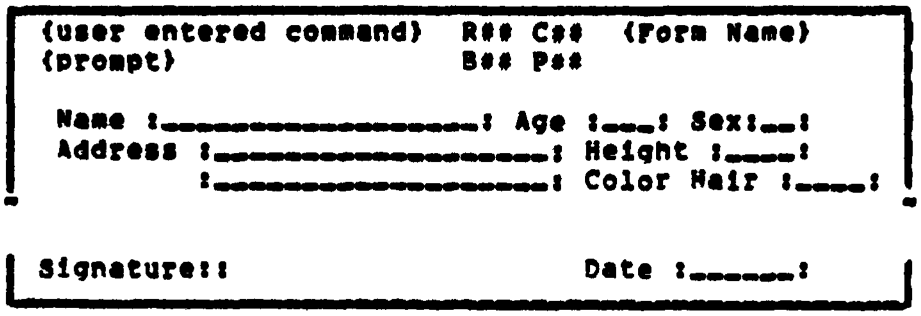
The Form Generator will be an important part of the IASS since it is reasonable to expect other applications, the DBMS for example, to make use of it to support their operations. It also has two modes of operation. "Design" time is when the new form is prepared by the user and all of its parts, called "blocks", are positioned and identified as to their function. "Use" time is when the previously designed form is called on to output the specified information in the prescribed format.

a. Form Display Format

There is one available display format for the Form Generator, called "page", and it is shown in Figure 6.6. This displays the form in the actual format for use-time. The prompts for each block are shown as well as their associated entry positions. The "function" and "i/o" values of each block appear on the command line when the block is referenced by the cursor.

b. Form Editing

In "page" format the user begins with a blank screen and is free to move the cursor to any position and make entries. Form editing is a much more formal procedure than in any of the other applications. Entries must consist of a set number of parts to be accepted by the system. First, a prompt of zero or more characters, which will appear on the display.



```
{user entered command}  R## C##  {Form Name}
{prompt}                B## P##

 Name :________________: Age :___: Sex:__:
 Address :_________________: Height :____:
         :_________________: Color Hair :____:


Signature::                   Date :______:
```

Figure 6.6 - "Page" format for Form Generator.

Second, the number of spaces reserved on the form for the entry, which will also appear on the display. Third, symbol indicating how this entry will be used by the form generator (input, output, call to text file, etc.). Fourth, the query statement upon which the output is based, the table and field name where the input is to stored, or the name of the text file to be output.

Extensive use is made of the command line while filling in each block. Only the first two parts of a block entry are shown on the actual form. The other two parts are displayed in the command line area when the block is referenced.

As in previous applications the actual changes made during editing are not effective until the user issues a "save" command.

c. Form Functions

(1) Arithmetic

Since the Form Generator can make use of DBMS queries and condition statements in acquiring the information to complete a block, it is necessary to provide basic arithmetic support. The following operations should be included: Addition, Subtraction, Multiplication, and Division. These are not intended as stand alone operations, but for inclusion in other operations.

(2) Aggregate

As the Form Generator can make use of DBMS queries and condition statements in acquiring the information to complete a block, it is necessary to provide some aggregate function support. The following functions should be included: Total, Count, Maximum, Minimum, and Average. These are not intended as stand alone operations, but for inclusion in other operations.

(3) Usage Indicators

Since the form and its blocks must be capable of supporting a wide range of uses, each form is tailor made by the user. The purpose of each block must be indicated by the "i/o" field in a manner that shows how the "function" block will be treated at "use" time. For example: The function field of an input-block might tell where the item is to be stored. For an output-block it may specify the database table and the query operation on it, necessary to get the item. For a text-block it may specify the text file that will be inserted in the form at that location.

d. Form Commands

(1) <u>Find</u>

Used to find a block object that meets a specified condition by moving the cursor to its start. The user is notified if there are any more blocks that meet the condition. This operation requires the use of a command-line format.

(2) <u>Insert</u>

New blocks are inserted by editing the blank area of a line in the proper manner as described in Subsection (b) above. This is done in an on-screen format. Other forms may be inserted into the present form at a completely blank line, but requires the use of a command-line format.

(3) <u>Move</u>

Used to move a display line(s), and the blocks on it, to a new position on the form. This operation requires the use of a command-line format.

(4) <u>Delete</u>

Used in reference to blocks, it eliminates the block and leaves the space on the line blank. In reference to lines, it eliminates all blocks on the line, removes the blank line, and all lower lines move up. This operation is performed in an on-screen format.

(5) <u>Copy</u>

Used to duplicate a line, or lines, on the form but in a different location. This operation is performed in a command-line format.

(6) <u>Add</u>

Adds a blank line, above or below the line referenced by the cursor. This operation is performed in an on-screen format.

(7) <u>Output</u>

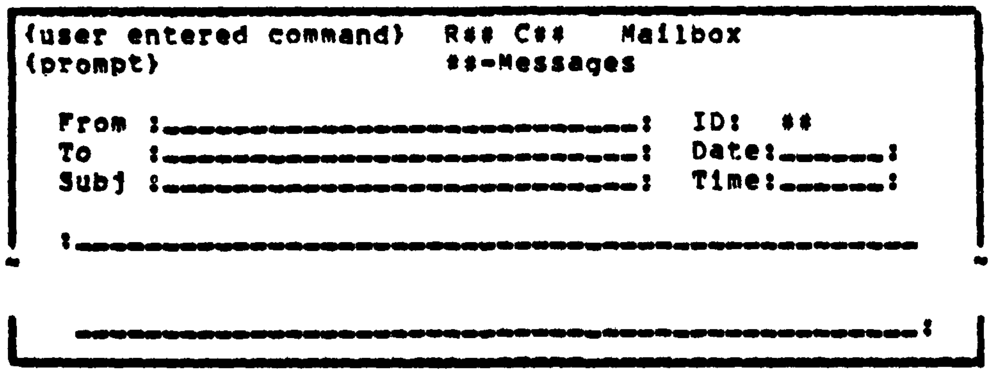
Used to send the contents of the form table to an indicated output device. The user can send the displayed version, as shown in Figure 6.6, or can opt to output the entire form table in tabular format so as to see all the information associated with each block.

5. IASS Electronic Mail

The Electronic Mail package supports the user in sending messages to other users, for reading at a later time. Upon entry to the IASS the user will be prompted if there is mail in the mailbox. By entering the Mail package the user will be greeted by a one line display synopsis of each message, which cannot be edited. The standard mail display format will be entered and the user will be free to read, edit, and/or delete current messages as well as compose new ones. Each message has a unique ID number and the user can refer to messages by the ID, originator, subject, or time-stamp. Outgoing messages are actually sent when the user leaves the Mail application, by removing all messages in the user mailbox that are not addressed to the user and routing them to their proper destination.

a. Mail Display Format

There is one display format available for Mail, and it is called "page" format. Figure 6.7 is an illustration of "page" format. Each message is displayed on the screen with its fields organized in a vertical direction. Each field has an associated entry position that is designated by delimiters.



```
(user entered command)  R## C##   Mailbox
(prompt)                   ##-Messages

  From :___________________________:  ID:  ##
  To   :___________________________:  Date:______:
  Subj :___________________________:  Time:______:

  :_____________________________________________
~                                                   ~

   _____________________________________________:
```

Figure 6.7 - "Page" format for Electronic Mail.

b. Mail Editing

The Mail package makes use of the standard UIM described in Section (A), above. The user may perform the standard editing functions on actual messages or on message "blanks". In "page" style display the user moves between messages by using the "scroll" command. On each message the user may perform editing operations in any of the entry positions. Movement between entry positions is possible only by using cursor motion keys.

Outgoing messages are created by editing one of the message "blanks" at the end of the table, or by using the "add" command to insert a "blank" message after the current one and then editing this "blank". Additionally, editing the contents of the "To" field in a current message, so that it no longer corresponds to the present user, turns the message into an outgoing one.

All editing changes are not actually implemented until the user issues a "save" command. Outgoing messages are sent when the user issues a "quit" command to leave the mail package. At that time the system finds all messages that are not addressed to the current user, updates the time-stamp on them, and then sends them to the appropriate user. Since users often have collective names, such as "oversight committee", that include more than one user, there is a special character tacked on to the standard destination address to indicate that the message is to the other users in that collective address.

c. Mail Functions

(1) <u>Multi-Hat Name Designator</u>

A special character that is placed in front of a name that corresponds to more than one user name. The multi-hat name actually refers to a database table that contains the names of the users who constitute the multi-hat name. At use time, the system will strip the multi-hat name from the message, make the proper number of copies of the message, insert the proper user names, and send the messages.

d. Mail Commands

(1) <u>Find</u>

Used to move the display to the message that meets a certain condition (e.g. From = 'Boss', Time < '2 Nov', Subj = 'Schedules'). Additionally, the user is notified if there are more messages that meet the condition.

(2) <u>Delete</u>

Deletes the message being referred to by the cursor. To delete multiple messages it can be used in a command line with a condition statement.

(3) <u>Copy</u>

Given a message object it will duplicate the object and insert it into the mailbox.

(4) <u>Add</u>

Places a blank message above, or below, the message being displayed.

In concluding Chapter 6 it is important to emphasize that the UIM is a very implementation dependent part of the IASS. What this chapter attempted to demonstrate was that the command categories defined in Chapter 2 could be implemented by a common, yet simplified, user interface by using a small command vocabulary coupled with a common display and editing format. This is not the only way to present the user interface, only a suggestion.

VII. CONCLUSION

The preceding six chapter have attempted to lay the groundwork for the possible design and implementation of what has been called an Integrated Application Software System (IASS). This thesis is the first small step toward the study of such a system, and the majority of the work remains to be done.

This thesis approached the topic from a broad perspective and did not seek to get down to specific implementation issues. Instead Chapter 2 reviewed the apparent characteristics of five application programs, and the appendices provided more detail on each. Chapter 3 took the characteristics of the logical file type associated with each application and formed them into a common data object. Chapter 4 took the common data object and explained a set of operations on it. Chapter 5 described how each of the included applications might interface to the common data object by using the operations of Chapter 4. Lastly, Chapter 6 attempted to illustrate how the user would interact with the applications in the IASS through a common interface.

One point must be emphasized and it is that an IASS is not a relational Database Management System (DBMS). There are enough DBMS applications already proven and available on the market. See Appendices (E) and (F) for two examples.

What the IASS does is try to use the DBMS approach to invisibly support the user's effort to utilize the various and unique applications. The IASS conceptual level is a common bond between all included applications and while it is the heart of the system, it should be kept hidden from the user, except in specialized applications like the DBMS.

If the user is always given direct access to the conceptual level and its operations, then the IASS is nothing more than a DBMS. In fact, such a capability is already present in the DBASE II system, Appendix (E), although it would be greatly improved by incorporating some of the better presentation ideas from the SEQUITUR system, Appendix (F).

It must be emphasized that this thesis is a limited, and very subjective, view of the IASS. From the study of this hypothetical IASS it seems clear that such a system could be implemented. However, no specific estimation can be made on the effectiveness or efficiency of such a system. It would be reasonable to expect the efficiency to be less than that of the individual application packages, but there is no way of determining how much less. These are very important considerations and will have to be studied before the true usefulness of an IASS can be estimated.

Much effort was placed on the conceptual level of the IASS, and yet it seem certain that the user interface will be the portion of the IASS that will determine its success or failure as an actual system. It is important to define the ultimate goals of an IASS in realistic terms so that an measurable objective exists. After some study there appear to be two important goals in the IASS design.

The first is to reduce the cost of owning the separate application programs by combining them into one IASS. Since it has been shown that the five given applications have much in common that can be factored out and placed in a common conceptual level, it would appear reasonable to expect the same from any future applications accepted for inclusion. This common conceptual level reduces the amount of duplication necessary to "own" the individual applications. Economic savings would hopefully be realized from the smaller amount of code needed, its more uniform structure, and the sharing of capabilities between applications. The design of the conceptual level and the individual application packages will have the major effect on achieving this goal.

The second goal is that the user must perceive an improvement in using the IASS over using the separate applications. The IASS must be more "user friendly" than the disjoint application programs it replaces. Simplicity and capability must be emphasized over system sophistication. Each capability that will be incorporated in an application must be measured as to its complexity and usefulness. It is not justifiable to increase system complexity just to add a fancy but little used feature. The design of the user interface module and the individual application packages will have the major effect on achieving this goal.

It would appear to be too early to attempt the implementation of such a system. Instead, more investigation needs to be done and the objectives more tightly defined.

APPENDIX A: WORD STAR

WORD STAR is a word processing program developed by Micro-Pro to combine the capabilities of a screen editor and an on-screen text formatter. The result is a very powerful text editor which displays the referenced file as it will appear on the printed page.

WORD STAR is primarily menu-driven. The commands which are presently valid are displayed in a menu, and are executed by keystroke combinations. On-line information is available to the user concerning many other aspects of WORD STAR. The menu driven feature eases user initiation to WORD STAR and is part of the Help facility. The level of help is selectable to match the users level of experience, and determines the extent to which the menus are displayed on the CRT.

WORD STAR is composed of a set of seven hierarchically organized menus or environments, as shown in Table A.1. The user enters WORD STAR in the No-File environment. At this point there is no file in reference, the object granularity is the file, and the menu options include commands to: change the logged disk drive, set the automatic directory display feature (on/off), set the help level, print a file, rename a file, copy a file, delete a file, run a program, open a document file, and open a non-document file.

Table A.1 - WORD STAR Menu Hierarchy.

LEVEL	MENU
1	No File
2	Main Menu
3	a. Help b. On-Screen Format c. Print Control d. Quick Edit e. File/Block

WORD STAR recognizes two types of files, "document" and "non-document". A document file can either be a text file processed by a word processor or a program run by a computer. A non-document file is a special purpose file which is used by another software product, and will not be discussed further.

The on-screen editor and formatter are invoked by selecting the menu option to open a document file. This causes WORD STAR to enter the Main Menu environment with a specific file in reference. If the file previously existed it is made current, otherwise a new file is created and made current. On entering the Main Menu environment, a status line and a rule are initialized. The status line contains information about the system - the name of the file, the page within the file, the column and row number the cursor is at, and the insertion mode (on/off). The rule indicates the right and left margin position as well as the tab positions. The Main Menu represents the basic file editing environment where the user will remain until it is decided to quit the current file and return to the No File Menu or the operating system. In any case, WORD STAR does not permit lateral movement between the sub-menus of the Main Menu.

A useful feature WORD STAR employs is "word wrap". With word wrap, the user does not have to insert carriage returns at the end of each line. As the text overruns the end of the line, WORD STAR automatically starts the next line. In this way, the user merely inputs an entire block of text as a continuous ASCII character string, and leaves the formatting to the system. In the Main Menu, the user can edit the file in granularities of character, word, and line. Insertion is a "toggled" operation (on/off), where the user is either in insert mode or overwrite mode. Any keystroke entered is either inserted in the text at the cursor position, shifting characters to the right to accommodate it, or overwrites the character at the cursor position. To facilitate on-screen editing, the Main Menu contains commands to control cursor movement and to scroll the screen. It is possible to insert tabs or end-of-paragraph markers. There is a "Find and Replace" command which can be repeated any number of times. Deletions can be done on a single character, a word, or an entire line. The Main Menu also contains options to select one of the five submenus.

The Quick Editing environment supports editing on higher levels of abstraction of text objects than the Main Menu. There are additional cursor movement commands to give a wider range of control and granularity. As in the Main Menu environment, the user can scroll the display, but now it is continuous at nine user selectable rates until stopped by command. Insertions are accomplished in the same way as in the Main Menu environment, but deletions are possible on a wider range of objects. There is a feature to allow a command to be repeated at one of nine user selectable rates, until stopped by command.

The Block environment provides the user a set of operations on a block of text. WORD STAR considers an entire file to be a special case of a block of text. Files can be saved by several menu options: save and resume the referenced file, save and quit to the operating system, save and exit the referenced file, and copy to another file. Files may also be renamed, deleted, printed, or quit without saving changes. To support these file operations, the Block Menu contains options to change the logged disk, and to turn the automatic directory listing on or off. In this capacity, the Block environment is used as a successor to the Main or Quick Editing environments after the cursor is positioned. Blocks in a file must be marked by the user.

As a delimited aggregation of text, a block can be moved within the same file. Copying blocks of text can either be within the referenced file or between the referenced file and an external file. Block copying between files are bi-directional. Copying a block to an external file entails overwriting an existing file or creating a new file. Copying a block from an external file entails moving the entire external file to the point in the text indicated by the cursor. Any marked block can also be deleted. As a precautionary measure, WORD STAR allows the user to hide block markers, and only blocks which are visibly marked can be deleted. In addition to a text block being organized into a continuous, unstructured string of text, WORD STAR supports a columnar organization.

The previously described menus contain operations to create, edit, position the cursor, or output a text file. The format of the file, either as it is visually displayed or printed out, is defined by a set of formatting parameters associated with the file or by commands embedded in the file. The formatting parameters associated with a file are initially set to default values and the set of embedded commands is initially empty.

Formatting in WORD STAR is primarily done on-screen with the options contained in the On-Screen Menu. The on-screen formatting commands are those whose effects can be visually displayed, and they are listed in Table A.2.

Table A.2 - WORD STAR On-Screen Formatting Commands.

1. Set left margin
2. Set right margin
3. Release margins
4. Set and clear tabs
5. Indent a paragraph
6. Create a special rule
7. Center text
8. Set line spacing

The On-Screen Menu also contains options in the form of (On/Off) toggles to control: word wrap, rule display, variable tabbing, hyphenation help, right margin justification, soft hyphen, print embedded control characters, and page break display. If an on-screen formatting operation needs to be applied to the previous contents of the file, the applicable portion of the file must be reformatted. Furthermore, these formatting parameters are only temporarily applied when the file is referenced. Any subsequent reference to a file requires that the on-screen formatting parameters be reset.

WORD STAR combines into one menu, the Print Menu, all options which create special printing effects not normally displayable on a video screen. There are options to: bold face, double strike, underline, strike out, subscript, and superscript. Since the effects of these options cannot be displayed on the video screen, a special character is used to mark the affected area. Additional special printing effects are selectable through this menu on a one time basis: overprint a character, indicate a non-break space, and overprint a line. The Print Menu also contains options which control the printer during output. The user may embed commands in the text file to cause the printer to change pitch, or cause a pause to allow the user to change the print element or ribbon.

Printing can also be directed through the use of embedded dot commands. These commands are placed in the text file and appear as regular text on the display, but are not output to a printer and force WORD STAR to change a printing parameter at print time. Dot commands alter the default parameters WORD STAR uses to format the printed page. Table A.3 provides a listing of these commands.

Dot-commands may be placed anywhere in the text, but since they are static and tend to destroy the relationship between what is displayed and what is printed, they are usually placed at the beginning of the text file. As with the options of the Print Menu, dot-command actions must be supported by the specific printer in use.

The last menu to be described is the Help Menu. Help is "on-line" in that it can be invoked at any time through the Main Menu, and is "dynamic" in that the level of help can be adjusted. The level will determine how much information is displayed when an option is selected. The Help Menu options display information on: paragraph reforming, flags in the right-hand margin, dot and print commands, status line, ruler line, how to set margins and tabs, and how to move blocks of text.

Table A.3 - WORD STAR Dot Commands.

1.	Set line height
2.	Set page length
3.	Set top margin
4.	Set bottom margin
5.	Generate headers
6.	Generate footers
7.	Set footer margin
8.	Reset page number
9.	Offset page from left side of printer
10.	Position page number
11.	Set character width
12.	Force a page break
13.	Prevent a page break

WORD STAR is an excellent and very popular word processing program. The screen-oriented and on-line formatting features are different from other systems in that they are extremely easy to use. Once experience is gained with WORD STAR it is difficult to use line-oriented editors or off-line formatting systems. The on-line help facility makes WORD STAR easy to learn and user friendly. One aspect of WORD STAR that could be considered a disadvantage is the large command set. However, being menu-driven, the commands not normally used do not have to be memorized since they are always listed in the menu.

APPENDIX B: VI

"VI" is a text editor used by the UNIX operating system and was created by the University of California at Berkeley, and Bell Laboratories.

VI (visual) is a display oriented interactive text editor with a command vocabulary size of about ninety one. The user sees the CRT screen as a window into the text file and all editing operations are immediately visible. Line numbers are not displayed and have no real use in VI, although it is possible to find out the number for a line. For the sake of protection the user does not actually edit the file, but a copy of it. At the completion of a session the user will indicate whether to keep the edited copy or the original.

There are forty seven movement commands for control of the cursor, which is the editor's point of reference, and the screen display. Scope of movement is possible over file, screen, paragraph, section, sentence, line, word, and character sized units. Up to twenty six locations in the file can be marked for later return, or specific locations found that match a desired character string. Table B.1 lists the cursor movement commands available in the VI system. Note that there is duplication, in that more than one command does the same thing.

Table B.1 - VI Cursor Movement Commands

1. Backward window
2. Forward window
3. Scroll down *
4. Scroll up *
5. Backspace one character *
6. Backspace a single character
7. Backup a word
8. Backup a word during insert
9. Backup to beginning of word
10. Retreat to previous line *
11. Retreat to beginning of sentence
12. Retreat to beginning of previous paragraph
13. Retreat to previous section boundary
14. Linefeed advance to next line
15. Advance to first non-white space on next line *
16. Advance to next line, first white space
17. Advance to next line, same column *
18. Advance to next character *
19. Advance to beginning of word
20. Advance to end of next word
21. Advance to section boundary
22. Advance to the next typed character
23. Advance to beginning of next paragraph
24. Move to previous line *
25. Move to end of current line *
26. Move to balancing parenthesis or brace
27. Moves cursor to last line on screen *
28. Moves cursor to middle of screen *
29. Move forward to beginning of word
30. Move forward to end of word
31. Move to first non-white space on current line
32. Move to line number # *
33. Search for word *
34. Search forward for string **
35. Search backward for string *
36. Search for next match **
37. Repeat last single character search
38. Find a single character, backwards *
39. Find a single character, forward *
40. Reverse direction of previous find

*, ** Useful - see page 133, paragraph (4)

Table B.1 - (Cont.)

41. Find first instance of next character
42. Repeat the last search command *
43. Homes the cursor
44. Mark the present position of the cursor *
45. Return to marked position *
46. Redraw the screen
47. Returns to previous context

* Useful - see page 133, paragraph (4)

The operations of insertion, modification and deletion are supported by thirty commands that permit the user a varied level of object control. Items that are inserted, modified or deleted are immediately updated on the screen to give the user a current view of the file status. The user also has the ability to undo the previous command if its effects were undesired. Most insertion and modification commands are structured so that they continue to operate until the user issues a command to terminate them. Normally during insertion the user has control of format in that new lines are started by entering a carriage return. However there is an option that will let VI determine when to start a new line, based on line length, and let the user just enter text as a continuous stream. Table B.2 lists the thirty edit commands.

In order to use VI the user issues the command "vi" followed by the name of the file to be edited. If this is a new file, then the name will not be found in the directory and VI will create an empty file. After entry, the user will issue cursor motion commands to maneuver through the file, and issue edit commands to change the contents of the file. There are no other modes or displays available in VI.

Table B.2 - VI Edit Command Summary

1. Insert a number of spaces
2. Insert nonprintable characters
3. Insert "shiftwidth" blank spaces
4. Insert at the beginning of line
5. Insert at end of line
6. Insert before the cursor **
7. Insert after the cursor **
8. Insert new line below current line
9. Insert new line above current line
10. Insert text below current line **
11. Insert text above current line **
12. Delete last character
13. Delete rest of the text on current line *
14. Delete character before cursor
15. Delete the following object
16. Delete single character under cursor **
17. Repeat last command **
18. Join together lines *
19. Replace single character under cursor
20. Replace characters at cursor **
21. Change the entire line
22. Change single character
23. Change the following object
24. Change rest of the text on current line
25. Undo last change to current buffer **
26. Restore current line to previous condition
27. Yank following object into buffer *
28. Yank a copy of current line into buffer
29. Repeat last text insertion
30. Named buffer specification follows *

*, ** Useful - see page 133, paragraph (4)

In addition to the two command categories already given there are additional commands of a miscellaneous nature. Table B.3 lists these additional commands.

Table B.3 - Miscellaneous VI Commands.

1.	Print file status message
2.	Clear and redraw the screen
3.	Redraw the current "logical" screen
4.	Suspend or restart output
5.	Cancel partially formed command
6.	Return to position in last edited file
7.	Reformat lines in buffer
8.	Indicate file and option manipulation
9.	Quit VI, enter line-oriented editor

Some very basic formating commands for line length and indenting are directly available. A macro creation capability is present to allow the user to create abbreviations for command strings. Table B.4 lists these formatting commands. VI makes no claim to supporting a formatting package, since the file will be output in the same format the user entered it. For special formatted output a VI generated file must be processed by an off-line word processor, like "NROFF -ME" described in Appendix (D).

VI provides a high degree of support to the user for restructuring a file, or files. There are nine buffers available for storing deleted text, and twenty six buffers to use as temporary holding spaces while reordering and editing. The text can be taken from other files and/or buffers, for use in the file currently being edited. If needed, previously deleted text from the current file can be recovered, and also other files.

Table B.4 - VI Formatting Commands.

1.	Reformatting command
2.	Shift lines left one "shiftwidth"
3.	Reindent lines
4.	Shift lines right one "shiftwidth"
5.	Prints current file contents

"VI" is a good screen oriented editor and has a wide range of capabilities, however it has some drawbacks.

(1) It has a poorly designed user interface since the command vocabulary is very large and the individual command strings are difficult to remember. There does not seem to have been much thought given to the design of the command vocabulary.

(2) It takes a fairly long time to learn the VI system and gain functional use. An on-line tutorial program is used to help beginners, since it is hard to become familiar with it on their own.

(3) VI does not inspire user confidence in that it is too easy to accidentally enter some unknown command string, and there is little correlation between what the user wants to do and the command(s) that must be issued.

(4) From personal use, about thirty three commands were considered to be generally useful (marked by * or **), and only ten of these accounted for the greater majority of all operations (marked by **). The remaining VI commands were generally treated as "window dressing" by all but the most sophisticated users.

(5) There is no help facility, of any kind, provided by the VI system. At the very least, an on-line listing of commands should be provided.

APPENDIX C: <u>EDIT</u>

EDIT is a text editor supported by the UNIX operating system. EDIT is a simplified version of another UNIX editor and contains a minimal set of operators. It is line oriented which means that the main object of EDIT is a line of text of some finite length.

EDIT merely supports text file creation and modification operations. The user inputs text into a file by lines, indicating the end of a line by a carriage return. A display of the file will show an ordered list of lines as they exist in the file. Ordering of lines is completely determined by the system and although the user can use line numbers as a reference, the line number is not directly accessible to the user to change or set. Any display of text by EDIT is done by line. Substrings can be referenced within a line, or lines. A formatted output display by EDIT can only be achieved if the user directly inputs the desired format line by line. No processing of the contents of a line is done by EDIT.

When invoked, EDIT sets aside a temporary copy of the referenced file in a working buffer. If the file does not already exist in the directory, then it is a new file and is created. The basic set of commands available to EDIT are listed in Table C.1.

Table C.1 - EDIT Command Summary.

1.	Edit a file
2.	Specify a file
3.	Append line(s)
4.	Insert line(s)
5.	Insert line(s) into an external file
6.	Insert line(s) from an external file
7.	Delete line(s)
8.	Copy line(s)
9.	Move line(s)
10.	Print line(s)
11.	Show line number
12.	List line(s)
13.	Substitute a string
14.	Search for string
15.	Undo last command
16.	Make effect of command global
17.	Move cursor
	- forward
	- backward
18.	Quit

Searching for a line has the effect of making the found line the current line. Any subsequent editing operations are done in relation to the current line. Lines can be found and displayed by line numbers, and ranges of lines can be specified. Lines can also be found and displayed forward or backward, relative to the current line. A line can be found by any substring of its contents, but the entire substring must be contained in one line. Because of this deficiency a substring may not be locatable merely because it exists in the text file. When searching EDIT will move forward or backward and will wrap around the buffer, so as to return to the starting line if the target object is not found.

New lines can be appended before the current line, or inserted after it. The user issues a command to specify that there are no more lines to add. Upon completion the current line is the last line added. Additions can also be made by moving or copying lines within the text file. Moving can be viewed as a combination of a deletion and an insertion. By specifying a range of lines to be changed, they are deleted and the system enters insert mode for the user to add the new lines. Additionally, insertions are possible from other text files.

Modifying a line is done by substituting a new string for an already existing target string on the line. If desired, the substitution can have global effect in that it will modify all occurrences of the target string on all lines.

Deletion is usually accomplished by indicating the line, or lines, to be deleted. A search command can be used with the deletion operation when the specific line numbers are not know.

EDIT protects the user from making inadvertent changes to a text file. The effects of the last executed command that effected the buffer can be reversed. Additionally, the effects of the editing session do not become permanent unless the user issues a command to make them permanent. At that point the edited copy, which is in the buffer, replaces the original file in the directory. Leaving EDIT without indicating to make the changes permanent is like the editing session never occurred.

In addition to writing a whole buffer out to the directory, subparts can be written to another text file. This is done by specifying the range of lines and the file to be written to.

The EDIT text editor is very basic which is both an advantage and a disadvantage. It has a minimal command set and therefore is easy to learn. The biggest problem is that it is line-oriented. As such, modifications are done a line at a time, where each line is a separate entity. It does not treat the file as a whole, but as a disjoint collection of lines. It imposes the idea of line numbers, which do not exist in the text file, in order to use the editor. There are fewer high level editing operations available, as compared to current screen-oriented editors, and they are limited to operating on lines and not the text file as a whole. While capable of producing satisfactory results, due to its line at a time limits, the operation becomes tedious if the file is large, and/or there are a lot of small changes which must be done. Given the advanced features of todays line-oriented editors, EDIT is a very archaic and frustrating way to create and modify a text file.

APPENDIX D: NROFF -ME

"NROFF -ME" is a text processing facility for files that are created on the UNIX operating system. It was created by the University of California at Berkeley, and Bell Laboratories. "NROFF" is a program that accepts an input file prepared by the user and outputs a formatted paper to the user's design. "-ME" is a macro package that enhances the capabilities of the "NROFF" program by adding additional formatting abilities and commands. The input file consists of the actual text entered by the user, through some editor system, and a series of embedded NROFF -ME commands.

There is a large vocabulary of "requests", which are really dot-commands consisting of a period followed by a two letter string. The basic NROFF package supports seventeen categories of commands, and has a total of eighty seven commands. The -ME package adds three categories and a total of sixty commands for a grand total of one hundred and forty seven commands. Table D.1 lists the NROFF and -ME command categories, and the number of commands in each.

NROFF -ME uses thirteen predefined general variables and twenty three predefined read-only variables to support its processing needs. The user is provided with a macro facility to define new commands in terms of the basic set of commands and operations on the variables. This allows the user to abbreviate a fairly long command stream into a single command.

Table D.1 - NROFF and -ME Commands.

COMMAND CATEGORY	COMMANDS NROFF	COMMANDS -ME
1. Font & Character Size Control	7	9
2. Page Control	7	0
3. Text Filling, Adjusting & Centering	6	0
4. Displays	0	22
5. Vertical Spacing	7	0
6. Line Length & Indenting	3	0
7. Paragraphing	0	4
8. Macros, Strings, Diversions, & Traps	13	0
9. Number Registers	3	0
10. Tabs, Leaders, & Fields	4	0
11. InputOutput Conventions	9	0
12. Hyphenation	4	0
13. Titles	3	13
14. Headings	0	6
15. Line Numbering	2	0
16. Conditional Input	8	0
17. Environment Switching	1	0
18. Standard Input Insertions	2	0
19. InputOutput File Switching	3	0
20. Miscellaneous	5	6
TOTAL	87	60

NROFF -ME is a good word processing system and it can produce some complex formatting actions. However, it does suffer from some drawbacks.

(1) Since the file is first created by the text editor and then run by NROFF, the user has a significant delay in determining if the desired format was achieved.

(2) In addition to depending on the text editor, NROFF must depend on other programs to preprocess the text file before NROFF can handle it for specialized requests. Two examples of preprocessors are packages to handle tables and complex equation symbology. While enhancing NROFF -ME's capabilities, they add more categories and commands, and increase the amount of time necessary for the user to see the actual results of commands.

(3) The user manual for the NROFF package is not presented in sufficient detail to completely understand the effect, or use, of all commands. It appears that the user is supposed to have a basic understanding of the system before reading the manuals!

(4) The command vocabulary is fairly large and they are not easy to remember. Based on personal use, only about twenty percent of the vocabulary is generally useful and therefore remembered. Table D.2 presents a simplified listing of the most used commands.

Table D.2 - Basic Commands NROFF -ME

1. Page length
2. Line spacing
3. Line length
4. Page headers
5. Indent
   - permanent
   - temporary
6. Begin next page
7. Need # lines
8. Insert # blank lines
9. Center the next # lines
10. Break
11. Define a macro
12. Fill/No-fill
13. Hyphenate/No-hyphenate
14. Underline
15. Section/Chapter headings
16. Quotations
17. Footnotes
18. Keep an index
19. Start paragraph
    - basic
    - left adjusted
    - body indented
    - numbered
20. Start display
    - list
    - block
    - floating block
    - delayed text
21. Table handler *
    - definition
    - start
    - body
    - end
22. Equation definition
23. Multiple column format
24. Default paper formats
    - thesis
25. Control constructs
    - read special variables
    - change special register
    - conditional formatting

* part of Table preprocessor

APPENDIX E: DBASE II

DBASE II is a relational database system created by Ashton-Tate of Los Angeles, California for microcomputer systems. For this review, the CP/M version of DBASE II was used, where the DBASE II program is an executable "command file" residing in the system.

The DBASE II system utilizes several different file types: database, report form, command, index, memory, and text. Each file type has a specific purpose that is identifiable by its type name. "Report form" files store the information, specified by the user, for describing the format (headings, fields, totals, subtotals, contents, etc.) in which a "database" file is to be output. "Command" files contain a sequence of DBASE II statements, commands, and control structures necessary to create a user defined view. "Index" files are a list of pointers to a specific "database" file. "Memory" files contain the values of memory variables and constants saved previously by the user. "Text" files are collections of ASCII characters for input into a "database" file, or created by output from a "database" file. DBASE II cannot directly use "text" files. Most of the files are stored in what is known as Standard Data Format (SDF), and they can be used directly by any other program that uses SDF files. Additionally, any text files in SDF can be used by the DBASE II system. The file is the largest data object supported by DBASE II which creates, deletes, or modifies the current file(s). A database file is brought into reference by user specification, and a maximum of two database files can be "open" at one time.

DBASE II can be used interactively or can be programmed to create a view of the database to support recurring applications. Regardless of method, DBASE II provides the user with the same basic high-level data definition (DDL) and data manipulation (DML) language. An English like command language with a very regular syntax is a user friendly feature of DBASE II. The commands are very powerful in that their operands and results are typically database files. The command structure is usually presented in the following form:

COMMAND [SCOPE] [CONDITION]

The scope modifier designates the number of records to be selected in response to the specific command. The condition modifier specifies a conditional statement that the record's field values must satisfy in order for the record to be included in the final result. Table E.1 provides a listing of the basic DBASE II commands, with duplicate commands having been factored out.

Table E.1 - DBASE II Basic Commands.

1. Display an expression on the screen
2. Format screen or printer output
3. Input a character string
4. Input a string to a memory variable
5. Wait for user input
6. List the records in a database
7. Display data from a database
8. Display the structure of a database
9. Rename a file
10. Erase a file
11. Generate a report
12. Execute a "command" file
13. Return from a "command" file
14. Display the contents of the memory variables
15. Store a value in a memory variable
16. Save memory variables to a file
17. Restore memory variables from a file
18. Select a specific database for use
19. Set specific DBASE II parameters
20. Abort a command

---

21. Create a new database
22. Edit a database
23. Modify a database's structure, or the contents of fields in selected records
24. Update a database from another database
25. Add data from a text file to a database
26. Copy data from a database to a text file
27. Insert record(s) into a database
28. Delete record(s) from a database
29. Unmark records marked for deletion
30. Locate a record based on key value, or condition
31. Goto a specified record
32. Move forward or backward in a database
33. Index a database
34. Sort a database based on a field
35. Perform JOIN operation on two databases

---

36. Count the number of records
37. Sum a field or subfield in a database

Default ordering for records in a database file is the sequence in which the records are entered. Ordering can be altered by inserting records into specific parts of the database, and by sorting or indexing the database. In the default order, the "database" file does not contain a recognized key.

By sorting or indexing a "database" file, keys are defined and the search time required to locate a record is reduced. Multiple indexing be done for the same database, but based on different keys. Sorting produces a new "database" file, which is a copy of the original database, only it is sorted. An "indexed" file is a virtual file of pointers to the original "database" file. Whereas lookup speed can be enhanced by indexing a database, there is overhead incurred in maintenance of the "index" file. Changes made to the original database file are not reflected in the new sorted "database" or "index" file. The original database must be sorted or indexed after each change in order to remain current.

The data definition language allows the user to define the organization of the data in a new database file by specifying the name of the database, and giving information on each of its fields (name, type, width, decimal places). The structure of a new database file can also be copied from that of another database file. Additionally, new structures can be created as the result of using the JOIN operator provided by the DBASE II system. At any time, the structure and/or contents of a file can be displayed or output. The structure of a database file can also be modified at a later time, but presents some problems in that all records currently in the database file are destroyed.

Besides using DBASE II interactively, it can be programmed in its own language through the use of "command" files. The DML statements are embedded in the file and iterative execution of DML statements are controlled by a set of DBASE II control structures (If-Then, If-Then-Else, Goto, and Do-While). "Command" files tend to make extensive use of memory variables and input/output functions which are also extensively supported by DBASE II. To create a user view the designer/programmer will edit a "command" file(s) to contain the correct DBASE II statements, commands, and control structures to manipulate the proper "database" files. The capabilities and limitations of any view is dependent on the design of the "command" file(s).

The reason for the great popularity of DBASE II is that it is a very easy database management system to learn and use. Its English-like command language is natural and user friendly. Although the command set is rather extensive, the command names accurately describe their action and use a regular syntax so they are easy to remember. The interactive nature and full screen display orientation makes user interaction simple and direct. With its set of predefined functions, input/output commands, "command" files, and programming constructs it is easy to create views for almost any application, DBASE II is a powerful relational database system yet it is obvious that the designers gave much thought to keeping it simple and did not introduce complexity for its own sake. However, there are a couple of problems with DBASE II which are worth mentioning, and they are all probably due to the justified emphasis on simplicity.

(1) At any one time, a maximum of two databases can be in reference. This limitation requires that databases be explicitly brought into and out of use. It would help if there was another method, besides using a "command" file, for performing operations on multiple tables.

(2) In modifying the structure of a database the contents are deleted. This requires that the database be explicitly saved to an external database and then be recopied back after structure modification. It is an inconvenience, to say the least.

(3) The only relational operation directly provided by the system is the JOIN command. It would greatly enhance the capability of the system to provide more of the operators.

(4) The display structure is a little bit too rigid, and the user does not have much direct control, sort of writing a "command" file, to effect the output format.

APPENDIX F: SEQUITUR

SEQUITUR is a relational database system designed by the Pacific Software Manufacturing Company of Berkeley, California.

SEQUITUR sees a database as a collection of named tables, each of which contains some kind of data related to the subject of the database. Each database has a set of system tables. The "Column" table lists the name, type, size, and display format of all columns authorized for use in the database's tables. The "Table" table lists the names of the columns that are included in each of the database's tables. Together the "Column" and "Table" tables act as part of a data dictionary system for the database.

SEQUITUR has a fairly large command vocabulary of over sixty seven commands. There are twenty five basic commands, forty two screen editor commands, and more formed by combinations of the previous commands. A multilevel "Help" facility is used to support the user.

SEQUITUR offers four kinds of help. There are status lines at the top of the screen. An "edit card" display can be called by the user in order to see a comprehensive list of cursor object and motion keys, and escape operations. The "help" command summons an on-line manual, that is preset by the user to provide no, medium, or maximum help. Lastly, there are situational help prompts that occur during the command process.

Table F.1 - SEQUITUR Basic Commands.

1. CHOOSE {database}
2. CREATE {database}
3. ADD to {table}
4. EDIT {table}
5. SHOW {table}
6. PRINT {table}
7. REPORT generator
8. FORMS generator
9. SELECT from {table} *
10. MANUAL select
11. JOIN {tables}
12. SORT {tables} *
13. UNION *
14. INTERSECTION *
15. DIFFERENCE *
16. UNIQUE rows *
17. DUPLICATE rows *
18. COPY
19. APPEND
20. REMOVE rows
21. RENAME column
22. COMPACT base
23. DUMP to {file}
24. LOAD from {file}
25. HELP from manual
26. EXIT

* = Member of SEQUITUR's "set" commands.

The twenty five basic commands cover the major operational capabilities of the SEQUITUR system. The commands are presented to the user in the form of a menu, and once a choice is made SEQUITUR enters the display mode necessary to support that choice. Table F.1 lists the basic commands, plus the command for exiting from SEQUITUR.

The SEQUITUR display modes are organized as "tables", or "pages". The table mode is similar to the approach taken by the "Query-by-Example" system (QBE), and presents the data in columns and rows with vertical lines separating the columns and indicators for new rows. Alternatively, the page mode presents the data one row at a time, with the column headings listed vertically. The user has the ability to flip back and forth between the two display modes at will.

Table F.2 - SEQUITUR Cursor Object & Motion Commands.

1.	Move cursor up one line
2.	Move cursor down one line
3.	Move cursor left one object
4.	Move cursor to next object
5.	Move cursor to beginning of object
6.	Move cursor to previous word
7.	Move cursor to end of current object
8.	Move cursor to next word
9.	Object = word
10.	Object = line
11.	Object = sentence
12.	Object = paragraph
13.	Object = view
14.	Object = page or screen
15.	Object = column
16.	Object = row
17.	Object = one character

Once in a desired display mode the user must make use of the editor commands to make changes to the table. All editor commands are single keys combined with the <Control>, <Escape>, or <Tab> keys. Table F.2 provides a list of the cursor object and motion commands available. Most operations require two commands since the object must be specified first, and then the actual operation.

Table F.3 - SEQUITUR Screen Editor Commands.

1. Delete left portion of object
2. Delete entire object
3. Delete right portion of object
4. Flips "insert" toggle
5. Shows rows marked for deletion
6. Flip "page-table" display style
7. Goto #-th object
8. Goto last object
9. Restores more recent version of row
10. Display earlier version of row
11. Executes a command
12. Search forward for column entry
13. Search backwards for column entry
14. Edit card display

The screen editor commands are used to make actual changes (additions, modifications, or deletions) to the displayed table on the screen. Table E.3 lists these commands which are used in conjunction with the cursor object and movement commands listed previously.

Additionally there are a number of miscellaneous commands that are provided to aid the user. These are listed in Table F.4.

Table F.4 - Additional SEQUITUR Commands

1. Get Edit Help
2. Scroll Forward
3. Scroll Backwards
4. Interrupt Present Operation
5. Lock/Unlock Cursor Object

There are an abundance of table types in SEQUITUR. "Virtual" tables consist of pointers to data in a "base" table(s), and are formed by conducting relational operations (e.g. JOIN) on the base table(s). Virtual tables are permanent additions to the database. All operations conducted on the virtual table effect the base table, but not all operations on the base table will reflected in the virtual table.

"Slice" tables consist of the data from a "home" table, and are formed by restricting or rearranging the columns in the home table. Actually, slice tables are just alternate ways of viewing the same home table. All operations conducted on the slice table effect the home table, and all operations on the home table effect the slice table.

"Template" tables are used to store control information on the operation(s) (SELECT, SORT, UNION, DUPLICATE, UNIQUE, INTERSECTION, and DIFFERENCE) desired to be performed on a set of "base" tables. The user specifies once the sequence of operations to be performed, and each time that result is desired the appropriate template table is called to create the desired virtual table.

SEQUITUR provides several methods of outputting data to the user:

(1) There is the "print" command which prompts the user to specify heading, page length, margins, page number, date, column/row divider symbol, etc. for either a "table" or "page" style output. The entire table is then output, one record at a time, in the specified format.

(2) There is the "form generator". The user creates a form letter or document by making an entry in the "forms" table in either "page" or "table" style, and answering several system prompts as to page size, width, margins. The form generator is intended for letter type generation since it only allows one text field in the form. All other entries are pulled from an appropriate table and the "form" repeated for each row in that table.

(3) There is the "report generator". The user creates a report table that is associated with a known data table. The report table specifies which data table columns are to be used, how they are positioned, what name they have on the

form, allotted width, and alignment. Again, the user must specify formatting items like page length, line length, margins, delimiters, and other related items. The individual columns in the report table can be marked for sorting, grouping, and/or arithmetic processing. If arithmetic processing is opted for, then another table, the "function" table is created to record what is to be done to each column - total, minimum, maximum, average, or count.

Based on a very short familiarization experience with SEQUITUR there is no doubt that it is a powerful and complete relational DBMS. However, it is not as user friendly as its advertisements would lead you to believe. Some of the problems encountered were:

(1) Too many commands to remember. This increased learning time and added to the confusion. Too many of the commands were just window dressing in that their effect could have be done using other commands. (Like the "Object =", extra cursor movement and deletion commands.) While using keys as commands leads to faster command input, it makes things more difficult when there are so many commands the symbol on the key has little or no relation to its effect.

(2) The structure of the user interface was unwieldy. It was easy to get lost and difficult to recover to a known location. Operations that worked under one condition did

not work in another, or produced completely different and unexpected results. (e.g. in some instances the "execute" command will return you to the main menu, in others it was ignored or treated as a mistake.)

(3) There were too many types of tables, ways of using tables, editing tables, and creating relations between tables. The user is being swamped with a level of detail that is better left to the system. It seems that SEQUITUR was created with simplicity and user support being lesser considerations to system sophistication.

APPENDIX G: VISICALC

VISICALC is an electronic spreadsheet program created by Software Arts, Inc. of Cambridge, Massachusetts and marketed by Personal Software Inc. of Sunnyvale, CA. Its purpose is to allow the user to easily model a wide range of numerical problems in a standard tabular format by replacing the user's pencil, calculator, and scratchpad.

The screen is divided into a grid of columns and rows that form addressable (column, row) entry positions. The columns, which run across the top of the grid, are lettered starting with "A" and the rows, which run down the side, are numbered starting with "1". Each entry position is an independent entity, and can contain a character string, a numeric value, or a function that must be calculated. Entry positions that contain functions are recalculated by VISICALC each time certain conditions are met. The functions will specify values in terms of constants, operators, and the values of other entry positions.

The screen is used as a "window" into the spreadsheet and is modifiable by the user. The user is given numerous commands, see Table G.1, with which to alter the display format of the screen.

Table G.1 - VISICALC Display Commands.

1. Clear Spread Sheet
2. Set Global Display Format To;
   - Integer
   - Dollars & Cents
   - Left/Right Justified
   - Graph
3. Set Entry Display Format To;
   - Integer
   - Dollars & Cents
   - Left/Right Justified
   - Graph
4. Reset Entry To Global Display Format
5. Set Column Width Within A Window
6. Set Order Of Recalculation;
   - Column Wise
   - Row Wise
7. Set Recalculation;
   - Automatic
   - Manual
8. Move An Entire Row Or Column
9. Window Control;
   - Split Screen Horizontal
   - Split Screen Vertical
   - Single Window
10. Window Synchronization;
   - Synchronized
   - Unsynchronized

The window can be "split" into two halves so as to look into nonadjoining areas of the spread-sheet simultaneously. The two windows can be "synchronized" so they move together, or unsynchronized so movement is independent. Display format may be globally set for the screen as a whole, or individual entry positions can be assigned their own format. Column width is variable from 3 to 37, but columns in the same window must have the same width. The value of each entry position is calculated by "column order" (A1, A2, ..., An, B1, B2, ..., Bn, C1, etc.) unless the user changes the recalculation order to "row order" (A1, B1, ..., n1, A2, B2, ..., n2, C2, etc.). By default VISICALC starts in "automatic" recalculation mode where the value of all entry positions are recalculated each time an entry is changed. As this can significantly slow down the model when large grids and/or complicated numerical expressions are used, the user can enter "manual" recalculation mode where a command must be issued to cause recalculation to occur.

VISICALC provides a command-line oriented editor that enters, modifies, or deletes data in a referenced entry position(s). A cursor is provided on the grid to indicate the current entry position referenced by VISICALC. There are screen commands to allow the user to scroll across the grid or to move to an exact (row, column) entry position. If needed, the numeric processing capability of VISICALC can be used like a calculator to support the user's computational needs. A powerful capability of VISICALC is the replicate command. This allows the user to define an entry once, and then have it entered in a range of successive column or row entry positions. Additionally, the user can specify if the original entry is to be replicated exactly, or should any references to other entry positions be updated at each new position to take into account relative position on the spreadsheet.

Table G.2 - VISICALC Cursor Movement & Entry Commands.

11. Move Cursor Right Or Up
12. Move Cursor Left Or Down
13. Change Cursor Direction;
    - Up/Down
    - Right/Left
14. Move Cursor To The Other Window
15. Move Cursor To A Specific Entry Position

16. Abort Last Command
17. Set An Entry Position To Blank
18. Delete An Entire Row Or Column
19. Inset A New Row Or Column
20. Replicate An Entry
21. Set Title Areas;
    - Horizontal Title
    - Vertical Title
    - No Title
22. Repeat A Label Entry
23. Make An Immediate Numerical Calculation
24. Enter A Label In An Entry Position
25. Enter A Value In An Entry Position
26. Save A Copy Of The Spread-Sheet

Since VISICALC is a numerical modeling tool it has a series of arithmetic and aggregate functions that it supports. Table G.3 provides a listing. VISICALC has been designed to store numbers in decimal format, not binary, and maintains them with up to eleven significant digits or decimal places.

Table G.3 - VISICALC Arithmetic & Aggregate Functions

a. Addition
b. Subtraction
c. Multiplication
d. Division
e. Exponentiation
f. Calculate The Sum Of A Range Of Values
g. Calculate The Minimum In A Range Of Values
h. Calculate The Maximum In A Range Of Values
i. Count The Number Of Entries In A List
j. Calculate The Average Of A Range Of Values
k. Calculate The Net-Present-Value Of A Range Of Values
l. Perform A Lookup Operation
m. PI (3.1415926536)
n. Calculate The Absolute Value
o. Calculate The Integer Portion Of A Value
p. Square Root
q. Logarithms, Base 2
r. Logarithms, Base 10
s. Trigonometric Functions (Sin, Cos, Tan, Asin, Acos, Atan)

VISICALC makes use of dynamic memory allocation so the actual dimensions of the spread-sheet depend on the amount of memory available and the complexity of the entries made by the user. The user does not have to worry about memory allocation since VISICALC takes responsibility for its use and efficiency. As entries shrink, or are deleted, VISICALC reclaims the extra memory space. The user is shown how much memory remains and a warning prompt occurs when memory space is nearly exhausted.

For a permanent copy of the contents of the spread sheet the user may send the output to a printer. A subpart of the total spread-sheet may be sent by designating the lower right corner to be printed.

VISICALC is a powerful and fairly simple modeling tool whose advantages seem to easily outweigh the disadvantages. The command vocabulary is low (26 commands, 19 functions) and the greater majority are actually useful and not just window dressing. The user manual is well written and easily understood, but is fairly long. VISICALC supports a known human weakness (small/fast short term memory, large/slow long term memory, and slow calculation speed) by remembering the details of a commonly reoccurring user problem (the situation to be modeled), limiting the user to providing a smaller and more select set of initial inputs, and performing the computations in a faster, more reliable, and repeatable manner. However it does have some problems:

(1) Command strings and their effect must be memorized since there is little relation to the string and the effect. Menus provided by the system are very poor, and require you to already know the meaning of the command string.

(2) A basic understanding of VISICALC and a high degree of operational capability can be obtained, in a fairly short time, by reading only the first third of the user manual. However, to gain maximum use of the system requires a

significant amount of time and effort to read the entire user manual and experiment with the operations. Some nice to know features that have a major effect on model validity (e.g. recalculation order) are discussed at the end of the user manual and might be easily missed.

APPENDIX H: ZIP

The relational data base management system "DBASE II", described in Appendix (D), contains a set of commands which, when embedded in a "command" file, define the output format used to generate the display on the screen, or output to the printer. In addition to generating the display form, the commands also direct the DBASE II system to either determine the values of the entries from a record in the referenced database, or from memory variables. If the input device is the screen/keyboard, DBASE II may retrieve a user entered value from the screen and store it in a field of a database record, or in a memory variable. These form definition commands can also be put into a new type of file, the "format" file, by ZIP. In this case the format, contained in the "format" file, is used as an display overlay to prompt the user to change data values in an existing record in a "database" file.

ZIP is a CP/M program used to generate, or modify, a DBASE II "command" or "format" file. It is a powerful tool in the sense that the user is not required to know the details of the DBASE II form generation capability ("command" files, and display commands). ZIP presents the user with a blank screen and an on-screen editor, which supports several levels of cursor movement and formatting commands, to help in the form design. Table H.1 lists the ZIP editor commands.

Table H.1 - ZIP Editor Commands.

```
1.  Screen commands
    - top
    - bottom
    - next
    - previous
    - first
    - last
2.  Middle of line
3.  Insert a space
4.  Add a line
5.  Delete
    - character
    - line
6.  Draw/Erase horizontal line
7.  Draw/Erase vertical line
8.  Erase/Save work file
9.  Insert DBASE II command expression
10. Change variable
    - vertical marker
    - horizontal marker
    - tab spacing
    - margin
    - page length
11. Quit
```

The cursor can be moved to any position on the blank screen where the user will enter the information required by the ZIP program. Information is conveniently limited to literal strings, memory variables, record field values, and fetching a value from the screen and storing it into a record field or memory variable. Interspersed between these ZIP formatting commands may be DBASE II executable commands if the file type is "command". There are special purpose commands to draw, or undraw, vertical and horizontal lines on the form.

The ZIP program may be viewed as a translator between the screen design made by the user and the operations of DBASE II. The screen contents associated with each screen position are translated into a sequence of DBASE II commands, statements, and control structures which are organized as either a "command" or "format" file. ZIP also places any embedded execution commands into the file and automatically sets, or resets, the appropriate system "toggles" as needed.

ZIP is a useful support tool for DBASE II in that it relieves the user from having to program a "command" file in order to create a desired display format. However, it must be pointed out that ZIP is a very basic formatter, is line oriented, and is incapable of the more complex types of displays.

APPENDIX I: MAIL

"MAIL" is an electronic mail facility produced by the University of California at Berkeley and Bell Laboratories for the UNIX operating system. It allows users to send messages to other users, or groups of users, on the system.

The basic unit of the MAIL system is the message, which is simply a special type of text file. The message is preformatted and contains fields for originator, destination, subject, copy to, and body. Messages are contained either in the users "private" mailbox or in the "system" mailbox. A "dead-letter" file is also maintained for each user to contain messages which cannot be delivered to a valid destination. The private mailbox and dead-letter file are maintained as text files in the UNIX directory and therefore can be used by other programs running under UNIX.

Upon logging into the UNIX system, a prompt appears at the terminal indicating that there is mail for the user. Messages addressed to a user are initially contained in the system mailbox, and can be read from the system mailbox by the MAIL facility. The messages already in the private mailbox and/or dead-letter file are text files and thus not directly accessible to the MAIL facility.

The user may elect to read the mail by invoking the MAIL facility. A one line summary of all messages in the system mailbox is presented to the user, and each message is given an integer identification number starting at one. At this point the user has a number of different options available as summarized in Table I.1.

Table I.1 - MAIL Command Summary

1. Alias a name +
2. Unalias a name(s)
3. Goto previous message * +
4. Goto next message * +
5. Display summary of commands +
6. Display out all currently defined aliases
7. Display a message
8. Display out headers of message list +
9. Display message list
10. Display size of each message
11. Display top few lines of each message
12. Execute the following UNIX shell command
13. Change directory
14. Delete message(s) +
15. Delete current message, print next message
16. Undelete messages marked for deletion
17. Reply to a received message *
18. Edit a list of messages in turn
19. Send message to designated users +
20. End-of-message +
21. Exit, don't change system mailbox *
22. Quit, save undeleted or unsaved messages in the user's mailbox, save unreferenced in the system mailbox.
23. Mark message(s) to be saved in system mailbox *
24. Save a message list by appending to a text file *
25. List current range of message headers
26. Help +
27. Set options +
28. Unset options

* MAIL facility has more than one command to perform this action.

+ Useful - see page 170, paragraph (2).

The user may select a message and read it. After reviewing the message the user may forget the message, save it in the system mailbox, delete it, or prepare a response. When the user quits the MAIL facility all messages which have not been deleted, saved, or reviewed are placed back into the system mailbox. The remaining messages, those reviewed but no special action indicated, are placed in the private mailbox. If the user desires, the MAIL facility can be exited and the system mailbox left unchanged. Additionally the user can create "alias" names that correspond to multiple users, ask for message summaries, append messages to files, or invoke an editor.

The MAIL utility does not contain its own editor, but depends on the editor(s) available to the UNIX system and on the user to set an option specifying which one is desired. When the user indicates that a message is to be created, the editor is invoked, the user enters the text, and when finished issues an end-of-message command to return control to the MAIL facility. While in the editor, the user can issue "escape" commands that directly effect the message processing. A listing of these escape commands is provided in Table I.2. Contents of other files may be inserted into the message, names of recipients added or changed, the header field edited, or an alternate editor invoked.

Table I.2 - MAIL escape commands

1. Execute UNIX shell command
2. Add names to recipients of copy
3. Read "deadletter" file into message
4. Invoke text editor
5. Abort the message being sent
6. Insert a named file into the message +
7. Create a subject field
8. Write the message into a named file
9. Pipe the message through a process as a filter
10. Insert a string into the message

+ Useful - see page 170, paragraph (2).

While in the MAIL facility, UNIX shell commands may be issued. The MAIL facility is temporarily interrupted, the command is executed, and then the MAIL facility is resumed without adverse effect.

Table I.3 - MAIL options.

1. (Append/Prepended) messages to private mailbox
2. (Yes/No) Subject line prompt
3. (Yes/No) Prompt for carbon copy recipients of message
4. (Yes/No) Modify delete command
5. (Yes/No) Ignore terminal interrupt signals
6. (Yes/No) Include sender in group message recipients
7. (Yes/No) Saving interrupted messages
8. Define default editor name
9. Define escape character
10. Define file to record outgoing mail
11. Define number of lines in the "top" of a message

Additionally, the MAIL facility has a series of options the user can change to tailor its operation. Table I.3 provides a listing of these options.

The MAIL facility is a good support program and is quite capable of accomplishing its goals. However, it has more than its fair share of problems.

(1) There is a very limited user manual, and experience must be gained from other users or by trial and error.

(2) There are too many commands, and too many of those duplicate each other. The number of commonly useful commands is low (marked with a +), with the rest being window-dressing.

(3) The facility is not user friendly. The user must be aware of location in the facility and what is expected next, because there are no special prompts and the help command only provides a command summary.

(4) If the message recipient is on line when the message arrives, whatever operation is in progress is rudely interrupted by the display of the message. This can be very disconcerting to the recipient.

(5) The user can't determine which message is going where (system mailbox, private mailbox, dead-letter file), prior to leaving the MAIL facility.

BIBLIOGRAPHY

Bricklin, D. & Franklin, B., VISICALC User Manual, Personal Software, Inc. 1979

Ghosh, S., Data Base Organization for Data Management, Academic Press 1977

Horowitz, E. & Sahni, S., Fundamentals of Data Structures, Computer Science Press, Inc. 1976

Kent, W., Data and Reality, North Holland Publishing Co. 1979

Naiman, A., Introduction to Word Star, Sybex Inc. 1982

Ullman, J., Principles of Database Systems, Computer Science Press 1980

DBASE II User Manual, Ashton Tate 1981

SEQUITUR User Manual, Pacific Software Manufacturing Company 1982

UNIX Programmer's Manual, Seventh Edition, Volume 2A Bell Telephone Laboratories, Inc 1979

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Professor Dusan Z. Badal, Code 52ZD Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	LT John Christopher Waters, USN 225 West 79th Street New York, New York 10024	1

ATE
LME
8