

134
*The Design and Use of a Data Dictionary System
for the Management of Data Bases and Forms
in an Office Automation System*

by

Mary M. Campbell

B.A. Montclair State College, 1971

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

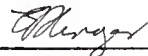
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:



Major Professor

LD
2668
.R4
CMSC
1788
C35
c. 2

CONTENTS

A11207 311851

1. Chapter One - Introduction.....	1
1.1 Data Base Management Systems - 'A historical perspective'	1
1.2 Data Dictionary System's	4
1.3 Forms Management Systems	12
1.4 Synopsis of the Problem	16
2. Chapter Two - Requirements for the Data Dictionary System	18
2.1 Overview.....	18
2.2 The Environment.....	18
2.3 The Physical Data Dictionary.....	19
2.4 Interfaces to the Data Dictionary System.....	21
2.5 Summary of the Requirements for the Data Dictionary System.....	27
3. The Data Dictionary Design	29
3.1 Overview.....	29
3.2 The Data Dictionary System Design.....	29
3.3 The Data Dictionary System Data Base Components.....	31
3.4 The Physical Dictionary Data Bases	41
3.5 Interfaces to the Data Dictionary System.....	47
3.6 Summary	54
4. The Implementation of the Data Dictionary System	55
5. Conclusions and Extensions to the Data Dictionary System.....	57
5.1 Conclusions.....	57
5.2 Extensions to the Data Dictionary System.....	58

LIST OF FIGURES

Figure 1. Conceptual Schema	28
Figure 2. Basic Storage Schema for the Data Dictionary System.....	31
Figure 3. Data Base Components for a Distributed Data Dictionary System.....	33
Figure 4. A Sample Tree Structure of the Data Dictionary Formats	49
Figure 5. A Sample Screen Format to Perform a Data Dictionary Function.....	50

Acknowledgments

I wish to thank Dr. Beth Unger, my major professor, for all the technical advice and time that she supplied in helping me with this report and project.

I would also like to thank my graduate committee: Dr. Virgil Wallentine and Dr. Austin Melton for helping review and edit this paper.

A special thanks is given to AT&T Technologies for sponsoring me in this graduate program and Bell Communications Research who continued to support me financially while I finished the program. Without the time off from my normal duties at work, and the financial help they supplied I would not have been able to complete the program.

1. Chapter One - Introduction

Since the inception of computers, the automation of office procedures has been one of the primary motivating factors for the development and enhancement of computer systems. During the last 30 years, we have witnessed the rapid transformation of computer systems in complexity and usage. The radical changes computer systems have brought about in office procedures throughout the world has not been without challenges to the designers of systems. Today more and more data is being maintained by computers. The enormous volume of data and applications requires new innovative tools to manage this data and make the data as meaningful and easy to access as possible. Offices need user friendly and fast methods for finding, storing and maintaining data. Integrity and security of data is essential. This paper presents an extension of a design of a tool - the Data Dictionary. It discusses why this tool is essential to all present and future Information Management Systems and how this tool can be intergrated with other areas. One area is forms management in an Office Automation System.

1.1 Data Base Management Systems - 'A historical perspective'

The primary function of the first computer systems was the simplification and handling of large amounts of numerical data. During the 1960's many organizations began to develop large information systems (accounting management systems, parts management systems, marketing information systems, etc.) The majority of these information systems executed in the batch (off-line) mode, used data that was entered via card decks and kept data on sequential tape files. They produced large amounts of paper reports. These information systems were moderately effective in keeping track of data but were not very friendly for the users or the programmers of the systems. As these information systems

proliferated, more powerful file organizations, data structures, programming tools and user interfaces were sought. During the late 1960's early 1970's much of the research effort was devoted to the development of new access methods, file organizations and programming languages. The advent of direct access storage devices (DASD) began the age of data bases. [car85] [how84]

Cardenas defines a data base to be:

"a collection of occurrences of a number of record types, where the record types and their occurrences are interrelated by means of specific relationships."[car85]

Martin enhances the concept of a data base when he says:

"A data base is a shared collection of interrelated data designed to meet the needs of multiple types of end users."[mar83]

There are 4 basic models of data bases:

- The hierarchical model
 - divides a file so that some records are subordinate to other records in a ordered tree structure. Every record has only one physical parent.
- The network model
 - establishes a relationship between records in which any record can be related as a child or a parent to any number of other records.
- The relational model
 - uses functions of relational algebra to manipulate relations stored in flat file table format. To make new relations available one manipulates relations

that share a common field.

- The inverted hierarchical model
 - is primarily a hierarchical data base in which fields of any file can be inverted to provide faster performance on the basis of complex data base content qualifications.

A new model of data bases called the semantic model is presently being explored by the artificial intelligence community.

- The semantic model
 - is a collection of constructs that are used to structure and access a data base in a way that reflects the meaning of the data.

In the last 10 years, data bases have become the primary storage medium for Office Information Systems [OIS]. The growing use of data bases created a whole new arena of problems in the area of data base management. Research efforts began to explore ways of controlling data bases and the various data base schemas that were emerging. Data Base Management Systems designers had to use design tools to deal with some of the problems associated with data bases, notably:

- How to control the proliferation of data bases.
- How to keep track of the kinds of data kept in data bases.
- How to interrelate all the data in various data bases.

- How to prevent redundancy in the data kept.
- How to ensure integrity of data entities.
- How to interface with conventional programming languages and files.
- How to provide more user friendly interfaces with data bases.
- How to intergrate various data bases schemas and storage methods.
- How to provide security for data bases.
- How to optimize performance of data bases and access to data bases.

A Data Dictionary is one of the tools used in Data Base Management Systems to help alleviate some of these problems.

1.2 Data Dictionary System's

The concept of Data Dictionaries is not new. Early incarnations of Data Dictionaries consisted of developing an inventory of data items used by a particular data base. With the advent of larger, centralized and integrated Data Base Management Systems, Data Dictionaries began to be enhanced. Besides just making an inventory of items in a particular data base, it became responsible for keeping track of duplication of data and associativity to interrelated data in the Data Base Management System. Incorporated within the Data Dictionary, were control mechanisms over how data was created and used. Security over who could access certain data bases was added to the dictionaries. Data Dictionary Systems were built around the Data Dictionaries that allowed the users of the Data Dictionaries on-line access to information in the dictionary and reports about

dictionary information. Interfaces to Data Dictionaries were created that generated a programmer's view of data. These interfaces would generate DSECTS and COPYLIB information for a variety of conventional programming languages such as PL/1, COBOL, IBM/Assembler. A DSECT is a term that is used in IBM Assembler language to represent an area in the program that can be established to hold a predefined global variable. These global variables may be called into any program and used as though they were actually defined in that program. Installations that use DBMS can predefine DSECTS to hold layouts of the fields in a data base. Programmers include these DSECTS and have variables defined for data base fields. COPYLIBS (copy libraries) perform the same function for COBOL and PL1. Data Dictionaries aided the Data Base Administrator (the person whose responsibility was to maintain the Data Base Management System) by generating control blocks* and parameters required by the Data Base Management System. The majority of Data Dictionaries that exist today were specifically written for one type of Data Base Management System.

1.2.1 Types of Data Dictionary Systems

Martin [mar85] classifies the Data Dictionary Systems that exist today into three groups:

* Control blocks are controlling information that may reside in a library or in the common storage area of the CPU. They assist certain Data Base Management Systems in performing data base functions.

1. Those that are passive - Data Dictionary Systems that provide features for defining and cataloguing data and program modules.
2. Those that can generate information - Data Dictionary Systems that provide automatic generation of programmers data descriptions and control blocks.
3. Those that are active - Data Dictionary Systems that enforce standards. They are drivers for high level data base languages and possibly control access to data in distributed Data Base Management System.

Data Dictionary Systems that are of type 1 or type 2 compose the majority of commercially available Data Dictionary Systems. Type 3 systems are what the best systems on the market are striving for.

1.2.2 Commercially Available Systems

Data Dictionary Systems available commercially tend to support one primary Data Base Management System model. The majority of the systems available were written for IMS* a hierarchical Data Base Management System. The reason for this is that IMS is one of the largest selling Data Base Management System used in business applications today. This is not to imply other systems are not used or desired. Relational model based systems did not come into the business market place until IMS had gained a majority of the market place. Today many companies, tried of the problems associated with IMS, are exploring alternatives.

* IMS (Information Management System) was written by IBM (International Business Machines)

Many companies are physically tied to IMS because of the complexity of the data relations and the volume of data being kept. What one does see, however, is business trying to distribute data by splitting off some logical fragments into smaller more manageable relational systems distributed on smaller processors or workstations.

A few large Data Dictionary Systems were designed to be free standing, that is, to be non-intergrated into one Data Base Management System. UCC-TEN by University Computing, Data Catalogue2 by Synergetics Corporation, Data Manager by MSP Inc. and LEXICON by Arthur Anderson & Co.* are Data Dictionary Systems that support different Data Base Management System models: hierarchical, network, relational and inverted hierarchical. The primary characteristic common to each of these Data Dictionary Systems is that they have interfaces that access the Data Base Management System's functions without being a part of the actual Data Base Management System. This allows them the flexibility to write front-end programs to whatever Data Base Management System they desire to communicate with.

Some of the functions that large Data Dictionary Systems share are [cun86] [car85] [mar83] [dun82] [lef77]:

- They all provide features for cataloguing data.

* Arthur Anderson & Co. is no longer marketing LEXICON.

- They all produce paper reports.
- Most support on-line queries. (Only one DATA DICTIONARY by Cincom Systems Inc. does not.)
- They all allow synonyms to be defined for field names.
- They all have cross referencing capabilities.
- They all allow error checking capabilities.
- They all allow data base definitions and descriptions.
- They all allow redundancy and inconsistency checking at the field level.
- They all provide some level of security. (The type and level of security varies with the system.)

1.2.3 An Overview of Important Features That Should be Considered in New Data Dictionary Systems

User Interfaces - All Data Dictionaries need the capability to interface in a friendly manner with the users of the Data Dictionary System. This implies that on-line terminal access be available to the Data Dictionary System. The users should not have to be computer scientists to access and understand the information obtained from the Data Dictionary System. System designers need an awareness that Data Dictionary Systems can be a useful tool for the management of data in expert systems (ES). One of their challenges in the future will be to meet the demands of users by holding more meaningful and accessible information.

Data Base Management System Interfaces - Data Dictionary Systems should be able to handle more than one Data Base Management System model. Interfaces between hierarchical, relational, network and inverted hierarchical models will be more common in the future. Modularity will be the key to a good Data Dictionary System. Purchasers of Data Dictionary Systems should be able to purchase specific interfaces that are required for whatever Data Base Management System they have.

Distributed Systems - Data Dictionaries should be able to handle Data Base Management System that are distributed across processors. Large centrally located Data Base Management Systems will still exist. However, more organizations are finding that distributed Data Base Management Systems promise better functionality and performance [mor84]. Data Dictionary Systems that expect to succeed in this distributed environment, should be prepared to handle problems like: cross referencing entities on different processors; keeping track of what is on what system and what interface is required for a particular model. As data gets replicated across processors (for performance reasons), the Data Dictionary System might be used to direct update transactions to all the places where the replicates exist. In this mode they would act as a data base server [dem85][bat85].

Security - Data base security is a very important issue that is being given attention lately. Organizations are realizing that the wealth of data that they have been collecting in their data bases is a company asset. Data Dictionaries are one place where security might be defined. There are 3 basic levels of security that may be maintained:

- Security at the data base level - This implies that access can be granted or denied to the whole data base. There are different methods of implementing

this such as defined access rights through passwords assigned to users of the system or terminal access (specific terminal names) assigned at the time the system is created or modified.

- Security at the field level - This implies that each field in the data base can be assigned a access rights. The same mechanisms that data base level security utilizes can be employed here.
- Security over a network - This implies that data is protected as it moves over transmission lines in a network. This kind of security might be implemented by applying encryption algorithms to the data before and after transmission. Encryption algorithms may be executed as programs that parse data and change it according to a pattern. Security over a network is an important issue being studied by corporations moving towards a more distributed environments.

Data Dictionaries can assist the security system by keeping track of passwords and encryption routines that need to be executed prior to data base processing.

Application Defined User Exit-Routines - Just as many large Data Base Management Systems have user exit-routines* embedded in their applications, Data Dictionary Systems should provide the same type of capability. This would give the Data Dictionary Systems the capability for attaching to location specific routines. An example of the types of routines that they may wish to call would

* A user exit-routine is a trap door left in the system so a procedure or set of procedures can be locally defined and specified at installation time. Until that point, it is processed as a call to a dummy routine.

be a Forms Management System. The Data Dictionary could be a useful tool in forms management if applications choose to implement them. Just as Data Dictionary Systems support a variety of Data Base Management Systems they could support a variety of user specific routines. A Forms Management System can be considered an extension to a Data Base Management System; they have many similarities.

1.3 Forms Management Systems

Forms Management Systems are systems designed for the complete control of a form. This includes form creation, storage, processing and dispersement.

1.3.1 Forms

Webster defines a Form to be a paper document with blank spaces.

Gehani [geh82] expands this concept to include electronic representations of forms and intelligent forms.

He defines an electronic form as "the computer analog of a paper form. Fields are filled in as in the paper version. However electric forms have more capabilities not found in paper forms."

Intelligent forms are defined as "electronic forms with routing capabilities attached."

McBride and Unger [mcb83] give more meaning to intelligent forms. They describe intelligent forms as a complete job in an office. Not only would they attach routing capabilities but also, instructions to individuals to whom the forms were to be routed, a distribution process and a list of associated actions that must be executed at routing termination points in a network. They view the physical forms as partitionable objects. Pieces of forms could be distributed and processed concurrently and then be brought back together during the final phase of processing.

Czejdo [cze84] envisions an intelligent form to be a Data Frame. A Data Frame is an artificial intelligence concept that encapsulates the idea of a data item with all its essential properties including computer representation, applicable contextual information, permissible operations and relationships with other data items. Data Frames can be represented as a relation. Cortese and Sirovitch [cor84] suggest forms* be kept in a relational data base where fields can be extracted and used to create other forms.

Data Frames and "modeling forms as jobs" are very similar concepts. They differ in their implementation but perform the same function for the intelligent form.

Intelligent forms can have many instances of representation. Initially, a form can be represented by a template, that describes its physical layout as represented on the screen of a CRT or paper output [tsi82]. This instance will at some point be transformed, so that specific fields in the form are associated with values or functions. The second instance may be thought of as the definition instance where meaning is attached to the template. This form definition may be categorized and stored. The third instance of a form, the processing instance, comes into existence when a routine begins processing the form. Daemon routines can be attached to specific partitions of form instances, where they guide the forms processing. These daemon routines can be thought of as background processes that only come into the foreground when intervention in processing is required. [cor84]

* By forms, I assume, form templates are meant.

1.3.2 Similarities between Forms and Data Bases

Forms and data bases have similar characteristics, some of these characteristics are:

- Forms and data bases are both made up of fields.
- Fields can have relationships with other fields.
- Fields can have functions associated with them.
- Fields can be grouped together into logical partitions.
- Fields may have constraints associated with them (i.e., verification of data types, ranges, etc.)
- Security may need to be attached to forms, data bases or fields within each.
- Not all fields on a physical form or data base may be required by everyone who uses the entity. Therefore, logical views or access rights might need to be established.
- Transactions defined for forms or data bases may need network routing instructions attached to them.
- The physical storage of fields in data bases and forms may be implemented in the same manner.
- The need for distributed processing may exist for both data bases and forms.

In a broad sense, form processing may be considered a subset of a deductive data base*. Development of knowledge based Forms Management Systems will be a direction that future research efforts will try to master. An important part in the development of a knowledge based Forms Management System is a means of controlling all the data definitions, form processing instructions, form templates, form instances and control information that will be required using intelligent forms. The logical place to control form information is an expanded version of a Data Dictionary System. A System Encyclopedia** was suggested as a possible model for data dictionaries [kon84]. However, some expert systems developers are treating systems as a completely new phenomenon and struggling again with classic problems of Management Information Systems, instead of expanding concepts already developed to meet the needs of their new requirements [ari85]. Artificial intelligence and data base research has co-existed for years. Only now that artificial intelligence wants to move into the real world with expert systems does the need for more interaction become apparent [myl84]. An expanded data dictionary is a bridge between these two worlds.

* A deductive data base is a data base, in a Data Base Management System, that has the capability to reason from stored facts or make heuristic judgements [fis84]. Deductive data bases are a relatively new concept. expert systems research efforts are trying to create data bases that perform these functions.

** A System Encyclopedia can be thought of as a group of data dictionaries interconnected to each other.

1.4 Synopsis of the Problem

This report is concerned with the design of a Data Dictionary System. The design will need to include the capability to interface with a Forms Management System. The literature review performed in this chapter highlights some of the concerns that need to be addressed in this design. The Data Dictionary System design will encompass the following design criteria:

- Be capable of being deployed on a distributed data base system.
- Support the definition of data bases and forms in the system.
- Support the definition of fields of data in a data base or form.
- Support the definition of processing information on a form.
- Be interactive with users of the system.
- Allow for security for the data dictionary functions normally reserved for the Data Base Administrator.
- Be a driver for a high level data base language.
- Assist end users by providing reports and views of forms and data bases in a friendly, meaningful form.
- Provide security checking capabilities and be able to support various levels of security.
- Define functions that will be needed to interface with a dictionary.

- Be modular so that expansions to the Data Dictionary System will easy to implement.

Chapter two of this report will detail the requirements for an expanded Data Dictionary System that can be designed to include control for Forms Management Systems. Chapter three will present the design of the Data Dictionary System. Chapter four will cover those portions of the Data Dictionary System that have been implemented and those portions that will be left for future implementation projects. Chapter five will present the conclusions of this study and discuss future projects that might be related.

2. Chapter Two - Requirements for the Data Dictionary System

2.1 Overview

The growth of distributed Data Base Management Systems, Office Information Systems and Expert Systems has caused an increased demand for better tools for the management of data. Existing Data Dictionary Systems do not meet the end users' needs for ease of use, distributed processing, logical interfacing capabilities, security or flexibility to expand the dictionary's functionality when required. The Data Dictionary System schema defined must control and define all operations on data for all systems using it. The requirement specifications for this Data Dictionary System may be divided into the following logical entities:

- The environment
- The physical data dictionary
- The interfaces to the data dictionary
- Security concerns

Requirements for each of these entities will be discussed in this chapter.

2.2 The Environment

This Data Dictionary System will initially be implemented under a UNIX* based operating system. The operating system executes on a DEC/VAX 11/780 processor,

* UNIX is a trade mark of Bell Telephone Laboratories

with 16 megabytes of main memory. The Data Base Management System currently available is Ingres**. Ingres is a Relational Data Base Management System. A variety of programming languages are available on the system (Pascal, C, Concurrent Pascal, Concurrent C. Lisp, ADA, MRS). The processor is linked together with several other processors (Interdata's 8/32, Perkin & Elmers 3220, AT&Ts 3b5 and 4 of AT&Ts 3b2 processors) which have UNIX based operating systems. There are various types of network communication services linking the processors in a Local Area Network. The main network used is AT&T's UUCP network, other networks used are Berknet, and AT&T's 3bnet. Although, presently, Ingres is only available on the DEC/VAX 11/780, the AT&T processors are new. Future distribution of Data Base Management Systems on these processors is a possibility. The design for the data dictionary system must take this into consideration. Flexibility in the design, to allow for additional Data Base Management Systems and routing capabilities should be address in the preliminary design.

2.3 The Physical Data Dictionary

The physical data dictionary sub-schema must provide a storage place for dictionary records. The design needs to take into account that these records may, at some point, need to be accessed from remote systems via a network. The design of the physical dictionary should contain a place to store the following types of records, which hold meta-data about the interfaced systems it will drive:

** Ingres is a Data Base Management System developed at the University of California, Berkley, presently marketed by Relational Technologies Inc..

- Data base records
- Forms records
- Tuple or segment records
- Field records
- Routing records
- Description records
- Security records

In addition to these records, consideration should be given to what level of security will be required on the meta-data in the dictionary.

Another topic of interest, in the design of the Data Dictionary System, will be "How to control routing information for distributed data bases that could be defined to the Data Dictionary System?" and "Should the Data Dictionary System be the controlling mechanism for routing functions?". The routing information that might be needed is: meta-data about routing to different processors; names of daemon routines to be executed at each system visited and routing status.

However the physical dictionary is implemented, performance issues should be addressed. The access of dictionary records is one place where bottlenecks can occur. The design should try to minimize this effect.

Because data dictionary records are so valuable to the functioning of the system, the design will consider mechanisms that can be employed to ensure system

integrity.

The environmental limitations, discussed in Section 2.2, have to be taken into account in the design of the physical dictionary and the Data Dictionary System itself. This is because, we would like to install a prototype in this environment. However, the design should address these limitations and explore possible alternatives if the environment could be changed.

2.4 Interfaces to the Data Dictionary System

The most complex component, of any Data Dictionary System schema, is the interfaces it makes with the world around it. The Data Dictionary System should be looked upon as an expandable entity. As new Data Base Management Systems or application interfaces come into the universe, it must have the ability to expand to meet their requirements. The various interfaces that the initial design will define are:

- The interface to Ingres
- The interface to a Forms Management System
- The interfaces to the users of the system.

The design must be able to accept new systems that need to interface to the Data Dictionary System.

2.4.1 Interfaces to Data Base Management Systems

The Data Dictionary System must be designed with the capability to drive certain Data Base Management System functions. It must do this, yet be able to keep it's flexibility to interface with more than one Data Base Management System model. The Data Dictionary System will be the repository for Data Base Management System's meta-data. During a creation or change in a data base, the Data Dictionary System will have to know what meta-data it must store.

The minimum requirements for the Data Dictionary System's meta-data will include the following.

- Data base name
- Tuple or segment name
- Column name or field name
- Alias names for any of the above entities
- Field types
- Field lengths
- Language conversions for the field types

Additional features that could be designed in the system include:

- Access rights on data bases, tuples, fields and Data Dictionary functions.
- Ranges for field validation.

- Routing meta-data for the Local Area Network (LAN) such as system identifiers, routing status, encryption routines, encryption status and daemon names for processing at various locations.
- Descriptive information: comments about entities.
- Usage statistics of tables or segments, number of fields in a table, table size, create date, last used date.
- Data base statistics: number of tables or segments; organization of the data base; last reorganized date; date created; sizing statistics; the system the data base resides on.
- Network statistics: Number of transactions sent over this line; response time of the transversal from point A to point B (if available).
- System information: types of Data Base Management Systems installed and their version; local applications using the dictionary; operating systems available and their version.

2.4.2 Interface to the Forms Management System

Another issue that must be addressed in the design phase is: "How to build a user specific interface that interacts with the dictionary?" Specifically, we would like the Data Dictionary System to act as a front-end processor for the management of forms in an Office Automation System. Current projects, being considered at Kansas State University, deal with office automation. The data dictionary design should have the flexibility to deal with the meta-data required for the Forms

Management System. It should drive the creation, deletion, and reorganization of form templates. It must have utilities for the Form Administrator, much like the DBA utilities. It should provide security for forms and form partitions. It should deal with routing needs of intelligent forms. All the similar meta-data for data bases (form names, field names, field size, types etc.) will also be kept for forms. The Data Dictionary Systems design will be critical to the future implementation of an intergrated Forms Management System. A design for a Forms Management System that can be intergrated with this data dictionary will be suggested along with the Data Dictionary System's design as an intergrated package.

2.4.3 Other Data Base Management Systems

Presently, there is no need to interface with other Data Base Management System. Ingres is the only Data Base Management System available on the processor the Data Dictionary System will be implemented on. This is a situation that may change at some point in the future. The preliminary design, for the Data Dictionary System, needs to address the way new applications will be added to the dictionary system.

2.4.4 User Interfaces

The last, perhaps the most challenging interface, to the Data Dictionary System is the user interfaces that need to be defined. Data Dictionary System users will fall into 3 categories:

- The system's designers/programmers

- The Data Base/Forms Administrator
- The application users

The needs of each type of user will be addressed.

Data Base/ Forms Systems Administrators

The Data Base Administrator and the Forms Systems Administrator will perform similar functions, with slight variations. On-line interfaces to these functions will need to be developed. Some of the functions are listed below:

- Data Integrity enforcement
- Standards enforcement
- Monitoring and Maintenance Activities
- Establishment of Security
- Definition of Logical Views
- Backup/Recovery functions
- Data base/ forms definition
- Routing definition

System Programmers / Designers

Systems programmers and designers need to interface to the Data Dictionary System to obtain meta-data about a data bases or forms that they may need for

their programs. The Data Dictionary System can be designed to assist them in creating record layouts of data base fields or form fields for their programs.

Application Users

Interfaces for application users need to be inherently simple. Most of the people who fit into the application user category are not going to be knowledgeable about computers, terminals, etc. They will need the ability to access information concerning forms or data bases. They may wish to generate batch reports or check routing status of forms or transactions. All user interfaces must be defined with help functions, comments and instructions visible on the entry screen.

The application user interfaces are the most difficult interface to design. The type of interfaces that exist, presently, go from the most simplistic listing functions, to the most complex generation of code. Idealistically, it would be nice to be able to generate any high level Data Base Management System language or forms language that the user needs. The user should not have to learn languages for requesting information. Instead on-line formats should be furnished to front-end the commands needed. This is not a trivial task. A portion of this type of activity will be implemented for the Data Base Administrator functions. The complete code generation procedure for any language will not be a part of the scope of this design. This could be considered as an extension to the Data Dictionary System at some later point in time.

Security

Most organizations are addressing the need for increased security on their systems. The Data Dictionary is one place that can be very useful in this area. Meta-data

contained in the system deals with most system data where security may be desired. The design of the Data Dictionary System should address how security for the following entities may be implemented using the Data Dictionary:

- Security for data bases
- Security for tuples or segments
- Security for field or groups of fields
- Security for forms
- Security for reserved dictionary functions (i.e., Data Base Administrator or Forms Administrator functions)
- Security for data that may be routed around a network

2.5 Summary of the Requirements for the Data Dictionary System

The Data Dictionary System will be a system that is active. It must be adaptable to new applications that want to use it's capabilities. To do this the system must be designed with modularity in mind. It must be a system that is ready for tomorrow's needs, not just what is required at this moment in time. Figure 1 shows a conceptual schema for the Data Dictionary System. Chapter 3 will discuss it's design based on the requirements presented in this chapter.

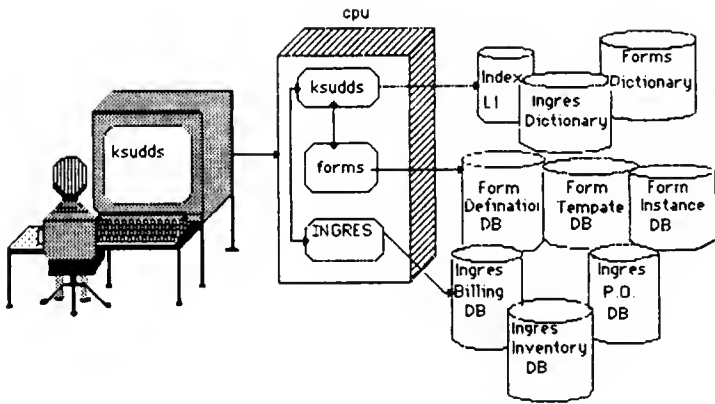


Figure 1. Conceptual Schema

3. The Data Dictionary Design

3.1 Overview

A prototype of the data dictionary, whose design is contained in this chapter, was implemented at Kansas State University for the management of Ingres data bases and forms in a Forms Management System for an Office Automation System. Although the design was intended for Kansas State University, the design is flexible enough to be deployed on any UNIX based system; provided that the system has the language C and the Data Base Management System Ingres. The programs for the Data Dictionary System are written in C and the primary storage medium for the Data Dictionary System's data are Ingres data bases.

This chapter will outline the design criteria for the Data Dictionary System along with it's interfaces. The system is designed so that it may be expanded to distributed processing, if the need arises. The design will initially be implemented on one processor.

3.2 The Data Dictionary System Design

Initially, the Data Dictionary System will be composed of four Ingres data bases and a series of C programs. The four Ingres data bases will hold the data required for the Data Dictionary System. The programs will control the Data Dictionary System's processing and user interfaces. The four data bases that comprise the Data Dictionary System primary storage facility are:

- The Master Index data base

- The Local Index data base
- The Local Ingres Dictionary data base
- The Local Forms Dictionary data base

The Master Index may reside on any processor where the Data Dictionary System is installed. The three local data bases define the local set. The Master Index and the three local data bases comprise what is needed to deploy the Data Dictionary System on one processor. If a distributed environment was required, the local data bases would need to be installed on each processor where the Data Dictionary System application was installed. Figure 2 presents a view of the storage configuration required for a one processor configuration.

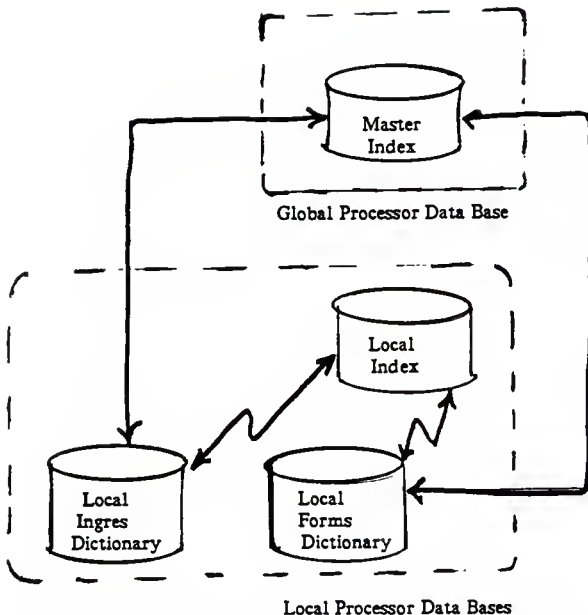


Figure 2. Basic Storage Schema for the Data Dictionary System

3.3 The Data Dictionary System Data Base Components

3.3.1 The Master Index Concept

The Data Dictionary System will have a Master Index data base. There will be only one Master Index. The Master Index is resident on any processor in the Data Dictionary System's network. There are a local set of data bases, that will be installed on any processor which implements the Data Dictionary System.

Initially, the set will contain a Local Index, an Ingres Dictionary and a Forms Dictionary. This set could be expanded to have dictionaries for other Data Base Management System or local applications. The Data Dictionary System is designed so that it may be installed locally or distributed across processors. For the most part, we will discuss the system as though it was going to be deployed in a distributed environment.

The Data Dictionary System's local dictionaries (Ingres and Forms) have been partitioned*. They may also be considered partitioned by structure, in that each local dictionary represents a different application system's meta-data.

The Local Indexes are partitioned by the Local Dictionaries that they represent. However, with respect to the distributed Data Dictionary System, they can be considered replicated because of the inclusion of the Master Index. The Local Dictionaries and their indexes will be homogeneously distributed over the network. This implies that several identical partitioned local dictionary sets will exist at each node in the network. A one processor deployment of the Data Dictionary System is a subset of a deployment on a distributed system. The distributed system is composed of a Master Index and more than one set of partitioned local components. The one processor system is composed of a Master Index and one set of local components. Figure 3 represents the Data Dictionary System's data bases in a system schema for the distributed environment. Figure 2 represents the data

* Draffan and Poole [dra80] define partitioned to be: "A homogeneous or heterogeneous distributed data base, where there is no duplication of data at various nodes of the network"

bases for a one processor configuration.

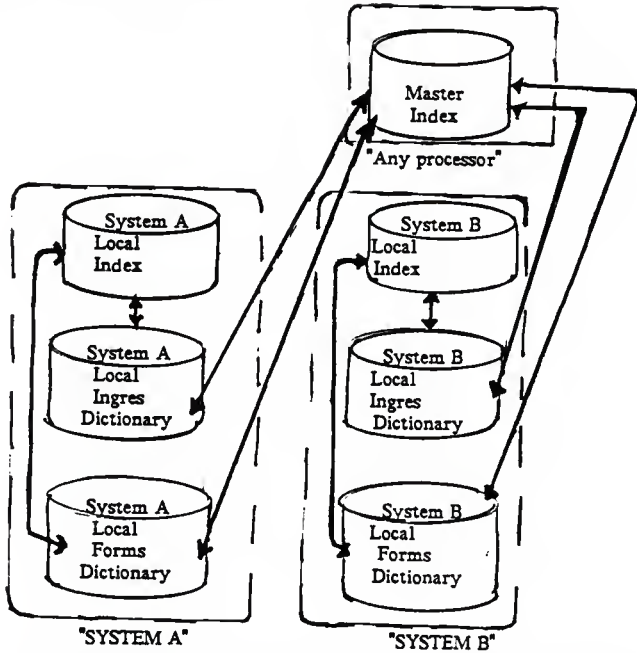


Figure 3. Data Base Components for a Distributed Data Dictionary System

The next four sections will discuss the Master Index, Local Index Ingres Dictionary and Forms Dictionary in more detail.

3.3.2 The Master Index Data Base

The Master Index will be allocated to all the processors in the Data Dictionary System system. Its functions are to direct requests and data to the correct Local Dictionary and to assure that no duplicate names are generated in a distributed environment.

The Master Index is, actually, an Ingres Data Base. The contents of the Master Index are as follows:

- Meta-data about names of the data bases, forms, and tuples for the application systems that are defined to the Data Dictionary System.
- The name of each Local Dictionary* data base that the Data Dictionary System must access.
- The name of a routine that is used to transfer requests for information over a network, to dictionaries on other processors. This routing routine may call an encryption routine to encrypt data being sent over the network. This feature is optional, depending on the application's need for network transmittal security.

When implementing the Data Dictionary System on a single processor, the Master Index is added overhead to the Data Dictionary System. An option to turn off the Master Index at system generation time, will be supplied for any single processor application. The Master Index is essential to the distributed environment,

* The Local Dictionaries are where the Data Dictionary System stores all the meta-data for each application that uses it.

however, and must always be implemented for it. In the case of the single processor system, the Local Index can function as the Master Index. This will be discussed further in the next section. When the system is converted to a distributed system, the Local Index may be copied. This copy will create the necessary base for the Master Index.

3.3.3 The Local System Indexes

The Local Indexes are defined to be a partitioned subset of the Master Index. The same information that is in each Local Index is replicated in the Master Index. The Master Index can be thought of as the union of all the Local Indexes. Although, the replication of index data adds overhead to the system, when new names are added to the dictionary. The overhead may be justified by the gains made during a retrieval operation. In most distributed processing, the majority of work is localized on one processor. Occasionally, the need will arise to access data on another processor.

The Data Dictionary System could have been designed to use only one index. However, there was a potential for bottlenecks, around the access to one index. Also, all the transactions generated from a processor where the index was not resident, would have to be transmitted over the network; even, if the dictionary information resided on the sending processor. Local Indexes distribute the function of the index, there by limiting the amount of network routing required. Any local processing, that does not require network services, can be handled on the home system where the request was issued. Only after unsuccessfully searching the Local Index for the appropriate name, will transactions be sent to the Master

Index.

The Local Indexes serve another purpose, they may act as backups to the Master Index. If the processor that the Master Index was on, was inaccessible, the non-effected processors would continue to process dictionary information. The advantages of replicating the index data can be summarized as follows:

- Faster transaction processing - Local processing, not requiring the network, could be contained on the local system.
- Better recovery in the event of system failures - System failures are bound to occur in any environment. Even with the advent of fault tolerant processors, crashes can occur. Without Local Indexes, these failures would prevent the Data Dictionary System from processing. With Local Indexes some processing can still continue. Algorithms to search the Local Indexes, if the Master Index was unavailable could be devised. Because the Local Indexes replicate the data in the Master Index, processing could continue. Searching the Local Indexes would result in slower processing, but would not cripple the whole Data Dictionary System. If you had 4 systems networked together and one system was inaccessible, then only the data dictionary on that machine would be inaccessible. Any transaction that could not be completed, could be queued for processing after the system was recovered. Appropriate messages would be sent to the users, to inform them of the unavailability. In an emergency, a destroyed Master Index could be recreated by joining all the Local Indexes to recreate the Master Index or extracting local information from the Master Index to recreate a Local Index.

Some of the disadvantages of replicating the index information are:

- More complex algorithms must be developed to assure data integrity and synchronization of updates to the indexes. However, by deploying the indexes as data bases in a commercial distributed Data Base Management System, one would minimize these problems. Any reputable distributed Data Base Management Systems vendor, would have solved these problems before marketing their product.
- Additional processing time is required because two updates must be made each time a data base or form is created, changed, or purged. This added overhead must be weighed against the performance gains made during retrieval.

3.3.4 The Dictionaries

Each application will have it's own dictionary. The application dictionaries will be located on each system that the application is distributed on. The design presented here includes two dictionaries, one for Ingres meta-data and one for Forms meta-data. One of the advantages of this distributed dictionary system is the flexibility to add new applications. For example if we wanted to install IMS, we could create a data dictionary to hold IMS type meta-data. The reason that we keep separate data dictionaries for each application can be broken into three categories:

- Each application has different dictionary needs. If we specify one dictionary, we might have to restrict the types of information that the dictionary

handles.

- By maintaining separate dictionaries, there is less contention for the same resource; response time should be better.
- It is easier to interface new application systems to the Data Dictionary System. When installing a new application, extensive changes to a working environment are not required. New code could be written or cloned, which will interface the Data Dictionary System to the new application; the working programs are not affected. User application systems can be designed to interface with the Data Dictionary System, as will be demonstrated in the section on the Design of the Forms Interface. The Data Dictionary System's configuration is designed in a way that allows many types of Data Base Management System dictionaries to be added. In this sense, the Data Dictionary System is expandable.

3.3.5 The Ingres Dictionary

The Ingres dictionary will use an Ingres data base to hold meta-data about the Ingres Data Base Management System. The meta-data is obtained by making the Data Dictionary System act as a front-end processor for Ingres create and define functions. The Ingres dictionary will contain information about the names of data bases, relations and fields defined in Ingres. It will store information, such as alias names defined to an Ingres data base, relation or field. It has the capacity to store English descriptions about each entity. Security in the dictionary will be supplied, to prevent unauthorized access to dictionary information.

3.3.6 A Schema of a Forms Management System

The main objective of this paper is to design a Data Dictionary System. However, to illustrate that user application systems could be designed to make effective use of this tool, a schema for a non-existent Forms Management System will be presented. In order to design the interface, it was beneficial to have a concept of what the Forms Management System's capabilities would be, along with the types of meta-data a Forms Management System would generate. The following schema will be used to model the interface to the Data Dictionary System.

The Forms Management System envisioned, is interactive with the users of the system. The forms are intelligent, in the sense that they can route themselves to people in a distributed environment, perform functions on their own, and interface when required to a user.

Forms themselves may be broken into three categories: form templates, form definitions, and form instances.

- **Form Templates** : Forms are made up of fields. Fields have templates that describe their physical characteristics (i.e., positioning on the form; labels; separators; group identifier types (single fields, block fields*, repetitive fields**)). A form template is composed of a set of field templates.

* A block field is a group of fields that are composed of different types of single fields.

** A repetitive field is a group of fields that have the same characteristics as a single field.

- Form Definitions : Forms have definitions. Because forms are composed of a set of fields, form definitions are composed of a set of field definitions plus some global attributes. Field definitions describe: field attributes, ranges for validation of fields, security, routing needs, descriptions, instructions, routines for performing functions (getting data from a Data Base Management System, computing field values, etc.). Form definitions can contain global definitions besides field definitions. Global definitions are attributes, instructions, security, routing needs, and functions to be performed, that define a complete form definition. An example of this would be the instructions that tell a form what to do at its completion. This might be to send copies of itself to certain individuals or file itself in a certain data base, etc. Form definitions have a n/1 relationship with a particular template. For every form template, multiple for definitions may exist.
- Form Instance A form instance is a form that has a template, a definition, and is being used by someone. Like office forms, many instances can exist.

In the Forms Management System envisioned, form templates, form definitions, and form instances are all stored in a Forms Management System data base. They are assigned a unique template number, definition number, and instance number. Forms may be categorized by a group type when initially defined in the Forms Management System. Forms can be viewed as tuples (segments, records) in any type of Data Base Management System. As such, forms have similar needs to that of a data base, with regards to data dictionary services.

3.3.7 The Forms Dictionary

The Forms Dictionary is designed to hold meta-data about forms. This meta-data is obtained by making the Data Dictionary System, a front-end processor to a Forms Management System. When creating/changing templates of forms or form definitions, the data dictionary programs extract needed information for the Data Dictionary System.

The Forms Dictionary is designed to store all form and field names used in the Forms Management System. It will keep track of other information generated by the Form Management System, such as: unique template numbers, definition numbers, the processor identifier and Form Management System data base name; where the Form Management System has stored this information. Alias names and English descriptions may be associated with forms and fields of forms. A category type, generated by the Form Management System, which defines each form or field according to a general type, will be kept with the associated name of the entity. If the user wishes to find all the names of templates that deal with "purchase orders", they could search on the defined category in the Data Dictionary System.

3.4 The Physical Dictionary Data Bases

The following section will describe the Ingres data bases that comprise the Data Dictionary System. A complete data base layout for each data base may be seen in Appendix A : "Data Dictionary System Data Base Layouts"

3.4.1 The Physical Master Index

The physical Master Index will be comprised of tuples in an Ingres data base. It may reside on any system that the Data Dictionary System is deployed on. It contains meta-data that is required to direct transmissions from one systems local dictionaries to another systems local dictionaries. The following tuples will be defined for use in the Master Index:

- **MINAME** : This tuple contains meta-data about names defined in the system. It is composed of the following information: the entity name, the application it belongs to (Ingres, Forms), the processor that the dictionary is defined on, and a password. The password is included because even names of data, sensitive to one application, might need to be protected.
- **MIAPPL** : This tuple contains application specific information, such as: what dictionary name is associated with which system and which processor the dictionary is located on. The name of a routine used to route requests to the dictionary is also included.
- **MIROU** : This tuple contains the following routing information: the name of a routing routine, the routing status (sent, received, waiting, queued, complete, etc.), the home system name, the last system visited, the next system it will sent to, an encryption status to tell the if the information needs to be encrypted before transmission, and the name of a daemon to be executed to perform the actual routing. This tuple will not be used until the Data Dictionary System is required to do distributed processing.
- **MIPWD** : This tuple represents a password table, in this table one can establish group passwords or single passwords. Given a valid user it either

maps to a group that has access rights or maps to itself. It contains a field that defines access rights, such as: read, all, system.

3.4.2 The Physical Local Indexes

The Local Indexes are exactly the same as the Master Index. The tuples will have the same names as the tuples in the Master Index, except instead of starting with "MI" they will start with "LI". The descriptions that are specified above under the Physical Master Index, also apply to the Local Master Indexes.

3.4.3 The Physical Ingres Dictionary

The Ingres dictionary will hold meta-data about Ingres data bases; defined to the Ingres Data Base Management System. This meta-data will be stored in an Ingres data base as tuples. The types of meta-data that will be kept are define in the following tuples:

- **IDDSDDB** : This tuple represents the meta-data kept about Ingres data bases. It contains the following information: the data base name, the home system that it is defined on, the application name (in this case Ingres) and a unique identification number that identifies this data base name to the Data Dictionary System.
- **IDDSTPL** : This tuple represents the meta-data kept about Ingres data base tuples. It contains the following information: the tuple name, the data base name the tuple is defined in and the unique identification number that identifies this tuple to the Data Dictionary System.

- IDDSFLD : This tuple represents the meta-data kept about Ingres data base fields. It contains the following information: the field name, the tuple which contains the field, the Ingres attributes that are associated with it, the length of the field and a unique identification number that identifies this field to the Data Dictionary System.
- IDDSALS : This tuple represents the meta-data about alias names defined to data bases, tuples or fields. It contains the following information: the identification number assigned to a data base, tuple or field, the alias itself and an alias number that associates multiple aliases defined to the same entity.
- IDDSDES : This tuple represents the meta-data about descriptions to entities. It contains the following information: the identification number assigned to a data base, tuple or field, the description itself and a description number that associates multiple descriptions with one entity.
- IDDSPWD : This tuple represents a password table. In this table one can establish group passwords or single passwords. The tuple contains the following information: a identification number assigned to a data base, tuple or field, a user's password, the associated users group, and the type of access permitted, such as read, all, update. Given a valid user, it either maps to a group that has access rights or maps to itself.
- IDDSROT : This tuple represents the routing tuple. It contains the following information: the identification number assigned to a data base, tuple or field, the destination of where the record is being sent, the home system that the

entity belongs to, the last place that the entity was sent to, the status of the transmission (sent, received, completed, waiting, unavailable etc.), a place to record encryption status (does the record need to be encrypted or de-encrypted during transmission) and a name of a Daemon to be executed which will perform the actual transmission.

3.4.4 The Physical Forms Dictionary

The physical forms dictionary will be defined as an Ingres data base. Meta-data captured by the Data Dictionary System while interfacing to the Forms Management System will be kept in the forms dictionary. The forms dictionary will be composed of the following tuples:

- **FDDSFMN** : This is a tuple that holds the meta-data about the data defined to the Forms Management System for a particular form. The forms tuple contains the following types of information: the name of the form, a unique template number generated by the Forms Management System (that is used as a key in the Forms Management System template data base to locate this specific form layout), a unique definition number that is generated by the Forms Management System (that is used as a key in the Forms Management System definition data base, where form definitions are kept), a category that describes what type of form this is and a unique identification number that identifies this form name to the Data Dictionary System.
- **FDDSFLD** : This is a tuple that holds the meta-data about the fields defined to a particular form. This tuple contains the following information: the field name, the form name that contains this field, the template number and

definition number generated by the Forms Management System, the category that describes what type of field this is and a unique identification number that identifies this field name to the Data Dictionary System.

- **FDDSTEMP** : This is a tuple that holds the meta-data about the the template data base in the Forms Management System. The tuple contains the following information: the template number generated by the Forms Management System, the name of the data base the Forms Management System is using to store templates, the processor that the Forms Management System data base is located on.
- **FDDSDEF** : This is a tuple that holds the meta-data about the the definition data base in the Forms Management System. The tuple contains the following information: the definition number generated by the Forms Management System, the name of the data base the Forms Management System is using to store forms definitions on, the processor that the Forms Management System data base is located on.
- **FDDSPWD** : This tuple represents a password table, in this table one can establish group passwords or single passwords. The tuple contains the following information: the Data Dictionary Systems identification number, a user's password, the associated users group, and the type of access permitted, such as: read, all, change etc.
- **FDDSRou** : This tuple represents the routing meta-data. It will not be accessed until the system is distributed, at that point it will contain the following types of information: the unique identification number that

identifies this form to the Data Dictionary System, the destination that this request is being sent to, the home system that the entity belongs to, the last system that the entity visited, the routing status (complete, processing, queued, waiting, etc.), the encryption status which informs the system that the record needs to be encrypted before transmission, and a name of a daemon to be executed to perform the actual transmission.

- **FDDSINST** : This tuple represents the meta-data about instances of forms. It contains the following information: the instance number associated with a particular form as generated by the Forms Management System, the form name that it is an instance of, the name of the user who has a instance, the system which the instance is on, the status of the form instance, such as: complete, processing, etc., and the date that it arrived at the present location.

3.5 Interfaces to the Data Dictionary System

3.5.1 Access to the Data Dictionary System

The Data Dictionary System will be accessed through on-line formats (CRT screens), that will operate from a menu driven system. These formats will be the primary way that a user interfaces to the Data Dictionary System. The formats are designed to receive information from a user about some data dictionary function that they wish to perform. This information is then extracted from a buffer (which holds the screen input) and is passed to a program. The program formats the information into the underlying language of the application system (in this case Ingres), performs the required commands, and returns information to the user's screen. In this manner, the data dictionary formats initiate a front-end

processor program, which interfaces to the underlying Data Base Management System or application system to perform data dictionary functions.

The screen formats will all be set up in, basically, the same manner. The formats that comprise the user interfaces are in a tree structure. A portion of this tree structure is shown in Figure 4 below. The user may traverse the tree sequentially. They may also, jump to lower or higher levels in the tree structure directly (once they become more familiar with the structure).

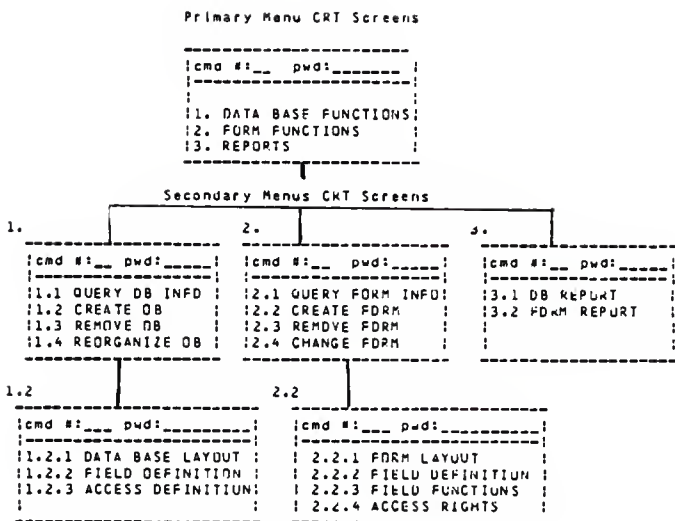


Figure 4. A Sample Tree Structure of the Data Dictionary Formats

The first format screen the user will see is the primary menu format, whose functions are described below. The primary menu screen is the root of the tree. Secondary menus are called from the primary menu. By entering a command displayed on the screen, the user is able to move to the next level in the tree structure. Each lower level either gives the user another menu of choices, or calls a

screen to perform a data dictionary function. Figure 5 represents a sample of a format that will perform a dictionary function.

FORMAT: Browse the Ingres Dictionary

```
COMMAND: _____
Data Base: _____ Relation: _____ Field: _____
.....

The following options are available:
Enter the name of any field with ( ) in the command line above
.....

Command Options:
1: List all data base names.
2: For data base(), relation(), list all relations.
3: For data base(), relation(), field(), list all field names.
4: For data base(), list all data base alias names.
5: For data base(), relation(), list all relation alias names.
6: For data base(), relation(), field(), list all field alias names.
7: For data base(), list all data base descriptions.
8: For data base(), relation(), list all relation descriptions.
9: For data base(), relation(), field(), list all field descriptions.
10: Get Help
11: Exit Options
```

Figure 5. A Sample Screen Format to Perform a Data Dictionary Function

Commands to transverse the tree structure are entered at the top of each screen, along with a user password. The user password will remain active while a particular user is logged on or until a new password is entered. Each screen will display the area where the password, optionally, may be entered again. The user does not have to enter the password until it is required. The bottom of each screen

will have a reserved line, where error messages will be displayed. All screens will be provided with "help" functions, that provide instruction to the users of the Data Dictionary System and act as an on-line manual. The types of screens that are possible are defined below. Appendix B : "Formats Used to Interface with the Data Dictionary System", will show the screen format that the prototype system will use.

3.5.1.1 The Primary Menu

The primary menu screen allows entry to the dictionary for an associated application. This entry begins the Data Dictionary System's interaction with an application. The dictionary will obtain meta-data from the application system. Another option that can be accessed from the primary menu, is a request for batch reports to be generated.

3.5.1.2 The Secondary Menus

The secondary menus are application specific. They act as transfer mechanisms to certain dictionary functions for the interfaced applications. Some of the options available at this point will be:

- Create a data base or form
- Reorganize a data base or change a form
- Delete a data base or form
- Browse the dictionary for information about the meta-data it has stored

3.5.1.3 Other Option Levels

Some functions in the second level of the Data Dictionary System may require more information. The secondary menus might generate another format for users to enter more specific data. The types of requests that may cause this to happen are listed below:

- Data base layout information - This format allows the user to enter the tuples required in the data base, along with their field names and any information about keys which are required.
- Data base field definition information - This format allows the user to enter fields, their attributes and lengths.
- Access definition information - This format allows the user to enter the dictionary security associated with data bases, tuples or fields.
- Form layout information - This format allows the user to enter the the form and field template information required to create a new form.
- Form definition - This format allows the user to enter definitions for form fields or global definitions for an entire form.
- Access definition information for forms - This format allows the user to enter the dictionary security associated with forms and fields on forms.
- Browse the dictionary - This format allows the user to query dictionary information, specific to their application.

- Reorganize a data base - This format allows the user to specify which data base they wish reorganized, and the structure of the new data base.
- Change a form - This format allows the user to edit existing form templates or definitions.

3.5.2 The Program Interfaces Required

The Data Dictionary System requires the following types of "C" programs to be written before the prototype can be functional:

- A program to manage the CRT formats that are used to enter data. This program transverses the tree structure of the formats, calling any interface programs required.
- A program to format the screens for display on the CRT. A subroutine of this program parses the buffer passed from the screen and extract user entered data.
- A command executor program, decides which function of the Data Dictionary System is need for each command entered and calls an appropriate routine to perform the requested function.
- A routine for each function of the Data Dictionary system. Some of these functions are browse the dictionary, create data bases, tuples, fields, perform DBA utilities, etc.
- A program to perform the "Help" and on-line manual functions of each screen.

- A code generation program, that generates Ingres code for data dictionary functions that are available to the users. The data that is extracted from the screen formats, is reformatted into correct syntax for calls to Ingres.
- A security program, to verify that users of the Data Dictionary System have authority to perform requested functions.

3.6 Summary

This chapter presented the design for a Data Dictionary System. The next chapter deals with a subset of this design that was implemented at Kansas State University. The full system has not been implemented at the present time, but may be considered for future implementation endeavors.

4. The Implementation of the Data Dictionary System

A prototype Data Dictionary System has been installed at Kansas State University on a DEC VAX 11/780 processor. The Data Dictionary System uses the Berkeley 4.2 Unix system, and is implemented in the language C. The Data Base Management System Ingres, version 7, is used for data dictionary data bases. The prototype Data Dictionary System is composed of two Ingres data bases that represent the primary storage mechanism for the Data Dictionary System's meta-data. The initial prototype was installed on one processor. Installation procedures are defined in Appendix C : "The Users Manual".

The implementation of a prototype Data Dictionary System builds a Local Index, a sample Local Ingres Data Dictionary (for the Data Base Management System Ingres). The only functions that have been completed at this time are the browse the dictionary function, the create a data base function, and the create a new relation function. Future additions are possible for other graduate students working on a project implementation.

Online user formats are available to perform the completed Ingres dictionary functions. These formats initiate a front-end processor program which interfaces with Ingres. The front-end processor program generates the Ingres Equel code to perform data dictionary functions. Some of the functions performed by the formats are: create an Ingres data base, relation or field. When a command is issued from a screen, an entry is made in the prototype dictionary. The entry consists of the meta-data that has been extracted from the format or generated by the Data Dictionary System itself. The user formats in the prototype, also, generate on-line queries against the data dictionary's meta-data about Ingres data

bases, relations, and fields.

It should be noted that the prototype data dictionary system illustrates how functions can be implemented on a smaller scale than is described in the masters report. The functions that were described that deal with the Forms Management System are not be implemented at this time. A distributed Data Dictionary System for Distributed processing is not required at this time and was not included in the prototype system. Only a sampling of major Ingres dictionary functions will be implemented at this time.

5. Conclusions and Extensions to the Data Dictionary System

5.1 Conclusions

A Data Dictionary System that may be used with the Ingres Data Base Management System and a Forms Management System has been designed. The dictionary's primary function is the management of meta-data, for the application systems that it interfaces with. The following design considerations were discussed in this report:

- The Data Dictionary System's meta-data storage facilities
- User friendly interfaces
- Security for the dictionary functions
- The types of meta-data that can be stored
- The modularity of the design to allow for extensions
- The possible extension for distributed processing
- The support for Data Base / Forms Administration functions.
- The ability to add more descriptive information about data bases and forms.

The Data Dictionary System presented, represents a small prototype of how this tool can benefit application users. For too long, we have allowed the management of data to be uncircumspect in the community of data processing. With the emergence of large amounts of data being stored and used in commercial environments and other organizations, we must assure the continuity of our

systems by providing easy to use mechanisms for their control. The data dictionary represents one of these tools. If implemented in a user friendly manner, it can be beneficial to the layman as well as the technician.

The data dictionary has potential for application in systems that deal with semantic information, rule based systems, and knowledge based systems of the future. Given the time and manpower necessary to implement a complete system, the potential for it's uses are impressive.

5.2 Extensions to the Data Dictionary System

The Data Dictionary System implemented, comprises a base for which a complete Data Dictionary System can be built. The topics that were discussed in this paper are significant for the environment where this Data Dictionary System is implemented. The following section will discuss some extensions to the implementation as well as the design.

An extension to make the Data Dictionary System into a rule based system is conceivable. Rules regarding application system functions could be stored in a data dictionary. These rules could be fired when a user entered the request for information, or executed a system function.

To illustrate how a rule based data dictionary might be used, we can use an example of the Intelligent Form. We could store rules for form manipulation in a data dictionary. Let's suppose that routing dispersements for a particular form definition were stored as rules in the data dictionary. It is conceivable that the entry of a command to create an instance of a form could cause an associated firing of a set of rules, these rules would partition the form into fields or groups of

fields. Another rule could be fired that would set off the concurrent processing of these form partitions and distribute the partitions to perform functions. When the form status for all the partitions changed to "completed," a set of rules to reconstruct the form could be fired. The Data Dictionary is a logical place to store rules for a rule based system.

Other extensions to the Data Dictionary System, that might be implemented are listed below:

- Interfaces to different Data Base Management Systems could be added to the Data Dictionary System.
- Interfaces to other local application systems could be added to the Data Dictionary System. The design for the forms management system, in chapter 3, could be implemented and a data dictionary interface created.
- The implementation of an editor for the Data Dictionary System's application functions might be added. The editor could be used to edit form descriptions or form templates.
- Deploying the Data Dictionary System onto a distributed network environment is another implementation extension.
- One could add semantics to the Data Dictionary System's meta-data. Knowledge based systems will require data dictionaries that can deal with semantic meaning and logical inferences.

BIBLIOGRAPHY

- [adi81] Adiba, M. and Andrade, J., "Update Consistency and Parallelism in Distributed Databases", The 2nd International Conference on Distributed Computing Systems, IEEE, April 1981, pp. 180-187
- [ari85] Ariav, G. and Ginzberg, M., "Design: A Systemic View of Decision Support", Communications of the ACM, Vol 28, No. 10, Oct 1985, pp. 1045-1052
- [ahe85] Ahearne, J. and O'Donnell, S., "Distributed Processing", Database and Network Journal, Vol 15, No. 2, 1985, pp. 8-9
- [bat85] Batini, C. and Ceri, S., "Database Design: Methodologies, Tools and Environments", Communications of the ACM, Jan 1985, pp. 148-150
- [ben81] Bennet, K. and Lunn, K., "A Highly Reliable Distributed Filestore Directory System", The 2nd International Conference on Distributed Computing Systems, IEEE, April 1981, pp. 299-307
- [bou81] Bouchet, P. and Chesnais, J. and Feuvre, J. and Jomier, G. and Kurinckx, A., "Pepin : An Experimental Multi-Microcomputer Data Base Management System", The 2nd International Conference on Distributed Computing Systems, IEEE, April 1981, pp. 211-217
- [cat86] Cattley, J., "Relational Data Base Design Guidelines", Proceedings of Share 66, Anaheim, Calif., March 1986, pp. 1-38
- [cod83] Codd, E., "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM, Vol 26, No. 1, Jan 1983, pp. 64-70
- [cor84] Cortese, G. and Sirovitch, F., "A Daemon-based Programming System for Office Procedures", Second ACM Conference of Office Information Systems, Vol 5, No. 1-2, June 1984, pp. 203-211
- [cun86] Cunningham, E. and Davis, R., "Data Dictionary Systems and Their Applications in Auditing", Data Processing, Vol 28, No. 1, Jan/Feb 1986, pp. 30-34
- [czd84] Czejdo, B. and Embley, D., "Office Form Definition and Processing Using a Relational Model", Second ACM Conference of Office Information Systems, Vol 5, No. 1-2, June 1984, pp. 123-131

- [dav81] Davida, G. and DeMillo, R. and Lipton, R., "Multilevel Secure Distributed Systems", The 2nd International Conference on Distributed Computing Systems, IEEE, April 1981, pp. 308-313
- [die83] Diener, A. and Bragger, R. and Dudler, A. and Zehnder, C., "Database Services for Personal Computers Linked by a Local Area Network", 1983 ACM Conference of Personal and Small Computers, Vol 6, No. 2, Dec 1983, pp. 32-35
- [don80] Donovan, R., "Managements' Role in a Successful Data Dictionary Implementation", Proceeding of Share 54, Anaheim, Calif., Vol 2, No. 2, March 1980 pp. 1924-1931
- [dra80] Draffan, I. and Poole, F., Distributed Data Bases, Cambridge University Press, 1980
- [duy82] Duyn, V., Developing a Data Dictionary System, Prentice Hall, 1982
- [geh82] Gehani, N., "The Potential of Forms in Office Automation", IEEE Transactions on Communications, Vol 1, No. 1, Jan 1982, pp. 120-125
- [gil82] Gillenson, M., "The State of Practices of Data Administration - 1981", Communications of the ACM, Vol 25, No. 10, Oct. 1982, pp. 699-707
- [hei84] Heimbegner, D., "Towards an Intergrated Environment for Accessing External Databases", Second ACM Conference of Office Information Systems, Vol 5, No. 1-2, 1984, pp. 143-151
- [jon80] Jones, T. and Donovan, R., "Managements Role in a Successful Data Dictionary", Proceedings of Share 54, Ahaheim, Calif., Vol 2, No. 2, Jan 1980, pp. 1924-1931
- [kan85] Kahn, B. and Garceau, L., "A Developmental Model of the Database Administration Function", Journal of Management Information Systems, Vol I, No. 4, Spring 1985, pp. 87-101
- [kit84] Kilagawa, H. and Gotoh, M. and Misaki, S. and Azuma, M., "Form Documentation Management System SPECDOQ: Its Architecture and Implementation", Second ACM Conference on Office Information Systems, Vol 5, No. 1-2, June 1984, pp. 132-142
- [kin85] King, R. and McLeod, D., "A Database Design Methodology and Tool for Information Systems", ACM Transactions on Office Information Systems, Vol 3, No. 1, Jan 1985, pp. 2-21
- [lef77] Lefkovits, H., Data Dictionary Systems, QED Information Sciences, Inc., 1977

- [leh86] Lehman, J., "Microcomputer Use of Mainframe Databases", Journal of Systems Management, Jan 1986, pp. 18-22
- [mai84] Maier, D., "Capturing More Meaning in Databases", Journal of Management Information Systems, Vol I, No. 1, Summer 1984 pp. 33-49
- Martin, J., Managing the Data Base Environment, Prentice Hall, Inc., 1983
- [mar85] Mark, L., "Self Describing Database Systems - Formalization and Realization", Technical Report #1484, Dept. of Computer Science University of Maryland, April 1985,
- [mcb83] McBride, R. and Unger, E., "Modeling Jobs in a Distributed System", 1983 ACM Conference of Personal and Small Computers, Vol 6, No. 2, Dec. 1983, pp. 32-35
- [mor83] Morland, D., "Human Factors Guidelines for Terminal Interface Design", Communications of the ACM, Vol 26, No. 7, July 1983, pp. 484-494
- [phi83] Phillips, R., "Dynamic Data Dictionary", Master's Thesis at Kansas State University, 1983
- [rus84] Ruspini, E., "An Intelligent Information Dictionary System", Journal of Systems and Software, Vol 4, No. 2-3, July 1984, pp. 197-205
- [sac84] Sacco, G., "OTTER - An Information Retrieval System for Office Automation", Second ACM-SIGOA Conference on Office Information Systems, Vol 5, No. 1-2, June 1984, pp. 104-111
- [sch83] Schmidt, J. and Brodie, M., Relational Database Systems: Analysis and Comparison, Springer-Verlag, Berlin Heidelberg N.Y., 1983
- [sto85] Stonebraker, M. Ingres Application Notes - Unix Relational Technology, Inc., 1985
- [sto86] Stonebraker, M. The Ingres Papers, Addison-Wesley Publishing Co., 1986
- [tsi82] Tschirtzits, D., "Form Management", Communications of the ACM, Vol 25, No. 7, July 1982, pp. 453-478
- [tsi82'] Tschirtzits, D. and Rabitti, F. and Gibbs, S. and Nierstrasz, O. and Hogg, J., "A System for Managing Structured Messages", IEEE Transactions on Communications, Vol Com30, No. 1, Jan 1982, pp. 66-73

- [ung83] Unger, E. and Jantz, D. and McBride, R. and Slonim, J., "Query Processing in a Distributed Data Base", 1983 Conference on Personal and Small Computers, Vol 6, No. 2, Dec. 1983, pp. 237-244
- [wal85] Walsh, M., "The 'Bread and Butter' Technology of the 1980s", Journal of System Management, Vol 36, No. 8, Aug. 1985, pp. 8-13
- [web84] Webster, "Distributed Software Personnel, When, How and Why?", Second ACM-Conference on Office Information Systems, Vol 5, No. 1-2, June 1984, pp. 80-87
- [win84] Winston, P., Artificial Intelligence, Addison Wesley Publishing Co., 1984
- [yan84] Yanike, M., "An Online Interface to a Relational Data Base for a Real-time Application", Proceedings of COMSAC 84 - The IEEE Computer Society's 8th International Computer Software and Applications Conference, Nov. 1984, pp. 146-151
- [zzz84] Excerpts from the speeches of the following authors:
- [tay84] Taylor, R. - A Workstation Perspective
 - [his84] Hsiao, D. - Improving Performance
 - [moh84] Mohan, C. - Distributed DBMS
 - [mai84] Maier, D. - Prolog in FGCS
 - [fis84] Fishman, D. - Expert Systems and DBMS
 - [ho184] Holsapple, C. and Wilson, A. - Building Blocks of DSS
 - [pic84] Pick, R. - Operating Systems
 - [kon84] Konsynski, B. - Information Systems
 - [mor84] Morgan, H. - Office Automation
 - [kat84] Katz, R. - CAD Databases
 - [afs84] Afsarmanesh, H. and McLeod, D. - Semantic DB Models
 - [my184] Mylopoulos, J. - AI and Databases
- "New Directions for Database Systems (A conference report)", FGCS, Vol 1, No. 1, July 1984, pp. 85-88

Appendix A. Data Dictionary Data Base Layout

The Master Index

MINAME:

name	appl	syson	pwd	id	num
------	------	-------	-----	----	-----

MIAPPL:

syson	appl	dictnm	routing
-------	------	--------	---------

MIPWD:

pwd	group	access
-----	-------	--------

MIROU:

routing	routstat	homesys	lastsys	nextsys	ecyptstat	daemon
---------	----------	---------	---------	---------	-----------	--------

The Local Index

LINAME:

name	appl	syson	pwd	id	num
------	------	-------	-----	----	-----

LIAPPL:

syson	appl	dictnm	routing
-------	------	--------	---------

LIPWD:

pwd	group	access
-----	-------	--------

LIROU:

routing	routstat	homesys	lastsys	nextsys	ecyptstat	daemon
---------	----------	---------	---------	---------	-----------	--------

The Ingres Data Dictionary

IDDSDDB:

<i>dbname</i>	<i>homesys</i>	<i>appl</i>	<i>id</i>	<i>num</i>	<i>con</i>	<i>qry</i>
---------------	----------------	-------------	-----------	------------	------------	------------

IDDSTPL:

<i>tplname</i>	<i>dbname</i>	<i>id</i>	<i>num</i>
----------------	---------------	-----------	------------

IDDNFLD:

<i>tplname</i>	<i>fieldnm</i>	<i>attrib</i>	<i>length</i>	<i>id</i>	<i>num</i>
----------------	----------------	---------------	---------------	-----------	------------

IDDALS:

<i>aliasno</i>	<i>id</i>	<i>num</i>	<i>alias</i>
----------------	-----------	------------	--------------

IDDSDDES:

<i>descno</i>	<i>id</i>	<i>num</i>	<i>description</i>
---------------	-----------	------------	--------------------

IDDSPWD:

<i>id</i>	<i>num</i>	<i>pwd</i>	<i>group</i>	<i>access</i>
-----------	------------	------------	--------------	---------------

IDDSDROU:

<i>id</i>	<i>num</i>	<i>nextsys</i>	<i>homesys</i>	<i>lastsys</i>	<i>roustat</i>	<i>ecrypst</i>	<i>daemon</i>
-----------	------------	----------------	----------------	----------------	----------------	----------------	---------------

The Forms Data Dictionary

FDDSFM:

formnm	templno	defno	catlg	idno
--------	---------	-------	-------	------

FDDSFLD:

formnm	fieldnm	templno	defno	catlg	idno
--------	---------	---------	-------	-------	------

FDDSTEMP:

templno	fmsdb	syson
---------	-------	-------

FDDSDEF:

defno	fmsno	syson
-------	-------	-------

FDDSPWD:

idno	pwd	group	access
------	-----	-------	--------

FDDSROU:

idno	nextsys	homesys	lastsys	roustat	ecryptst	daemon
------	---------	---------	---------	---------	----------	--------

FDDSALS:

aliasno	idno	alias
---------	------	-------

FDDSDES:

descno	idno	description
--------	------	-------------

FDDSINST:

instno	formnm	user	syson	inststat	arrivdte
--------	--------	------	-------	----------	----------

Appendix B. Screen Format Layouts

Primary Menu

COMMAND: _____ PASSWORD: _____
.....

The following options are available:
Enter the command number that you desire above

.....

Command Options

- 1: Ingres Data Dictionary
- 2: Forms Data Dictionary
- 3: Generate Data Dictionary Reports
- 4: Get Help
- 5: Exit Options

Ingres Main Menu

COMMAND: _____ PASSWORD: _____
.....

The following options are available:
Enter the command number that you desire above

.....
Command Options:

- 1: Browse the Ingres Data Dictionary
- 2: Create Ingres Data Base / Relations
- 3: Remove Ingres Data Base / Relations
- 4: Reorganize an Ingres Data Base
- 5: Other DBA Options
- 6: Create Aliases for Data Bases, Relations or Fields
- 7: Create Descriptions for Data Bases, Relations or Fields
- 8: Establish Security for Data Dictionary Functions
- 9: Get Help
- 10: Exit Options

Browse the Ingres Dictionary

COMMAND: _____
Data Base: _____ Relation: _____ Field: _____
.....

The following options are available:
Enter the name of any field with () in the command line above
.....

Command Options:

- 1: List all data base names.
- 2: For data base(), relation(), list all relations.
- 3: For data base(), relation(), field(), list all field names.
- 4: For data base(), list all data base alias names.
- 5: For data base(), relation(), list all relation alias names.
- 6: For data base(), relation(), field(), list all field alias names.
- 7: For data base(), list all data base descriptions.
- 8: For data base(), relation(), list all relation descriptions.
- 9: For data base(), relation(), field(), list all field descriptions.
- 10: Get Help
- 11: Exit Options

Create Ingres Data Base Menu

COMMAND: _____

The following options are available:
Enter the command number that you desire above.

Command Options:

- 1: Create a new Ingres data base.
- 2: Create a new data base relations, copy from an old relation.
- 3: Create a new data base relations, copy from multiple relations.
- 4: Create a new data base relation, enter new information.
- 5: Create a new secondary index for a relation.
- 6: Get Help
- 7: Exit Options

Create Ingres Data Base or Change Existing Data Base Options

COMMAND: _____
Data Base: _____ Concurrency: __ Query Modification: __
.....

The following options are available:
Enter the name of any field with () in text on the line above.

.....

Command Options:

- 1: Create a new Ingres data base(),
with concurrency(y/n),
with query modification (y/n).
- 2: Change an existing data base() options,
with concurrency(y/n),
with query modification (y/n).
- 3: Get Help
- 4: Exit Options

Create A Secondary Index for a Relation

COMMAND: _____
Data Base: _____ Relation: _____
Fields: _____

The following options are available:
Enter the name of any field with () in text on the line above.

- Command Options:
- 1: Create in data base(), relation() a secondary index on fields specified above.(Max. 6 fields)
 - 2: Get Help
 - 3: Exit Options

Create A New Relation

COMMAND: _____

Data Base: _____ Relation: _____
.....

The following options are available:

Enter the name of any field with () in text on the line above.
.....

Command Options:

1: In data base () create a new relation () whose fields are listed below. Valid attributes are (I1, I2, I4, F4, F8, C1 -> C255)

2: Get Help

3: Exit Options

Fields Name	Attribute	Field Name	Attribute
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

Create A New Relation by Copying an Existing Relation

COMMAND: _____
Data Base: _____ Copy All(y/n) ___ (default yes)
New Relation: _____ Old Relation: _____

Create a new relation() from an old relation () in data base ()
use the fields listed below or copy all fields in the table();
use the where clause listed below to perform the copy. The format
is as follows: where {field qualifier constant} and/or {field
qualifier field}. Valid qualifiers are: >, <, >=, <=, =, not =.

Where: _____

New Field	Old Field	New Field	Old Field
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

- Command Options:
1: Create a new relation by copying an existing relation.
2: Get Help
3: Exit Options

Create A New Relation by Copying Multiple Relations

COMMAND: _____
Data Base: _____ New Relation: _____ Copy All(y/n) __
Rel. 1: _____ Rel. 2: _____ Rel. 3: _____

.....
Create a new rel.() from rel. 1 (), rel. 2 (), rel. 3() by
matching relation#.#field listed below, satisfying the where
conditions listed below. Valid qualifiers are { >, <, >=, <=, =, not = }
.....

Matching: {#.#field = #.#field,} { and {#.#field and #.#field} and ...

Matching: _____

Where {#.#field qualifier constant} and/or {#.#field qualifier #.#field}

Where: _____

New Field	1.#field	2.#field	3.#field
-----------	----------	----------	----------

Command Options:

1: Create a new relation by copying multiple relations

2: Get Help

3: Exit Options

Destroy Data Base or Relations in a Data Base

COMMAND: _____

Data Base: _____

.....

Enter the name of any field with () in the text on the line above
or enter options on the line specified.

.....

Command Options:

- 1: Destroy a data base ().
destroy the complete unix file also (y/n).
- 2: Destroy a relation or relations {listed below} in data base().
- 3: Destroy permit relations or relations{listed below} in data base().
all(y/n) ___ List Other options _____
- 4: Destroy integrity on relation in data base().
all(y/n) ___ List Other options _____
- 5: Get Help
- 6: Exit Options

List Relations: _____

Reorganize a Data Base

COMMAND: _____
Data Base: _____ Relation: _____
Max. number of pages _____, Min number of pages _____, Fill Factor: _____
Types(y/n): ISAM: __, CISAM: __, Hash: __, CHash: __, Heap: __, Cheap: __
Cheap Sort: __, Heap Sort: __, Truncate: __
Options for Sort(y/n): Ascending order: __, Decending order: __
Keys: _____

.....
Enter the name of any field with () in the text on the line above
or enter options on the line specified.
.....

Command Options:

- 1: Reorganize a relation () in data base () to storage structure()
using keys in order specified above and a new max # of pages (),
a new min # of pages () and a new fill factor ().
- 2: System modification of a data base() to a Hash structure using
the options specified below.
Options(y/n): relation: __, attribute: __, indexes: __, tree: __
protection: __, integrities: __, super user: __

3: Get Help

4: Exit Options

Define Aliases

COMMAND: _____
Data Base: _____ Relation: _____ Field: _____
.....

The following options are available:
Enter the name of any field with () in the command line above
.....

Command Options:

- 1: Define the alias names listed below for data base().
- 2: Define the alias names listed below for relation () in data base().
- 3: Define the alias names listed below for field () in data base (),
relation ().
- 4: Get Help
- 5: Exit Options

Aliases: _____

Define Descriptions

COMMAND: _____
Data Base: _____ Relation: _____ Field: _____
.....

The following options are available:
Enter the name of any field with () in the command line above
.....

Command Options:

- 1: Define the alias names listed below for data base().
- 2: Define the alias names listed below for relation () in data base().
- 3: Define the alias names listed below for field () in data base (),
relation ().
- 4: Get Help
- 5: Exit Options

Descriptions: _____

Other DBA Functions

COMMAND: _____
Data Base: _____ Relation: _____
Month: _____ Day: _____ Year: _____
Unix full path name: _____

.....
The following options are available:
Enter the name of any field with () in the command line above
.....

Command Options:

- 1: Override 7 day retention period for a relation: Keep the relation()
in the data base () till month() day () year ().
- 2: Unload a copy of data base () to backup unix file ().
- 3: Reload a back copy of data base () from unix file ().
relation ().
- 4: Purge all expired and temporary relations in data base (),
use the following options (y/n):
delete all expired users:
delete all unreconizable files: __
- 5: Restore data base () after a system crash.
- 6: Help
- 7: Exit Options

Establish Security

COMMAND: _____
User id: _____ Group id: _____ Password: _____
Data Base: _____ Relation: _____ Field: _____
.....

The following options are available:

Enter the name of any field with () in the command line above

.....

Command Options:

1: Establish Security for selective data dictionary functions(y/n):

*****Note: any option not specified below defaults to no*****

Security Level allowed-

Allow all authority to all data bases: __, relations: __, fields: __

Create data bases authority: __

Create new relations in any data base: __ in a data base (): __

Create aliases for any: __, for a data bases(): __,

for a relations(): __, for a field (): __

Create definitions for any: __, a data base(): __,

for a relation (), for a fields: __

Browse any: __, a data base(): __, a relation(): __, a field (): __

Purge a data base(): __ any: __ Reorg a data base(): __, any: __

Destroy a data base(): __, any : __ Unload a data base(): __, any: __

Destroy a relation(): __, any : __ Save a data base(): __, any: __

Modify a data base(): __, any : __ Restore a data base(), any: __

Modify a relation(): __, any : __

Access batch reporting facility: __

2: Help

3: Exit Options

Exit Options

COMMAND: _____

.....
The following options are available:
Enter the command number above in the command field.
.....

Command Options:

- 1: Continue with present function
- 2: Exit the KSUDDS Data Dictionary System
- 3: Return to the KSUDDS Data Dictionary Main Menu
- 4: Return to the Ingres Data Dictionary Main Menu
- 5: Return to the Create Main Menu
- 6: Help
- 7: Exit Options

APPENDIX C

Users Guide and Installation Instructions

This appendix will discuss what is necessary to install a sample prototype of the data dictionary system that is implemented to date. The prototype dictionary "ksudds" consists of 4 "C" programs, 2 shell programs, and 3 header files. The "C" programs source is contained in Appendix D and online source will be available to KSU students for extensions that they may wish to add, anyone requiring the online source should contact the KSU computer science department. The main program uses the Curses routines, and the other 3 programs are Ingres Equal programs. To compile these programs use the commands listed below:

```
equal -d ksudds1.q
equal -d brow.q
equal -d icrel.q
cc -o ksudds ksuddsm.c ksudds1.c
    brow.c icrel.c -lq -lcurses -ltermcap
```

The Data Dictionary data base presently install is called : "ksudds1". Below is a layout of the relations and fields in the data base.

ksudds1 : main data base

Relation: liname

attribute name type length keyno.

name	c	13	
id	c	2	
num	i	2	
appl	c	10	
syson	c	8	
pwd	c	13	

Relation: lipwd

attribute name type length keyno.

pwd	c	13	
group	c	13	
access	c	2	

Relation: liappl

attribute name type length keyno.

syson	c	8	
appl	c	10	
dictnm	c	13	

Relation: iddsdb

attribute name	type	length	keyno.
dbname	c	13	
homesys	c	8	
appl	c	10	
id	c	2	
num	i	2	
con	c	2	
qry	c	2	

Relation: iddstpl

attribute name	type	length	keyno.
tplname	c	13	
dbname	c	13	
id	c	2	
num	i	2	

Relation: iddsfid

attribute name	type	length	keyno.
fidname	c	13	
tplname	c	13	
attrib	c	5	
id	c	2	
num	i	2	

Relation: iddspwd

attribute name type length keyno.

id	c	2	
num	i	2	
pwd	c	13	
group	c	13	
access	c	2	

Relation: iddsals

attribute name type length keyno.

alno	i	2	
id	c	2	
num	i	2	
entity	c	13	
alias	c	25	

Relation: iddsdes

attribute name type length keyno.

descno	i	2	
id	c	2	
num	i	2	
entity	c	13	
desc	c	255	

Relation: seedrec

attribute name type length keyno.

startno	i	2	
curno	i	2	
maxno	i	2	

Users Guide

The KSUDDS data dictionary system is accessed through the Kansas State University system KSUVAX1. The user must have an account on this system to access the programs that run the data dictionary. To access the data dictionary system the user types "ksudds" on the screen and the main menu for the data dictionary system prototype is displayed.

In the prototype system the only option available from the main menu is the Ingres Data Dictionary functions. The user enters the number 1 in the command line at the top of the screen. The screens accessed are in a tree structure, where the main menu is the top node. Each command on the screen represents a branch of the tree. The terminal nodes of a branch will perform a desired function. To traverse the tree structure one either enters the desired command number or chooses an exit option. The exit options screen will get the user to another level in the tree or will exit the dictionary. A help function is available at every implemented screen that describes the type of data to be added and the format of the data.

Each screen is composed of a command field and entry fields for the required data. The fields are accessed sequentially. The user types the required information in a field and depresses the return key or the space bar. The cursor

will then move to the next entry field. If no input is required, depressing the space bar or the enter key inserts a null value in the field and tabs to the next field required. Corrections are made by depressing the backspace key before the entering a field. Once a field has been entered no further corrections may be made. If incorrect data or commands are entered the user is notified and the Exit Options screen is displayed. The user has the options of reentering the present screen, going to another screen, or exiting the dictionary entirely.

Although some screens are designed to have a password entered, at this time the password routine is not implemented. Users should just depress the enter key or the space bar which will give them Data Base Administrator status.

The prototype Ingres data dictionary is installed with the following options available. The user may create a new data base, create a new relation, and browse the data dictionary. If any other option is selected a error message will be displayed to tell the user that the option is not implemented and the Exit Options screen will be displayed.

The screens were designed to be self documenting. An online manual is available for each screen by using the help option for that screen. Appendix B shows the layouts for each screen.

The following screen formats are implemented in the prototype data dictionary system:

- The Primary KSUDDS Main Menu
- The Ingres Options Main Menu
- The Browse the Ingres Data Dictionary Format
- The Create Ingres Data Base Menu
- The Create Ingres Data Base or Change Data Base Options Format
- The Create a New Relation Format
- The Exit Options Format
- The Help Formats (one per screen above)

Appendix D. Code for the Prototype System

This appendix contains the code written for the prototype data dictionary system KSUDDS, built at Kansas State University. This prototype system does not implement the entire system described in the Masters Report. It implements a subset of the Ingres Data Dictionary system. Although the design in the Masters Report discusses the implementation on a distributed data base management system, this program does not run in a distributed environment.

The following code is written in the "C" programming language. It makes use of the Curses routines for screen formatting and terminal I/O. It accesses the Ingres Data Base Management System through the Equal option of Ingres. It also accesses Ingres through Unix shell programs which execute the Ingres commands that run at the Unix level outside of Ingres.

In the Berkley 4.2 Unix environment, operating at Kansas State University, not all the Curses routines worked correctly. Parts of this code was added to make the prototype work under that system. The code added compensated for the tabbing to various fields on the same line. For correct tabbing to work, a tab to another line was required prior to a tab to the same line. These excess tabs may be removed once a new version of the system corrects these prob-

lems. Also, in the Ingres application the Equal "param" statements were not working as documented in the Ingres manual. Anyone who wishes to add to this code or use this code should be aware of these limitations.

ksuddsm.c

The following code contains the main program for the KSUDDS data dictionary system. The program is written in "C". The main program contains all the Curses screen management routines for the prototype system. Curses routines are used to handle cursor movement and I/O through a crt screen. The Curses routines put the terminal in raw mode so "C" routines were written to handle back spacing and any entry key the program needed. If the program abends while in these Curses routines, the user must turn the program off and on again to reset the terminal mode to a normal Unix setting. Any screen input field verification is also performed in this main program. The following external routines are called from this main program:

- ksuddsi1.q - an Ingres Equel routine to update the Ingres KSUDDS dictionary
- brow.q - an Ingres Equel routine to browse the Ingres KSUDDS dictionary.
- icrel.q - an Ingres Equel routine to create an Ingres data base relation.
- crshell - a Unix shell routine to create an Ingres data base.
- chshell - a Unix shell routine to change Ingres data base options.

The program uses the header routines: ksudds.h, ksuddsd.h, ing.q.h.

```

/*****
/* Main Program for the KSUMMS data dictionary system
/*
/***** author: Mary Campbell
/*****
/* The KSUMMS data dictionary system is a sample prototype for a design
/* documented in the Masters Report written at Kansas State University.
/*
/* This program will perform data dictionary functions for the data base
/* management system Ingres. This is a small prototype of the original
/* design, as such it does not implement all the options discussed in the
/* design. If anyone wishes to add routines to this code they should first
/* read the Master Report.
/* Although the design talks about a distributed environment
/* the implementation is not done on a distributed system.
/*
/* The program makes use of the curses routines in the unix system for
/* handling terminal I/O. Curses does funny things to your terminal setting
/* so if this program abends you must disconnect your terminal and turn it
/* on again to reset the standard unix shell terminal modes. Also, not all
/* the curses options work in this Berkeley 4.0 version of the operation
/* system, so some funny code has been added to this program in an effort
/* to compensate for these problems.
/*
/* This program also interfaces with the data base management system Ingres
/* through the EQUEL option and through a shell program to execute Ingres
/* unix commands.
/*****
#include <urses.h>
#include <string.h>
#include "/usr/att/campbell/proj.impl/ksudds.h"
#include "/usr/att/campbell/proj.impl/ksuddsd.h"

```

```

WINDOW #scr0;
WINDOW #scr1;
WINDOW #scr2;
WINDOW #scr3;
WINDOW #scr4;
WINDOW #scr5;
WINDOW #scr6;
main ()
{
void cleanup();
void init();
init(); /* call routine to initialize the program variables */
while (logflag=='y' && badflag=='n')
{
display_logon();
scrmanjno=10000; /* screen indication no. Main dds screen */
read_logon();
dds_pwd_rtn(); /* call data dictionary system password validation */
if (badflag=='y')
break;
cmdno=str2int(cmd);
switch (cmdno)
{
case 1:
ingflag = 'y';
while (ingflag == 'y' && badflag == 'n')
ingressdds(); /* call ingres data dictionary application */
break;
}
}
}

```

```

case 2: case 3:
    msgno=14
    display_msg();
    display_exit();
    break;
case 5:
    display_exit();
    break;
default:
    help_rtn();
    break;
}
/* end case */
/* end while */
/*call the exit clean up routine */
/* end main */
/* *****
/* clean up before exiting the program */
/* *****
void
cleanup ( )
{
    clear();
    refresh();
    endwin();
    /*print("qry -Zs\n",ii,qry);*/
    exit(0);
}
/* end curses
*/

```

```

/* *****
/* Initialization Routine
/* *****
void
init()
{
    initscr();
    crmode();
    /* set break mode */
    scr0 = newwin(LINES-1,COLS-1,CSTARTR,CSTARTC);
    scr1 = newwin(LINES-1,COLS-1,CSTARTR,CSTARTC);
    scr2 = newwin(LINES-1,COLS-1,CSTARTR,CSTARTC);
    scr3 = newwin(LINES-1,COLS-1,CSTARTR,CSTARTC);
    scr4 = newwin(LINES-1,COLS-1,CSTARTR,CSTARTC);
    scr5 = newwin(LINES-1,COLS-1,CSTARTR,CSTARTC);
    scr6 = newwin(LINES-1,COLS-1,CSTARTR,CSTARTC);
    ip = &i;
}
/* *****
/* convert string digit to numeric digit
/* *****
int str2int(string)
char string[];
{
    int i,iv,r=0;
    for (i=0; ((string[i] >= '0') && (string[i] <= '9')); i++)
        {

```



```

iv=string[i] - '0';
r = r * 10 + iv;
}
return (r);
}
/* *****
/* Display screen Primary Options KSUIDS
/* *****
void
display_logon()
{
clear();
refresh();
wclear(scr0);
wrefresh(scr0);
wmove(scr0,CSTARTR,CSTARTC);
wprintw(scr0,"
printw(scr0,"COMMAND:
printw(scr0,"*****
*****\n");
printw(scr0,"The following commands are available\n");
printw(scr0,"Enter the command number and a valid KSUIDS password\n");
printw(scr0,"*****
*****\n\n");
printw(scr0," 1 = INGRES Data Dictionary Functions\n\n");
printw(scr0," 2 = Forms Data Dictionary Functions\n\n");
printw(scr0," 3 = Generate Data Dictionary Reports\n\n");
printw(scr0," 4 = Get Help\n\n");
printw(scr0," 5 = Exit Choices\n\n");
wmove(scr0,CHOMER,CHOMECL);
touchwin(scr0);
wrefresh(scr0);
}
- D -

```

```

/*****
/* read the logon screen - scro
*****/
void
read__logon()
{
    void bsscr0();
    int i,k;
    /* *****/ read command field *****/
    theRow = CMOROW;
    ccol = CMICOL;
    wmove(scro,theRow,ccol);
    touchwin(scro);
    wrefresh(scro);
    for (i=0;i<2;i=i+1)
    {
        k=ccol + i;
        cmd[i] = wgetch(scro);
        if (cmd[i] == '\b')
        {
            bsscr0(k,theRow);
            i=i-2;
        }
        else if (cmd[i] == '\n' || cmd[i] == '\r') /* return or blank exit */
    }
}

```

```

{ cmd[i] = '\0'; /* set last element in string to null*/
  break;
}

} /* end read command loop */
/***** read password *****/
cmdno = strtoint(cmd);
if (cmdno == 5) return;
therow = PWDROW;
ccol = PWDCOL;
wmove(scr0,therow,ccol);
wrefresh(scr0);
for (i=0;i<13;i=i+1) /* read up to 12 char from screen */
{
  k=ccol + i;
  pwd[i] = wgetch(scr0); /* read dictionary password field */
  if (pwd[i] == '\b') /* if backspace then backup
  {
    bsscr0(k,therow); /* backspace routine */
    i=i-2; /* set array back to mistake point */
  }
  else if (pwd[i] == '\n' || pwd[i] == '^')
  {
    pwd[i] = '\0';
    break;
  }
} /* end for password loop */
} /* end read-logon routine */

```



```

cmdno=str2int(cmd);
switch (cmdno)
{ case 1:
  browflag = 'y';
  while (browflag=='y' && badflag=='n')
    browse_rtn(); /* call browse dictionary routine*/
  break;
case 2:
  createflag = 'y';
  while (createflag=='y' && badflag=='n')
    createdb(); /* call create main menu */
  break;
case 3: case 4: case 5: case 6: case 7: case 8:
  msgno=1;
  display_msg(); /* exit choices displayed */
  display_exit();
  break;
case 10:
  display_exit(); /* exit choices displayed */
  break;
default:
  help_rtn(); /* get help */
  break;
}
}

/* *****
/* Display screen for INGRES data dictionary system's primary menu
/* *****
void
display_ing()
{

```

```

clear();
refresh();
wclear(scr1);
wrefresh(scr1);
move(scr1,CSTARTR,CSTARTC); /* move cursor to output line
crow = CSTARTR;
ccol = CSTARTC;

/* format the screen output */
wprintw(scr1,"
wprintw(scr1,"COMMAND:
wprintw(scr1,"*****\n");
wprintw(scr1,"The following commands are available\n");
wprintw(scr1,"Enter the command number and a valid KSUDDS password\n");
wprintw(scr1,"*****\n");
. *****\n");
D-13. wprintw(scr1," 1 : Browse the INGRES data dictionary \n");
wprintw(scr1," 2 : Create INGRES data bases / relations \n");
wprintw(scr1," 3 : Remove INGRES data base / relation\n");
wprintw(scr1," 4 : Reorganize Ingres data base\n");
wprintw(scr1," 5 : Other DBA Options\n");
wprintw(scr1," 6 : Create aliases for data bases, relations, fields\n");
wprintw(scr1," 7 : Create definitions for data bases, relations,fields\n");
wprintw(scr1," 8 : Establish security for data dictionary functions\n");
wprintw(scr1," 9 : Get Help\n");
wmove(scr1,"10 : Exit Choices\n");
touchwin(scr1);
wrefresh(scr1);
}

```

```

/*****
/* read the ingres main menu screen scr1
*****/
void
read_ingmm()
{
    int i,k;
    void bsscri();
    /* ***** read ingres cmd field *****/
    theRow = CMDROW;
    ccol = CMDCOL;
    move(scri,theRow,ccol);
    touchwin(scri);
    wrefresh(scri);
    for (i=0;i<3;i=i+1) /* read 1 character command */
    {
        k=ccol + i;
        cmd[i] = wgetch(scri);
        if (cmd[i] == '\b')
        {
            bsscri(k,theRow);
            i=i-2;
        }
        else if (cmd[i] == '\n' || cmd[i] == ' ') /* return or space exit */
        {
            cmd[i] = '\0';
            break;
        }
    }
}
/* end read command loop */

```

```

/* ***** read inqres dictionary password field ***** */
cmdno = str2int(cmd);
if (cmdno == 10) return;
therow = PWDROW;
ccol = PWDCOL;
wmove(scr1,therow,ccol);
touchwin(scr1);
wrefresh(scr1);
for (i=0;i<13;i=i+1)
{
    k=ccol + i;
    ipwd[i] = wgetch(scr1); /* read password field */
    if (ipwd[i] == '\n')
    {
        bsscr1(k,therow);
        i=i-2;
    }
    else if (ipwd[i] == '\n' || ipwd[i] == ' ') /* set array back to mistake point */
        break; /* return or space exit */
}
} /* end for password loop */
/* ***** end read routine */
/* backspace for screen */
/* ***** */

```



```

void
bsscri(k,theRow)
int k,theRow;
{
    int j,row;
    j=k;
    row=theRow;
    j=j-1;
    /* reposition cursor and blank out bad input*/
    wmove(scr1,1,j);
    wrefresh(scr1);
    wmove(scr1,row,j);
    wrefresh(scr1);
    wprintw(scr1,"%s",blanks);
    touchwin(scr1);
    wrefresh(scr1);

    wmove(scr1,row,j);
    wrefresh(scr1);
}
/* ***** */
/* Browse the Ingres data dictionary
/* ***** */
void
browse_rtn()
{
    extern int Ibroff();
    int rc;
    scrwango=11100;
    accesscode = 'd'; /* temp till security added */
    if ((accesscode != 'd') && (accesscode != 'a'))
    {
        badflag='y';
        return;
    }
}

```



```

wprintw(scr2," 6 : For data base (), relation (), field (), list all alias field
names.\n");
wprintw(scr2," 7 : For data base (), list all associated discriptions\n");
wprintw(scr2," 8 : For data base (), relation (), list all associated descriptio
ns.\n");
wprintw(scr2," 9 : For data base (), relation (), field (), list all assoc. desc
riptions.\n");
wprintw(scr2,"10 : Get Help\n");
wprintw(scr2,"11 : Exit Choices \n");
wmove(scr2,CHOMER,CHOMEC);
touchwin(scr2);
wrefresh(scr2);
}
/* end display */
/***** read the browse screen - scr2 *****/
19
/* read the browse screen - scr2
*/
void
read_browse()
{
int i,k;
void bsscr2();
/* ***** read the command field ***** */
nofld = 'n';
therow = CMDROW;
ccol = CMDCOL;
wmove(scr2,therow,ccol);
touchwin(scr2);
wrefresh(scr2);
for (i=0;i<3;i=i+1)
{
k=ccol + i;
cmd[i] = wgetch(scr2);
}
}

```

```

if (cmd[i] == '\b')
{
    bscr2(k, theRow);
    i=i-2;
}
/* reposition command array */
else if (cmd[i] == '\n' || cmd[i] == '\r') /* return or blank get out of loop
/
{
    cmd[i]='\0';
    break;
}
}
/* end read command loop */
/* ***** read data base field ***** */
cmdno = strtoint(cmd);
if ((cmdno == 11) || (cmdno == 10) || (cmdno == 1)) return;
theRow = DBROW;
ccol = DBCOL;
wmove(scr2, theRow, ccol);
wrefresh(scr2);
for (i=0; i<13; i=i+1)
{
    k=ccol + i;
    ii.db[i] = wgetch(scr2); /* read database field*/
    if (ii.db[i] == '\b') /* if backspace then backup */
    {
        bscr2(k, theRow);
        i=i-2;
    }
    else if (ii.db[i] == '\n' || ii.db[i] == '\r') /* set array back to mistake point */
    {
        ii.db[i] = '\0';
        break;
    }
}
}

```

```

}
/* ***** end for database loop */
/* ***** read relation field ***** */
if (cmdno == 2) return;
if (cmdno != 3) return; /* temp. till other options added */
theRow = RELROW;
ccol = RELCOL;
wmove(scr2,theRow,ccol);
wrefresh(scr2);
for (i=0;i<13;i=i+1) /* read up to 12 char from screen */
{
k=ccol + i;
ii.rel[i] = wgetch(scr2); /* read relation field */
if (i==0)
{
if(ii.rel[0] == ' ' || ii.rel[0] == '\n')
{
ii.rel[0] = '\0';
nofld = 'y';
break;
}
}
if (ii.rel[i] == '\b') /* if backspace then backup
{
bsscr2(k,theRow);
i=i-2;
}
else if (ii.rel[i] == '\n' || ii.rel[i] == ' ') /* return or blank */
{
ii.rel[i] = '\0';
break;
}
} /* end for relation loop */
}

```

```

/* ***** read the field field ***** */
if ((cmdno == 3) || (nofld == 'y')) return;
ccol = BFLDCOL;
wmove(scr2, theRow, ccol);
wrefresh(scr2);
for (i=0; i<13; i=i+1)
{
    k=ccol + i;
    ii.fld[i] = wgetch(scr2); /* do read from field field */
    if ((i==1) && (ii.fld[0] == ' ' || ii.fld[0] == '\n'))
    {
        ii.fld[i] = '\0';
        nofld = 'y';
        break;
    }
    if (ii.fld[i] == '\b') /* if backspace then backup
    {
        bsscr2(k, theRow);
        i=i-2;
    }
    else if (ii.fld[i] == '\n' || ii.fld[i] == ' ')
    {
        ii.fld[i] = '\0';
        break;
    }
}
/* end for field loop */
/* end read browse routine */
}
}

```

```

/* ***** */
/* backspace screen 2
/* ***** */
void
bscr2(k,theRow)
int k,theRow;
{
int j,row;
j=k;
row=theRow;
j=j-1;
wmove(scr2,1,j);
wrefresh(scr2);
wmove(scr2,row,j);
wrefresh(scr2);
wprintw(scr2,"%s",blanks);
touchwin(scr2);
wrefresh(scr2);

wmove(scr2,row,j);
wrefresh(scr2);
}
/* ***** */
/* create a new data base or change data base main menu
/* ***** */
void
createdb()
{
if (accesscode != 'd')
{
badflag='y';
return;
}
}
/* ***** */
/* reposition cursor and blank out bad input*/

```



```

scrmanqno=11200;
display_create();
read_createmm();
cmdno=str2int(cmd);
switch (cmdno)
{
case 1:
    endflag='y';
    while (endflag=='y' && badflag == 'n')
        cdbrtln();
    break;
case 4:
    curflag='y';
    while (curflag=='y' && badflag == 'n')
        cme1rtn();
    break;
case 2: case 5:
    msgno=1;
    display_msg();
    display_exit();
    break;
case 7:
    display_exit();
    break;
default:
    help_rtn();
    break;
}
/* end switch */
/* end create db */
}

```

```

/* *****
/* Display Screen for Create Ingress Data Base Main Menu
/* *****
void
display_create()
{
clear();
refresh();
wclean(scr4);
wrefresh(scr4);
wmove(scr4,CSTART,CSTARTC);
crow = CSTART;
ccol = CSTARTC;

wprintw(scr4,"
wprintw(scr4,"COMMAND:
\n");
Dwprintw(scr4,"*****
*****\n");
wprintw(scr4,"The following commands are available\n");
wprintw(scr4,"Enter the command number and a valid KSUIDS password\n");
wprintw(scr4,"*****
*****\n");
wprintw(scr4,"COMMAND : Options available\n");
wprintw(scr4," 1 : Create a new data base.\n");
wprintw(scr4," 2 : Create new data base relations, copy from old relation.\n");
wprintw(scr4," 3 : Create new data base relations, copy from multiple relations.
\n");
wprintw(scr4," 4 : Create new data base relations, enter new information.\n");
wprintw(scr4," 5 : Create secondary Index for a relation.\n");
wprintw(scr4," 6 : Get Help\n");
wprintw(scr4," 7 : Exit Choices\n");
wmove(scr4,CHOMER,CHOMEC);
touchwin(scr4);
wrefresh(scr4);
}

```

```

/*****
/* read the ingres create main menu screen scra
*****/
void
read_createMM()
{
void bsscr4();
int i,k;
char ch;
theRow = CMDROW;
ccol = CMICOL;
wmove(scra,theRow,ccol);
touchwin(scra);
wrefresh(scra);
for (i=0;i<2;i=i+1)
{
k=ccol + i;
cmd[i] = wgetch(scra);
if (cmd[i] == '\b')
{
bsscr4(k,theRow);
i=i-2;
}
else if (cmd[i] == '\n' || cmd[i] == '^') /* reposition command array */
{
cmd[i]='\0';
break;
}
}
}
/* end read command loop */
/* end read-logon routine */

```

```

/* ***** */
/* backspace screen 4
/* ***** */
void
bsscr4(k, theRow)
int k, theRow;
{
int j, row;
j=k;
row=theRow;
j=j-1;
wmove(scr4, 1, j);
wrefresh(scr4);
wmove(scr4, row, j);
wrefresh(scr4);
wprintw(scr4, "Zs", blanks);
touchwin(scr4);
wrefresh(scr4);

wmove(scr4, row, j);
wrefresh(scr4);
}
/* ***** */
/* create/change an Ingres data base
/* ***** */
void
cddbbtn (
{
if (accesscode != 'd')
{
badflag = 'y';
return;
}
}

```

/* reposition cursor and blank out bad input*/

/* reposition cursor */

```

}
scrmanqno=11210;
display_ccdb();
read_createdb();
cmdno=str2int(cmd);
switch (cmdno)
{
case 1:
    if (ii.db == '\0') /* test for entry of db name */
    {
        msgno =2;
        display_msg();
        display_exit();
    }
    else
    {
        createshell(); /* call shell routine to do create */
        display_exit();
    }
    break;

case 2:
    if (ii.db == '\0') /* test for entry of db name */
    {
        msgno =2;
        display_msg();
        display_exit();
    }
    else
    {
        changeshell();
        display_exit();
        break;
    }
}
}

```

```

case 4:
    display_exit();
    break;
default:
    help_rtn();
    break;
    /* end switch */
}
/* end create/change routine */

/* *****
/* Display Screen for Create Ingress Data Base or Change Existing Options */
/* *****
void
display_ccdb()
{
clear();
refresh();
wclear(scr3);
wrefresh(scr3);
wmove(scr3,CSTARTR,CSTARTC);
crow = CSTARTR;
ccol = CSTARTC;

wprintw(scr3,"
");
wprintw(scr3,"COMMAND:
\n");
wprintw(scr3,"Data Base:
\n");
wprintw(scr3,"*****
*****\n");
wprintw(scr3,"The following commands are available\n");

/* format the screen output */
CREATE NEW DATA BASE or CHANGE EXISTING DATA BASE \

/* clear screen
/* write screen buffer to crt
/* move cursor to output line

```

```

CONCURRENCY: QUERY MODIFICATION: \n");
*****
*****

```

```

wprintw(scr3,"Enter the command number and a valid KSUDDS password\n");
wprintw(scr3,"Enter the name of any field with ( ) in text in the space above.\n"
);
wprintw(scr3,"Enter additional options after depressing the return key\n");
wprintw(scr3,"*****\n");
*****\n";
wprintw(scr3,"COMMAND: Options available\n");
wprintw(scr3," 1 : Create a new data base ().\n");
wprintw(scr3,"      with concurrency          \n");
wprintw(scr3,"      with query modification       \n");
wprintw(scr3," 2 : Change an existing data base ().\n");
wprintw(scr3,"      add concurrency              \n");
wprintw(scr3,"      add query modification      \n");
wprintw(scr3," 3 : Get Help\n");
wprintw(scr3," 4 : Exit Choices\n");
wmove(scr3,CHOMER,CHOMED);
Dtouchwin(scr3);
wrefresh(scr3);
}
/*****
/* read the createdb screen -- scr3
/*****
void
read_createdb()
{
void bsscr3();
int i,k,q,t,ct;
/* ***** read the command field *****
theRow = CMDROW;
ccol = CMDCOL;
wmove(scr3,theRow,ccol);
touchwin(scr3);
wrefresh(scr3);
}

```

```

for (i=0;i<2;i=i+1)
{
    k=ccol + i;
    cmd[i] = wgetch(scr3);
    if (cmd[i] == '\b')
    {
        bsscr3(k,theRow);
        i=i-2;
    }
    /* reposition command array */
    else if (cmd[i] == '\n' || cmd[i] == '\r') /* return or blank */
    {
        cmd[i] = '\0';
        break;
    }
}
/* end read command loop */
/* ***** read data base field ***** */
cmdno = strtoint(cmd);
if ((cmdno == 4) || (cmdno == 3)) return;
theRow = RRROW;
ccol = RRRCOL;
wmove(scr3,theRow,ccol);
wrefresh(scr3);
for (i=0;i<13;i=i+1)
{
    k=ccol + i;
    u.dbc[i] = wgetch(scr3);
    /* read database field*/
    if (u.dbc[i] == '\b')
    {
        bsscr3(k,theRow);
        i=i-2;
    }
    /* if backspace then backup */
    /* set array back to mistake point */
}

```



```

else if (ii.dbc[i] == '\n' || ii.dbc[i] == ' ') /* return or blank */
{
    ii.dbc[i] = '\0';
    break;
}
}
/* ***** */ /* end for database loop */
/* ***** */ read concurrency field *****
theRow = CONROW;
ccol = CONCOL;
wmove(scr3, theRow, ccol);
wrefresh(scr3);
for (i=0; i<2; i=i+1)
{
    k=ccol + i;
    ii.con[i] = wgetch(scr3); /* read concurrency field */
    if (ii.con[i] == '\b') /* if backspace then backup */
    {
        bsscr3(k, theRow);
        i=i-2;
    }
    else if (ii.con[i] == '\n' || ii.con[i] == ' ') /* set array back to mistake point */
    {
        ii.con[i] = '\0';
        if (ii.con[0] == 'n') ct = 0;
        else
        {
            ct = 1; /*default yes*/
            ii.con[0] = 'y';
        }
        break;
    }
}
/* end for concurrency loop */
}

```

```

/* ***** read the query field ***** */
theRow = QRYROW;
ccol = QRYCOL;
wmove(scr3,theRow,ccol);
wrefresh(scr3);
for (i=0;i<2;i=i+1)
{
    k=ccol + i;
    ii.qry[i] = wgetch(scr3);
    if (ii.qry[i] == '\b') /* if backspace then backup */
    {
        bsscr3(k,theRow);
        i=i-2;
    }
    else if (ii.qry[i] == '\n' || ii.qry[i] == ' ') /* set array back to mistake point */
    {
        ii.qry[i] = '\0';
        if (ii.qry[0] == '\n') qt = 0;
        else
        {
            ii.qry[0] = 'y'; /*default yes*/
            qt = 1;
        }
        break;
    }
} /* end for query loop */

```

```

switch (ct)
{
    case 0:          /* no concurrency */
        if (qt == 0) /* no query modification */
            optno = 4;
        else
            optno = 2; /* with query modification */
        break;
    case 1:          /* with concurrency */
        if (qt == 0) /* no query modification */
            optno = 3;
        else
            optno = 1; /* with query modification */
        break;
}
/* ***** */
/* end read createdb routine */
/* Backspace for screen 3
/* ***** */
void
bescr3(k, theRow)
int k, theRow;
{
    int j, row;
    j=k;
    row=theRow;
    j=j-1; /* reposition cursor and blank out bad input*/
    wmove(scr3,1,j);
    wrefresh(scr3);
    wmove(scr3,row,j);
    wrefresh(scr3);
    wprintw(scr3,"%s",blanks);
}

```

```

touchwin(scr3);
wrefresh(scr3);

wmove(scr3,row,j); /* reposition cursor */
wrefresh(scr3);

}
/* *****
/* create a new relation
/* *****
void
cmrelrtn()
{
extern int iappidx();
extern void icrel();
int rc;
if ((accesscode != 'd') && (accesscode != 'u') && (accesscode != 'a'))
{
badflag='y';
return;
}
scrmanqno=11240;
display_cnr();
read_relation();
cmdno=str2int(cmd);
switch (cmdno)
{
case 1:
if (fldcntr == 0)
{
if (msgno != 3)
{
msgno = 4; /* invalid fields entered */
display_msg();
}
display_exit();
}
}
}

```

```

    }
    else
    if ((ii.dbf[0] == '\0') || (ii.rel[0] == '\0'))
    {
        * msgno = 2; /* invalid data base or relation name */
        display_msg();
    }
    else
    {
        clear();
        refresh();
        move(CSTARTR,CSTARTC);
        printw("Processing your request\n");
        refresh();
        rc = lappendx(2);
        if (rc == 0)
            icrel();
    }
    display_exit();
    break;
case 3:
    display_exit();
    break;
default:
    help_rtn();
    break;
}
/* end switch */
/* end create relation routine */
/* ***** */
/* Display Screen for Create Ingress create new relation */
/* ***** */
void
display_cnr()

```



```

/* ***** */
void
dds_pwd_rtn()
{
    /* dummy routine that can be revised to add data dictionary password security */
    badflag='n';
}
/* ***** */
/* display error messages to screen routine */
/* ***** */
void
display_msg()
{
    char c;
    clear();
    refresh();
    move(2,2);
    switch (msgno)
    {
        case 1:
            printf("Sorry, this function is not implemented in the prototype system\n\n");
            printf("Hit any key when you wish to continue.\n");
            refresh();
            c = getch();
            break;
        case 2:
            printf("No entry was made in a required field, please reenter.\n\n");
            printf("Hit any key when you wish to continue.\n");
            refresh();
            c = getch();
            break;
    }
}

```

```

case 3:
    printf("Bad attribute entered in a field, please reenter the screen\n\n");
    printf("Hit any key when you wish to continue.\n");
    refresh();
    c = getch();
    break;
case 4:
    printf("Invalid fields or attributes entered, reenter the screen\n\n");
    printf("Hit any key when you wish to continue.\n");
    refresh();
    c = getch();
    break;
} /* end switch */
} /* end routine */

/* *****
/* Exit options routine
/* *****
void
display_exit()
{
    void bsscr6();
    int i,k;
    clrscr();
    refresh();
    wclear(scr6);
    wrefresh(scr6);
    wmove(scr6,CSTARTR,CSTARTC);
}
/* clear screen
/* write screen buffer to crt
/* move cursor to output line
/* format the screen output */

```



```

wprintw(scr6, "
\n");
wprintw(scr6, "COMMAND:
\n");
wprintw(scr6, "*****\n");
wprintw(scr6, "*****\n");
wprintw(scr6, "The following commands are available\n");
wprintw(scr6, "Enter the command number above in the command field
\n");
wprintw(scr6, "*****\n");
wprintw(scr6, " 1 : Continue with the present function
\n");
wprintw(scr6, " 2 : Exit the KSUMMS Data Dictionary System
\n");
if (scrmanngo != 10000)
{
wprintw(scr6, " 3 : Return to the KSUMMS Data Dictionary Main Menu
\n");
if (scrmanngo != 11000)
{
wprintw(scr6, " 4 : Return to the Ingres Data Dictionary Main Menu
\n");
if ((scrmanngo != 11100) && (scrmanngo != 11200))
{
wprintw(scr6, " 5 : Return to the Create Main Menu Screen
\n");
}
}
}
touchwin(scr6);
wrefresh(scr6);
/* ***** read command field ***** */
theRow = CMDROW;
ccol = CMDCOL;
move(scr6, theRow, ccol);
touchwin(scr6);
wrefresh(scr6);

```

```

for (i=0;i<2;i=i+1)
{
/* read 1 character command */
k=ccol + i;
cmd[i] = wgetch(scr6);
if (cmd[i] == '\b')
{
bsscr6(k,theRow);
i=i-2;
}
else if (cmd[i] == '\n' || cmd[i] == ' ') /* return or blank exit */
{
cmd[i] = '\0';
break;
}
}
/* **** test choices set flags for exit level *****/
cmdno = str2int(cmd);
switch (cmdno)
{
case 1: /* do nothing return to where you were */
break;
case 2: /* get out of everything */
badflag = 'y';
break;
case 3: /* return to KSUIDS main menu*/
badflag = 'n';
logflag = 'y';
ingflag = 'n';
browflag = 'n';
createflag = 'n';
cnrflag = 'n';
cnbflag = 'n';
break;
}

```

```

case 4: /* return Ingres main menu*/
    badflag = 'n';
    logflag = 'y';
    ingflag = 'y';
    browflag = 'n';
    createflag = 'n';
    cmrflag = 'n';
    cndbflag = 'n';
    break;
case 5: /* return create main menu*/
    badflag = 'n';
    logflag = 'y';
    ingflag = 'y';
    browflag = 'y';
    createflag = 'y';
    cmrflag = 'n';
    cndbflag = 'n';
    break;
default:
    break;
}
}
/* *****
/* routine to emulate backspacing
/* *****
void
bsscr6(k,theRow)
int k,theRow;
{
int j,row;

```

```

j=k;
row=theRow;
j=j-1;
wmove(scr6,1,j);
wrefresh(scr6);
wmove(scr6,row,j);
wrefresh(scr6);
wprintw(scr6,"%s",blanks);
touchwin(scr6);
wrefresh(scr6);

wmove(scr6,row,j);
wrefresh(scr6);
}
/*****
/* display help messages to screen routine
*****/
void
help_rtn()
{
char c;
clear();
refresh();
move(2,2);
switch (scrmango)
{
case 10000:
printw("This is the KSU Data Dictionary Main Menu.\n");
printw("Options implemented are 1,4 + 5. \n");
break;
case 11000:
printw("You are at the Ingres Dictionary Main Menu screen.\n");
printw("Options implemented are 1,2,9 + 10. \n");
break;
}
}

```

```

case 11100:
    printw("You are at the Browse the Dictionary screen.\n");
    printw("Options implemented are 1,2,3,10 + 11. \n");
    printw("Max # characters for data base, relation, field is 12ch\n");
    printw("You must depress the return key after each entry\n");
    printw("Valid names are expected for required fields\n");
    break;

case 11200:
    printw("You are at the Create Main Menuscreen.\n");
    printw("Options implemented are 1,4,6 + 7. \n");
    break;

case 11210:
    printw("You are at the create a new data base screen.\n");
    printw("Options implemented are 1,2,3, + 4. \n");
    printw("Max # characters for the data base field is 12ch\n");
    printw("The data base may not be in the dictionary already\n");
    printw("For concurrency or query modification enter y or n.\n");
    printw("Concurrency means multiple people can use the same data base\n");
    printw("Query Modification means access to relations is restricted to the\n");
    printw(" creator of the relation. Once turned on this can not be changed.\n");
    printw(" Con. and QM default to yes if no entry. \n");
    printw("You must depress the return key after each entry\n");
    break;

case 11240:
    printw("You are at the create a new relation screen.\n");
    printw("Options implemented are 1,2 + 3 \n");
    printw("Max # characters for the DB, Rel., Fid fields is 12ch\n");
    printw("The relation may not be in the dictionary already for that DB\n");
    printw("The valid attributes are listed on the screen if an invalid entry\n");
    printw("is made you must reenter the whole screen, be careful.\n");
    printw("The present implementation only allows 10 field entries.\n");
    printw("Depressing the return key with no entry will exit field entry mode\n");

```

```

    printw("You must depress the return key after each entry\n");
    break;
default:
    printw("No help available, sorry.\n");
    break;
} /* end case */
printw("Enter the command number desired and hit return key.\n");
printw("You will automatically go to the next field for entry or \n");
printw("    or the next screen. \n");
printw("The return key enters the field information, \n");
printw("    back spacing is permitted only before entry. \n");
printw("Once a field has been entered no further changes are permitted.\n");
printw("Exit options allows you to change screens or exit the program. \n");
printw("Password Security is not implemented, depress return key to bypass\n");
};
printw("\n
refresh();
c = getch();
refresh();
}
void
ingpwrtn()
{
    /* dummy routine to validate ingres dictionary passwords */
    badflag='n';
    accesscode='d';
}

```

```

/*****
/* read the create relation screen
*****/
void
read_relation()
{
    short i,k,j,x,rc;
    char getout,efld;
    void bsscr5();
    int validate_attr();
    /* *****/
    getout = 'n';
    theRow = CMDROW;
    ccol = CMICOL;
    wmove(scr5,theRow,ccol);
    touchwin(scr5);
    wrefresh(scr5);
    for (i=0;i<2;i=i+1) /* read command */
    {
        k=ccol + i;
        cmd[i] = wgetch(scr5);
        if (cmd[i] == '\b') /* backspace */
        {
            bsscr5(k,theRow);
            i=i-2;
        }
        else if (cmd[i] == '\n' || cmd[i] == ' ') /* new line or space */
        {
            cmd[i]='\0';
            break;
        }
    }
}
/* end read command loop */

```

```

/* ***** read data base field ***** */
cmdno = str2int(cmd);
if ((cmdno == 2) || (cmdno == 3)) return;
therow = DBROW;
ccol = DBCOL;
wmove(scr5,therow,ccol);
wrefresh(scr5);

for (i=0;i<13;i=i+1) /* read data base field */
{
    k=ccol + 1;
    ii.db[i] = wgetch(scr5); /* read database field*/

    if (ii.db[i] == '\b') /* backspace */
    {
        bscr5(k,therow);
        i=i-2;
    }
    else if (ii.db[i] == '\n' || ii.db[i] == '\0') /* newline or space */
    {
        ii.db[i] = '\0';
        break;
    }
} /* end read db field loop */

if (ii.db[0] == '\0') return; /* no db entered getout */

```



```

/* ***** read relation field ***** */
theRow = RELROW;
ccol = RELCOL;
wmove(scr5,theRow,ccol);
wrefresh(scr5);

for (i=0;i<13;i+1) /* read relation */
{
    k=ccol + i;
    ii.relc[i] = wgetch(scr5);
    if (i==0)
    {
        if(ii.relc[0] == ' ' || ii.relc[0] == '\n')
        {
            ii.relc[0] = '\0';
            getout = 'y';
            break;
        }
    }
    if (ii.relc[i] == '\b') /* backspace */
    {
        bscr5(k,theRow);
        i=i-2;
    }
    else if (ii.relc[i] == '\n' || ii.relc[i] == ' ') /* return or blank */
    {
        ii.relc[i] = '\0';
        break;
    }
} /* end for relation loop */
if (getout == 'y') return;

```

```

/* ***** read the fields ***** */
theRow = SFLDR;
fldcntr = 0;
x = 1;
efld = 'n';

while (efld == 'n')
{
  for (j=0;j<10;j=j+1) /* read up to 10 fields and attributes*/
  {
    if (efld == 'y') break; /* getout of read fields*/
    ccol = SFLDC;
    if (x==2) /* 2nd pass move to col 30 */
      ccol = SFLDC + 29;

    wmove(scr5,theRow,ccol);
    touchwin(scr5);
    wrefresh(scr5);

    for (i=0;i<13;i=i+1)
    {
      k=ccol + i;
      flds[j].mfldci = wgetch(scr5);
      if (flds[j].mfldci == '\n')
      {
        bsscr5(k,theRow);
        i=i-2;
      }
      else if ((flds[j].mfldci == '\n') || (flds[j].mfldci == ' '))
      {
        flds[j].mfldci = '\0';
        if (i == 0) efld = 'y';
        break;
      }
    }
  } /* end for field loop */
}

```

```

if (efld == 'y') break; /* if end get out j loop + while */
if (x==1)
    /* set column position */
    ccol = ccol + 14;
else
    ccol = ccol + 15;

wmove(scr5, theRow, ccol);
touchwin(scr5);
wrefresh(scr5);

for (i=0;i<5;i=i+1) /* read attribute */
{
    k=ccol + i;
    flds[j].attr[i] = wgetch(scr5);
    if (flds[j].attr[i] == '\b')
    {
        bscr5(k, theRow);
        i=i-2;
    }
    else if (flds[j].attr[i] == '\n' || flds[j].attr[i] == '\f')
    {
        if (x==1) x=2; /* what field position do I need */
        else
        {
            x = 1;
            theRow = theRow + 1;
        }
    }
    flds[j].attr[i] = '\0'; /*null terminate */
}

```

```

if (i == 0)          /* no entry get out*/
{
    efld = 'y';
    break;
}
rc = validate_attr(j,i); /* good attribute? */
if (rc == 0)
{
    msgno = 3;
    display_msg();
    fldcntr = 0;
    efld = 'y';
}
break;
}
} /* end for field loop */

if (efld == 'n')
    fldcntr = fldcntr + 1;
} /* end for j loop */

efld = 'y'; /* all done get out of field attr loop */
} /* end while */
} /* end read create relation routine */

```

```

/* ***** */
/* backspace screen 5
/* ***** */
void
bscr5(k, theRow)
int k, theRow;
{
    int j, row;
    j=k;
    row=theRow;
    j=j-1;
    wmove(scr5, 1, j);
    wrefresh(scr5);
    wmove(scr5, row, j);
    wrefresh(scr5);
    wprintw(scr5, "%s", blanks);
    touchwin(scr5);
    wrefresh(scr5);

    wmove(scr5, row, j);
    wrefresh(scr5);
}
/* ***** */
/* reposition cursor and blank out bad input*/
/* ***** */

```

```

/*****
/* Validate the attribute field is correct
*/
/*****
int
validate_attr(a,b)
int a,b;
c
char ch,ch2[4];
int lth;

ch = flds[a].attr[0];
ch2[0] = flds[a].attr[1];
ch2[1] = flds[a].attr[2];
ch2[2] = flds[a].attr[3];
ch2[3] = flds[a].attr[4];
lth = strtoint(ch2);
/* int length 1,2,4 */
if ((ch=='i') && (lth != 1) && (lth !=2) && (lth != 4))
return(0);
/* float 4 or 8 */
if ((ch=='f') && (lth !=4) && (lth != 8))
return(0);
/* char 1->255 */
if ((ch=='c') && (lth <= 0) || (lth > 255))
return(0);
/* ok send back true */
return(1);
}

```

```

/* *****
/* create a new data base
/* *****
void createshell()
{
extern int iappidx();
char array[25];
int rc;
clear();
refresh();
move(CSTARTR,CSTARTC);
printw("Processing your request\n");
refresh();
rc = iappidx(1);
if (rc == 0)
{
printf (array, "crshell %o %s",optno,ii.db);
system(array);
printw("transaction completed\n");
refresh();
sleep(2);
}
else
{
printw("A error has occurred during create DB process.\n");
refresh();
sleep(2);
}
}
}

```

```

void changeshell()
{
extern int iappidx();
char array[25];
int rc;
clear();
refresh();
move(CSTARTK,CSTARTC);
printw("Processing your request\n");
refresh();
if ((optno == 3) || (optno == 4))
{
printw("Can't turn off query modification\n");
refresh();
sleep(2);
}
else
{
rc = iappidx(3);
if (rc == 0)
{
sprintf (array,"chshell %d %s",optno,ii.db);
system(array);
printw("transaction completed\n");
refresh();
sleep(2);
}
else
{
printw("A error has occurred during Change DB process.\n");
refresh();
sleep(2);
}
}
}
}

```


ksudds.h

The following routine is a header file for the main program. It contains all the global variable definitions and all the global routine definitions.

```

/*****
/* Header file for the ksudds data dictionary system
*/
/*
/***** author: Mary Campbell
/*****
*/
/*****
/* Screen Buffer Adresses
*/
/* Global Variables
/*****
int crow; /* current row
int ccol; /* current column
int orow; /* old row
int ocol; /* old column
int msgno; /* error message number
int scrmanqno; /* screen management number
int optno; /* option number for create db
char accesscode; /* access code : a=all,d=dba,u=update,r=read
char ch;
int theRow; /* a row on the screen
int cmdrow; /* current row command field
int pwdrow; /* current row password field
int cmdcol; /* current column command field
int pwdcol; /* current column password field
char blanks[3] = {' ',' ',' '}; /* used to correct backspace in curses
int cmdno; /* command field from screens
char cmd [3]; /* dictionary password field for ksudds system
char ipwd [13]; /* ingres password field for ingres dictionary
char badflag="n"; /* flag for breaking out of anything your doing
char logflag="y"; /* flag for logon data dictionary system
char lngflag="y"; /* flag for ingres data dictionary system
char browflag="y"; /* flag for browse function in ingres dds
char createflag="y"; /* flag for create function in ingres dds
char cmdbflag="y"; /* flag for create new data base in ingres dds
char cnflag="y"; /* flag for create new relation in ingres dds
char ingpwdoflag="y"; /* ingres password flag
/*****

```

```

char nofld;
char noqry;
int fldcnt; /* field counter */
struct field /* structure to hold fields for create relation */
{
    char mfld[13];
    char attr[5];
};
struct field flds[20],*fp; /* structure to hold ingres data */
struct inginfo
{
    char dbc[13];
    char con[2];
    char qry[2];
    char rel[13];
    char fld[13];
    char pwd[13];
} ii, *ip;

```

```

/*****
/* Routines
*****/
/*****
/* KSUDDS Primary Menu
/* Ingres data dictionary
/* Displays message on the screen
/* gives help
/* Display exit coices
/* display create main menu
/* Browse ingres dds
/* read the logon screen fields input
/* KSUDDS INGRES Primary Menu
/* KSUDDS INGRES Browse
/* KSUDDS INGRES Create/change data base
/* KSUDDS INGRES Create Menu
/* KSUDDS INGRES Create new relation
/* create/change data base routine
/* create new relation routine
/* read database fields
/* read relation fields
/* call shell routine to create a data base
/* call shell to change a db option
*****/
void display_logon();
void ingresdds();
void display_msg();
void help_rtn();
void display_exit();
void createdb();
void browse_rtn();
void read_logon();
void display_ing();
void display_browse();
void display_ccdb();
void display_create();
void display_cnr();
void ccdb_rtn();
void cnr_rtn();
void read_dbflds();
void read_rel_flds();
void createshell();
void changeshell();
void read_createdb();
void read_createm();
void read_ingmm();
void ingpwd_rtn();
void dds_pwd_rtn();
void read_relation();
int str2int();

```

ksuddsd.h

The following routine is a global define header file. It contains the global constants used in the program.

```

/* ksuddsd.h is a header file for define constants
/* this file is included in the ksudds data dictionary system
/*
/*
#define CHOMER 2
#define CHOMEC 9
#define CMDROW 2
#define CMDCOL 9
#define PWDRROW 2
#define PWDCOL 49
#define CSTARTR 1
#define CSTARTC 1
#define DBCOL 11
#define DBROW 3
#define BFLPCOL 53
#define BFLPDRROW 3
#define SELDR 13
#define SELFC 0
#define RELCOL 33
#define RELROW 3
#define CONCOL 38
#define CONROW 3
#define GRFCOL 60
#define GRFDROW 3
#define MAXFLDS 20
/* cursor home row */
/* cursor home column */
/* cursor cmd field */
/* cursor cmd column */
/* password field row */
/* password field column */
/* cursor start row */
/* cursor start column */
/* cursor db row */
/* cursor fld column */
/* cursor fld row */
/* cursor mult flds row */
/* cursor mult fld start col */
/* cursor rel column */
/* cursor rel row */
/* cursor concurrency column */
/* cursor concurrency row */
/* cursor query column */
/* cursor query row */
/* max. no. fields */

```

ing.q.h

The following routine is a header file for the Ingres sub-routines. It contains the external structures passed to these programs.

```

## extern int fldcnt; /* field counter */
## extern struct field /* structure to hold fields for create relation */
## {
##     char mfid[13];
##     char attr[5];
## }
## extern struct field flds[20],*fp;
## extern struct inginfo
## {
##     char db[13];
##     char cont[2];
##     char gry[2];
##     char rel[13];
##     char fld[13];
##     char pwd[13];
## } ii;
##
##
##
##
##
##
##
##
##
##
##
##

```


ksuddsi1.q

The following routine is an Ingres Equel program. The program updates the Ingres Data Dictionary and the Local Index when adding new data bases, relations, or fields through the KSUDDS data dictionary system. The Ingres data dictionary in the KSUDDS data dictionary prototype system consists of the an Ingres data base which holds relations for the Local Index and relations for the data portion of the Ingres dictionary. The Master Index is not implemented and is not required until the KSUDDS system is implemented in a distributed environment.

```

#include "/usr/att/campbell/proj.impl/ing-q.h"
#include <stdio.h>
#define idbn[13];
#define applname[10];
#define sysonname[8];
#define sno,cno,mno,dno;
#define tempname[13];
#define ireln[13];
#define ifldn[13];
#define iattr[5];
#define type[2];
#define concur[2];
#define qrym[2];
int
.iappidx(op)
D int op;
C
int i,f;
## ingres ksudds1
## range of s is seedrec
## range of l is lname
## range of i is iddsdb
## range of t is iddstol
## range of f is iddsfld
strcpy(applname, "ingres");
strcpy(sysonname, "ksuvax1");
printf(idbn, "%s", ii.db);

```

```

if (op == 1)      /* if adding a new data base */
{
  sprintf(concur,"%s",ii-con);
  sprintf(qryM,"%s",ii-qry);
  strcpy(type,"d");      /* get current id no */
## retrieve (sno = s-startno, cno=s-curno, mno=s-maxno)
## where s-startno=1
  if (cno >= mno)      /* any more room in dictionary */
  {
##      exit
##      printf("The KSUDDS! Ingres Data Dictionary is full!\n");
##      return (5);
  }
  else
  {
##      f = 0;      /* see if already exists */
##      retrieve (tempname = i.dbname) where i.dbname = idbn
##      {
##          f = 1;
##      }
##      if (f==1)      /* yes get out */
##      {
##          exit
##          printf("The requested data base already exists!\n");
##          return(6);
##      }
##      else      /* no update curent no, index, and dictionary */

```

```

    {
        cno = cno + 1;
        replace s(curno=cno) where s.startno =1
        append to iddsdb(
            dbname=idbn, appl=applname,
            homesys=syssonname, id=type, num=cno,
            con=concur, dry=qryM)
        append to lname(
            name=idbn, appl=applname,
            syson=syssonname, id=type, num=cno)
        exit
        return(0);
    }
}

if (op==2) /* new relation and fields */
{
    strcpy(type,"d");
    f = 0; /* see if already exists */
    ## retrieve (dbno = 1.num) where l.name = idbn and l.id = type
    ## {
        f = 1;
    }
    ## if (f==0) /* no get out */
    {
        ## exit
        printf("The requested data base: %s is not valid \n",ii.db);
        return(1);
    }
    f=0;
    sprintf(iireln,"%s",ii.rel);
    sprintf(type,"%t");
}

```

```

## retrieve (tempname = l.name) where l.name = ireln and l.num = dbno
##
##      f = 1;
##    }
##  if (f==1) /* yes get out */
##    {
##      exit
##      printf("The relation: %s already exists in data base: %s\n",ii.rel,ii.db);
##      return(1);
##    }
##
##      append to iddstbl(
##        tplname=ireln, dbname=idbn,
##        id=type, num=dbno)
##      append to lname(
##        name=ireln, apcl=apclname,
##        syson=sysonname, id=type, num=dbno)
##    for (fp=flds;fp<<(flds+fldcptr);fp++)
##    {
##      sprintf(ifldn,"%s",fp->mfld);
##      sprintf(iattr,"%s",fp->attr);
##      strcpy(type,"f");

```

```

##      append to iddsfld(fldname=ifldn, tblname=ireln,
##      attrib=iatr, idstype, num=dbno)
##      }
##      exit
##      return(0); /* end oc2 */
}

if (op == 3) /* if changing data base options */
{
    sprintf(concur,"%s",ii.con);
    sprintf(qrym,"%s",ii.qry);
    f = 0; /* see if already exists */
    strcpy(type, "d");
    retrieve (tempname = i.dbname) where i.dname = idbn
    {
        f = 1;
    }
    if (f==0) /* get out no data base */
    {
        exit
        printf("The requested data base doesn't exist!\n");
        return(1);
    }
    else /* update the data base */
    {
        replace i (con=concur, qry=query) where i.dname=idbn
        exit
        return(0);
    }
} /* end op 3 */
} /* end program */

```

brow.q

The following routine is an Ingres Equel program. The program browses the Ingres Data Dictionary system when requests are made to list the data bases, relations, or fields in the Ingres data dictionary. This sub-routine is called from the main program.

```

#include "/usr/att/cambell/proj.impl/ing.q.h"
#include <stdio.h>
##char oddl[13];
##char tdk[13];
##char ofid[13];
##char tfid[13];
##char otbl[13];
##char oattr[5];
##char tip1[13];
##char type[2];
##int dbno;
int
Ibrow(cmdx)
int cmdx;
{
int i,f;
char ch;
## ingres ksudds1
## range of a is lname
## range of b is iddsdb
## range of c is iddsfld
## range of d is iddstpl
if (cmdx == 1) /* list all data base names */
{
f = 0;
i = 1;
strcpy(type,"d");
retrieve (odb = a.name)
where a.id = type
}
}

```



```

f=1;
if (i==20)
{
    printf ("\nRepress any key, when you wish to continue.\n");
    scanf ("%c",ch);
    i = 0;
}
printf ("%Zs\n",odb);
i = i + 1;
}
## if (f==0)
{
    printf ("\nNo data bases found in the dictionary!! \n");
    sleep(2);
    exit
    return(-1);
}
else
{
    printf ("\nRepress any key, when you wish to continue.\n");
    scanf ("%c",ch);
    exit
    return(0);
}
} /* end list db */
##

```

```

if (CMDX == 2)      /* list all relations names */
{
    f = 0;
    i = 1;
    sprintf(tdb,"%s",ii.db);
    printf("\n\nListing relation names for data base: %s\n\n",ii.db);
    retrieve (otpl = d.tblname)
        where d.dbname = tdb
    {
        f=1;
        if (i==20)
        {
            printf ("\nRepress any key, when you wish to continue.\n");
            scanf ("%c",&ch);
            i = 0;
        }
        printf("%s\n",otpl);
        i = i + 1;
    }
    if (f==0)
    {
        printf("\nNo relations in data base: %s \n",ii.db);
        sleep(2);
        exit
        return(-1);
    }
    else
    {
        printf ("\nRepress any key, when you wish to continue.\n");
        scanf ("%c",&ch);
        return(0);
    }
} /* end list relation */

```

```

if (cmdx == 3) /* list all field names */
{
    printf(tdb,"%s",ii.db);
    printf(ttcl,"%s",ii.rel);
    printf("\n\n Listing fields for data base: %s, relation %s\n\n",ii.db,ii.r
el);
    f = 0;
    i = 1;
    retrieve (dbno = b.num) where b.dbname = tdb
    retrieve (ofld = c.flname, oattr =c.attr) where
    c.tblname = ttcl and
    c.num = dbno
    {
        f=1;
        if (i==20)
        {
            printf ("\nDepress any key, when you wish to continue.\n");
            scanf ("%c",&ch);
            i = 0;
        }
        printf("%s : %s\n",ofld,oattr);
        i = i + 1;
    }
    if (f==0)
    {
        printf("\nNo fields found for data base: %s, relation %s\n",ii.db,ii.rel);
        sleep(2);
        exit
        return(-1);
    }
}

```

```
else
{
    printf ("\nDepress any key, when you wish to continue.\n");
    scanf ("%c",ch);
    exit
    return(0);
}
} /* end list field */
} /* end browse dictionary */
```

crel.q

The following routine is an Ingres Equel program. The routine is used when creating a new relation through the Ingres data dictionary. The routine will create the desired relation in the data base name passed to it from the main program.

```

#include "/usr/att/campbell/proj.impl/ing.d.h"
#include <stdio.h>
#include <string.h>
#define dbn dbn[13];
#define tail tail[13];
void
icrel()
{
    int i,k;
    char *otrs[20];
    sprintf(dbn,"%s",ii.db);
    sprintf(tbl,"%s",ii.rel);
    i = 0;
    for (fp=flds;fp<(flds+fldcnt);fp++)
        {
            ptrs[i] = fp->mfld;
            ptrs[i+1] = fp->attr;
            i=i+2;
        }
    ## ingres dbn
    switch (fldcnt)
    {
        case 1:
            ## param create tbl("%c=%c",ptrs)
            break;
        case 2:
            ## param create tbl("%c=%c,%c=%c",ptrs)
            break;
        case 3:
            ## param create tbl("%c=%c,%c=%c,%c=%c",ptrs)
            break;
    }
}

```

```

case 4:
    param create tol("Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc",ptrs)
    break;
case 5:
    param create tol("Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc",ptrs)
    break;
case 6:
    param create tol("Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc",ptrs)
    break;
case 7:
    param create tol("Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc",ptrs)
    break;
case 8:
    param create tol("Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc",ptrs)
    break;
case 9:
    param create tol("Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc",ptrs)
    break;
case 10:
    param create tol("Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc,Zc=Zc",ptrs)
    break;
}
## exit
return;
}

```

crshell

The following routine is a Unix Shell program that will perform the actual creation of a new Ingres data base when requested in the KSUDDS data dictionary system. This shell is called through the main program.


```
case $1 in
  1)
    creatdb -c +q $2 > crfile
    exit 0
    ;;
  2)
    creatdb -c +q $2 > crfile
    exit 0
    ;;
  3)
    creatdb -c -q $2 > crfile
    exit 0
    ;;
  4)
    creatdb -c -q $2 > crfile
    exit 0
    ;;
  *)
    echo "error occurred during create db: $1 : $2 " > erfile
    exit 1
    ;;
esac
```

chshell

The following routine is a Unix Shell program that will perform the a change to an Ingres data base that already exists. The routine will change the concurrency options or the query modification options of an existing data base. This routine is call through the main program.

```

case $1 in
  '1')
    creatdb -e +c +q $2 > crfile
    exit 0
    ;;
  '2')
    creatdb -e -c +q$2 > crfile
    exit 0
    ;;
  '3')
    creatdb -e +c $2 > crfile
    exit 0
    ;;
  '4')
    creatdb -e -c $2 > crfile
    exit 0
    ;;
  *)
    echo "error occurred during create db: $1 : $2 " > erfile
    exit 1
    ;;
esac

```

*The Design and Use of a Data Dictionary System
for the Management of Data Bases and Forms
in an Office Automation System*

by

Mary M. Campbell

B.A. Montclair State College, 1971

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Abstract

The master report "Design and Use of a Data Dictionary System for the Management of Data Bases and Forms in an Office Automation System" presents a design for a data dictionary system in a distributed processor environment. The data dictionary was designed to interface to a data base management system and a forms management system. The design is modular. As the need to interface to other data base management systems or application systems arises, the data dictionary can be expanded to incorporate the new systems meta data. The data dictionary employs menu driven screen entry formats that generate the data dictionary's query language. This allows non computer science personnel a simple method for entry and query of the data dictionary's meta data.

Other topics covered in this masters report include: design criteria for the data dictionary system, security functions, interactive user interfaces and the similarity between data base management functions and forms management functions.