NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A280 131

DTIC QUALITY INSPECTED 2

THESIS

The Design and Implementation of a Funtional/Daplex Data Interface for the Multimodel and Multilingual Database System

by

William Aime Demers
Jon Mark Rogelstad

March 1994

Thesis Advisor: David K Hsiao

Approved for public release; distribution is unlimited.

94-17773

1979

DTIC ELECTE JUN 10 1994 S B D

94 6 9 086

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
TheDesign and Implementation of the Funtional/Daplex Data Interface for the Multimodel and Multilingual Database System (U)

12. PERSONAL AUTHOR(S)
Demers, William Aime and Rogelstad, Jon Mark

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 03/93 TO 03/94	1994, March, 24	198

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Functional Database, Multilingual and Multimodel Database System, MultiBackend Database Computer, Heterogeneous Database System

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

The Multi-Lingual and Multi-Model Database System (MDBS), at the Naval Postgraduate School, is the only database system in which five different database model interfaces operate on a single computer system. The purpose of the MDBS is to prove that multiple database models can coexist and share data within a single computer system. The MDBS provides a platform in which diverse database models can share data, via access to other models' base data, stored in the MDBS common format. The problem was that a Functional database model had not previously been implemented on the MDBS. The goal of this thesis was to provide a Functional/DAPLEX interface for the MDBS.

The approach was to design and implement a set of programs to encapsulate the Functional/DAPLEX language as introduced by D.W Shipman in 1981. It is imperative that the interface maintain the look, feel and characteristics of the Functional data model.

The result is a set of programs, written in C, which implement a Functional/DAPLEX interface, thus a sixth interface, for the MDBS. The programs parse DAPLEX data definition language and data manipulation language constructs, convert them to an MDBS common format, store the data with other models' base data and can reconvert MDBS common data back to a Functional form.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: David K. Hsiao	22b. TELEPHONE (Include Area Code): (408) 656-2253; 22c. OFFICE SYMBOL: CS/Hq

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

Approved for public release; distribution is unlimited

The Design and Implementation of a Functional/Daplex Interface for the Multimodel and Multilingual Database System

by

William Aime Demers
Commander, United States Navy Reserve
Bachelor of Science, Worcester Polytechnic Institute, 1975
and
Jon Mark Rogelstad
Lieutenant, United States Navy Reserve
Bachelor of Science, North Dakota State University, 1979

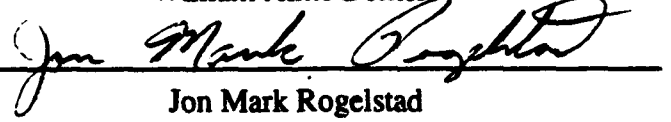
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

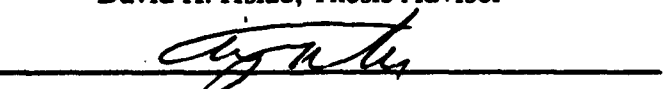
from the

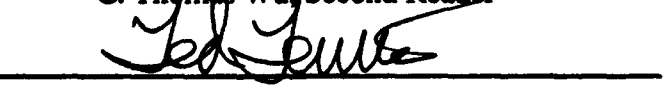
NAVAL POSTGRADUATE SCHOOL
March 1994

Authors:
William Aime Demers


Jon Mark Rogelstad

Approved By:
David K. Hsiao, Thesis Advisor


C. Thomas Wu, Second Reader


Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The Multi-Lingual and Multi-Model Database System (MDBS), at the Naval Postgraduate School, is the only database system in which five different database model interfaces operate on a single computer system. The purpose of the MDBS is to prove that multiple database models can coexist and share data within a single computer system. The MDBS provides a platform in which diverse database models can share data, via access to other models' base data, stored in the MDBS common format. The problem was that a Functional database model had not previously been implemented on the MDBS. The goal of this thesis was to provide a Functional/DAPLEX interface for the MDBS.

The approach was to design and implement a set of programs to encapsulate the Functional/DAPLEX language as introduced by D.W Shipman in 1981. It is imperative that the interface maintain the look, feel and characteristics of the Functional data model.

The result is a set of programs, written in C, which implement a Functional/DAPLEX interface, thus a sixth interface, for the MDBS. The programs parse DAPLEX data definition language and data manipulation language constructs, convert them to an MDBS common format, store the data with other models' base data and can reconvert MDBS common data back to a Functional form.

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

TABLE OF CONTENTS

I. THE INTRODUCTION 1

A. THE MOTIVATION 1
B. THE BACKGROUND 2
C. THE OBJECTIVE 5

II. THE MDBS SYSTEM 6

A. THE BACKGROUND AND DESIGN 6
B. INTERPROCESS COMMUNICATIONS 10

III. THE MODEL LANGUAGE INTERFACE 13

A. STRUCTURE OF THE MODEL LANGUAGE INTERFACE 13
B. THE LANGUAGE INTERFACE LAYER SECTION 13
C. THE KERNEL MAPPING SYSTEM SECTION 14
D. THE KERNEL CONTROLLER SECTION 15
E. THE KERNEL FORMATTING SECTION 15
F. COMMON USE STRUCTURES 15

IV. THE LANGUAGE INTERFACE LAYER (LIL) 18

V. THE KERNEL MAPPING SYSTEM (KMS) 22

A. THE DATA DEFINITION LANGUAGE PARSER (DDLP) 22
1. DECLARE entity ENTITY 24
2. DEFINE function(entity) = *primitive* data type 25
3. DEFINE function(entity1) = entity2 26
4. DEFINE function1(entity1) = INVERSE OF function2(entity 2) 27
5. DEFINE function1(entity1) = function 2(entity 2) 29

B. THE DATA MANIPULATION LANGUAGE PARSER (DMLP) 31
1. FOR 31
a. FOR with NEW Modifier 32
b. FOR without the NEW Modifier 34
2. RETRIEVE 34
a. RETRIEVE function1(entity1) and ... and function n (entity1) .35
b. RETRIEVE with Simple SUCH THAT Clause 35
c. RETRIEVE Statement with Compound SUCH THAT Clause. .36
d. RETRIEVE with WHERE clause 38
3. DELETE Transactions 40

VI. THE KERNEL CONTROLLER (KC) AND KERNEL FORMATTING SYSTEM (KFS) 41
A. THE KERNEL CONTROLLER 41
B. THE KERNEL FORMATTING SYSTEM (KFS) 42
VII. CONCLUSION 45
A. IMPLEMENTATION 45
B. LESSONS LEARNED 46
C. RECOMMENDATIONS FOR FUTURE RESEARCH 47
D. SUMMARY 48
APPENDIX A -USER DEMONSTRATION GUIDE 49
APPENDIX B -PROGRAM CODE 61
LIST OF REFERENCES 188
INITIAL DISTRIBUTION LIST 189

LIST OF FIGURES

Figure 1 The Multimodel, Multilingual and Cross-Model Capability 7
Figure 2 The Multibackend Database Supercomputer (MDBS) 8
Figure 3 MDBS Communication Channels 11
Figure 4 The MDBS Model Interface Layer 14
Figure 5 Structure of MDBS dbid_node 16
Figure 6 Structure of dap_dbid_node 16
Figure 7 Structure of dap_info 17
Figure 8 Module Selection Menu 18
Figure 9 Database Selection Menu 18
Figure 10 Functional Model LIL Interface Menu 19
Figure 11 Sample DAPLEX Schema Display 20
Figure 12 Query Selection Menu 21
Figure 13 Structure of dap_dbid_node 23
Figure 14 Structure of dap_db_entity_node 24
Figure 15 Attribute Node Structure 25
Figure 16 Representative DDL Constructs 26
Figure 17 Entity to Entity Mapping 28
Figure 18 Structure of dap_alias_node 28
Figure 19 Sample Database Template File MAINT.t 30
Figure 20 Sample Database Descriptor File 30
Figure 21 DAPLEX FOR Statement with NEW Modifier 32
Figure 22 DAPLEX FOR Query Example 33
Figure 23 DAPLEX RETRIEVE with Simple SUCH THAT Clause 36
Figure 24 ABDL RETRIEVE COMMON Statement 36
Figure 25 DAPLEX RETRIEVE with Complex SUCH THAT Clause 37
Figure 26 Complex RETRIEVE Example 37
Figure 27 DAPLEX RETRIEVE with WHERE Clause 38
Figure 28 DAPLEX RETRIEVE with WHERE Modifier 39
Figure 29 Comparison of RETRIEVE and DELETE Statements 40
Figure 30 Format of the Retrieved Kernel Data 43

ACKNOWLEDGEMENTS

We would first and foremost like to thank Dr. David K. Hsiao, for allowing us the opportunity to work on the MDBS. His insightful direction and assistance made it possible for us to continue to work on the MDBS project even after his departure for Hong Kong. His insistent push for us to create a clear concise plan and obtain a thorough understanding of the MDBS insured we encountered minimal problems in his absence. We are truly sorry that he will not be able to attend our graduation.

We would like thank to Dr. C. Thomas Wu for working so diligently with us on a project not of his own design. Through his dedication and knowledge of database systems, he provided us with the necessary guidance to successfully complete this area of research. His humor kept our motivation high.

Last, but certainly not least, we would like to thank Mike Williams, Sue Whalen and Rosie Johnson. Their hard work and prompt assistance kept the many computer gremlins from taking-up permanent residence in these older systems. Finally, to John Locke, the only person left here with any historical knowledge and expertise in the code written for the MDBS.

I. THE INTRODUCTION

A. THE MOTIVATION

The amount of data being stored by the United States military continues to grow. The expansion is diverse in both format and content. The content of the data may range from personnel records to supplies to representations of battles to methods of repair. The storage medium used to retain the information is equally diverse utilizing paper, microfiche, photographs, electronic media and very often only within the mind of an individual. The current trend in information management is to store as much data as is possible, electronically. Data stored in electronic format requires significantly less physical storage space than its paper or plastic counterparts. What methods do we use to store the information? Personnel records are often stored in a relational database format consisting of fields and tuples. Supplies may be stored in a network database format with pointers and multi-valued attributes. Battle simulation information may be stored in an object oriented format utilizing methods and inheritance. To store and retrieve the data effectively and efficiently is the task of a database management system.

As each of the terms associated with its corresponding electronic format is varied so are the computer languages used by the database management systems to process the information. Each of the languages has its individual advantages based on its intended usage. Many have been implemented for a long time and contain vast amounts of data. To convert them to a newer and more efficient language would require substantial financial and human resources. In this era of budget reductions and personnel drawdowns, neither is readily available. The databases are normally segregated requiring diverse commands often incomprehensible to other database languages (Hsia92a). This is similar to having an economist and a physicist trying to discuss their area of expertise. While both may use English, the terms and idioms used will probably not be understood by the other. Significant amounts of time and money have been invested into the capabilities of these systems as well as training the personnel to use them. Unfortunately the skills and

knowledge of one system do not normally transfer to those of another system. Not only is this true of the human users, it is also true of the computer systems that must interact with each other. This is the main problem of today's heterogeneous database storage. Since the diverse models cannot communicate directly with each other, managers are required to duplicate the data in multiple systems. There are other problems associated with this type of approach to data management too, specifically, data redundancy (multiple space for same data) and accuracy (ensuring updates are promulgated to every system). As a direct result of the data redundancy and the lack of inter-model communications, data inconsistencies are bound to occur, sometimes with disastrous results (Hsia92b).

B. THE BACKGROUND

There are different approaches to solving this problem. One method may be to require the users to become proficient in all the languages necessary to access the data. They would then have to put any interrelated data together via their own methods. This would require a huge undertaking to train all current users as well as training each new user in multiple languages. It does not solve the problem of inter-model communications, data redundancy and inconsistent data. Another method could be to require all data to be in a specific format so that the users would only be required to know one language. This method would require a phenomenal amount of effort and programming to change over all data, programs and retrain the users to become proficient in this new database language. In addition, as new database methods are developed, this same undertaking must be done to use the new database model. In this time of military budget reductions and fiscal austerity, it is probably not a viable option. Another option could be to allow the user to continue operating as normal and let the computer do the required translations.

Allowing the computer to perform the translations would permit the users to continue using the data language they are trained in and would require no changes to previously written interface programs. The translations could be accomplished in one of two ways. First keep the data in the native language format and let each language translate into and

out of the other languages. The problem with this approach is the number of translation routines grows at a rate of $O(n^2)$. With a very small number of database languages, this could be an acceptable method. But how small? In a system with only two database languages, to add a third language would require eight new translation routines (one in-translation routine and one out-translation routine for each of the old languages to the new language and then two in-translation and two out-translation routines for the new language to the old languages). As with the user learning multiple languages, the data sharing and intercommunications would be severely limited. For example, the computer would have to know in advance what language each piece of data was stored in so that it could issue the appropriate retrieval command (Hsia92a).

A second alternative is to use one common language for storage of the data and provide one set of translators for each language. The disadvantages of this approach are the requirement to initially translate all the current data into the storage data format and finding a data model that is sufficiently robust to support such an undertaking. The advantages of this approach are the initial translation is a one time requirement and the number of translation routines required for any new language is two regardless of the number of other languages already present. For any new language added to the system the operator must design one in-translation routine and one out-translation routine from the new language to the storage or kernel data language. The users are not required to learn any new language constructs or requirements and previously written programs do not have to be modified to allow access to any of the stored data. The required data storage area will be the sum of all these individual computers placed into one computer system (Hsia91).

One database machine, known as the Multi-backend Database System (MDBS), is at the Naval Postgraduate School's Laboratory for Database Systems Research. This system which started development in the early 1980's (Hsia91), is fundamentally based upon the Database Computer (DBC) as described in (Bane79). This system not only addresses a solution to the problem of a multilingual and multimodel database systems, but is also capable of processing large amounts of data efficiently through parallel processing

(Hall89). The MDBS system has shown that it is possible for these data languages to coexist on a single system, interact compatibly, and even share data without data duplication. The user can use whatever database language and format that is most convenient or fits the situation. The database language can range from early models such as hierarchical to modern languages such as object oriented data models. Efforts have been undertaken to show that each of the major groups of database models can be implemented. One rapidly growing database related field, not yet implemented on the MDBS, is Artificial Intelligence or AI. AI programs often consist of many simple observations and facts linked by a set of rules and stored as a large database of information. Common forms of AI language constructs are analogous to the functional data model. In the functional data model, the data are entered in a mathematical form similar to:

$$f(x) = y$$

where each of the variables, f, x and y, are not limited to solely numeric values and functions. The name functional is misleading and constructs are more aptly associated with mathematical relations than functions. In a function the value of x can be mapped to one and only one value y. With the functional data model this is not the case. This can be illustrated as follows. Suppose a user entered into the database the fact

$$son(Adam) = Cain$$

which is read as "the son of Adam is Cain". It is also possible and correct to enter:

$$son(Adam) = Abel$$

or "the son of Adam is Abel". If a user were to query the system as to who is the son of Adam, the response should be the set {Cain, Abel}. As a mathematical *relation* this is acceptable, but as a mathematical *function* it is not. Unfortunately the name relational data

model already exists and so the name functional data model is used for these type of expressions.

The concept of an AI system is to gather information through experience and base future queries and decisions based on previous inputs, very similar to the way most humans make decisions. As with the human intelligence, AI requires the storage of huge amounts of base data as well as the rules to follow in making intelligent choices. Applications of AI are wide and varied. It is used in Abstract Object Recognition, Fuzzy Logic, Intelligent Decision Making, Computer Aided Instruction, Expert Systems and many other areas. For demonstration purposes, this research designed and implemented an Expert System model. An Expert System is a program designed to allow the computer to act as an expert in a specific area. Data is initially entered based on human expert observation and analysis. Over time, through user inputs and feedback, the computer will be able to provide insight and suggestions based on previous conditions and results. To make effective use of this data, they must be stored and retrieved accurately, quickly and efficiently. This type of storage and retrieval requirements are the purview of a database management system. The system that best fills this void is the Naval Postgraduate School's MDBS.

C. THE OBJECTIVE

The objective of this thesis is to show that a functional data model interface can be made compatible with the MDBS and the other data models implemented on it. We selected a DAPLEX type data language based on the description by (Ship81) to implement the functional data model in the MDBS (Lim86). Using a DAPLEX data language structure, we will show that it is possible to construct an AI interface which does coexist and interact with other data models on a single system. To demonstrate that it is compatible with the MDBS kernel data language we selected some representative primitive language constructs from which other more complex language constructs may be implemented. To this end we have created a small model of an expert system which exercises the necessary language functionality.

II. THE MDBS SYSTEM

A. THE BACKGROUND AND DESIGN

Conceptually the requirements of the MDBS are two fold. The first essential requirement is to create an environment in which multiple data languages can compatibly coexist, and interact within a single computer operating system. Access to many different database languages on a single computer is not uncommon. It is similar to finding different word processing software on a single computer. What is unique in the MDBS is that the different database languages are stored and maintained utilizing a single, common data format (see Figure 1). The obvious advantage of such a system, is the reduction of duplicated data throughout the various databases. This was previously necessitated by the requirement to store the data, both base and meta data, in a format understandable by the individual database languages. If the data are stored in the same format for all the languages it should be relatively easy, regardless of the interface language used, to be able to retrieve and present the data in manner consistent with the interface language. For example, within the MDBS it is possible to create and input database information utilizing the hierarchical language constructs. The behavior of the database and the transaction commands are all consistent with the DL/I language of a hierarchical database. Another user can write transactions, update the same hierarchical database and output any queries in a relational format using SQL constructs. The second user merely needs to inform the MDBS at the start of the session that the relational format is going to be used. As far as the first user is concerned the database is a hierarchical data model while the second user is convinced that the database is a relational data model. Programs do not have to be modified and users do not have to be retrained.

The second central requirement of the MDBS is to be able to operate using very large databases without significant performance degradation. If we are to integrate multiple heterogeneous databases as proposed the amount of base data can become voluminous. This feat is accomplished utilizing multiple off the shelf computer systems working in

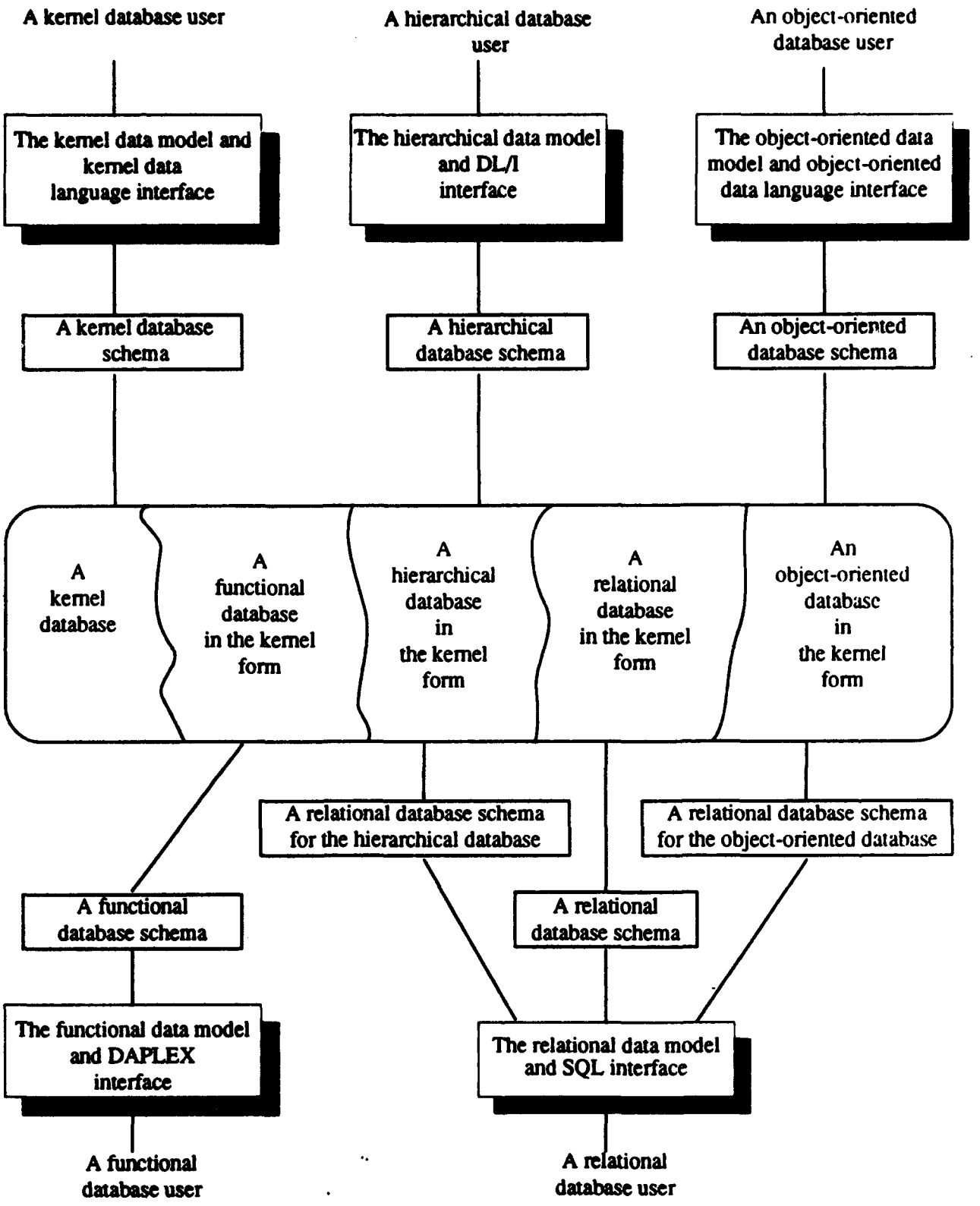


Figure 1 The Multimodel, Multilingual and Cross-Model Capability

parallel (see Figure 2). In the MDBS these are known as backend systems. The use of

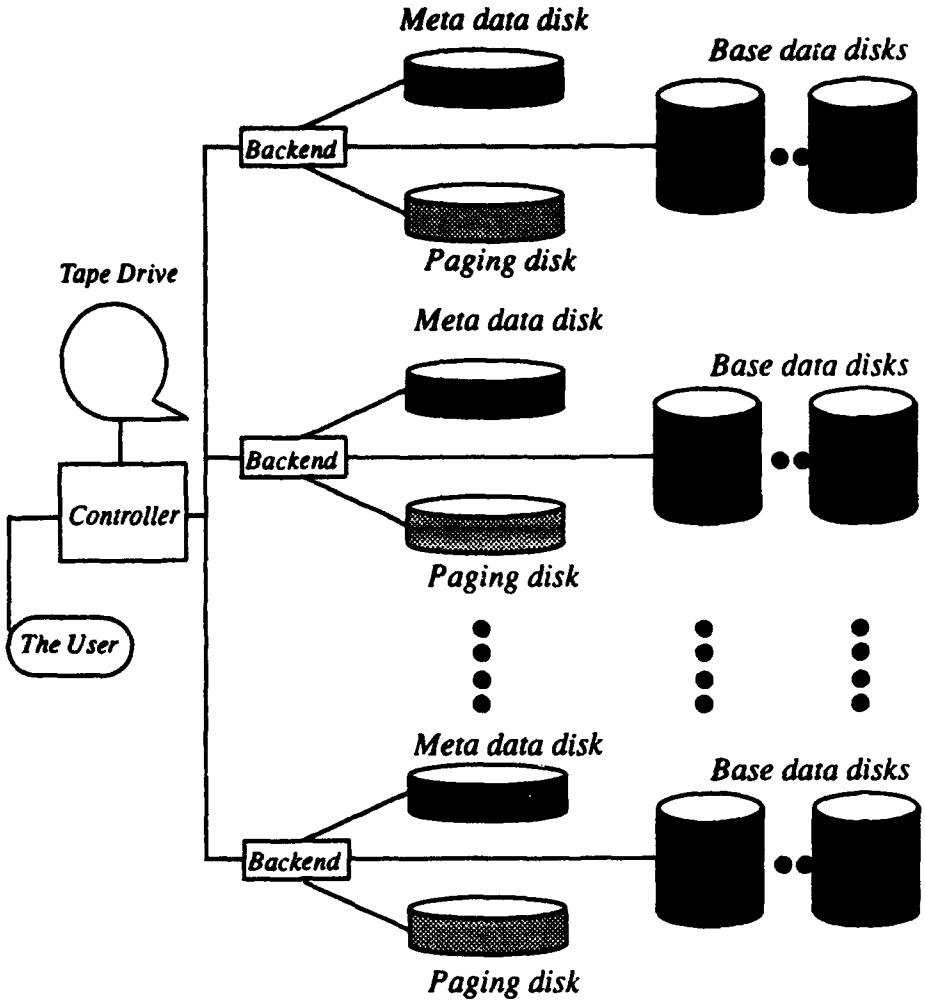


Figure 2 The Multibackend Database Supercomputer (MDBS)

multiple backend systems accommodates the expansion of the databases by providing a large physical repository in which to store the data (Meek93). Using off the shelf technology ensures that the MDBS does not need a specialized computer system, while keeping the cost down and the availability high. The parallel aspect of the backends provides improved response and speed in data retrieval by having each system working simultaneously and independently to complete the transaction (Hall89). Consequently, the

base data can be distributed among the various backends, thereby minimizing the search space of each individual backend system. As a transaction is generated, the query is distributed to each of the backend systems. The query is then executed simultaneously on each backend system and the information is passed via the network to the frontend controller. One small draw back to the multi-backend system is the need to duplicate the meta-data. However, when compared to the storage requirements for base data, the stora requirements for the meta-data is an order of magnitude smaller in a typical databa application. This data are retrieved, processed and stored in a variety of formats.

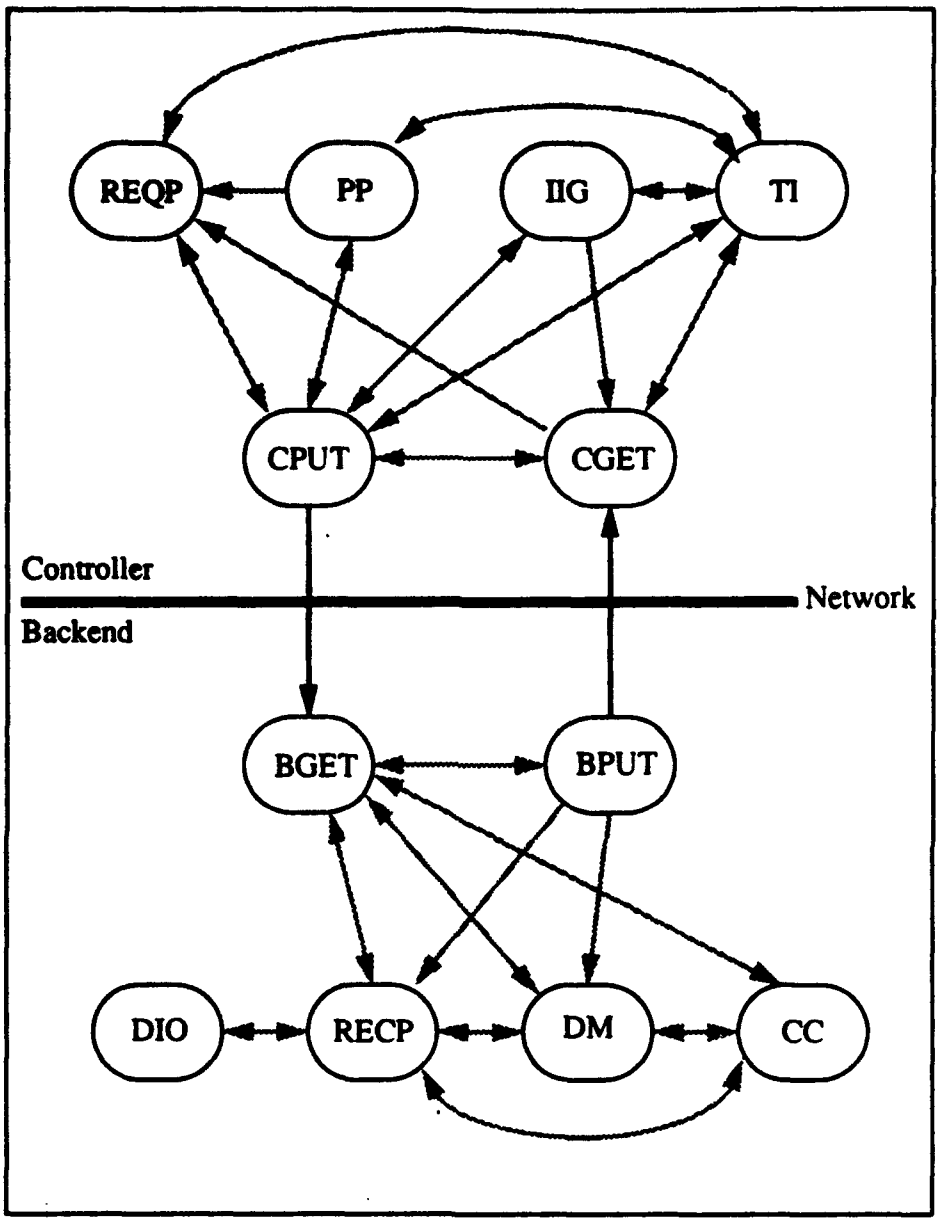
Regardless of the system used, the amount of base data in the database system will continue to grow. As the storage requirements near capacity, if no action is taken, the user will note a degradation in response time and possibly the loss of critical data. The traditional response for normal database management systems would be the purchase of a new larger and, hopefully, faster computer system. The cost of purchasing such systems and the resulting disruption while switching (not considering that it may require a new database system) is substantial. Additionally, although it may be possible to purchase a larger system it may not be possible to acquire a faster computer system. The results will be a negligible increase in response time even though storage capacity is increased. For the MDBS it is a relatively simple matter to improve response and/or capacity. The user needs only to backup the current database (a normal undertaking in modern computer systems), add another readily available backend system and then restore the database from the backup. The system will automatically redistribute the base data during the restoration. The results will be a significant increase in the overall response time and a quantum increase in the storage capacity of the MDBS. The cost and disruption in the MDBS is minimal. Previous studies have shown that doubling the number of backends can nearly double both the speed and capacity of the MDBS (Hall89).

Another advantage of the MDBS multi-system set up is in communications. Again, the MDBS makes use of no specialized equipment and is therefore more readily compatible with changing technology. The frontend controller communicates with the backends via a

standard LAN setup. The LAN is supported by standard ethernet equipment employing TCP/IP protocol with UDP for improved communications reliability. This is discussed in greater detail in other documents (Watk93). The extensive use of off-the-shelf technology makes the MDBS financially attractive when taking into account availability, supportability, reliability and maintainability. These four areas traditionally account for the lion's share of the cost of owning and operating any database system.

B. INTERPROCESS COMMUNICATIONS

Communication within the MDBS are preformed via the interaction of twelve separate processes (see Figure 3). Six processes are used for the controller or frontend and six for the database storage area or backend. The process names may not be indicative of the functionality of the procedures contained. The names have other historical significance and are retained in the interest of maintaining compatibility. The six controller processes are controller get (CGET), controller put (CPUT), test interface (TI), request processing (REQP), insert-information generation (IIG), and post processing (PP). The CPUT process is responsible for the sending of messages across the network to the backend system processes. The CGET process is responsible for the receipt of messages from the backend. The TI process handles all the user interface processing. The TI which contains the translation algorithms for each of the language interfaces, is responsible for interfacing with the user and selecting the appropriate interface displays based on the users request. All translations to and from the kernel language are performed within the TI. The REQP process parses the kernel translated transactions ensuring the query is in a kernel acceptable format and syntax before it is sent to the backend. The IIG process assists with the setup and distribution of the base data for the backends. The distribution is an integral function when dealing with large databases to ensure improved retrieval performance from the parallel processing completed by the backend systems. The PP process properly ensures a consistent format for the data received from the backend. This data must again be provided to and processed by the TI to ensure the output is correct and in the user selected format.



REQP - Request Processing
PP - Post Processing
IIG - Insert Information Generation
TI - Test Interface
CPUT - Controller Put
CGET - Controller Get
BGET - Backend Get
BPUT - Backend Put
DIO - Disk Input/Output
RECP - Record Processing
DM - Directory Management
CC - Concurrency Control

Figure 3 MDBS Communication Channels

The six backend processes are backend get (BGET), backend put (BPUT), record processing (RECP), concurrency control (CC), directory management (DM), and disk input/output (DIO). All six processes run independently on each backend machine participating in MDBS. The BPUT and the BGET perform the same functions for the backends as the CPUT and CGET, respectively, performed for the frontend. The RECP process is responsible for the processing of the data records. The CC process ensures data (record) integrity is maintained during the processing of transactions. The data in this case includes both the meta-data and the base data. To accomplish this for the base data the CC must communicate with the RECP. The DM process and the DIO process are responsible for the actual reading and writing of the data to the backend storage units. The DM handles the meta-data and works with the meta-data disks only while the DIO performs similar functions working only with the much larger base data and associated drives. The DIO communicates only with the RECP.

The majority of our research dealt with the TI process. To add an additional module to the MDBS, it is essential to understand the overall inter-process communications to ensure the proper functionality of the data model. Each process has its own unique interface requirements which, if not met, may result in failure of the whole system.

III. THE MODEL LANGUAGE INTERFACE

As stated earlier, in the MDBS the CNTRL section contains all the processes which are necessary for the frontend system to establish an interface between the user and the data which are stored in backend computers. The process with the CNTRL section which contains the specific user interface commands is the Test Interface or TI. The TI is actually a collection of multiple processes and programs compiled as libraries and then joined together into a single entity. A portion of this collection of libraries are the specific interpreters and translation routines for each of multiple languages compatible with MDBS. The source code for each translator is located in the LANGIF/SRC subdirectory of the TI directory.

A. STRUCTURE OF THE MODEL LANGUAGE INTERFACE

There are six subareas within SRC: The COM subdirectory which maintains some common routines shared by all the interface languages, the SQL subdirectory for the relational database model, the DML subdirectory for the network database model, the DLI subdirectory for the hierarchical database model, the OBJ subdirectory for the object oriented database model and the DAP subdirectory for functional database model. The names are abbreviations for the data manipulation languages used for the individual database models. Each subdirectory of a specific database model is further divided into an allocation or ALLOC area, the language interface layer or LIL area, the kernel mapping system or KMS area, the kernel controller or KC area and the kernel formatting system or KFS area [Figure 4]. The purpose behind the division into the subareas of ALLOC, LIL, KMS, KC and KFS is to facilitate software maintenance by dividing the programs into functional areas (Meek93).

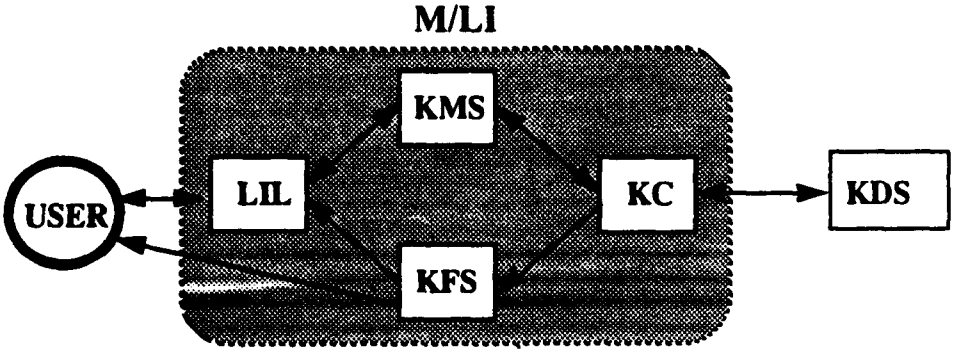
B. THE LANGUAGE INTERFACE LAYER SECTION

The ALLOC area contains the code for the allocation of memory for model specific pointer types. Calls to the ALLOC area routinely are made from all the other areas within

the module. Inter-model common pointers are contained in the COM directory. The LIL area contains the code necessary for direct user interfaces and presentations with the exception of the responses to database queries. These interfaces include asking the user for the module to be used (i.e. attribute based, relational based, functional based, etc.), requests for file name and obtaining queries from the user. What the user sees on the screen is provided by the LIL, except for the actual database transaction response displays.

C. THE KERNEL MAPPING SYSTEM SECTION

The KMS is the heart of the database module (Kloe85). It is responsible for ensuring the information passed from the LIL is semantically and syntactically correct, parsing of the information, ensuring the data is consistent with previous inputs (e.g. the user cannot declare a function for en entity that has not yet been declared), translation of the information from the user input language to the kernel based language and passing the translated information to the KC. Except during retrievals, the KMS receives replies from



LIL : Language Interface Layer
KMS : Kernel Mapping System
KFS : Kernel Formatting System
KC : Kernel Controller
M/LI : Model/Language Interface
KDS : Kernel Database System

Figure 4 The MDBS Model Interface Layer

the KC and passes the replies back to the LIL.

D. THE KERNEL CONTROLLER SECTION

The KC handles all interfaces to the backend system. It sends the kernel formatted queries to the backend Kernel Data System (KDS) and ensures the responses from the KDS have been processed without problem. If an error is detected in the backend the KDS passes that information to the KC which will conduct the appropriate exception handling procedures. During retrievals the KC places the KDS responses into the proper structures and passes the data and control to the KFS.

E. THE KERNEL FORMATTING SECTION

The KFS is responsible for the receipt and acknowledgment of retrieved responses from the backend via the KC. The data received are in a stream format. The KFS examines the stream, determines whether the information is an attribute name or value, converts the data into an acceptable format for the user selected database model and then displays it to the user. Control is passed back to the LIL and the cycle is complete.

We have shown there are many similarities between the various database models. With this commonality there are numerous structures and variables that are shared by all the database models. These structures and variables are contained in the LANGIF subdirectory in a file called LICOMMDATA.H (for Language Interface COMMon DATA). The extensive use of union structures make it possible for each of the database models to access other models base data. Two important types used by each module are the current database info or CURR_DB_INFO and the language interface info or LI_INFO. Both are unions defined in the LICOMMDATA.H.

F. COMMON USE STRUCTURES

CURR_DB_INFO is a structure consisting of the name of the user (CDI_DBNAME), a database node for the specific module (CDI_DB), a group node (CDI_GRP) which indicates how the attributes for a particular module are grouped (e.g. relations for the relational model), an attribute node (CDI_ATTR) which indicates the name and type of each attribute and an integer (CDI_DBTYPE) to indicate which model type is being

accessed. The CDI_DB is of type dbid_node, a union of database nodes, which provides a

```
union dbid_node {
        struct rel_dbid_node *dn_rel;
        struct hie_dbid_node *dn_hie;
        struct net_dbid_node *dn_net;
        struct dap_dbid_node *dn_dap;
        struct obj_dbid_node *dn_obj;}
```

Figure 5 Structure of MDBS dbid_node

pointer for a specific module database node depending on the value of CDI_DBTYPE. For the functional data model, dn_dap is used and points to functional data model specific information [Figure 5]. Regardless of the model, each database node contains at a minimum the name of the database, a pointer to the first group of attributes and a pointer to the next database node. For this project this database node is the dap_dbid_node [Figure 6]. The specifics for the functional database node will be discussed later.

```
struct dap_dbid_node {
        char                *dap_db_name;
        int                 number_of_entitys;
        int                 number_of_aliases;
        int                 object_counter;
        dap_db_entity_node  *first_entity;
        dap_db_alias_node   *first_alias;
        dap_dbid_node       *next_db;}
```

Figure 6 Structure of dap_dbid_node

The LI_INFO provides a common interface construction to facilitate the passing of data and parameters throughout the procedures of each module by actually passing only the one object. This structure contains a pointer to its appropriate CURR_DB_INFO, pointers to kernel language translation files, transaction files, response files, etc. As this is also a union structure, each module has its own specific name and parts, but in general are of the same construction. For the Functional database model, this corresponds to LI_DAP and is of type DAP_INFO. The construct of DAP_INFO, the names of the fields and the intended

usage is indicated below [Figure 7]. The structure CURR_DB_INFO was previously

Data Type	DAP INFO	Data Use
curr_db_info	dpi_curr_db	Current DB & structure in use
file_info	dpi_file	Transaction File Pointer/Info
tran_info	dpi_dml_tran	Individual transaction
tran_info	dpi_abdl_tran	Kernel language transaction
ddl_info	dpi_ddl_files/	Kernel DDL files (.t and .d)
int	dap_operation	Current operation type
int	dap_answer	Current kernel response
int	dap_error	Error type indicator
int	dap_buff_count	Transaction buffer count
kms_info	dpi_kms_data	KMS special data
kms_info	dpi_kfs_data	KFS special data
temp_str_info	dpi_kc_data/	Linked list for KC
int	dpi_subreq_stat	Status of nested requests
temp_str_info	dap_query/	Linked list for KFS

/Indicates the item is a pointer type

Figure 7 Structure of dap_info

discussed. The FILE_INFO type consists of a string to contain the file name and a pointer to that file. The rest of the structured types are linked list structures. Each of the non-primitive types (i.e. not int, char, etc.) within any of the structures is further defined within LICOMMDATA.H.

IV. THE LANGUAGE INTERFACE LAYER (LIL)

Once the user opts to use the functional data model by selecting *f* from the initial Module Selection menu, Figure 8, program control is given to the Language Interface

```
Select an operation:

    (a) - Execute the attribute-based/ABDL interface
    (r) - Execute the relational/SQL interface
    (h) - Execute the hierarchical/DL/I interface
    (n) - Execute the network/CODASYL interface
    (f) - Execute the functional/DAPLEX interface
    (o) - Execute the Object-Oriented interface
    (x) - Exit to the operating system

Select->f
```

Figure 8 Module Selection Menu

Layer (LIL). The principle function of the LIL is to act as the interface between the user, the operating system and the MDBS. The LIL presents the interface menus, and accepts the user inputs (normally single character selections) as the user navigates through the functional data model.

The first functional data model menu encountered is the Database Selection menu,

```
Enter type of operation desired
    (l) - load new database
    (p) - process existing database
    (x) - return to the MLDS/MDBS system menu
```

Figure 9 Database Selection Menu

Figure 9. The user can define a new database name (selection *l*), access a previously named database (selection *p*) or return to the Module Selection menu (selection *x*). If *l* is selected the user is prompted for a unique name for the database. This name is used extensively throughout the frontend and backend as a means of identifying database specific files. The name of the database is case sensitive.

Once a unique name is provided, the next menu permits the user to designate a file or the terminal as the input source for defining the schemata for the database. If a file is to provide the definition, the LIL assumes the named file is in the UserFiles directory. If the user decides to input the schemata from the terminal, a blank screen is provided and the entered data are placed in a temporary file which is then read the same as if the file source was selected. The format for the schema creation statements can be one of the following:

1. DECLARE *entity* ENTITY
2. DEFINE *function(entity) = primitive data type*
3. DEFINE *function(entity1) = entity2*
4. DEFINE *function 1(entity1)* = INVERSE OF *function2(entity2)*
5. DEFINE *function 1(entity1) = function2(entity2)*

The reading and parsing of these transactions will be discussed in detail in a later chapter on the Kernel Mapping System (KMS). Once the new database schema has been accepted the LIL redisplays the Database Selection menu. The user would next select the *p* option.

Once the metadata is defined, written to the metafiles and copied to the backends, by the KMS, the user is presented with the Functional Model LIL Interface menu shown in

Enter your choice
(d) - display schema
(m) - mass load from a data file
(s) - send data to a file for mass load
(f) - read in a group of queries from a file
(t) - read in queries from the terminal
(x) - return to previous menu

Figure 10 Functional Model LIL Interface Menu

Figure 10. Selection *d* will cause the LIL to display a textural representation of the metadata, entered in the previous step, on the screen. A sample, based on the data used in Appendix A, is shown in Figure 11. Selection *m* (mass load) allows the user to enter a large amount of data quickly and easily. The user will be prompted for the name of the file which contains the mass load data. The system expects the mass load file to be located in the UserFiles directory. The format of the DAPLEX mass load file is slightly different from the other MDBS modules. In our module the data are laid out in a format that more closely resembles the expected format of the ABDL used for data storage in the backends. As will

Database Name : MAINT

Entity: MALFUNCTION
ROOT(MALFUNCTION) = EQUIPMENT
INDICATION(MALFUNCTION) = INOPERATIVE
DISCREPANCY(MALFUNCTION) = STRING

Entity: INOPERATIVE
DIAGNOSIS(INOPERATIVE) = MALFUNCTION
WRA(INOPERATIVE) = EQUIPMENT
PROBLEM(INOPERATIVE) = STRING

Entity: EQUIPMENT
NUMBER(EQUIPMENT) = STRING
NAME(EQUIPMENT) = STRING

Figure 11 Sample DAPLEX Schema Display

be discussed later in this document, many of the functions maintain their relations via user transparent intermediate relations, referred to as pointer entities. To ensure the functions are correctly maintained, these pointer entities must be included in the mass load. Other MDBS interface modules do not depend on this mechanism. Also, the mass load for other modules assumes the order of the attribute values exactly matches the order the attribute names were entered in the schema. The DAPLEX mass load makes no such assumptions and each line of data in the file can be entered in any order. We feel this is a much more flexible entry format and can be readily applied to fit any of the other MDBS interface modules. Lastly, the DAPLEX mass load format meshes with an option, which is not available in any other interface module. This new option is the selection *s* which sends all the current base data from this database to a user designated file in the UserFiles directory. This is especially useful in the current MDBS configuration since the data are kept in non permanent backend storage and changes are made which the user wishes to maintain after the MDBS is shutdown. The mass dump also provides a quick method for creating a mass load file, without worrying about the correct format.

The next two options, *f* and *t*, permit the user to enter a set of DAPLEX queries. If the *f* option is selected the user is prompted for the name of a previously created file of queries. The file is expected to be located in the UserFiles directory. If the *t* option is selected the

user is presented with a blank screen to enter the desired queries. These queries are saved to a temporary file and then processed in the same manner as if the *f* option had been selected. Once the queries have been read, the LIL displays the queries with numeric indexes. The user is then presented with the Query Selection menu. The last selection in the LIL Interface menu is *x* and it's selection will return the user to the Database Selection menu.

The last menu in the functional model LIL is the query selection menu, Figure 12.

```
Pick the number or letter of the action desired
    (num) - execute one of the preceding queries
    (d)   - redisplay the list of queries
    (x)   - return to the previous menu
```

Figure 12 Query Selection Menu

Entering one of the numbers corresponding to the queries listed prior to this menu, will cause that query to be processed by the KMS. No validity checks of the queries either syntactically or semantically are done until processing begins by the KMS. In the functional module, the LIL redisplays the query in case the original display has scrolled off the screen. Entering a *d* will redisplay the queries with their numeric indices. The Query Selection menu is redisplayed following either of the two previous menu selections. The last selection *x* will return the user to the Functional Model LIL Interface menu.

V. THE KERNEL MAPPING SYSTEM (KMS)

Contained within the Kernel Mapping System (KMS) is the DAPLEX Parser Module. The DAPLEX Parser Module converts the user's DAPLEX Query language constructs into equivalent Attribute-Based Data Language (ABDL) statements. The new ABDL constructs are made available to other modules by appending them into the structure dap_info. The control and dap_info are then passed to the backend data manager via the Kernel Controller (KC).

The DAPLEX statements can be subdivided into two types: Data Definition Language (DDL) statements and the Data Manipulation Language (DML) statements. The DDL statements define the functional database schema and instructs the backend how the data will be stored. The DML permits the user to conduct other basic database functions such as the insertion and retrieval of the base data.

A. THE DATA DEFINITION LANGUAGE PARSER (DDLP)

The Data Definition Language Parser (DDLP) reads DAPLEX DDL constructs from a file or as queries entered through the terminal. From the DDL constructs the parser generates dap_info pointer information and creates two metadata files that are used by the MDBS to establish and maintain a database. As the DDLP reads and translates the DAPLEX schema constructs, it stores the information about the database (entities, entity types, attributes and attribute types, etc.) in the dap_dbid_node (see Figure 13). This node contains the schema information necessary to create the database template and descriptor files. It also used during the processing of DDL and DML statements to ensure their semantic and syntactic correctness.

The pointer dap_id_name points to the user entered name of the database. This name is used by the MDBS in the confirmation of numerous files and pointers required for the database created by DDLP as well as database verification for the backend processes. The next three data structures are integers. The first two, number_of_entitys and

char* dap_db_name
int number_of_entitys
int number_of_aliases
int object_counter
dap_db_entity_node* first_entity
dap_db_alias_node* first_alias
dap_db_id_node* next_db

Figure 13 Structure of dap_dbid_node

number_of_aliases, are used to store the current count of entities and aliases within the schema. The third integer, object_counter, is used to track the number of entity instances created by the user. As a new instance of each entity is created, it is assigned a unique identifier which the program generates. The last used entity identification value is stored in object_counter. Entity identification numbers will be discussed in more detail later. The next three structures are pointers, which as their name suggests point to, respectively, the first entity, the first alias and the next DAPLEX database for the current DAPLEX database.

The Data Definition Parser searches for one of two key words, DEFINE and DECLARE, as precursors to possible DAPLEX DDL statements. "DECLARE" precedes a DAPLEX entity creation statement while "DEFINE" is used prior to function, alias and inverse descriptions. The DAPLEX DDL statements must be in one of the following five forms:

1. DECLARE *entity* ENTITY
2. DEFINE *function(entity) = primitive data type*
3. DEFINE *function(entity1) = entity2*
4. DEFINE *function 1(entity1)* = INVERSE OF *function2(entity2)*
5. DEFINE *function 1(entity1) = function2(entity2)*

1. DECLARE *entity* ENTITY

The simplest form of DDL statement is DECLARE *entity* ENTITY. When this statement form is encountered, the name of the new entity is compared against the entity names already entered by previous DECLARE statements. If the name already exists the construct is rejected, an appropriate error message is displayed and the processing stops. Our DAPLEX data model and the MDBS do not allow for duplicate entity names. If the new entity name is unique a dap_db_entity_node is created and appended to the linked list of entities indicated by the pointer, first_entity within the dap_dbid_node. The new entity's

char* dap_entity_name
char* dap_entity_addr
char dap_entity_type
int number_of_attribs
dap_db_attrib_node* first_attrib
dap_db_entity_node* next_entity

Figure 14 Structure of dap_db_entity_node

data structure can be seen in Figure 14.

The entity name, in capital letters, is copied into a character string pointed to by dap_entity_name. The dap_entity_addr contains the actual name, in ABDL format, that will stored as part of the database schema. In its current configuration, the MDBS only allows strings of sixteen characters or less. Therefore, if the length of any string is more than sixteen characters the DAPLEX module will truncate it prior to storage. In the case of the entity name it is stored in a location pointed to by dap_entity_addr. The dap_entity_addr is in the standard ABDL attribute value format with the first letter capitalized and all other letters in lower case. This is also true of the name of the templates as stored in the backend. In reality, they are values of the attribute name TEMPLATE. Later, as functions (attributes) are added to the entity, the number_of_attrib will be incremented. The structure first_attrib is a pointer to the first of a linked list of attribute nodes.

Each time a new entity is declared, an ***identification attribute*** is automatically added. The attribute name assigned to the identification attribute is created using the first four letters of the entity name concatenated with a "QQ". This attribute is essential when mapping entities to other entities and ensures the uniqueness of each instance. The final element of the entity node, next_entity, is the pointer to the next entity within the current database. Once the data fields for the dap_db_entity_node are filled, it is added to the linked list in dap_dbid_node and the database node's number_of_entitys variable is incremented. Once an entity has been created other functions (attributes) can be assigned to the entity.

2. **DEFINE *function(entity) = primitive data type***

The statement DEFINE *function(entity) = primitives data type* creates a simple function (attribute) for the entity name found within the parentheses. An entity with several simple functions can be construed as a relation with an attribute for each defined function. Within this paper we will use the two terms interchangeably.

This DEFINE statement is parsed leaving three tokens, *function*, *entity* and *primitive data type*. The DDLP first checks the dap_dbid_node's linked list of entities (first_entity) until the *entity* name is found or the end of the list is encountered. An entity must already be declared before any functions can be assigned to it. If no such entity is located, an appropriate error message is displayed and the parsing/schema-creation process stops. If the entity node is found, the parser compares the *function* name to other existing attribute nodes assigned to that entity. If the entity does not have an attribute node with this name, a new attribute node will be created and appended to the list. The attribute node data structure can be seen in Figure 15.

char* dap_attrib_name
char* dap_ attrib_addr
char dap_attrib_type
dap_db_attrib_node* next_attrib

Figure 15 Attribute Node Structure

The function name found in the DEFINE construct is stored in a location pointed to by dap_attrib_name. The dap_attrib_addr points to the storage area which contains the first sixteen characters of the function name. This address represents the actual value used in the metadata. The primitive data type which follows the equal sign must be STRING, CHARACTER, INTEGER or FLOAT. The first letter of these data types will be stored in dap_attrib_type. The characters "E","A" or "G" corresponding to entity, alias and composite function, respectively, can also be stored in the dap_attrib_type. These will be discussed in later forms of the DEFINE statement. A short example may clarify some points of confusion.

(1) DECLARE equipment ENTITY
(2) DEFINE name(equipment) = STRING
(3) DEFINE the_purchase_year(Equipment) = INTEGER

Figure 16 Representative DDL Constructs

The last DEFINE construct, Figure 16(3), can be interpreted as: "When the function THE_PURCHASE_YEAR is applied to the entity EQUIPMENT it will yield an integer number." The entity name is stored as EQUIPMENT, while the entity address is stored as Equipment. The string *THE_PURCHASE_YEAR* is stored as the attribute name and *THE_PURCHASE_YEA* as the attribute address. The DAPLEX data model allows for function overloading or polymorphic functions. As long the input parameter, in this case the entity, is different, the functions can utilize the same name.

3. **DEFINE *function(entity1) = entity2***

The statement DEFINE *function(entity1) = entity2* is a more complex definition of a function. In this case, the function maps one entity to another. Once DDLP verifies that *entity1* and *entity2* have been defined it creates a new attribute node with an attribute name *function*. The new attribute node's dap_attribute_type is assigned the character "E", which identifies it as an entity type attribute.

Unfortunately, when using the ABDL of the MDBS, there is no easy method of storing pointers to entities. To accomplish the necessary mapping, we chose to create an intermediate entity, henceforth referred to as a *pointer entity*. Remember that each entity instance has a unique identifier assigned as an attribute, the name of this identifier consisting of the first four letters of the entity name with a "QQ" appended, (e.g. the entity equipment in Figure 16(1) will have an identification attribute called "EQUIQQ".) When a set of base data is entered into the database to create a new instance of the entity, equipment, the object_counter in the dap_dbid_node is used to produce a unique identification number for the new instance. The object_counter is then incremented to ensure each instance has a unique identification number.

When a pointer entity is created its name is formed by the concatenation of the three tokens: *function*, *entity1* and *entity2* separated by an underscore and, if necessary, truncated to a maximum of sixteen characters. The only attributes types in this entity will be identification attributes, its own and those entities it maps. As with all entities, the first attribute is automatically generated at the time of creation of the entity type. The other two attributes identify the entity instances which are to be mapped from one to another. For example the statement, DEFINE indication(malfunction) = inoperative, would produce a new entity type named "indication_malfunction_inoperative" with attributes INDIQQ, MALFQQ and INOPQQ. When mapping from the entity "malfunction" to the entity "inoperative", the object number assigned to each instance entered is copied into the identification attributes in the pointer entity (see Figure 17). The instances are now connected.

4. **DEFINE *function1(entity1)* = INVERSE OF *function2(entity 2)***

With the DEFINE *function1(entity1)* = INVERSE OF *function2(entity* 2) statement the DDLP must ensure that *entity1*, *entity2* and *function2* have been previously declared. In addition *function2* must already have been defined as an attribute of *entity2*.

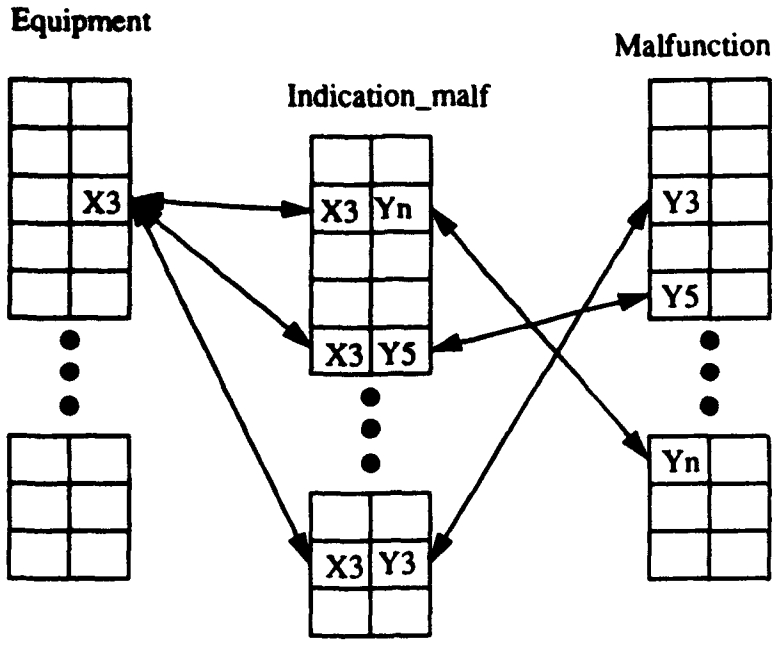


Figure 17 Entity to Entity Mapping

Function1 must be a new attribute of *entity1*. An appropriate error message will be output if any of the above conditions are violated and the processing of the schema will cease.

Another advantage of using the pointer entity type of data structure is that it contains all the information necessary to instantly implement an inverse function. No additional data structures need to be created. However, when creating an inverse function an attribute node and an alias node are created and the new attribute node is assigned to *entity1*. The attrib_type of the attribute node is assigned the character "A" indicating that it corresponds to an alias. The newly created alias node data structure can be seen in Figure

char* dap_alias_new_name
char* dap_alias_old_name
char* dap_alias_ent1
char* dap_alias_ent2
dap_db_alias_node* next_alias

Figure 18 Structure of dap_alias_node

18. *Entity1* and *function1* are stored in dap_alias_ent1 and dap_alias_new_name respectively. Dap_alias_ent2 and dap_alias_old name point to the storage location of copies of *entity2* and *function2*, respectively.

Once designated as an alias the new name can be used in data manipulation statements in place of the actual entity and function name. No actual data or data storage needs to be allocated in the backend for an alias. If data associated with the alias are to be stored in the backend they are stored based on the original entity and function name.

5. DEFINE *function1(entity1) = function 2(entity 2)*

The final DDL construct, DEFINE *function1*(*entity1*) = *function2*(*entity2*), allows for another definition of an alias. This alias is nothing more than another name for the same entity. Once the parser verifies that *entity2* and *function2* exist, the alias function name, *function1*, is stored in a location pointed to by dap_alias_new_name. The entity node, *entity1* is identified by dap_alais_ent1. While the predefined *entity2* and function name are in locations pointed to by dap_alias_ent2 and dap_alias_old_name respectively.

Once all the DDL constructs are read and processed, the DAPLEX Data module must create the two metadata files. The first metadata file is the database template file. This file contains information about the number of templates and their names, as well as the number of attributes and their names and types [see Figure 19]. The system name for this file is constructed by appending a ".t" to the end of the database name (e.g. MAINT.t). The location of this file is expected to be in u/mdbs/UserFiles.

The second metadata file is the database descriptor file. This file contains the name of the database and all its templates [see Figure 20]. There is also instructional information on how the backend is to divide the data into equivalent classes. In Figure 20, the letter b following TEMP indicates the values listed below it are distinct, unique values. This means the values that TEMP contains, at the time of the insert, will determine where the data are stored. The system name for the file is determined combining the database name with an extension of ".d" (e.g. MAINT.d). The Database Template file and the

```
MAINT
6
3
Malfunction
TEMP s
DISCREPANCY s
MALFQQ i
4
Wra_inoperative
TEMP s
EQUIQQ i
INOPQQ i
WRA_QQ i

 • • •
4
Equipment
TEMP s
NUMBER s
NAME s
EQUIQQ i
```

Figure 19 Sample Database Template File MAINT.t

```
MAINT
TEMP b s
! Root_malfunctio
! Indication_malf
! Malfunction
! Wra_inoperative
! Inoperative
! Equipment
@
$
```

Figure 20 Sample Database Descriptor File

Database Descriptor file represent all the metadata required by the MDBS. Once these files have been created and copied to the backend metadata disks the base data can be entered, manipulated and retrieved.

B. THE DATA MANIPULATION LANGUAGE PARSER (DMLP)

Data Manipulation Language statements are those statements used to enter, retrieve and delete the base data. DML statements obviously can be processed only after the schema has been defined via the DDL statements and the metadata has been copied to the backend disks. The DAPLEX DML statements implemented for our model are:

1. The "FOR" statement that allows base data to be entered,.
2. The "RETRIEVE" statement for recalling and displaying the base data
3. The "DELETE" statement which removes the previously entered base data.

The complexity of general DML statements make it infeasible to give a generic example for each of the above type statements.

The DMLP parses and translates all DML statements utilizing the same algorithm. The DAPLEX query begins with one of the three keywords, listed above, followed by an entity name and /or function name. The rest of the query consists of statements and clauses that qualify and quantify the data to be entered for the specified entity. The DMLP verifies syntactically and translates the request into an equivalent ABDL statement. A single DAPLEX statement may result in numerous ABDL statements being promulgated. The translated ABDL statements are loaded into the structure dap_info data, the appropriate operation indicators are set and then control is passed to the KC. The KC is responsible for the execution of the generated ABDL statements. The DML statements are discussed in more detail in the following sections.

1. FOR

The "FOR" statement allows the user of the DAPLEX model to enter and update the base data for the previously defined schema. In other words, these statements are used to set the values a function returns when it is applied to an entity. The "FOR" instruction has two general formats.

a. FOR with NEW Modifier

The FOR statement with a NEW modifier requires the DAPLEX module to create a new instance of the entity. The user must provide the base data for each of the entity's functions in the other statements following. A generic form of the FOR statement with the a NEW modifier is demonstrated in Figure 21.

```
FOR NEW entity name
    BEGIN
LET function1(entity) = literal1
LET function2(entity) = literal2
...
LET function k(entity) = entity 2
    SUCH THAT function m (entity 2) = literal m
...
LET function n(entity) = literal n
    END
```

Figure 21 DAPLEX FOR Statement with NEW Modifier

Anytime the DMLP encounters the key word FOR, it begins to accumulate the information necessary to create an ABDL INSERT statement. The entity name in the first line is the template attribute value of where the base data will be stored in the backend. The entity name found in the LET statements must be the same as the first occurrence (exceptions occur if it encounters a SUCH THAT clause). In our model and in the MDBS, multiple entities with a single INSERT statement are not allowed. FOR statements may result in multiple ABDL INSERT statements, but they must be passed individually to the backend. The key word, BEGIN, is used to mark the start of a block of LET and SUCH THAT statements which contain other amplifying information for the entity.

As the DMLP parses each individual LET clause it verifies that the *function* has been declared and is associated with the appropriate entity. Next the parser checks to see that the base data following the equal sign is of the correct data type, previously defined for the *function*. For example, given a DAPLEX FOR query, as written in Figure 22, the primitive data type of the function problem_number in line (4) must be of type "I" (i.e. integer). If the function problem_number had been defined as any other primitive data

```
(1) FOR NEW inoperative
(2) BEGIN
(3) LET problem(inoperative) = "UHF_No_Receive"
(4) LET problem_number(inoperative) = 27
(5) LET wra(inoperative) = equipment
(6) SUCH THAT number(equipment) = "ARC-51"
(7) END
```

Figure 22 DAPLEX FOR Query Example

type, an appropriate error message will be output. Once verification is complete, (i.e. the function name and base data type are legitimate and matching), the parser adds the appropriate ABDL addresses, from the corresponding dap_dbid_node entity and attribute nodes, along with the base data values, as specified, to a linked list.

The LET clause followed by SUCH THAT does not add data to the new entity instance. Rather, it instantiates a pointer entity mapping the new instance to the other entity. The LET/SUCH THAT combination indicates an entity to entity relationship (see Figure 17). The DMLP first retrieves the base data then generates the INSERT necessary to establish the mapping between the two entities. Therefore a separate ABDL RETRIEVE and INSERT statement will be generated in addition to the INSERT generated for the new entity instance. Using the example in Figure 22 and assuming the object_counter is set to 52, we could expect the following ABDL statements:

```
[RETRIEVE ((TEMP=Equipment)and(NUMBER=Arc-51))(EQUIQQ) by EQUIQQ]
[INSERT <TEMP, Wra_inoperative><EQUIQQ,10><INOPQQ,52><WRA_QQ,52>]
[INSERT<TEMP,Inoperative><PROBLEM,Uhf_no_rx><PROBLEM_NUMBER,27><INOPQQ,52>]
```

The value of 10, for the EQUIQQ, would have been a result of the initial retrieve statement.

The DMLP obtains the identification number via calls to a subroutine which executes a modified retrieval, or as it is named in the DAPLEX module a *short_term_retrieve*. Normally, when an ABDL RETRIEVE statement is executed, control is ultimately passed to the Kernel Formatting System which is responsible for outputting the retrieved data onto the terminal screen. The *short_term_retrieve* subroutine, however, executes the statement and places the retrieved identification attributes into a linked list but does not pass control to the KFS.

The parser continues processing the LET clauses until the keyword END is found. The user must include a LET clause for every function associated with the entity, except the user transparent identification attribute. The current object_count found in dap_info will be appropriately assigned to the identification attribute and incremented by the parser. If every function for an entity is not included the MDBS backend will reject the INSERT statement.

All necessary information has been assembled to generate the ABDL INSERT statement. The function name and base data were stored in a linked list as the LET clauses were processed. Combining this data with the entity name and identification attributes, it is straight forward to create the remainder of the ABDL INSERT transactions. This list, now consisting of only INSERT statements, is then placed in the dap_info structure and passed to the KC for further execution.

b. FOR without the NEW Modifier

This form of the DAPLEX FOR statement allows the user to map data from one entity to another without establishing a new entity instance. The main difference between transactions without the NEW modifier as opposed to with the modifier, is that there is no new base data to be entered. All the statements in the block following begin must be LET/SUCH THAT clauses. As with the example from Figure 22, numerous RETRIEVE statements will be initially generated to obtain the necessary entity identification numbers to establish the relationship mapping. The INSERT statements which result, will be only for storing data in the pointer_entities. Another difference is there will be a LET/SUCH THAT statement which includes the initial target entity (e.g. inoperative in Figure 22).

2. RETRIEVE

The DAPLEX and ABDL languages use the same keyword to denote the return of data from the database although the format of the two RETRIEVE statements are quite

different. This DAPLEX statement allows the user to see the results of applying a function to an entity. Three forms of the RETRIEVE statement were implemented into our model.

a. RETRIEVE function1(entity1) and ... and function n (entity1)

This is the simplest RETRIEVE form. When the function is applied to the entity all values are returned. After identifying the keyword RETRIEVE the DMLP processes the first function and entity name. It checks the names against the list of valid entity and attribute names to verify that they have been defined within the schema. If the next token is "and" the DMLP validates the second set of names, and so on. The entity names must be the same or the DAPLEX RETRIEVE statement is rejected and processing of the query ends. The function names are concatenated together separated by a ",". These steps are repeated for each "and" clause until the end of the query.

Once the end of query has been reached the DMLP can assemble the ABDL RETRIEVE statement. It has retained the entity name and the concatenated list of function names. A copy of the first function name in the list is inserted into the required BY clause of the statement (the backend parser will fault without a BY clause). The final ABDL statement will look like:

[RETRIEVE(TEMP=*entity*) (function1, function2, ...function n) BY function1]

This RETRIEVE statement is loaded into the dap_info data structure, the appropriate operation is set and the control of the program is passed to the KC for execution. The data are returned to the KFS which outputs the results to the screen for the user.

b. RETRIEVE with Simple SUCH THAT Clause

This type of RETRIEVE allows the user to tailor the specification to retrieve the data required. The user can input the condition on what data are to be retrieved. The simplest form of the SUCH THAT clause can be see in Figure 23.

The DMLP processes the first line of the query as described in the preceding section. After verifying that all the function and entity names have been properly defined,

1) RETRIEVE function 1(entity1) and function2(entity1).....
2) SUCH THAT function x(entity1) = literal

Figure 23 DAPLEX RETRIEVE with Simple SUCH THAT Clause

it concatenates the function names in the first line, separated by commas, to form the list of attribute names to be retrieved. When a SUCH THAT clause is encountered, the parser will begin to create an ABDL RETRIEVE COMMON statement instead of the standard RETRIEVE. Figure 24 displays the format of an ABDL RETRIEVE COMMON statement.

1) RETRIEVE (TEMP=entity1) (function1, function2,....)
2) COMMON(identification attribute, identification attribute)
3) RETRIEVE(TEMP=entity1) and (function x = literal) (identification attribute)]

Figure 24 ABDL RETRIEVE COMMON Statement

The first line of Figure 23 is translated to create the first line in Figure 24. The statement immediately following the keywords SUCH THAT (i.e. line2 in Figure 23) form the final line of the RETRIEVE COMMON. The second line of Figure 24 specifies which attributes link the two templates together, for this simple case the same template. In almost all of the DAPLEX RETRIEVE COMMON statements it will be the identification attribute which will link the two templates. This more complex ABDL RETRIEVE COMMON statement used for this particular retrieve could have been replaced with a simple RETRIEVE, but it would have required a separate set of procedures to generate the transaction and the results would still be the same. The generated ABDL statement is loaded into the dap_info data structure, the operation is set to reflect RETRIEVE COMMON and control is passed to the KC.

c. RETRIEVE Statement with Compound SUCH THAT Clause.

A DAPLEX RETRIEVE statement with a compound statement following the SUCH THAT clause can be seen in Figure 25. As usual, the DMLP first verifies that all functions and entities have been previously defined in the schema. For this query to

```
RETRIEVE function 1(entity1) and function2(entity1).....
    SUCH THAT function x(entity2) = entity3 AND
    function (entity4) = literal
```

Figure 25 DAPLEX RETRIEVE with Complex SUCH THAT Clause

make any logical sense, entity1 must to be related to some part of the SUCH THAT clause. Therefore one of the following two sets of conditions must be true:

a) entity1 = entity2 and entity3 = entity4
   or
b) entity1 = entity3 and entity2 = entity4

If not the query will be rejected. As with the previous retrieve type, a RETRIEVE COMMON is generated for each SUCH THAT clause. The retrieved attribute type will be an identification attribute, based on the conditional data statement which follows AND. Once the set of identification attributes is retrieved, the DMLP creates a disjunctive ABDL RETRIEVE utilizing those identification attributes. We will use Figure 26 for an example.

```
Retrieve problem(inoperative)
SUCH THAT
indication(malfunction) = inoperative AND
discrepancy(malfunction) = "Generator_Inop"
```

Figure 26 Complex RETRIEVE Example

In this example, the final goal is to retrieve a set of strings corresponding to the attribute *problem*. To start the DMLP will generate a RETRIEVE COMMON statement which will obtain the INOPQQ from the template Indication_malf. The actual statement would be similar to:

```
[RETRIEVE ((TEMP=Malfunction) and (DISCREPANCY = Generator_inop)) (MALFQQ)
COMMON (MALFQQ,MALFQQ)
RETRIEVE (TEMP=Indication_malf)(INOPQQ)]
```

Within our sample database (see Appendix A), this transaction would retrieve a set of three INOPQQ numbers corresponding to the three pieces of equipment that will have problems if the generator does not work. Utilizing this set of numbers, the DMLP will next generate a disjunctive retrieve similar to the following (assuming the three retrieved INOPQQ values are 3, 6, 8):

[[RETRIEVE ((TEMP=Inoperative) and (INOPQQ = 3)) or ((TEMP=Inoperative) and (INOPQQ = 6)) or ((TEMP=Inoperative) and (INOPQQ = 8)) (PROBLEM) BY PROBLEM]

d. RETRIEVE with WHERE clause

The final type of RETRIEVE transaction that can be translated by the

(1) Retrieve function1(entity1) WHERE
(2) BEGIN
(3) function2(entity1) = entity2
(4) function3(entity2) = literal1
(5) function4(entity2) = literal2
...
(6) functionx(entity2) = literal x
(7) END

Figure 27 DAPLEX RETRIEVE with WHERE Clause

DMLP is a RETRIEVE with a WHERE clause (see Figure 27). The DMLP first validates all entity and function names, ensuring they have been previously defined in the schema. The function in line (3) of Figure 27, must correspond to one of our pointer entities. If any of the above stipulations are not met, an error message is output and further processing of the query ceases.

Line (1) provides the information to generate the final RETRIEVE statement. The final ABDL RETRIEVE statement will be based upon retrieved entity1 identification attribute values. From the BEGIN until the END clause is reached, the DMLP will create one RETRIEVE COMMON transaction for each conditional statement read. These queries will locate the identification attributes necessary for the final retrieve. Between successive retrieves, the attributes are stored as an intersection set utilizing a linked list. If at anytime the set of attributes becomes the null set processing stops, since every intersection thereafter will also produce a null set. This of course means that there is no data which matches our original query.

Figure 28 is an example of a RETRIEVE statement with the WHERE modifier. This example is taken from the tutorial found in Appendix A. When the DMLP reads line (4) it could generate a RETREIVE statement as follows:

(1) Retrieve discrepancy(malfunction) WHERE
(2) BEGIN
(3) indication (malfunction) = inoperative
(4) problem(inoperative) = "UHF_No_Xmit/Rx"
(5) problem(inoperative) = "TACAN_No_Hd/DME"
(6) problem(inoperative) = "RDR_No_Xmit/Rx"
(7) END

Figure 28 DAPLEX RETRIEVE with WHERE Modifier

[RETRIEVE((TEMP=Inoperative) and (PROBLEM="Uhf_no_xmit/rx") (INOPQQ)
COMMON(INOPQQ,INOPQQ)
RETRIEVE((TEMP=Indication_malf) (MALFQQ))]

Lines (5) and (6) from Figure 28 will cause the DMLP to generate statements similar to the above with the string "UHf_no_xmit/rx" replaced with "Tacan_no_hd/dme" and "Rdr_no_xmit/rx", respectively.

Assume for this example that the first RETRIEVE COMMON statement returns a set of entity1 identification attributes {7,10,12,14,15} and the second RETREIVE COMMON returns the set {10,11,12,15}. The intersection of these two sets result in the new set {10,12,15}. If the final RETRIEVE COMMON returns the set of identification attributes {7,8,10,12,14} the final intersection produces the new set {10,12}.

After all the RETRIEVE COMMON statements have been executed and their results merged, the final simple RETRIEVE can be created. Line (1) provides the template name (i.e. entity1) and the list of attributes to be retrieved. The set of identification attributes are used to form the disjoint clauses of the final RETRIEVE statement. Utilizing our sample data from the previous paragraph, the final RETRIEVE would look like:

[RETRIEVE((TEMP=Malfunction)(MALFQQ=10)or(TEMP=Malfunction)(MALFQQ=12))
(PROBLEM) by PROBLEM]

This RETRIEVE statement is loaded into the dap_info data structure, the appropriate operation is set and the control of the program is passed to the KC for execution. When the backend returns the data and control of the system, the KFS parses the output and prints the results on the screen for the user.

3. DELETE Transactions

The final major requirement of any database system is the DELETE transaction. Using INSERT, RETRIEVE, RETRIEVE-COMMON and DELETE the database system should be functionally complete. In our DAPLEX model we took advantage of the similarities of the RETRIEVE and the DELETE statements for both the functional data

[RETRIEVE ((TEMP=Inoperative) and (INOPQQ = 3)) (PROBLEM) BY PROBLEM]

[DELETE ((TEMP=Inoperative) and (INOPQQ = 3))]

Figure 29 Comparison of RETRIEVE and DELETE Statements

model and the ABDL (see Figure 29). The only real difference, besides the obvious RETRIEVE and DELETE, is the missing list of attributes to be output. Otherwise all processing and collection of identification attributes is exactly the same. When control is passed to the DELETE process, the program simply modifies the original query by replacing DELETE with RETRIEVE. It then pass the modified query to the DMLP which processes it as though it is a RETRIEVE statement. When the DMLP is finished, the DELETE process intercepts the final ABDL transaction, replaces RETRIEVE with DELETE, chops off the ending attributes (by searches for the double right parens), and send the query on to the KC for execution. It also has to ensure the operation indicator is set to ExecDelReq, so the KC will handle the query properly and doesn't wait around for data that will never come.

VI. THE KERNEL CONTROLLER (KC) AND KERNEL FORMATTING SYSTEM (KFS)

A. THE KERNEL CONTROLLER

The Kernel Controller (KC) is responsible for the inter-communications between the backend system controller and the Language Interface module. It accomplishes this via the Test Interface (TI). All procedure calls to the TI are characterized by TI_ at the start of the procedure name followed by an R or an S for receive or send functions, respectively (e.g. TI_S$TrafUnit is a process which sends message traffic to the backend.) After the DAPLEX queries are parsed and translated by the KMS, they are passed with operations indicator to the KC.

The KC first ensures there are no current problems or faults with any communications with the backend. If there are fault messages the KC attempts to process them prior to initiating any new transactions. If it is unable to do so the system will fault and the MDBS program is terminated. This is an extreme situation, however, and normally the result of some catastrophic backend failure. In a normal situation, the KC will ensure the communications channel is clear before sending the transaction.

The KC does no processing of the transactions. It assumes they are correct as received from the KMS. The transactions are passed to the KC as a linked list of ABDL transactions. Each transaction within the linked list is assumed to be of the same type. Under the current configuration, a backend problem will result if any type of transaction types are mixed (e.g. you cannot mix INSERT transactions with RETRIEVE transactions.)

Transactions are executed one at a time by forwarding them to the backend system, requesting reports of any errors between each transaction, until the list is exhausted. If the transaction was a user requested retrieve and no errors were reported by the backend system, program control, along with the database information pointer, is then passed to the KFS. Program control is returned to the KMS in all other cases.

There are numerous occasions when it is necessary to retrieve some information so the initial query may be completed by the KMS. This information, normally entity identification attributes, is of a transient nature and is not part of actual response requested by the user. A normal retrieve will have control eventually passed to the KFS for display. We developed a quick retrieval mechanism which passes the information back to the KMS instead of to the KFS. To reduce the complexity of follow on transactions which utilize the data, it is stored as a set, so duplicate identification attributes are suppressed. The quick retrieval processing of the returned data is very similar to the processing done in the KFS which will be discussed shortly. The major difference between the two is that the KFS places the data on the screen while the quick retrieve places the data into sets.

B. THE KERNEL FORMATTING SYSTEM (KFS)

As its name suggests the KFS is responsible for placing the data received from the kernel data system onto the screen in a format consistent with the data model being used. In our case, we try to present the data as a list of attributes for a set of entities. Obviously we cannot display our entities which represent complex objects. How would you display a malfunction? We can display the name (description) of the malfunction, we can give a list of the names representing the equipment affected by the malfunction but it is not really possible to display the actual malfunction itself. In more complex systems we may be able to display a bitmap representation of a malfunctioning piece of equipment (our example of a non working electrical generator would not be very exciting.) Therefore in the KMS we do not attempt to display entities, but rather the resultant of the functions applied to the entity as long as the resultant is a primitive type (e.g. Integer, Character, etc.) Even though the user is supposed to be unaware of the existence of the entity identification values, it is possible to display them via the KFS.

When the KFS obtains control of the system a copy of the dap_info is passed along with a linked list of functions expected to be received from the kernel. As the data are read, they are compared against the list of expected values to ensure we only display the

attributes requested by the user. The entities have already been compared and screened during other operations to ensure that duplicates are not displayed. This does not mean, however, that we could not have separate entities with equal attribute values. These entities would *appear* to the user to be duplicates, when in reality they are the same values for different entities. For example, we could request the first name of a set of people. If the name BILL showed twice it would indicate that there were two distinct people with the same first name.

Future models may find it desirable to process and allow duplicate entities so the frequency of a particular one could be determined and the display order based on that frequency. In our sample database design, that would be equivalent to finding the most probable entity.

The data received from the kernel is passed in one of two similar formats (see Figure

[TEMP\0attribute_value\0... ATTRIBUTE_NAME\0attribute_value\0]\0
[COMMON\0TEMP\0attribute_value\0... ATTRIBUTE_NAME\0attribute_value\0]\0

Figure 30 Format of the Retrieved Kernel Data

30). As can been seen the beginning delimiter is [and the ending delimiter is], very similar to the format the KMS uses in sending the data to the backend. Each piece of data, however, is separated by a \0 character. In the programming language C, the \0 is the end of string character marker. Any attempts to make use of the native C string library functions would only apply to the first string (i.e. [TEMP and [COMMON in Figure 30). Knowing that the returned data is in an extended character array we were able to make use of the string functions in a less standard manner.

Previous interface modules utilized individual character manipulations to parse and reformat the data. The procedures used were lengthy, difficult to follow and did not make use of pre-existing functions. These procedures basically examined each character being returned from the kernel. If the character was a] they knew it was the end of the retrieved data. If the character was a \0 they new it was the end of a word. In every other case the

character was copied to another character array for later comparison and/or display. Often the other modules would display attributes not necessarily requested by the user or duplicates of the same attribute within a record or entity. These attributes may have been required by the kernel to locate certain data (especially in retrieve-common transactions). Our procedure makes use of the string functions already available in ANSI C and alleviates many of the previous problems.

As stated earlier, as part of the retrieve transaction process, the KMS prepares a linked list of attribute names that have been explicitly requested by the user. The number of attribute values being retrieved will be the same for each entity. Using this information, we simply match the current portion of a retrieved string to the words in our linked list or check to see if the string contains a *|* (end-of-response) character. If the latter is true we know the array has been completely processed. If the former is true then we know the string which follows will be an attribute value we wish to display. If neither is true, we simply skip the next string.

We advance the pointer position past the string presently being pointed at utilizing the ANSI C strlen function. From earlier sections you may remember that the attribute names are stored as all upper case characters while the values are stored with the first character capitalized and the rest lower case. Because of this, there is no problem in string comparisons when the name and the value may have been the same. When we have examined and processed the same number of attribute names as was passed from the KMS we insert a carriage return and line feed. Since the initial display of the transactions by the LIL may scroll off the screen we also redisplay the query so the user to can more easily compare the query with the response. The resultant KFS code is much cleaner and easier to comprehend. We also avoided the problem of displaying data that was not requested. As the entities have been pre-screened by the KMS by comparing the unique entity identification attributes, no duplicate entities are displayed.

VII. CONCLUSION

This thesis involved the implementation of the Functional/DAPLEX database interface to the Multi-Lingual, Multi-backend Database System (MDBS). The Functional model is well suited for the storage of data used in artificial intelligence and expert system applications. MDBS is an effective tool for managing a data base systems growth, performance, data sharing and resource consolidation.

A. IMPLEMENTATION

In this thesis, we presented the specification and implementation of a functional language interface. The DAPLEX model compliments the five previously implemented database models: Attribute-based, Relational, Hierarchical, Network and Object-Oriented. It was developed to conform with the data structure and other model conventions established by Kloepping and Mack (Kloe85). The MDBS implemented at the Naval Postgraduate School is the only known system that incorporates and integrates six diverse database management systems into a single system.

The implementation's modules were written in the C language. Substantial attention and effort was expended to produce computer code which was well documented and structured. Unlike previous models we avoided the use of global variables and pointers whenever possible. Variables to be utilized by subprograms and functions were passed within procedure calls.

Although we only implemented a subset of Shipman's DAPLEX model (Ship81), the interface provides the necessary features to adequately demonstrate that the Functional model is a viable, practical component of the MDBS. It is sufficiently robust to enable database users to develop expert model applications of real world phenomena.

The program was written to assist with syntactical and semantic error checking capabilities. In as many cases as feasible, messages commensurate with the error are output to the screen to assist the user. The program's error handling routines significantly reduce the number of catastrophic program crashes and core dumps.

B. LESSONS LEARNED

As the Functional interface module parses the DAPLEX transactions, it simultaneously creates equivalent ABDL statements. The statements are checked for syntactic and semantic accuracy at the same time the ABDL statement is being constructed. In other words it is similar to a one pass compiler. In retrospect, we feel a two pass procedure may have been a better strategy. Making modifications, to allow for additional capabilities, were difficult. Changes to the way the DAPLEX statements were validated for correctness often produced many undesirable side-effects on the ABDL statement construction process. A two pass algorithm could have eliminated some of these problems while making the program more flexible.

Our representation of the Functional data model does allow the use of the same functions over different domains. It does not allow for functions with multiple arguments. For example, we should be able to designate a function such that

offspring(husband, wife) = person

to represent the set of children produced in a marriage. It is obviously possible for either spouse to have children from another relationship but it would not be possible for either to have children alone. It would of course be desirable to retrieve based on a single spouse, e.g. retrieve all the children a wife had regardless of the husband.

The program was written in a non- ANSI standard (i.e. K and C) version of C. No ANSI standard compiler is currently available on the Sun Microcomputer for the MDBS. This version lacked many of the features of ANSI C which would have made the program easier to write and understand. One such limitation was in defining the functions. Modern C compilers allow the definition of a function similar to:

```
char * func_name(int var1,char var2)
```

The format we were required to use was similar to:

```
char* func_name(var1, var2)
    int var1;
    char var2;
    {...
    ...}
```

This distinction prevented us from utilizing the software on a more robust compiler and debugger and porting the code unto the Sun 4 microcomputer. It would be highly beneficial to obtain a more capable compiler for future applications.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

The cross model access capability which accomplishes data sharing among database models, is one of the primary goals of the MDBS. The DAPLEX to ABDL interface obviously exists. Other models have additional cross modeling capability, allowing access to another model's base data. The goal is to allow the user of one model to view another models base data in a format they are most familiar. Similar cross model capabilities should be implemented between the functional data model and the other MDBS modules. In some situations this may not be possible as it may not be able to logically represent older models in newer models. For example, the concept of representing a network as a hierarchy would only be realistically possible unless the network was already very specialized, i.e. a hierarchy. To represent the hierarchical model as a network model is, of course, no problem.

Upgrading to an ANSI standard compiler would have a positive impact on the MDBS. It would eliminate the inconsistencies and provide additional programming tools. It is our belief that the next system upgrade should include a standard C++ compiler. This compiler would allow the MDBS data modules improved structures with no global variables and pointers. The use of objects, classes and inheritance would better complement the individual data module sharing, since each module uses many shared and as well similar data structures.

While implementing a new compiler, It would also be an excellent time to review and evaluate the data structures utilized by the various models. Some of the structures contain variables that are no longer used or have misleading names. Eliminating and renaming variables would improve the understandability of the program. A comparison of each

module's data structure, could lead to single standard database node with improved inter-module data sharing.

The MDBS backend operating system could be revamped. Designed many years ago it still has many of the systems software limitations of the time that should not be present today. For instance, we were limited to a character string length of 16 characters. We believe that this was a limitation of the old system to keep the size to two bytes or less. Also currently, the metadata files (template and descriptor) must be manually copied to the backend before they can be used for the first time. This feature limits the users capability to dynamically change the database schema during a session.

D. SUMMARY

The results of this thesis demonstrated that the Functional data model is an integral module of the MDBS. The Functional module interface provides an artificial intelligence and expert system aspect to the MDBS, that it did not have previously, while maintaining the integrity of MDBS's kernel database management system. The MDBS is capable of understanding and processing multiple diverse database models. The concept of a homogeneous mixture of heterogeneous database models remains a viable option through implementation of a system similar to the MDBS.

APPENDIX A - USER DEMONSTRATION GUIDE

User inputs are denoted by *italicized* entries. All user files are expected to be located in /u/mdbs/UserFiles. Comments are in this font (12 pt Times).

```
db11/u/mdbs> begin
111213abd1.930809.txt       start.check
FAMILY.cover                start.cntrl*
Family.cover.backup         start.cntrl.screen.trace*
MAINT.r                     stop.check
daplex.demo                 stop.db11*
list2.stop                  stop.db12*
main*                       stop.db13*
master.run.be*              temp.txt
max_n_cmds.cntrl            tempjunk/
min_n_cmds.be*              trace/
min_n_cmds.cntrl            zero.db11*
old/                        zero.db12*
sstop*                      zero.db13*


Welcome to MDBS, today is  Mon Jan 31 09:33:22 PST 1994



Check the time each file was last compiled:

-rwxrwxr-x  1 mdbs       122880 Jan 22 15:03 ../BE/bget.exe
-rwxrwxr-x  1 mdbs       122880 Jan 22 15:03 ../BE/bput.exe
-rwxrwxr-x  1 mdbs       204800 Jan 22 14:59 ../BE/cc.exe
-rwxrwxr-x  1 mdbs        57344 Jan 22 15:00 ../BE/dio.exe
-rwxrwxr-x  1 mdbs       294912 Jan 22 15:02 ../BE/dirman.exe
-rwxrwxr-x  1 mdbs       262144 Jan 22 15:05 ../BE/recp.exe

-rwxrwxr-x  1 mdbs       122880 Jan 22 14:52 ../CNTRL/cget.exe
-rwxrwxr-x  1 mdbs       122880 Jan 22 14:52 ../CNTRL/cput.exe
-rwxrwxr-x  1 mdbs       114688 Jan 22 14:52 ../CNTRL/iig.exe
-rwxrwxr-x  1 mdbs       131072 Jan 22 14:53 ../CNTRL/pp.exe
-rwxrwxr-x  1 mdbs       180224 Jan 22 14:55 ../CNTRL/reqp.exe
-rwxrwxr-x  1 mdbs       835584 Jan 27 09:30 ../CNTRL/ti.exe

There should be 12 files listed, if not you need to recompile.
```

```
Do you need to recompile any executable and/or copy the 6
 executable files to each Back End
(bget, bput, cc, dio, dirman, recp.exe)? (y/n) n

The Current Configuration is:

Version Name:   greg

Controller:     db11

1 Back End:
                db13

 WARNING:  All data will be lost if you reconfigure

 Do you wish to reconfigure the Back Ends? (y/n) n


 Do you wish to use current database? (y/n) n

Zeroing backend meta disk on back end, db13...
File to zero = /dev/sd2c
Bytes to zero = 1000000
Bytes written...
     102400
     204800
     307200
     409600
     512000
     614400
     716800
     819200
     921600
    1000000
Zeroing backend data disk on back end, db13...
File to zero = /dev/sd4c
Bytes to zero = 200000
Bytes written...
     102400
     200000
Removing CINBT and IIG AT tables on controller, db11...

 Do you wish to run the Multi Modal, Multi Lingual,
                Multi Backended Database System? (y/n) y

stopping processes on back end db13
```

```
no processes to kill on db13
stopping processes on db11, the controller
stop.db11: syntax error at line 13: '(' unexpected
EXECUTING: start.cntrl
rm: trace/*.tr: No such file or directory
starting 5 of 6 controller processes on db11...
EXECUTING: rsh db13 -n /u/mdbs/be.greg/run.be &
EXECUTING: /u/mdbs/greg/CNTRL/ti.exe  1
PID written to /u/mdbs/.ti.exe.pid
**** Unlink error: No such file or directory
No match.
Running backend on db13...
[1] 13961
[2] 13962
[3] 13963
[4] 13964
[5] 13965
[6] 13966
[0] 0000        System configured for 1 backend(s).
```

NOTE: Ensure there are six processes running plus the [0] 0000 otherwise the MDBS may suddenly quit.

```
MBDS: Initializing communications...

Seconds remaining:  5  4  3  2  1
```

```
          The Multi-Lingual/Multi-Backend Database System

     Select an operation:

             (a)  -  Execute the attribute-based/ABDL interface
             (r)  -  Execute the relational/SQL interface
             (h)  -  Execute the hierarchical/DL/I interface
             (n)  -  Execute the network/CODASYL interface
             (f)  -  Execute the functional/DAPLEX interface
             (o)  -  Execute the Object-Oriented interface
             (x)  -  Exit to the operating system

     Select-> f

     Enter type of operation desired
             (l) - load new database
             (p) - process existing database
             (x) - return to the MLDS/MDBS system menu

     Action --- > 1


     Enter name of database ----> MAINT

     Enter mode of input desired
             (f) - read in a group of creates from a file
             (t) - read in creates from the terminal
             (x) - return to the main menu

     Action --- > f
```

Enters the database schema. A copy is printed at the end of this section.

```
     What is the name of the CREATE/ QUERY file ----> MAINTdapdb

     Enter type of operation desired
             (l) - load new database
             (p) - process existing database
             (x) - return to the MLDS/MDBS system menu

     Action --- > p


     Enter name of database ----> MAINT
```

```
Enter your choice
        (d) - display schema
        (m) - mass load from a data file
        (s) - send data to a file for mass load
        (f) - read in a group of queries from a file
        (t) - read in queries from the terminal
        (x) - return to previous menu

Action --- > d

Database Name : MAINT

Entity: MALFUNCTION
ROOT(MALFUNCTION) = EQUIPMENT
INDICATION(MALFUNCTION) = INOPERATIVE
DISCREPANCY(MALFUNCTION) = STRING

Entity: INOPERATIVE
DIAGNOSIS(INOPERATIVE) = MALFUNCTION
WRA(INOPERATIVE) = EQUIPMENT
PROBLEM(INOPERATIVE) = STRING

Entity: EQUIPMENT
NUMBER(EQUIPMENT) = STRING
NAME(EQUIPMENT) = STRING

Enter your choice
        (d) - display schema
        (m) - mass load from a data file
        (s) - send data to a file for mass load
        (f) - read in a group of queries from a file
        (t) - read in queries from the terminal
        (x) - return to previous menu

Action --- > m


What is the name of the CREATE/ QUERY file ----> MAINT.r
5  10  15  20  25  30
```


Indicates the system is working.

```
Enter your choice
        (d) - display schema
        (m) - mass load from a data file
        (s) - send data to a file for mass load
        (f) - read in a group of queries from a file
        (t) - read in queries from the terminal
        (x) - return to previous menu

Action --- > f


What is the name of the CREATE/ QUERY file ----> MAINTdapreq1

        1       Retrieve name(equipment)

        2       Retrieve number(equipment)

        3       Retrieve problem(inoperative)

        4       Retrieve discrepancy(malfunction)

        5       Retrieve name(equipment) and number(equipment)

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > 5


Retrieve name(equipment) and number(equipment)

NAME(EQUIPMENT)          NUMBER(EQUIPMENT)
Generator                Wra_1
Radar                    Aps-138
Tacan                    An-125
Uhf_cb                   Wra_2
Uhf_radio                Arc-51
Uhf_trx                  Wra_7

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu
```

```
Action --- > 4


Retrieve discrepancy(malfunction)

DISCREPANCY(MALFUNCTION)
Generator_inop
Uhf_cb_popped
Uhf_trx_malf

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > 3


Retrieve problem(inoperative)

PROBLEM(INOPERATIVE)
Rdr_no_xmit/rx
Tacan_no_hd/dme
Uhf_no_xmit/rx
Uhf_no_rx
Uhf_no_xmit

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > x

Enter your choice
        (d) - display schema
        (m) - mass load from a data file
        (s) - send data to a file for mass load
        (f) - read in a group of queries from a file
        (t) - read in queries from the terminal
        (x) - return to previous menu

Action --- > f

What is the name of the CREATE/ QUERY file ----> MAINTdapreq2
```

```
1       Retrieve name(equipment) and number(equipment)

2       Delete equipment
        SUCH THAT
        number(equipment) = "WRA_2"

3       Retrieve problem(inoperative)
        SUCH THAT
        indication(malfunction) = inoperative AND
        discrepancy(malfunction) = "Generator_Inop"

4       Retrieve discrepancy(malfunction)
        SUCH THAT
        indication(malfunction) = inoperative AND
        problem(inoperative) = "UHF_No_Xmit/Rx"

5       Retrieve discrepancy(malfunction)
        SUCH THAT
        indication(malfunction) = inoperative AND
        problem(inoperative) = "UHF_No_Rx"

6       Retrieve discrepancy(malfunction) WHERE
        BEGIN
        indication (malfunction) = inoperative
        problem(inoperative) = "UHF_No_Xmit/Rx"
        problem(inoperative) = "TACAN_No_Hd/DME"
        problem(inoperative) = "RDR_No_Xmit/Rx"
        END

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu
```

```
Action --- > 3


Retrieve problem(inoperative)
SUCH THAT
indication(malfunction) = inoperative AND
discrepancy(malfunction) = "Generator_Inop"

PROBLEM(INOPERATIVE)
Rdr_no_xmit/rx
Tacan_no_hd/dme
Uhf_no_xmit/rx

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > 4


Retrieve discrepancy(malfunction)
SUCH THAT
indication(malfunction) = inoperative AND
problem(inoperative) = "UHF_No_Xmit/Rx"

DISCREPANCY(MALFUNCTION)
Generator_inop
Uhf_cb_popped
Uhf_trx_malf

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu
```

What problems exist if the generator is inop.

What could cause the UHF radio to not transmit or receive.

```
Action --- > 5

Retrieve discrepancy(malfunction)
SUCH THAT
indication(malfunction) = inoperative AND
problem(inoperative) = "UHF_No_Rx"

DISCREPANCY(MALFUNCTION)
Uhf_trx_malf

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > 6

Retrieve discrepancy(malfunction) WHERE
BEGIN
indication (malfunction) = inoperative
problem(inoperative) = "UHF_No_Xmit/Rx"
problem(inoperative) = "TACAN_No_Hd/DME"
problem(inoperative) = "RDR_No_Xmit/Rx"
END

DISCREPANCY(MALFUNCTION)
Generator_inop

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > 1

Retrieve name(equipment) and number(equipment)

NAME(EQUIPMENT)          NUMBER(EQUIPMENT)
Generator                Wra_1
Radar                    Aps-138
Tacan                    An-125
Uhf_cb                   Wra_2
Uhf_radio                Arc-51
Uhf_trx                  Wra_7
```

What could cause the UHF to be transmit only.

What could cause all of these together.

```
Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > 2


Delete equipment
SUCH THAT
number(equipment) = "WRA_2"


Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > 1


Retrieve name(equipment) and number(equipment)

NAME(EQUIPMENT)          NUMBER(EQUIPMENT)
Generator                Wra_1
Radar                    Aps-138
Tacan                    An-125
Uhf_radio                Arc-51
Uhf_trx                  Wra_7

Pick the number or letter of the action desired
        (num) - execute one of the preceeding queries
        (d)   - redisplay the file of queries
        (x)   - return to the previous menu

Action --- > x

Enter your choice
        (d) - display schema
        (m) - mass load from a data file
        (s) - send data to a file for mass load
        (f) - read in a group of queries from a file
        (t) - read in queries from the terminal
        (x) - return to previous menu

Action --- > x
```

```
Enter type of operation desired
        (l) - load new database
        (p) - process existing database
        (x) - return to the MLDS/MDBS system menu

Action --- > x


        The Multi-Lingual/Multi-Backend Database System

Select an operation:

        (a)  -  Execute the attribute-based/ABDL interface
        (r)  -  Execute the relational/SQL interface
        (h)  -  Execute the hierarchical/DL/I interface
        (n)  -  Execute the network/CODASYL interface
        (f)  -  Execute the functional/DAPLEX interface
        (o)  -  Execute the Object-Oriented interface
        (x)  -  Exit to the operating system

Select-> x
All done with MBDS
db11/u/mdbs/greg/run--2>
```

MAINTdapdb DECLARATIONS

```
DECLARE equipment ENTITY
DEFINE name(equipment) = STRING
DEFINE number(equipment) = STRING
DECLARE Inoperative ENTITY
DEFINE problem(inoperative) = STRING
DEFINE wra(inoperative) = equipment
DECLARE malfunction ENTITY
DEFINE discrepancy(malfunction) = STRING
DEFINE indication(malfunction) = inoperative
DEFINE root(malfunction). = equipment
DEFINE diagnosis(inoperative) = INVERSE OF indication(malfunction)
```

APPENDIX B - PROGRAM CODE

```
/*

* File Name: alloc.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Alloc/alloc.c
 * This file contains procedures for allocating space for Daplex
  Interface common structures
 *
 */

#include <stdio.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"


                    dap_dbid_node   *dap_dbid_node_alloc()

{
                dap_dbid_node *new_dbid_ptr;

                /* allocate an area of memory for the structure of
        type dap_dbid_node                      */
#ifdef EnExFlag
                printf("Enter dap_dbid_node_alloc\n");
#endif

                if ((new_dbid_ptr = (dap_dbid_node*)
                    malloc (sizeof(dap_dbid_node))) == NULL)
                  {
                                printf("*** dap_dbid_node_alloc problem with malloc\n");
                                sleep(10);
                  }
      new_dbid_ptr->dap_db_name = (char*)malloc(sizeof(char)*DBNLength);

#ifdef EnExFlag
                printf("Exit dap_dbid_node_alloc\n");
#endif
                new_dbid_ptr->next_db = NULL;
                return new_dbid_ptr;

} /* end dap_dbid_node_alloc */
```

```
                dap_db_entity_node   *dap_entity_node_alloc()

{
            dap_db_entity_node   *new_dap_entity_ptr;

            /* allocate an area of memory for the structure of
      type dap_entity_node                          */

#ifdef EnExFlag
            printf("Enter dap_db_entity_node_alloc\n");
#endif

            if ((new_dap_entity_ptr = (dap_db_entity_node *)
                malloc (sizeof (dap_db_entity_node))) == NULL)
                {
                        printf("*** dap_db_entity_node_alloc problem with
malloc\n");
                        sleep(10);
                }

#ifdef EnExFlag
            printf("Exit dap_db_entity_node_alloc\n");
#endif

            new_dap_entity_ptr->next_entity = NULL;
            new_dap_entity_ptr->number_of_attribs=0;
            return new_dap_entity_ptr;

} /* end dap_db_entity_node_alloc */

                dap_db_attrib_node  *dap_db_attrib_node_alloc()

{
            dap_db_attrib_node   *new_dap_db_attrib_ptr;

            /* allocate an area of memory for the structure of
      type dap_db_attrib_node                       */

#ifdef EnExFlag
            printf("Enter dap_db_attrib_node_alloc\n");
#endif
```

```
                if ((new_dap_db_attrib_ptr = (dap_db_attrib_node *)
                    malloc (sizeof (dap_db_attrib_node))) == NULL)
                  {
                                printf("*** dap_db_attrib_node_alloc problem with
malloc\n");
                                sleep(10);
                  }

#ifdef EnExFlag
                printf("Exit dap_db_attrib_node_alloc\n");
#endif

                new_dap_db_attrib_ptr->next_attrib = NULL;
                return new_dap_db_attrib_ptr;

} /* end dap_db_attrib_node_alloc */



                        struct dap_req_info *dap_req_info_alloc()

{
                struct dap_req_info *new_dap_req_ptr;

                /* allocate an area of memory for the structure of type dap_req_info */

#ifdef EnExFlag
  printf ("Enter dap_req_info_alloc\n");
#endif

                if ((new_dap_req_ptr = (struct dap_req_info *)
                    malloc (sizeof (struct dap_req_info))) == NULL)
                  {
                    printf ("*** dap_req_info_alloc problem with malloc\n");
                    sleep(10);
                  }

     new_dap_req_ptr->dpri_in_req=NULL;
                new_dap_req_ptr->dpri_next_req=NULL;
                new_dap_req_ptr->dpri_req=NULL;

#ifdef EnExFlag
  printf ("Exit dap_req_info_alloc\n");
#endif
```

```
        return new_dap_req_ptr;

} /* end dap_req_info_alloc */



                struct dap_kms_info *dap_kms_info_alloc()

{
  struct dap_kms_info *new_dap_kms_info_ptr;

  /* allocate an area of memory for the structure of type dap_kms_info */

#ifdef EnExFlag
  printf ("Enter dap_kms_info_alloc\n");
#endif

     if ((new_dap_kms_info_ptr = (struct dap_kms_info *)
        malloc (sizeof (struct dap_kms_info))) == NULL)
       {
        printf ("*** dap_kms_info_alloc problem with malloc\n");
        sleep(10);
       }

#ifdef EnExFlag
  printf ("Exit dap_kms_info_alloc\n");
#endif

     return new_dap_kms_info_ptr;

} /* end dap_kms_info_alloc */
```

```
/*

* File Name: chk_res_left.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kc/chk_res_left.c $
 * This file contains procedures utilized in processing queries to
  the backends of the MDBS.
 *
 */

#include <stdio.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"


/* An External from TI in MDBS - needed so that we can return to
  attribute-based interface without any problems */
extern  int  reqs_left_count;


                        dap_chk_responses_left(dap_ptr)


 /*  This procedure accomplishes the following:              */
 /*  (1) Receives the message from MBDS by calling           */
 /*     TI_R$Message() which is defined in the Test Int.     */
 /*  (2) Gets the message type by calling TI_R$Type.         */
 /*  (3) If not all responses to the request have been       */
 /*     returned, a loop is entered.  Within this loop a     */
 /*     case statement separates the responses received by   */
 /*     message type.                                        */
 /*  (4) If the response contained no errors, then procedure */
 /*     TI_R$ReqRes() is called to receive the response from */
 /*     MBDS.                                                */
 /*  (5) A check is then made to determine if this is the last */
 /*     response.  If it is, then the results are processed. */
 /*  (6) If the message contained an error then procedure    */
 /*     TI_R$ErrorMessage is called to get the error message */
 /*     and then procedure TI_ErrRes_output is called to     */
 /*     output the error mesage.                             */

struct dap_info  *dap_ptr;
{
 int       OddMark = TRUE;
```

```
 int      msg_type,
          done;
 char     *response;

 /* values for these two depends on the defines in tstint.def (CNTRL/TI) */
 char     request[ONE_K],   /* Added request length */
                  err_msg[ONE_K];
 struct   ReqId  rid;  /* Defined in licommdata.h */

#ifdef EnExFlag
 printf("Enter dap_chk_responses_left \n");
#endif

/* koi_response is now created in newuser.c in Lil          */
 response = dap_ptr->dpi_kfs_data.kfsi_dap.kdi_response;
 done = FALSE;
 while (!done)/*Not all responses for the current request have been received*/
  {
   TI_R$Message();/*receive message from MBDS*/

   msg_type = TI_R$Type(); /* get the message type of the received message*/

   switch(msg_type) /* Is the response correct or are there errors? */
    {
     case CH_ReqRes: /* The response is correct */

      TI_R$ReqRes(&rid,response); /* Receive the results */

      done = f_chk_if_last_response(dap_ptr); /*Are we done */
      /* What kind of operation is this */
      switch (dap_ptr->dap_operation)
       {
       case ExecRetCReq:
       case ExecRetReq:

        f_kernel_formatting_system(dap_ptr);

        break;

       case  ExecInsReq:
       case  ExecDelReq:
       case  ExecUpdReq:
```

```
            break;
          }

        break;

    case ReqsWithErr:

        TI_R$ErrorMessage(request,err_msg);
        TI_ErrRes_output(request,err_msg);
        done = TRUE;
        break;

    default:
        printf("Illegal msg_type = %d received.\n", msg_type);
        break;

    } /* End switch */

  } /* End while */

 /* Reset the External from MBDS */
 reqs_left_count = 0;

#ifdef EnExFlag
 printf("Exit dap_chk_responses_left \n");
#endif

} /* end dap_chk_responses_left */


                        f_chk_if_last_response(dap_ptr)

struct dap_info  *dap_ptr;

 /* This procedure accomplishes the following:                    */
 /* (1) Determine length of response.                             */
 /* (2) Determines if this is the last response to a given request and   */
 /*     returns a boolean indicating such.                        */

{
 int response_length;

#ifdef EnExFlag
```

```
 printf("Enter f_chk_if_last_response \n");
#endif

 for (response_length = 0; /* Calculates response length */
    dap_ptr->dpi_kfs_data.kfsi_dap.kdi_response[response_length] != EOResult;
    response_length++);

 ++response_length;

 /* Checks if this is the last response */
 if (dap_ptr->dpi_kfs_data.kfsi_dap.kdi_response[response_length - 3]
    == CSignal)
   {
#ifdef EnExFlag
 printf("Exit1 f_chk_if_last_response \n");
 printf("response = %d\n", response_length);
#endif

     return TRUE;
   } /* end if */
 else  /* It is not the last response */
   {
#ifdef EnExFlag
 printf("Exit2 f_chk_if_last_response \n");
#endif

     return FALSE;
   } /* end else */

 } /* end f_chk_if_last_response */
```

```
/*
 * File Name: dap_checks.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/dap_checks.c
 * This file contains utility procedures for finding information about
   the database currently in use
 *
 */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#ifndef DAPLEX_INFO
#include <licommdata.h>
#include <dap_info.h>
#endif


                        int   entity_search(name, ptr)

/* Sees if an entity exists */
  char *name;
  dap_db_entity_node *ptr;
  {
    while (ptr)
    {
      if (!strcmp(name,ptr->dap_entity_name)) return 1;
      ptr=ptr->next_entity;
    }
    return 0;
  }


                        int  attrib_search(attrib, entity, ptr)

/* Sees if an attribute in an entity exists */
  char *attrib;
  char *entity;
  dap_db_entity_node *ptr;
  {
    dap_db_attrib_node *at_ptr;
```

```
  while (ptr)
  {
   if (!strcmp(entity, ptr->dap_entity_name))
   {

    at_ptr = ptr->first_attrib;
                while(at_ptr)
                {
                 if(!strcmp(attrib,at_ptr->dap_attrib_name)) return 1;
                 at_ptr=at_ptr->next_attrib;
                }
   return 0;
   }
   ptr=ptr->next_entity;
  }
  return 0;
 }



                              int   alias_search(name,ptr)

/* Sees if an alias exists */
 char *name;
 dap_db_alias_node *ptr;
 {
  while (ptr)
  {
   if (!strcmp(name,ptr->dap_alias_new_name)) return 1;
   ptr=ptr->next_alias;
  }

  return 0;
 }


int
 illegal_char_search(name)
/* Sees if an valid characters are used in names*/
 char *name;
 {
 if (!isalpha(*name++))
 {
   printf("Non alphanumeric at start of token.\n");
```

```
   return 1;
  }
  for(;*name;*name++)
  {
   if (!(isalnum(*name)|| *name == '_'))
   {
    return 1;
   }
  }
  return 0;
  }
```

int pre_parser(db_node, query, start)

```
/* Sees if the function passed is a composite and expands it*/
  dap_dbid_node *db_node;
  char *query;
  int start;
{
  int composite=FALSE;
  char token1[ONE_K], *tok1, *tok2;

#ifdef EnExFlag
   printf("Enter the pre_parser Function.\n");
   fflush(stdout);
#endif
  strcpy(token1,query);
printf("HELP value passed in to check is %s\n",token1);
  if (strstr(token1,"("))
  {
    tok1=strtok(token1," (");
    tok2=strtok(NULL,")");
    if (attrib_type_search(tok1,tok2,db_node->first_entity)=='G')
    {
      strcpy(token1,get_addr(db_node,'A',tok2,tok1));
printf("HELP value passed is now %s\n",token1);
      composite=TRUE;
    }
  }
  if (composite)
  {
    tok1=strtok(token1,"&");
    tok2=strtok(NULL,"&");
```

```
    if (start)
    {
      sprintf(query,"%s # %s &",tok1,tok2);
    }
    else
    {
      sprintf(query,"%s # %s &",tok2,tok1);
    }
  }
  return composite;
}
```

```
/*
 * File Name: dap_define.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/dap_define.c
 * This file contains the procedure for processing DDL constructs.
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <licommdata.h>
#ifndef DAPLEX_INFO
#include <dap_info.h>
#endif

/* following variable need to be passed  dap_db_id_node *db_node */

                        int  ddl_parse(input, dap_info_ptr)

 struct dap_info *dap_info_ptr;
 char *input;
{
  dap_dbid_node *db_node;
  char  keyword[InputCol], string[InputCol], string2[InputCol];
  char  token1[InputCol];
  char  token2[InputCol];
  char  token3[InputCol];
  char  token4[InputCol];
  char  temp[InputCol];
  char  substring[InputCol];

#ifdef EnExFlag
  printf ("Enter ddl_parse\n");
  fflush (stdout);
#endif


  db_node = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;

  strcpy(string,input);
```

```
  /* get operation keyword from string */
  keyword = strtok(string," ");

/* Process transaction beginning with DECLARE*/
  if (!(strcmp(keyword,"DECLARE")))
  {

   if (strchr(input, '=')) /* "=" -> alias declaration else new entity */
   {
     /* New Alias Declaration */
     token1 = strtok(NULL, " =");
     if (illegal_char_search(token1))
               {
       ERROR("Illegal character in word:",token1);
       dap_info_ptr->dap_error = ErrCreateDB;
       return 0;
               }

     if (alias_search(token1, db_node->first_alias)
             ||entity_search(token1, db_node->first_entity))
               {
       ERROR("Previously declared : ", token1);
       dap_info_ptr->dap_error = ErrCreateDB;
       return 0;
     }

     /*see if token 1 is a reserved word*/
     if (reserve_search(token1))
     {
       ERROR("Entity Name is reserved word: ",token1);
       dap_info_ptr->dap_error = ErrCreateDB;
       return 0;
     }

     /* process the token that follows = in the alias declaration */
     token2 = strtok(NULL," ");
     if (token2[0] == '=' )
     {
       if (token2[1]=='\0')
       {
         /* token2 was only "=". Get next token */
         token2 = strtok(NULL," ");
                 }
```

```
    else
    {
      /* token2 has "=" (no spaces) immediately preceeding token */
      *token2++; /* remove preceeding '=' */
    }
  }

  if (token2)
          {
    if (illegal_char_search(token2))
          {
      ERROR("Illegal character in word:",token2);
      dap_info_ptr->dap_error = ErrCreateDB;
      return 0;
          }
    else
          {
      if (entity_search(token2, db_node->first_entity))
          {
        /* Succesful Alias Declaration*/
        add_new_alias(db_node,token1,token3,token2,token4);
        add_new_attrib(db_node,token2,token1,'A',token4,token3);
      }
      else
          {
        ERROR("Entity has not been defined : ",token2);
        dap_info_ptr->dap_error = ErrCreateDB;
                    return 0;
      }
          }
        }
  else
          {
    ERROR("Nothing found after = ",string);
    dap_info_ptr->dap_error = ErrCreateDB;
    return 0;
  }

}
else  /* Not an entity New Entity Declaration */
{

  if (token1 = strtok(NULL, " "))
```

```
    {

      if (illegal_char_search(token1))
                {
        ERROR("Illegal character in word:",token1);
        dap_info_ptr->dap_error = ErrCreateDB;
        return 0;
                }

      if (entity_search(token1,db_node->first_entity))
                {

        ERROR("Entity has previously been defined : ",token1);
        dap_info_ptr->dap_error = ErrCreateDB;
        return 0;
                }

      /*see if token is reserved word*/
      if (reserve_search(token1))
      {
        ERROR("Entity Name is reserved word: ",token1);
        dap_info_ptr->dap_error = ErrCreateDB;
        return 0;
      }

    }
    else
/* token1 is null -> DECLARE with no arguments*/
              {
      ERROR("Too few arguments - Nothing follows DECLARE");
      dap_info_ptr->dap_error = ErrCreateDB;
      return 0;
    }

    /* have valid entity name (token1). Now the word ENTITY must follow */
    if (token2 = strtok(NULL, " "))
    {
      if (!(strcmp(token2, "ENTITY"))) /*token2 = "ENTITY" */
      {
        /* Succesfully declared entity */
        add_new_entity(db_node, token1);
      }
      else
```

```
        {
        ERROR("Illegal DECLARE statement", string);
        dap_info_ptr->dap_error = ErrCreateDB;
        return 0;
        }
      }
      else
              {
        ERROR("Too few arguments - Nothing follows DECLARE");
        dap_info_ptr->dap_error = ErrCreateDB;
        return 0;
      }
    }

    /* There should be no more tokens in string */
    if (strtok(NULL," "))
    {
      ERROR("Too many arguments in Declare Statement", input);
      dap_info_ptr->dap_error = ErrCreateDB;
      return 0;
    }

}
else if (!(strcmp(keyword,"DEFINE")))
 {
 if (!(strchr(input, '=')))
 {
   ERROR("Illegal Definition format. No equal sign",input);
   dap_info_ptr->dap_error = ErrCreateDB;
   return 0;
 }

 substring = strtok(NULL,"=");

 if (!(strstr(substring,")")))/* No Right Parens*/
 {
    ERROR("Illegal Definition format. Improper Left Hand side",input);
    dap_info_ptr->dap_error = ErrCreateDB;
    return 0;
 }

 if (strstr(substring," ("))/* Case one or two*/
 {
```

```
  token1 = strtok(substring," ");
  token2 = strtok(NULL," ");
  if (strcmp(token2,"("))/*Means token2 is not Left Parens by itself*/
  {
    token2++;
  }
  else
  {
    token2 = strtok(NULL," ");
  }
}
else/*Case three or four*/
{
  token1 = strtok(substring," (");/*Use space to clear any leading*/
  token2 = strtok(NULL," (");
}/*Should be down to good tokens only otherwise token will be messed up*/
if (strstr(token2,")"))/*It had best be the last character*/
{

            token2[strlen(token2)-1]='\0';/*Get rid of the last character*/
}

if(strtok(NULL," )"))/*Means more after right parens*/
{
   ERROR("Illegal Definition format. Improper Left Hand side",input);
   dap_info_ptr->dap_error = ErrCreateDB;
   return 0;
}

if (reserve_search(token1))
{
   ERROR("Illegal Definition format. Reserved word on Left Hand side",string);
   dap_info_ptr->dap_error = ErrCreateDB;
   exit(-1 );
}

if (illegal_char_search(token1))
{
   ERROR("Illegal Definition format. Attribute name on Left Hand side",token1);
   dap_info_ptr->dap_error = ErrCreateDB;
   return 0;
}
```

```
if (!(entity_search(token2,db_node->first_entity)))
{
   ERROR("Illegal Definition format. NO entity name on Left Hand side",token2);
   dap_info_ptr->dap_error = ErrCreateDB;
   return 0;
}

if (attrib_search(token1,token2,db_node->first_entity))
{
   ERROR("Illegal Definition format. Attribute name used",token1);
   dap_info_ptr->dap_error = ErrCreateDB;
   return 0;
}/*Now get the right hand side of the equation*/


strcpy(string2,input);
strtok(string2,"=");
if (!(token3 = strtok(NULL," ")))/*Means nothing after the equal sign*/
{
  ERROR("Illegal Definition format. No Right Hand side",input);
  dap_info_ptr->dap_error = ErrCreateDB;
  return 0;
}


if (!strcmp(token3,"INVERSE"))/*We may have got an alias*/
{

  if (!(token3 = strtok(NULL," "))|| strcmp(token3,"OF"))/*Means nothing after Inverse*/
  {
    ERROR("Illegal Definition format. Incorrect info after Inverse",input);
   dap_info_ptr->dap_error = ErrCreateDB;
    return 0;
  }/*Up to here it looks pretty good for an alias*/


  if (!(substring = strtok(NULL,"=")))/*Best not be another equal sign*/
  {
  ERROR("Illegal Definition format. No/Incorrect Info after Inverse Of",input);
  dap_info_ptr->dap_error = ErrCreateDB;
  return 0;
  }
```

```
if (!(strchr(substring,')')))/* No Right Parens*/
{
   ERROR("Illegal Definition format. Improper Right Hand side",substring);
   dap_info_ptr->dap_error = ErrCreateDB;
   return 0;
}

if (strstr(substring," ("))/* Case one or two*/
{
  token3 = strtok(substring," ");
  token4 = strtok(NULL," ");
  if (strcmp(token4,"("))/*Means Left Parens by itself*/
  {
    token4++;
  }
  else
  {
                token4 = strtok(NULL," ");
  }
}
else/*Case three or four*/
{
  token3 = strtok(substring," (");
  token4 = strtok(NULL," (");
  if (strstr(token4,")"))/*It had best be the last character*/
  {
                token4[strlen(token4)-1]='\0';/*Get rid of the last character*/
  }
}/*Should be down to good tokens only otherwise token will be messed up*/

if (strtok(NULL," )"))/*Means more after right parens*/
{
   ERROR("Illegal Definition format. Improper Right Hand side",substring);
   dap_info_ptr->dap_error = ErrCreateDB;
   return 0;
}


if (!(entity_search(token4,db_node->first_entity)))
{
   ERROR("Illegal Definition format. NO entity name on right Hand side",token4);
   dap_info_ptr->dap_error = ErrCreateDB;
```

```
      return 0;
    }

    if (!(attrib_search(token3,token4,db_node->first_entity)))/*This had better already
exist*/
    {
                ERROR("Illegal Definition format. No attribute on Right Hand
side",token3);
       dap_info_ptr->dap_error = ErrCreateDB;
       return 0;
    }
    add_new_alias(db_node,token1,token3,token2,token4);
    add_new_attrib(db_node,token2,token1,'A',token4,token3);
  }

  else /*otherwise should be a composite function or a datatype or an entity and already in
token3*/
  {
    if (strstr(token3,"("))
    {
      make_composite(db_node,token1,token2,strtok(token3,"("),
       strtok(NULL,"("),strtok(NULL,")"));
      return 1;
    }

    if (data_type_search(token3))/*Proper definition of basic attribute*/
    {

                add_new_attrib(db_node, token2, token1, token3[0], token3);
    }
    else
    {

                add_new_attrib(db_node, token2, token1, 'E', token3);
                if (entity_search(token3,db_node->first_entity))/*Pointer for entity*/
                {
        sprintf(temp,"%s_%s_%s",token1,token2,token3);
                   add_new_entity(db_node,temp);
        add_new_attrib(db_node,temp,ID(token2),'I', "INTEGER");
                   add_new_attrib(db_node,temp,ID(token3),'I', "INTEGER");

#ifdef EnExFlag
  printf ("Exit ddl_parse\n");
```

```
  fflush (stdout);
#endif

        return 1;

      }
                else
                {
                  ERROR("Illegal Definition format. Improper expression on Right Hand
side",input);
        dap_info_ptr->dap_error = ErrCreateDB;
        return 0;
                }
    }
  }
 }
}
```

```
/*

 * File Name: dap_general.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/dap_general.c
 * This file contains general utility procedures for operations in the
  Daplex Interface.+
 *
 */

#include <stdio.h>  /* printf found here */
#ifndef DAPLEX_INFO
#include <dap_info.h>
#include <licommdata.h>
#include "flags.def"
#endif

#define RESERVE_LENGTH 10
#define RESERVE_NO 24
#define DATA_TYPE_NO 4

char RESERVE_LIST [RESERVE_NO][RESERVE_LENGTH]=
/* List of reserve words*/
   {"FLOAT"
   ,"INTEGER"
   ,"STRING"
   ,"CHARACTER"
   ,"AND"
   ,"OR"
   ,"DECLARE"
   ,"DEFINE"
   ,"ENTITY"
   ,"INVERSE"
   ,"OF"
   ,"LET"
   ,"FOR"
   ,"A"
   ,"NEW"
   ,"BEGIN"
   ,"END"
   ,"LET"
   ,"INCLUDE"
   ,"EXCLUDE"
   ,"RETRIEVE"
```

```
 ,"SUCH
 ,"THAT"
 ,"WHERE"};

                        ERROR(error_msg, token)

/* Error handler for outputing messages to a file*/
 char * error_msg;
 char * token;
 {
 /*file error_msg must be opened before this function is used */
 printf("\n%s %s\n", error_msg, token);
 }

/* Determines if the token is a reserved token */
reserve_search(token)
 char * token;
 {
  int i;

  for (i=0;i<RESERVE_NO;i++)
  {
    if (!strcmp(token,RESERVE_LIST[i]))
    {
      return 1;
    }
  }
  return 0;
}

                        data_type_search(token)

/* Checks to see if the token is a standard datatype */
 char *token;
 {
  int i;

  for (i=0;i<DATA_TYPE_NO;i++)
  {
    if (!strcmp(token,RESERVE_LIST[i]))
    {
      return 1;
    }
```

```
 }
 return 0;
}
```

abdl_form(string)

```
/* Put a string into the ABDL attribute value format */
 char string[];
{
 int i,j;

 string[0]=toupper(string[0]);
 j=strlen(string);
 for (i=1;i<j;i++)
  string[i]=tolower(string[i]);
 return;
}
```

char* get_addr(db_node,type,entity,attribute)

```
/* Gets the ABDL formatted address of whatever */
  dap_dbid_node* db_node;
  char type;
  char* entity;
  char* attribute;
  {
    dap_db_entity_node* entity_node;
    dap_db_attrib_node* attrib_node;

    entity_node = db_node->first_entity;
    while (entity_node && strcmp(entity_node->dap_entity_name,entity))
      entity_node = entity_node->next_entity;
    switch (type)
    {
      case 'e':
      case 'E':
      if (entity_node)
      {
        return entity_node->dap_entity_addr;
      }
      else
      {
```

```
        return NOT_FOUND;
      }
      break;
      case 'a':
      case 'A':
      if (!entity_node)
      {
        return NOT_FOUND;
      }
      else
      {
       attrib_node = entity_node->first_attrib;
       while (attrib_node && strcmp(attrib_node->dap_attrib_name,attribute))
         attrib_node = attrib_node->next_attrib;
       if (attrib_node)
       {
         return attrib_node->dap_attrib_addr;
       }
       else
       {
         return attribute;
       }
      }
      default:
      return entity;
    }
  }
```

char* range_value (db_node, attribute_name, entity_name)

```
/*Returns the range of a given function*/
   struct dap_db_id_node *db_node;
   char *attribute_name;
   char *entity_name;

{
 dap_db_entity_node *entity;
 dap_db_attrib_node *attribute;

#ifdef EnExFlag
  printf ("Enter range_value\n");
  fflush(stdout);
#endif
```

```
 entity = db_node->first_entity;
  while (entity && strcmp(entity->dap_entity_name, entity_name))
  {
    entity = entity->next_entity;
  }

  if (!entity)
  {
 #ifdef EnExFlag
   printf ("Exit2 range_value\n");
   fflush(stdout);
#endif
       return NULL;
  }
  attribute = entity->first_attrib;
  while (attribute && strcmp(attribute->dap_attrib_name, attribute_name))
  {
    attribute = attribute->next_attrib;
  }
  #ifdef EnExFlag
   printf ("Exit range_value\n");
   fflush(stdout);
#endif

  if (!attribute) return NULL;
  return attribute->range;
}
```

```
* File Name: k_cont.c
* Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kc/k_cont.c
 *
*/

#include <stdio.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"

                        f_Kernel_Controller(dap_ptr)

struct dap_info *dap_ptr;


 /* This procedure accomplishes the following:                    */
 /* (1) Checks dpi_operation to determine whether we are creating a    */
 /*    database or querying the database or if there are errors.     */
 /* (2) Depending on the dpi_operation the corresponding              */
 /*    procedure is called.                                          */


{
 struct   file_info     *f_ptr;

#ifdef EnExFlag
 printf("Enter f_Kernel_Controller \n");
 fflush(stdout);
#endif

   dap_ptr->dpi_subreq_stat = LASTSUBREQ;

  /*  look at operation to determine what action to take   */

   switch (dap_ptr->dap_operation)
    {
      case CreateDB:   /*case where we are creating a database*/
      /* .d and .t files are already created*/
  f_ptr = &(dap_ptr->dpi_ddl_files->ddli_temp);
  f_ptr->fi_fid = fopen(f_ptr->fi_fname, "r");
  dbl_template(dap_ptr->dpi_curr_db.cdi_dbname);
```

```
  fclose(f_ptr->fi_fid);

#ifdef EnExFlag
 printf("Exit1 f_Kernel_Controller \n");
 fflush(stdout);
#endif

  break;

        case ExecRetReq:
        case ExecRetCReq:
        case ExecInsReq:
        case ExecDelReq:
        case ExecUpdReq:
            dap_req_execute(dap_ptr);
#ifdef EnExFlag
 printf("Exit2 f_Kernel_Controller \n");
 fflush(stdout);
#endif
            break;

        default:     /* This handles any errors */

            /* Error handling code ********/
            printf("Error - Unknown operation type in Kc.\n");

#ifdef EnExFlag
 printf("Exit3 f_Kernel_Controller \n");
 fflush(stdout);
#endif
            break;
      }  /* end switch */

}  /* end procedure f_Kernel_Controller */
```

```
/*

 * File Name: kfs.c
 *  Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kfs/kfs.c
 * This process is to receive the requested output from the back-
 * ends and output it in a Functional/DAPLEX type form.
 */

#include <stdio.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"


                        f_kernel_formatting_system(dap_ptr)

struct dap_info   *dap_ptr;

{
 int   done, msg_type, number_of_answers;
 int   OddMark = TRUE;
 char   *response, function[InputCols];
 struct temp_str_info* temp;

#ifdef EnExFlag
 printf("Enter f_kernel_formatting_system\n");
#endif

 response = dap_ptr->dpi_kfs_data.kfsi_dap.kdi_response;
 ++response;   /* skip '[' character in response */

 get_header(dap_ptr->dap_query,function);
 temp=dap_ptr->dap_query;
 while (temp!=NULL)
 /*Set the headers at 25 characters apart*/
 {
   printf("%-25.25s",temp->tsi_str);
   number_of_answers++;
   strtok(temp->tsi_str,"(");
   temp=temp->tsi_next;
 }

 /*Responses look like [string\0string\0string\0 ... ] such that
attribute values are separated by attribute names. There may also
```

```
be other undesireable information which must be cleaned.*/

  while(*response != CSignal && strcmp(function,response))
  {
    response+=(strlen(response)+1);
  }
  temp=dap_ptr->dap_query;
  /*Everyother word is an attribute name & CSignal is the end*/
  while(*response != CSignal)
  {
    if (!strcmp(response,function))
      printf("\n");
     if (OddMark)
 /* Then it is an attribute name */
      {
       if (check_list(temp,response))
 /*Then it is one of the attribute names wanted so just skip the name */
        {
         response+=(strlen(response)+1);
         OddMark = FALSE;
        }
       else
 /* Else we skip the attribute name AND value we don't want*/
      {
         response+=(strlen(response)+1);
         response+=(strlen(response)+1);
        }
      }
     else
 /* Then it is an acceptable attribute value to print */
      {
       printf("%-25.25s",response);
       response+=(strlen(response)+1);
       OddMark = TRUE;
      }
   }
   printf("\n");

/* Clean up the linked list of headers */
   while (dap_ptr->dap_query)
   {
     temp = dap_ptr->dap_query;
     dap_ptr->dap_query = temp->tsi_next;
```

```
    free(temp->tsi_str);
    free(temp);
  }
#ifdef EnExFlag
 printf("Exit f_kernel_formatting_system\n");
#endif
  } /* end f_kernel_formatting_system */
```

check_list(list, word)

```
/* Sees if a word is in a linked list */
struct temp_str_info *list;
char *word;
{

 while (list)
 {
   if (!strcmp(list->tsi_str,word)) return TRUE;
   list = list->tsi_next;
 }
 return FALSE;
}
```

get_header(query,function)

```
/* Retrieves the first header's attribute name */
struct temp_str_info* query;
char* function;
{
 char* temp;

#ifdef EnExFlag
 printf("Enter get_header\n");
#endif

 temp = (char*)malloc(sizeof(char)*(strlen(query->tsi_str)+1));
 strcpy(temp,query->tsi_str);
 strcpy(function,strtok(temp,"("));

#ifdef EnExFlag
 printf("Exit get_header\n");
#endif
}
```

```
/*

 * File Name: kms.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/kms.c
 * This file contains procedures for operations redirection within the
   KMS of the Daplex Interface.
 *
 */

#include <stdio.h>
#include <licommdata.h>
#include <dap_info.h>
#include <string.h>
#include "flags.def"


                    translate_requst(dap_info_ptr, f_curr_req_ptr)

  struct dap_info  *dap_info_ptr;
  struct req_info  *f_curr_req_ptr;

{

int i;
char temp_str[InputCols+1];

#ifdef EnExFlag
   printf("Enter translate_request\n");
   printf("dap request = %s\n", f_curr_req_ptr->ri_dap_req->dpri_req);
#endif

/* Get rid of any junk at the start */
for (i=0;i<strlen(f_curr_req_ptr->ri_dap_req->dpri_req) &&
   f_curr_req_ptr->ri_dap_req->dpri_req[i]==' ';i++) ;
  strncpy(temp_str,(f_curr_req_ptr->ri_dap_req->dpri_req)+i,InputCols);
  temp_str[InputCols] = '\0';
  strtok(temp_str," ");

/* What operation is being requested */
/* If it says DELETE */
  if (!strcmp(temp_str,"DELETE"))
  {

   do_DELETE(dap_info_ptr,f_curr_req_ptr->ri_dap_req->dpri_in_req);
```

```
#ifdef EnExFlag
    printf("Exit translate_request after delete\n");
#endif

  return;
  }
  else
  {
   /* If it says DEFINE or DECLARE */
    if ((!strcmp(temp_str,"DEFINE")) ||
        (!strcmp(temp_str,"DECLARE")))
    {

      ddl_parse(f_curr_req_ptr->ri_dap_req->dpri_req,dap_info_ptr);

#ifdef EnExFlag
    printf("Exit translate_request after create\n");
#endif

      return;
     }
  }

  /* Otherwise it is a normal transaction*/
  /*  Check to ensure the right format for the backend!!!*/
  dap_to_abdl(dap_info_ptr,f_curr_req_ptr->ri_dap_req->dpri_in_req);

#ifdef EnExFlag
  printf("Exit translate_request\n");
#endif

return;

} /* end translate_request */
```

f_abdl_cleanup(dap_info_ptr)

```
/* This function ensures we free up any structures created in the query
  processing */
struct dap_info  *dap_info_ptr;

{
```

```
#ifdef EnExFlag
 printf("Enter f_abdl_cleanup\n");
#endif

 dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req =
       dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req;

 while (dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req)
  {
   dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req =
    dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_next_req;

   free(dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_req);

   free(dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req);

   dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req =
            dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req;
  }

 free(dap_info_ptr->dpi_abdl_tran);

#ifdef EnExFlag
 printf("Exit f_abdl_cleanup\n");
#endif

} /* end f_abdl_cleanup */
```

do_DELETE(dap_info_ptr,query_ptr)

```
/* This procedure does the lazy work for the Delete transaction */

struct dap_info *dap_info_ptr;
struct temp_str_info *query_ptr;
{
char *temp1;
char temp_str[ONE_K], temp2[ONE_K];
dap_dbid_node *node_ptr;
dap_db_entity_node *entity;
int found =FALSE;

#ifdef EnExFlag
  printf("Enter do_DELETE\n");
```

```
#endif

 node_ptr = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;
 entity = node_ptr->first_entity;

 strcpy(temp2,query_ptr->tsi_str);
 strtok(temp2," ");
 temp1=strtok(NULL," ");
 to_caps(temp1);

 while(entity && !found)
 {
/* To use query.c it must have a field to retrieve. Use the ID pointer*/
   if (!strcmp(entity->dap_entity_name,temp1))
   {
     sprintf(temp_str,"RETRIEVE %s(%s)%s",
   ID(temp1),temp1,(query_ptr->tsi_str)+(strlen(temp1)+8));
     found=TRUE;
   }
   entity = entity->next_entity;
 }
 if (!found)
 {
   ERROR("Entity not found",temp1);
   return 0;
 }

 strcpy(temp2,query_ptr->tsi_str);
 strcpy(query_ptr->tsi_str,temp_str);

 dap_to_abdl(dap_info_ptr,query_ptr);
 strcpy(query_ptr->tsi_str,temp2);

/* Replace the word RETRIEVE with DELETE */
 dap_info_ptr->dap_operation = ExecDelReq;
 sprintf(temp_str,"[DELETE%s",(dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req-
>ari_req)+9);
 strcpy(dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req->ari_req,temp_str);

/* Need to end the request before the retrived field names
  This hould occur right after '))'. Can only delete one
  instance at a time */
 temp1 = strstr(dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req->ari_req,"))");
```

```
 if (temp1 != NULL)
 {
   temp1[2] = ']';
   temp1[3] = '\0';
 }

#ifdef EnExFlag
printf("HELP the request is %s\n",dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req-
>ari_req);
  printf("Exit do_DELETE\n");
#endif

}
```

```
/*

* File Name: lil.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Lil/lil.c
 * This file contains procedures which control the menu selection
  routines for the user, accept inputs and redirect required action
  else where
 *
 */

#include <stdio.h>
#include <licommdata.h>
#include <lil.dcl>
#include <lil.ext>
#include <dap.dcl>
#include <dap.ext>
#include <dap_info.h>
#include "flags.def"


                        f_language_interface_layer()

  /* This proc allows the user to interface with the system. */
  /* Input and output: user DAPLEX*/

{
  int    num;
  int    stop;      /* boolean flag */


#ifdef EnExFlag
  printf ("Enter f_language_interface_layer\n");
  fflush(stdout);
#endif

 dap_init();

 /* initialize several ptrs to different parts of the user structure
   for later use */

 dap_info_ptr  = &(cuser_dap_ptr->ui_li_type.li_dap);/*Which should be NULL*/
 tran_info_ptr = &(dap_info_ptr->dpi_dml_tran);
 first_req_ptr = &(tran_info_ptr->ti_first_req);
```

```
 curr_req_ptr = &(tran_info_ptr->ti_curr_req);

/* the followings are inserted for testing build_ddl_files only */
/*
 dap_info_ptr->dpi_curr_db.cdi_db.dn_fun = dbs_dap_head_ptr.dn_fun;
 f_build_ddl_files();
*/
/* end test code */

 stop = FALSE;
 while (stop == FALSE)
  {
   /* allow user choice of several processing operations */
   printf ("\nEnter type of operation desired\n");
   printf ("\t(l) - load new database\n");
   printf ("\t(p) - process existing database\n");
   printf ("\t(x) - return to the MLDS/MDBS system menu\n");
   dap_info_ptr->dap_answer = get_ans(&num);

   switch (dap_info_ptr->dap_answer)
    {
     case 'l': /* user desires to load a new database */
           f_load_new(dap_info_ptr);
           break;
     case 'p': /* user desires to process an existing database */
           f_process_old(dap_info_ptr);
           break;
     case 'x': /* user desires to exit to the operating system */
           /* database schemas must be saved back to files */
           /* and also associated memory must be freed up
           f_save_catalogs();*/
           stop = TRUE;
           break;
     default:  /* user did not select a valid choice from the menu */
           printf ("\nError - invalid operation selected\n");
           printf ("Please pick again\n");
           break;
    } /* end switch */

  /* return to main menu */

  } /* end while */
```

```
#ifdef EnExFlag
  printf ("Exit f_language_interface_layer\n");
  fflush(stdout);
#endif
} /* end f_language_interface_layer */
```

f_load_new(dap_info_ptr)

```
struct dap_info  *dap_info_ptr;

 /*  This proc accomplishes the following:                    */
 /*  (1) determines if the new database name already exists, either  */
 /*     in the schema linked list or as a '.DBname.cat' file.      */
 /*  (2) adds a new header node to the list of schemas,          */
 /*  (3) determines the user input mode (file/terminal),         */
 /*  (4) reads the user input and forwards it to the parser, and    */
 /*  (5) calls the routine that builds the template/descriptor files */

{
   int    num;
   int    more_input;
   char   filler[MaxPathLen+FNLength+2];
   struct ddl_info *ddl_info_alloc();
   dap_dbid_node  *new_ptr,
                  *db_ptr,
                  *dap_dbid_node_alloc();
   FILE   *cat_fd;

#ifdef EnExFlag
  printf ("Enter f_load_new\n");
  fflush(stdout);
#endif

 dap_info_ptr->dap_operation = CreateDB;

   /* prompt user for name of new database */
   printf ("[7;7m\nEnter name of database ---->[0;0m ");
   readstr (stdin, dap_info_ptr->dpi_curr_db.cdi_dbname);
   to_caps (dap_info_ptr->dpi_curr_db.cdi_dbname);

   /* See where this comes from in the DAP unit*/
```

```
  db_ptr = dbs_dap_head_ptr.dn_dap;/*I don't know what this does but hopefully points
to a NULL space Could have had data from before*/

  while (db_ptr)
  {
   /* determine if new database name already exists */
   /* by traversing list of entity-oriented db schemas   */
   if (!(strcmp(db_ptr->dap_db_name,
            dap_info_ptr->dpi_curr_db.cdi_dbname)))
    {
     printf ("\nError - db name already exists\n");
     printf ("[7;7mPlease reenter db name ---->[0;0m ");
     readstr (stdin, dap_info_ptr->dpi_curr_db.cdi_dbname);
                to_caps (dap_info_ptr->dpi_curr_db.cdi_dbname);
     db_ptr = dbs_dap_head_ptr.dn_dap;/*Ensures we start at the top of the list to look*/
    } /* end if */
   else
     /* increment to next database */
     db_ptr = db_ptr->next_db;
  } /* end while */

  dap_info_ptr->dpi_ddl_files = ddl_info_alloc();
  strcpy(DDESCFname, dap_info_ptr->dpi_curr_db.cdi_dbname);
  strcat(DDESCFname, ".d");
  strcpy(DTEMPFname, dap_info_ptr->dpi_curr_db.cdi_dbname);
  strcat(DTEMPFname, ".t");

/* continue - user input a valid 'new' database name */
/* add new header node to the list of schemas and fill-in db name */
/* and init relevant user stucture ptrs */

new_ptr = dap_dbid_node_alloc();


/*  Ensure all our constructs are ready to go*/
strcpy (new_ptr->dap_db_name, dap_info_ptr->dpi_curr_db.cdi_dbname);
dap_info_ptr->dpi_curr_db.cdi_dbtype = DAP;
new_ptr->number_of_entitys = 0;
new_ptr->first_entity = NULL;
new_ptr->object_counter = 0;
new_ptr->first_alias = NULL;
new_ptr->number_of_aliases = 0;
new_ptr->next_db = dbs_dap_head_ptr.dn_dap;
```

```
dbs_dap_head_ptr.dn_dap = new_ptr;
dap_info_ptr->dpi_curr_db.cdi_db.dn_dap = new_ptr;

/* check for user's mode of input */
more_input = TRUE;
while (more_input)
 {
  /* determine user's mode of input */
  printf ("\nEnter mode of input desired\n");
  printf ("\t(f) - read in a group of creates from a file\n");
  printf ("\t(t) - read in creates from the terminal\n");
  printf ("\t(x) - return to the main menu\n");
  dap_info_ptr->dap_answer = get_ans(&num);

  switch (dap_info_ptr->dap_answer)
   {
    case 'f': /* user input is from a file */
          f_read_transaction_file(dap_info_ptr);
/* At this point we should have a pointer to our input file*/
          if (dap_info_ptr->dap_error != ErrReadFile)
           {
            /* file contains transactions */

            read_a_file(dap_info_ptr);
                                        fclose(dap_info_ptr->dpi_file.fi_fname);
/* Make sure we are looking in the correct area */
            strcpy(dap_info_ptr->dpi_ddl_files->ddli_temp.fi_fname,
                          add_path(filler,DTEMPFname));
            strcpy(dap_info_ptr->dpi_ddl_files->ddli_desc.fi_fname,
                          add_path(filler,DDESCFname));
            f_Kernel_Controller(dap_info_ptr);
            more_input = FALSE;
           } /* end if */
          break;
     case 't': /* user input is from the terminal */
          f_read_terminal(dap_info_ptr);
                        if (dap_info_ptr->dap_error != ErrReadFile)
                         {
                                /* user input transactions */

            read_a_file(dap_info_ptr);
/* Make sure we are looking in the correct area */
            strcpy(dap_info_ptr->dpi_ddl_files->ddli_temp.fi_fname,
```

```
                                add_path(filler,DTEMPFname));
                strcpy(dap_info_ptr->dpi_ddl_files->ddli_desc.fi_fname,
                                add_path(filler,DDESCFname));
                f_Kernel_Controller(dap_info_ptr);
                                        more_input = FALSE;
                                } /* end if */
            break;

        case 'x': /* exit back to LIL */
            more_input = FALSE;
            break;
        default:  /* user did not select a valid choice from the menu */
            printf ("\nError - invalid input mode selected\n");
            printf ("Please pick again\n");
            break;
       } /* end switch */

    if (dap_info_ptr->dap_error == ErrCreateDB)
            more_input = FALSE; /* errors in creates so exit this loop */
    dap_info_ptr->dap_error = NOErr;
    } /* end while */

#ifdef EnExFlag
  printf ("Exit f_load_new\n");
#endif

} /* end f_load_new */



                        f_process_old(dap_info_ptr)

/* This procedure is for using predefined databases */
struct dap_info  *dap_info_ptr;
{
  FILE              *cat_fd;
  int               num, stop;
  int               found = FALSE;
  int               exist = FALSE;
  char              file_name[FNLength + 3],
                   *record_file;
  char              filler[MaxPathLen+FNLength+2];
  dap_dbid_node        *db_ptr;
  struct ddl_info      *ddl_info_alloc();
```

```
  struct tran_info      *tran_info_alloc();
  struct ab_req_info    *ab_req_info_alloc();

#ifdef EnExFlag
  printf ("Enter f_process_old\n");
#endif

  record_file = (char*)malloc(sizeof(char)*100);
  /* create the template and descriptor structure if it doesn't already exist*/
  if (dap_info_ptr->dpi_ddl_files == NULL)
    dap_info_ptr->dpi_ddl_files = ddl_info_alloc();

  /* prompt user for name of existing database */
  printf ("[7;7m\nEnter name of database ---->[0;0m ");
  readstr (stdin, dap_info_ptr->dpi_curr_db.cdi_dbname);
  to_caps (dap_info_ptr->dpi_curr_db.cdi_dbname);
  db_ptr = dbs_dap_head_ptr.dn_dap;

/* These files had best already exist */
  strcpy(DDESCFname, dap_info_ptr->dpi_curr_db.cdi_dbname);
  strcat(DDESCFname, ".d");
  strcpy(DTEMPFname, dap_info_ptr->dpi_curr_db.cdi_dbname);
  strcat(DTEMPFname, ".t");

  while (db_ptr)
    {
     /* determine if given database name already exists */
     /* by traversing list of db schemas   */
     if ((strcmp(db_ptr->dap_db_name,
            dap_info_ptr->dpi_curr_db.cdi_dbname))== 0)
       {
                found = TRUE;
                dap_info_ptr->dpi_curr_db.cdi_db.dn_dap = db_ptr;
                dap_info_ptr->dpi_curr_db.cdi_dbtype = DAP;
        /* assuming it has already its own temp & desc files, otherwise
          build them need to be called.                 */
                strcpy(dap_info_ptr->dpi_ddl_files->ddli_temp.fi_fname,
                                add_path(filler,DTEMPFname));
                strcpy(dap_info_ptr->dpi_ddl_files->ddli_desc.fi_fname,
                                add_path(filler,DDESCFname));
                break;
       } /* end if */
     else
```

```
      /* increment to next database */
      db_ptr = db_ptr->next_db;
  } /* end while */

if (!found)
 {
  strcpy(DDBCat, ".");
  strcat(DDBCat, dap_info_ptr->dpi_curr_db.cdi_dbname);
  strcat(DDBCat, ".t");
  if (cat_fd = fopen(DDBCat, "r"))
   {
    exist = TRUE;
    printf("\nI have already a database .d and .t files with\n");
    printf("the same name from previous sessions.\n");
    printf("\nDo you want to use them(y/n)?");
    dap_info_ptr->dap_answer = get_ans(&num);

    if (dap_info_ptr->dap_answer == 'y')
     {
      found = TRUE;
      fclose(cat_fd);
      dap_info_ptr->dap_operation = CreateDB;
      /*Need to create a function to make the DB from the .d and .t files*/
      f_load_catalog(dap_info_ptr);
                dap_info_ptr->dpi_curr_db.cdi_db.dn_dap = dbs_dap_head_ptr.dn_d[illegible];
                dap_info_ptr->dpi_curr_db.cdi_dbtype = DAP;

                /* assume it has read its own temp & desc files
                  else other problems */

                strcpy(dap_info_ptr->dpi_ddl_files->ddli_temp.fi_fname,
                        add_path(filler,DTEMPFname));
                strcpy(dap_info_ptr->dpi_ddl_files->ddli_desc.fi_fname,
                        add_path(filler,DDESCFname));

      f_Kernel_Controller(dap_info_ptr);
     }
    else
      fclose(cat_fd);
   } /* end if cat_fd */
  } /* end if !found */

if (found)
```

```
  {
   /* TI in MBDS - need to set the user and database id for the user */
/*   DBL_S$Use(dap_info_ptr->dpi_curr_db.cdi_dbname);
   TI_S$AssignDB(cuser_obj_ptr->ui_uid, dap_info_ptr->dpi_curr_db.cdi_dbname);
*/
   stop = FALSE;
   while (!(stop))
   {
    printf("\nEnter your choice\n");
    printf("\t(d) - display schema\n");
    printf("\t(m) - mass load from a data file\n");
    printf("\t(s) - send data to a file for mass load\n");
    printf("\t(f) - read in a group of queries from a file\n");
    printf("\t(t) - read in queries from the terminal\n");
    printf("\t(x) - return to previous menu\n");
    dap_info_ptr->dap_answer = get_ans(&num);

    switch (dap_info_ptr->dap_answer)
    {
     case 'd':
          f_display_schema(dap_info_ptr);
          break;

     case 'm':
          f_read_transaction_file(dap_info_ptr);
          dap_mass_load(dap_info_ptr);
          break;

     case 's':
          f_read_reciept_file(dap_info_ptr);
          dap_mass_dump(dap_info_ptr);
          break;

     case 'f':
          f_read_transaction_file(dap_info_ptr);
          f_read_file(dap_info_ptr,f_tran_info_ptr);
          f_queries_to_KMS(dap_info_ptr,f_tran_info_ptr);
          f_free_requests(f_tran_info_ptr);
          break;
     case 't':
          f_read_terminal(dap_info_ptr);
          f_read_file(dap_info_ptr,f_tran_info_ptr);
          f_queries_to_KMS(dap_info_ptr,f_tran_info_ptr);
```

```
            f_free_requests(f_tran_info_ptr);
            break;
        case 'x':
            stop = TRUE;
            break;
        default:
            printf("\nError invalid operation selected\n");
            printf("Please pick again\n");
            break;
       } /* end switch */
     } /* end while !stop */

   } /* end if found */
  else
    if (!exist)
      printf ("Error - db name does not exist\n");

#ifdef EnExFlag
  printf ("Exit f_process_old\n");
#endif
} /* end f_process_old */



                        f_display_schema(dap_info_ptr)

/* Procedure for displaying a functional database schema */
/* It places it in a file and does a cat |more on the file */
struct dap_info  *dap_info_ptr;
{
  dap_dbid_node   *db_ptr;
  dap_db_entity_node *entity,*tmpent;
  dap_db_attrib_node *attrib;
  int          ind;
  FILE          *fid;
  char          tmpstr[200];
  char          *temp;
  temp=(char*)malloc(sizeof(char)*SNLength);
  db_ptr = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;
  fid = fopen("schema","w");
  fprintf(fid,"\nDatabase Name : %s\n",db_ptr->dap_db_name);

  entity = db_ptr->first_entity;
/* For each ENTITY real entity */
```

```
  while (entity)
   {
    ind = FALSE;
    attrib = entity->first_attrib;
/* For each attribute that's not a QQ */
    while (attrib)
    {
      if (strcmp(attrib->dap_attrib_name,"TEMP") && !strstr(attrib-
>dap_attrib_name,"QQ"))
      {
        if(!ind)
        {
          ind=TRUE;
          fprintf(fid,"\nEntity: %s\n",entity->dap_entity_name);
        }
      fprintf(fid,"%s(%s) = %s\n",attrib->dap_attrib_name,entity->dap_entity_name,
attrib->range);
     }/*End of if not TEMP or name with #*/
     attrib = attrib->next_attrib;
   }/*end of Attribute search*/
   entity = entity->next_entity;
  }/* end of Entity scarch*/

  fclose(fid);
  system("more schema");
  system("rm schema");

} /* end f_display_schema */



                                attr_type(entity,string,temp)

/* Facilitates the search for an entity during display schema*/
   dap_db_entity_node *entity;
   char *string;
   char *temp;
  {
   char *temp2;
   int  length;

#ifdef EnExFlag
 printf("Enter attr_type \n");
```

```
 fflush(stdout);
#endif

    length = strlen(string);
   while (entity)
   {
     if(!strncmp(entity->dap_entity_name,string,length))
     {
       temp2= entity->dap_entity_name;
       temp2+=length;
       strcpy(temp,temp2);

#ifdef EnExFlag
 printf("Exit attr_type good\n");
 fflush(stdout);
#endif
         return;
      }/*End if strncmp */

      entity = entity->next_entity;
   }/*End While entity*/

#ifdef EnExFlag
 printf("Exit attr_type bad\n");
 fflush(stdout);
#endif

   strcpy(temp,"Unable to Locate");
return;
 }/* End of attr_type*/

                        find_alias(alias,entity,attrib,temp)

/* Gets alias information for output or other usage */
 dap_db_alias_node *alias;
 char *entity, *attrib, *temp;
 {

#ifdef EnExFlag
 printf("Enter find_alias \n");
 fflush(stdout);
#endif
```

```
while (alias)
{
  if (!strcmp(entity,alias->dap_alias_ent1) &&
     !strcmp(attrib,alias->dap_alias_new_name))
  {
   strcpy(temp,alias->dap_alias_ent2);

#ifdef EnExFlag
 printf("Exit find_alias good\n");
 fflush(stdout);
#endif
    return;
  }
  alias = alias->next_alias;
}
strcpy(temp,"Unable to Locate");

#ifdef EnExFlag
 printf("Exit find_alias bad\n");
 fflush(stdout);
#endif
return;
}
```

```
/*

 * File Name: lilcommon.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/DAP/Lil/lilcommon.c
 * This file contains the procedures that are used to handle the
  transactions that are entered by the user.
 */

#include <stdio.h>
#include <ctype.h>
#include <strings.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"


                    f_queries_to_KMS(dap_info_ptr,f_tran_info_ptr)

  struct tran_info  *f_tran_info_ptr;
  struct dap_info   *dap_info_ptr;

                /* This routine causes the queries to be listed on the screen. */
                /* The selection menu is then displayed allowing any of the    */
                /* queries to be executed.                                      */


{
      struct req_info  *trn_info;
                int proceed:    /* boolean flag */
                int num;

#ifdef EnExFlag
   printf ("Enter f_queries_to_KMS\n");
#endif

  num = 0;
  f_list_queries(f_tran_info_ptr);
  proceed = TRUE;
  while (proceed == TRUE)
    {
     printf ("\nPick the number or letter of the action desired\n");
     printf ("\t(num) - execute one of the preceeding queries\n");
     printf ("\t(d)   - redisplay the file of queries\n");
     printf ("\t(x)   - return to the previous menu\n");
```

```
dap_info_ptr->dap_answer = get_ans(&num);

switch (dap_info_ptr->dap_answer)
          {
           case 'n' : /* execute one of the queries */
             if (num > 0 && num <= f_tran_info_ptr->ti_no_req)
                          {
                           f_find_query (num,f_tran_info_ptr);

       /* This is the default value for operation */
       /* If not a retrieve request, this value is reset */
       /* in dap_kernel_mapping_system */

       dap_info_ptr->dap_operation = ExecRetReq;

                          /*Entry to query transaction programs */
       translate_requst(dap_info_ptr,f_tran_info_ptr->ti_curr_req);
       if (dap_info_ptr->dap_operation != ExecNoReq)
         {
                             if (dap_info_ptr->dap_error == NOErr)
          {
                              f_Kernel_Controller(dap_info_ptr);
          }
         else
                              dap_info_ptr->dap_error = NOErr;
         }
       f_abdl_cleanup(dap_info_ptr);
                          } /* end if */
             else
                          {
                           printf ("\nError - the query for the number you ");
       printf ("selected does not exist\n");
                           printf ("Please pick again\n");
                          } /* end else */

    break;
           case 'd' : /* redisplay queries */
    f_list_queries(f_tran_info_ptr);
             break;
           case 'x' : /* exit to mode menu */
             proceed = FALSE;
    f_tran_info_ptr->ti_no_req = 0;
             break;
```

```
                default : /* user did not select a valid choice from the menu */
                  printf ("\nError - invalid option selected\n");
                  printf ("Please pick again\n");
                  break;
      } /* end switch */

    } /* end while */

#ifdef EnExFlag
  printf ("Exit f_queries_to_KMS\n");
#endif

} /* end f_queries_to_KMS */



                        f_list_queries(f_tran_info_ptr)

 struct tran_info  *f_tran_info_ptr;

                /* This routine actually prints the query list to the screen */

{
                struct req_info  *f_curr_req_ptr,
                 *f_first_req_ptr;

                struct  temp_str_info *req_ptr;  /* ptr to a line of a query */
                int             i;              /* the number of the query  */
                int     first_line;         /* boolean flag          */
     FILE    *qry_fid;           /* file id for query print file */


#ifdef EnExFlag
  printf ("Enter f_list_queries\n");
#endif

 f_curr_req_ptr  = &(f_tran_info_ptr->ti_curr_req);
 f_first_req_ptr = &(f_tran_info_ptr->ti_first_req);

 i = 1;
 f_curr_req_ptr->ri_dap_req = f_first_req_ptr->ri_dap_req;

 qry_fid = fopen(".qry_file", "w");
```

```
/* loop and print the queries until there are no more */
while (f_curr_req_ptr->ri_dap_req)
 {
  req_ptr = f_curr_req_ptr->ri_dap_req->dpri_in_req;
  first_line = TRUE;
  fprintf (qry_fid,"\n");
  while (req_ptr)
              {
    if (first_line)
                {
                 /* first line of a query so print the number of it first */
                 fprintf (qry_fid, "\t%d\t %s\n", i, req_ptr->tsi_str);
                 first_line = FALSE;
                } /* end if */
    else
                 fprintf (qry_fid, "\t\t %s\n", req_ptr->tsi_str);
               req_ptr = req_ptr->tsi_next;
              } /* end while */

  f_curr_req_ptr->ri_dap_req = f_curr_req_ptr->ri_dap_req->dpri_next_req;
  i++;
 } /* end while */

 fclose( qry_fid ); /* close the query file */

 system("more .qry_file"); /* print out the queries */
 system("rm .qry_file");

#ifdef EnExFlag
  printf ("Exit f_list_queries\n");
#endif

} /* end f_list_queries */
```

f_find_query(num,f_tran_info_ptr)

```
int num;        /* specified query to be executed */
struct tran_info  *f_tran_info_ptr;

{
              struct temp_str_info    *temp_str;
```

```
    struct req_info  *f_curr_req_ptr,
                     *f_first_req_ptr;
                    int i;              /* counter */

                    /* This function walks down the query list to the (num)th  */
                    /* query and passes back the ptr to that query */

#ifdef EnExFlag
  printf ("Enter f_find_query\n");
#endif

 f_curr_req_ptr  = &(f_tran_info_ptr->ti_curr_req);
 f_first_req_ptr = &(f_tran_info_ptr->ti_first_req);

 /* set the current ptr to the first ptr */
 f_curr_req_ptr->ri_dap_req = f_first_req_ptr->ri_dap_req;
 for (i = 1; i < num; i++)
   f_curr_req_ptr->ri_dap_req = f_curr_req_ptr->ri_dap_req->dpri_next_req;


f_tran_info_ptr->ti_curr_req = *f_curr_req_ptr;
temp_str = f_curr_req_ptr->ri_dap_req->dpri_in_req;
/*printf ("%s \n", f_curr_req_ptr->ri_dap_req->dpri_req );*/
while (temp_str)
{
  printf ("%s \n", temp_str->tsi_str);
  temp_str = temp_str->tsi_next;
}
printf ("\n");

#ifdef EnExFlag
  printf ("Exit f_find_query\n");
#endif

} /* end f_find_query */

                              f_free_requests(f_tran_info_ptr)

struct tran_info *f_tran_info_ptr;

{
  /* This function frees all memory reserved by the transaction list */

  struct temp_str_info *temp_str_ptr;
```

```
  struct dap_req_info *curr_req_ptr;

#ifdef EnExFlag
  printf ("Enter f_free_requests\n");
#endif

 /* set the current ptr to the first ptr */
 curr_req_ptr = f_tran_info_ptr->ti_first_req.ri_dap_req;
 while (curr_req_ptr != NULL)
  {
   curr_req_ptr = curr_req_ptr->dpri_next_req;
   free (f_tran_info_ptr->ti_first_req.ri_dap_req->dpri_req);

   temp_str_ptr = f_tran_info_ptr->ti_first_req.ri_dap_req->dpri_in_req;
   while (temp_str_ptr)
    {
     temp_str_ptr = temp_str_ptr->tsi_next;
     free (f_tran_info_ptr->ti_first_req.ri_dap_req->dpri_in_req);
     f_tran_info_ptr->ti_first_req.ri_dap_req->dpri_in_req = temp_str_ptr;
    } /* end while */

   free (f_tran_info_ptr->ti_first_req.ri_dap_req);
   f_tran_info_ptr->ti_first_req.ri_dap_req = curr_req_ptr;
  } /* end while */

#ifdef EnExFlag
  printf ("Exit f_free_requests\n");
#endif

} /* end f_free_requests */
```

```
/*

 * File Name: load_data.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Lil/load_data.c
 * This file contains procedures for making an input file of mass data

 */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <licommdata.h>
#ifndef DAPLEX_INFO
#include <dap_info.h>
#include "flags.def"
#endif


                        dap_mass_dump(dap_info_ptr)

/* Places backend base data into a file */

 struct dap_info *dap_info_ptr;

{
 dap_db_entity_node *entity;
 dap_db_attrib_node *attribute;
 dap_dbid_node *db_ptr;

 struct temp_str_info  *value,
                       *temp, *temp2,
                       *temp_str_info_alloc();
 int       OddMark = TRUE;
 int       msg_type,
           done,
           ind = 0;
 char      *query;
 char      *response;
 FILE      *load_file;
 /* values for these two depends on the defines in tstint.def (CNTRL/TI) */
 char      request[ONE_K],   /* Added request length */
           err_msg[ONE_K];
 struct    ReqId     rid;  /* Defined in licommdata.h */
```

```
#ifdef EnExFlag
 printf("Enter dap_mass_dump\n");
#endif

 db_ptr = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;
 entity = db_ptr->first_entity;
 value = temp_str_info_alloc();
 temp = value;
/* For each ENTITY get ALL its data */
 while (entity)
 {
/* ABDL query format */
   sprintf(temp->tsi_str,"[RETRIEVE(TEMP=%s)(TEMP",entity->dap_entity_addr);
   attribute = entity->first_attrib;
/* We need EVERY attribute */
   while (attribute)
   {
     if (attribute->dap_attrib_type != 'E' &&
         attribute->dap_attrib_type != 'G' &&
         attribute->dap_attrib_type != 'A')
     {
       strcat(temp->tsi_str,",");
       strcat(temp->tsi_str,attribute->dap_attrib_addr);
     }/*end if attrib_type*/
     attribute = attribute->next_attrib;
   }/*end while attribute*/
   strcat(temp->tsi_str,")]");
   entity = entity->next_entity;
   if (entity)
   {
     temp->tsi_next = temp_str_info_alloc();
     temp = temp->tsi_next;
   }/*end if entity*/
 }/*end while entity*/

 temp = value;
 load_file=dap_info_ptr->dpi_file.fi_fid;
 while (temp)
 {
/* Send it to the backend for retrieval */
/* The rest is a combo of KC and KFS */
 Tl_S$TrafUnit(dap_info_ptr->dpi_curr_db.cdi_dbname,temp->tsi_str);
```

```
response = dap_info_ptr->dpi_kfs_data.kfsi_dap.kdi_response;

done = FALSE;
while (!done)/*Not all responses for the current request have been received*/
 {
  TI_R$Message();/*receive message from MBDS*/

  msg_type = TI_R$Type(); /* get the message type of the received message*/

  switch(msg_type) /* Is the response correct or are there errors? */
  {
    case CH_ReqRes: /* The response is correct */
     TI_R$ReqRes(&rid,response); /* Receive the results */

     done = f_chk_if_last_response(dap_info_ptr); /*Are we done */
     ++response;/*Skip initial [ */
query = response;
while(*response != CSignal)
{
  if (!strcmp(query,response) && ind)
    fprintf(load_file,"\n");
  fprintf(load_file,"%s ",response);
  response+=(strlen(response)+1);
  ind++;
}
    break;

    case ReqsWithErr:
     TI_R$ErrorMessage(request,err_msg);
     TI_ErrRes_output(request,err_msg);
     return 0;
     break;

    default:
     printf("Illegal msg_type = %d received.\n", msg_type);
     break;

 } /* End switch */

} /* End while not done */
temp = temp->tsi_next;
}/*end of while temp loop*/
fprintf(load_file,"\n$$\n");
```

```
 /* Reset the External from MBDS */
 dap_info_ptr->dpi_kc_data = value;
 fclose(load_file);

#ifdef EnExFlag
 printf("Exit dap_mass_dump\n");
#endif
 }
```

```
/*
 * File Name: meta_d_t.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/meta_d_t.c
 * This file contains procedures for the creation of the two metadata
  files, the descriptor and the template files, from the dap_db_node.
 *
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <licommdata.h>
#ifndef DAP_INFO
#include <dap_info.h>
#endif

                        create_t_file(db_ptr)

/* Function that creates the .t and .d files that contain metadata for MDBS */
int
  dap_dbid_node *db_ptr;
  {
  dap_db_entity_node *current_entity_node;
  dap_db_attrib_node *current_attrib_node;
  FILE * t_out;
  FILE * d_out;
  char filler[MaxPathLen+FNLength+3];

  /* open metadata .t file. Format is FILENAME.t */
  d_out = fopen(add_path(filler,DDESCFname), "w");

  /* open metadata .d file. Format is FILENAME.d */
  t_out = fopen(add_path(filler,DTEMPFname), "w");

  /* output database name and number of entities for .t */
  fprintf(t_out,"%s \n", db_ptr->dap_db_name);
  fprintf(t_out, "%d\n", db_ptr->number_of_entitys);

  /* output database name and "TEMP b s" for .d */
  fprintf(d_out,"%s \n", db_ptr->dap_db_name);
  fprintf(d_out,"TEMP b s\n");
```

```
/* traverse the entity linked list */
current_entity_node = db_ptr->first_entity;
while(current_entity_node)
{
 /* for each entity output it's name and number of attributes for .t */
 /* Add Temp as first attribute (also add 1 to attrib count) */
 fprintf(t_out,"%d \n", current_entity_node->number_of_attribs+1);
 strcpy(filler, current_entity_node->dap_entity_addr);
 abdl_form(filler);
 fprintf(t_out,"%s \n", filler);
 fprintf(t_out,"TEMP s\n");
 current_attrib_node = current_entity_node->first_attrib;
  /* for each entity output "! " followed by it's name for .d */
 fprintf(d_out,"! %s \n", filler);

 /* traverse the attribute linked list for current entity */
 while (current_attrib_node)
 {
  /* for each attribute type out it's name and type .t */
  /* Some attributes produce no ouput for .d */
  if (current_attrib_node->dap_attrib_type!='E' &&
      current_attrib_node->dap_attrib_type!='G' &&
      current_attrib_node->dap_attrib_type!='A')
  {
    fprintf(t_out,"%s %c \n",current_attrib_node->dap_attrib_addr.
       tolower(current_attrib_node->dap_attrib_type));
  }
  current_attrib_node = current_attrib_node->next_attrib;
 }
   /* get next entity and repeat loop */
 current_entity_node = current_entity_node->next_entity;
} /* end while */

/* End of file marker for .d is '@' and '$' each on seperate line */
fprintf(d_out,"@\n");
fprintf(d_out,"$\n");

/* close .d and .t output files */
fclose(t_out);
fclose(d_out);
return 1;
}
```

```
/*

 * File Name: mss_load.c
 * Source: /u/mdbs/grcg/CNTRL/TI/LangIF/src/Dap/Kms/mss_load.c
 * This file contains procedures for the loading of base data directly
   from a data file to the MDBS.
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <strings.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"


                    dap_mass_load(dap_info_ptr,file_name)

  struct dap_info  *dap_info_ptr;
  char *file_name;

{
FILE  *mass_load_fid;
char insert[MaxPathLen], temp1[InputCols];
char temp[InputCols],*token1,*token2,*strtok();
char query[ONE_K];
dap_dbid_node  *db_ptr;
dap_db_attrib_node   *attrib;
int counter=0;

#ifdef EnExFlag
  printf("Enter dap_mss_load\n");
#endif

/* Finds the location of the file containing the data */
mass_load_fid = dap_info_ptr->dpi_file.fi_fid;
db_ptr = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;
fgets(temp, InputCols, mass_load_fid);
counter++;
dap_info_ptr->dap_operation = ExecInsReq;

/* Special end of data file marker */
while (!strstr(temp,"$$"))
```

```
{
  strtok(temp,"\n");/*Removes any \n if present*/
  token1=strtok(temp," ");
/*  Places it in the ABDL transaction format */
  sprintf(query,"[INSERT(<%s,%s>",token1,strtok(NULL," "));
  token1=strtok(NULL," ");
  while(token1)
  {
    sprintf(temp1,",<%s,%s>",token1,strtok(NULL," "));
    strcat(query,temp1);
    token1=strtok(NULL," ");
  }
  strcat(query,")]");
/* Sends the data to the backends */
  TI_S$TrafUnit(dap_info_ptr->dpi_curr_db.cdi_dbname,query);
/* Ensures the ABDL is done */
  dap_chk_responses_left(dap_info_ptr);
  TI_finish();
/* An indicator to let users know they've not been forgotten */
  if(!(counter%5)){ printf("%d  ",counter); fflush(stdout);}
  counter++;
  fgets(temp, InputCols, mass_load_fid);
}
/* Reset the object/entity instance counter */
db_ptr->object_counter=counter;
printf("\n");

#ifdef EnExFlag
  printf("Exit dap_mss_load\n");
#endif

}
```

```
/*
 * File Name: node_entry.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/node_entry.c
 * This file contains procedures for creating and adding new nodes to
  the dap_dbid_node structure.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <licommdata.h>
#ifndef DAPLEX_INFO
#include <dap_info.h>
#include "flags.def"
#endif


        void    add_new_attrib(dap_start, entity, name, kind, entity2, name2)

/* Adds a new attribute to an entity */
  struct dap_db_id_node *dap_start;
  char *entity;
  char *name;
  char kind;
  char *entity2;
  char *name2;
  {
  char temp_str[ONE_K];
  dap_db_entity_node *start;
  dap_db_attrib_node *temp_ptr,*dap_db_attrib_node_alloc();

  start = dap_start->first_entity;
  while (strcmp(start->dap_entity_name,entity))
    start=start->next_entity;
  if (kind !='E' && kind !='A')
  {
    (start->number_of_attribs)++;/*Increase the number of attributes*/
  }

  /*Make room for new node and names*/
  temp_ptr = dap_db_attrib_node_alloc();
  temp_ptr->dap_attrib_name = (char*) malloc (sizeof(char)*strlen(name)+1);
  temp_ptr->range = (char*) malloc (sizeof(char)*strlen(entity2)+1);
```

```
  temp_ptr->dap_attrib_addr = (char*) malloc (sizeof(char)*(ANLength+2));

  /*put values into strings*/
  strcpy(temp_ptr->range, entity2);
  strcpy(temp_ptr->dap_attrib_name, name);
/If it is an entity to entity function fix the address*/
  if (kind == 'E')
  {
    sprintf(temp_str,"%s_%s",name,entity);
    abdl_form(temp_str);
    strncpy(temp_ptr->dap_attrib_addr, temp_str,ANLength);
  }
/If it is an alias function fix the address*/
  else if (kind == 'A')
  {
    sprintf(temp_str,"%s_%s",name2,entity2);
    abdl_form(temp_str);
    strncpy(temp_ptr->dap_attrib_addr, temp_str,ANLength);
  }
/* Otherwise it is straight forward. Composites done elsewhere*/
  else strncpy(temp_ptr->dap_attrib_addr, name,ANLength);
  temp_ptr->dap_attrib_addr[ANLength]='\0';
  temp_ptr->dap_attrib_type = kind;

  /*Adjust the pointers*/
  temp_ptr->next_attrib = start->first_attrib;
  start->first_attrib = temp_ptr;

  return;
  }

                          void  add_new_entity(start, name)

  struct dap_db_id_node *start;
  char *name;
  {
  dap_db_entity_node *temp,*dap_entity_node_alloc();
  char* attrib_name;

  (start->number_of_entitys)++;/*Increase the number of entities*/

  /*Make room for new node and names*/
  temp = dap_entity_node_alloc();
```

```
temp->dap_entity_name = (char*) malloc (sizeof(char)*(strlen(name)+1));
temp->dap_entity_addr = (char*) malloc (sizeof(char)*(ANLength+2));
attrib_name = (char*) malloc (sizeof(char)*(strlen(name)+2));

/*put values into strings*/
temp->number_of_attribs = 0;
temp->first_attrib = NULL;
strcpy(temp->dap_entity_name, name);
strncpy(temp->dap_entity_addr, name,ANLength);
temp->dap_entity_addr[ANLength]='\0';
/* Remember it is really an attribute value */
abdl_form(temp->dap_entity_addr);

/*Adjust the pointers*/
temp->next_entity = start->first_entity;
start->first_entity = temp;

/*New entity needs a new identification attribute*/
strcpy(attrib_name, ID(name));
add_new_attrib(start, name, attrib_name, 'I');
free(attrib_name);
return;
}


    void   add_new_alias(start, new_name, old_name, new_entity, old_entity)

struct dap_db_id_node* start;
char *new_name;
char *old_name;
char *new_entity;
char *old_entity;
{
dap_db_alias_node *temp;

(start->number_of_aliases)++;/*Increase the number of aliases*/

/*Make room for new node and names*/
temp = (dap_db_alias_node*) malloc (sizeof(dap_db_alias_node));
temp->dap_alias_new_name = (char*) malloc (sizeof(char)*strlen(new_name)+1);
temp->dap_alias_old_name = (char*) malloc (sizeof(char)*strlen(old_name)+1);
temp->dap_alias_ent1 = (char*) malloc (sizeof(char)*strlen(new_entity)+1);
temp->dap_alias_ent2 = (char*) malloc (sizeof(char)*strlen(old_entity)+1);
```

```
  /*put values into strings*/
  strcpy(temp->dap_alias_new_name, new_name);
  strcpy(temp->dap_alias_old_name, old_name);
  strcpy(temp->dap_alias_ent1, new_entity);
  strcpy(temp->dap_alias_ent2, old_entity);

  /*Adjust the pointers*/
  temp->next_alias = start->first_alias;
  start->first_alias = temp;

  return;
  }

            void  make_composite(db_node, tok1, tok2, tok3, tok4, tok5)

/* composite function attribute nodes have special requirements */
  dap_dbid_node *db_node;
  char *tok1;
  char *tok2;
  char *tok3;
  char *tok4;
  char *tok5;
{
  dap_db_entity_node *entity;
  dap_db_attrib_node *temp, *dap_db_attrib_node_alloc();
  char *range_value(), *range;

#ifdef EnExFlag
  printf ("Enter make_composite\n");
  fflush(stdout);
#endif
  entity = db_node->first_entity;
/* Find the proper entity */
  while (strcmp(entity->dap_entity_name, tok2))
  {
    entity = entity->next_entity;
  }
/* Get ready to add a new attribute to the entity */
  temp = dap_db_attrib_node_alloc();
  temp->dap_attrib_name = (char*)malloc(sizeof(char)*(strlen(tok1)+1));
/* Place the data into the appropriate areas */
  strcpy(temp->dap_attrib_name, tok1);
```

```
  temp->dap_attrib_type = 'G';
  range = range_value(db_node,tok4,tok5);
  temp->range = range_value(db_node,tok3,range);
  temp->dap_attrib_addr = (char*)malloc(sizeof(char)*(InputCols+1));
  sprintf(temp->dap_attrib_addr,"%s(%s) & %s(%s)=%s ",tok3,range,tok4,tok5,range);
/* Readjust the pointers */
  temp->next_attrib = entity->first_attrib;
  entity->first_attrib = temp;
#ifdef EnExFlag
  printf ("Exit make_composite\n");
  fflush(stdout);
#endif

}
```

```
/*

 * File Name: query.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/query.c
 * This program parses and processes the DAPLEX statements FOR and
   RETRIEVE.
*/
#define ONE  1
#define REST 0
#define TRUE  1
#define FALSE 0
#define LET  0
#define SUCH_THAT  1
#define RETRIEVE   2
#define TOKEN_LENGTH 80
#define ELEMENT_NAME_LENGTH 6
#define MAX_LEN 200
#define MAX_LEN_ABDL 200

#include <stdio.h>
#include <strings.h>
#include <licommdata.h>
#include <stdlib.h>
#include "flags.def"

#ifndef DAPLEX_INFO
#include <dap_info.h>
#endif


                                        int is_integer(i)

/* This program checks to see if the passed character value
   represents an integer. If it is an integer a 1 is returned
   otherwise a 0 is returned. */
    char *i;
    {
    int j;

#ifdef EnExFlag
    printf("Into the Integer Function.\n");
    fflush(stdout);
#endif
```

```
 for(j=0;j<strlen(i);j++)
    {
    if(!isdigit(i[j])) return 0;
    }

#ifdef EnExFlag
  printf("Exiting the Integer Function.\n");
  fflush(stdout);
#endif

  return 1;
  }
```

int is_real(r)

```
/* This program checks to see if the passed character value represents
  a real number. If it is a real number then a 1 is returned otherwise
  a 0 is returned.*/
  char *r;
  {
  int j;
  int decimal =0;
  for(j=0;j<strlen(r);j++)

#ifdef EnExFlag
  printf("Into the Real Function.\n");
  fflush(stdout);
#endif

  for(j=0;j<strlen(r);j++)
  for(j=0;j<strlen(r);j++)
  {
    if(!isdigit(r[j])) return 0;
    {
      if(r[j] == '.' && !decimal)
        decimal++;
      else
      {
#ifdef EnExFlag
  printf("Exit the Real Function returning a 0.\n");
  fflush(stdout);
```

```
#endif

        return 0;
       }
     }
   }

#ifdef EnExFlag
   printf("Exit the Real Function returning a 1.\n");
   fflush(stdout);
#endif

   return 1;
   }
```

linked_list_node* add_linked_list(name, value, linked_list_head_ptr)

```
/* This program adds an attribute name and value to the linked list
  pointed to by a pointer passed by the user. These linked lists are
  then used to create compound ABDL RETRIEVES and INSERT statements
  (done by other functions).*/
 char *name;
 char *value;
 linked_list_node *linked_list_head_ptr;
{
 linked_list_node *linked_list_ptr;

#ifdef EnExFlag
   printf("Into the Insert linked list Function.\n");
   fflush(stdout);
#endif

 /* create space for new linked_list node */
 linked_list_ptr = (linked_list_node*)malloc(sizeof(linked_list_node));
 linked_list_ptr->attrib_name=(char*)malloc(sizeof(char)*strlen(name)+1);
 linked_list_ptr->attrib_value=(char*)malloc(sizeof(char)*strlen(value)+1);

 /* copy the attribute name and value to be inserted into db */
 strcpy(linked_list_ptr->attrib_name, name);
 strcpy(linked_list_ptr->attrib_value, value);

 /* update the necessary pointers in list */
```

```
linked_list_ptr->next = linked_list_head_ptr;
linked_list_head_ptr = linked_list_ptr;

#ifdef EnExFlag
   printf("Exit the Insert linked list Function.\n");
   fflush(stdout);
#endif

 return linked_list_head_ptr;
}
```

char* ID(tok_name)

```
/* This program makes a 4 letter entity id from a string passed by
   the user. e.g. If the user passes the string(entity) "Equipment" then
   a pointer to "EquiQQ" is returned. If the string is less than 4
   characters then a "QQ" is concatenated to the entire string.
   e.g. if "WRA" is passed then a pointer to "WRAQQ" is returned.*/
 char *tok_name;
{
   static char entity_id[MAX_LEN];
   int tok_size = 4;

   /* tok_size=4 unless token size is < 4  (i.e. string length) */
   if (strlen(tok_name) < 4) tok_size = strlen(tok_name);
   else tok_size = 4;

#ifdef EnExFlag
    printf("Into the Insert ID Function.\n");
    fflush(stdout);
#endif

   strncpy(entity_id, tok_name, tok_size);
   entity_id[tok_size]= 'Q';
   entity_id[++tok_size]= 'Q';
   entity_id[++tok_size]= '\0';

#ifdef EnExFlag
    printf("Exit the Insert ID Function.\n");
    fflush(stdout);
#endif
```

```
  return entity_id;
} /*End of make entity id function */
```

get_ptr_entity_addr(db_node,substring,tok1,tok2,tok3,tok4)

```
/* This program makes a pointer entity name. A pointer entity has
   attributes which are entity id's which contain the object number
   of the entity pointed to. The user passes a pointer to 3 strings
   which are the fuction name, and two entity names. The pointer
   entity name is these 3 strings concatenated together (seperated by
   '-'). The program also verifies that the two entity names exist.
   For example if the user sends the 3 strings "owner", "Items",
   "Location" then the pointer entity name is: owner_Items_Location
   The program also checks that Items and Location are previouly
   defined entiy names. The entity pointer will have the attributes
   Item# and Loca#. These attributes will contain the integer value
   corresponding to the object counter number assigned previously.*/

 dap_dbid_node* db_node;
 char *substring;
 char *tok1;
 char *tok2;
 char *tok3;
 char *tok4; /*pts to storage area for new name */
{

#ifdef EnExFlag
   printf("Into the Insert make ptr entity name  Function.\n");
   fflush(stdout);
#endif

  if (substring)
  {
    strcpy(tok3,strtok(substring," "));
    strcpy(tok4,get_addr(db_node,'A',tok2,tok1));
    if (!strcmp(tok4,NOT_FOUND))
    {
      ERROR("Unknown attribute ",tok1);
      return (0);
    }
```

```
  }
  else
  {
    ERROR("Incomplete Let Statement :\n",substring);
    return (0);
  }

#ifdef EnExFlag
   printf("Exit the Insert make ptr entity name function.\n");
   fflush(stdout);
#endif


} /*End of make_ptr_entity_name*/


                        char *get_token(req_ptr, token_type)

/* This program returns the next token of a Daplex request query. If
   the user passes a NULL req_ptr then the previous query is used. A
   non-NULL string means get token from the new query string. Token
   type 1 implies get the next individual token, a 0 implies get the
   rest of the query line. Queries are made up of numerous query lines
*/
struct temp_str_info *req_ptr;
int token_type;/*1 means one token zero means rest of tokens on
line*/{
   static int token_location;
   static struct temp_str_info *current_line_ptr;
   static char process_line[MAX_LEN];
   static char token[MAX_LEN];
   static char null_string[1] = '\0';
   int last_token;
   int i = 0;

#ifdef EnExFlag
   printf("Into the Get Token Function.\n");
   fflush(stdout);
#endif

   if (req_ptr) /*if true then new query to parse */
   {
     strcpy(process_line,req_ptr->tsi_str);
     current_line_ptr=req_ptr;
```

```
    token_location=0;
  }

  /* remove leading blanks */
  for (;process_line[token_location]==' ';token_location++);

  /*get new query if at end of line*/
  if (process_line[token_location] == '\0'||
     process_line[token_location]== '\n')
  {
    if (current_line_ptr->tsi_next)
    {
      current_line_ptr=current_line_ptr->tsi_next;
      strcpy(process_line, current_line_ptr->tsi_str);
      for(token_location=0;process_line[token_location]==' ';
      token_location++); /*remove leading spaces*/
    }
    else
    {

#ifdef EnExFlag
   printf("Exit the Get Token Function return NULL.\n");
   fflush(stdout);
#endif

      return(null_string);
    }
  }

  /*The head pointer is at the first reasonable character*/
  if (token_type)
  {
    for(last_token=token_location;process_line[last_token]!=' ' &&
      process_line[last_token]!='\0'&& process_line[last_token]!='\n';
      last_token++)
    {
      token[i++] = process_line [last_token];
    }
  }
  else
  {
    for(last_token=token_location; (last_token < strlen(process_line) &&
      process_line[last_token]!='\0'&& process_line[last_token]!='\n');
```

```
      last_token++)
    {
      token[i++] = process_line [last_token];
    }
  }

  token[i]='\0';
  token_location = last_token;

#ifdef EnExFlag
   printf("Exit the Get Token Function return token(s).\n");
   fflush(stdout);
#endif

  /*capitalize token */
  for(i=0;i<strlen(token);i++) token[i] = toupper(token[i]);
  return token;
} /* end of get_token function */



                              parse_lhs(str,tok1,temp)

/* This function parses the Left Hand side of the = sign. The user
   passes a string in the form of token1 (token2) = ?. This function
   parses & copies token1 and token2 to their designated storage areas.
   ? denotes that there are numerous possibilities following the "="
*/
 char *str;
 char *tok1;
 char *temp;
{
 char string[MAX_LEN];
 char substring[MAX_LEN];
 char *tok2;
 int i;

#ifdef EnExFlag
   printf("Into the Insert parse_lhs Function.\n");
   fflush(stdout);
#endif

 tok2=(char*)malloc(sizeof(char)*(strlen(temp)+1));
 strcpy(tok2,temp);
```

```
i = strcspn(str, "=");
strncpy(substring,str,i);
substring[i] = '\0';

if (!strstr(substring,")"))/* No Right Parens */
{
  ERROR("#1 Illegal Definition format. Improper Left Hand side",string);
   return (0);
}

if (strstr(substring," ("))/* Case one or two*/
{
  strcpy(tok1, strtok(substring," "));
  strcpy(tok2, strtok(NULL," "));
  if (strcmp(tok2,"("))/*Means tok2 is not Left Parens by itself*/
  {
     tok2++;
  }
  else
  {
                strcpy(tok2, strtok(NULL," "));
  }
}
else/*Case three or four*/
{
  strcpy(tok1, strtok(substring,"("));
  strcpy(tok2, strtok(NULL," ("));
 }/*Should be down to good tokens only otherwise token will be messed up*/
 if (strcspn(tok2,")"))/*It had best be the last character*/
 {
   tok2[strlen(tok2)-1] = '\0';/*Get rid of the last character*/
 }

 if (strtok(NULL," "))/*Means more after right parens and before = */
 {
   ERROR("#2 Illegal Definition format. Improper Left Hand side",string);
   return (0);
 }
strcpy(temp,tok2);
free(tok2);

#ifdef EnExFlag
   printf("Exit the Insert parse_lhs Function.\n");
```

```
   fflush(stdout);
#endif

} /*end parse_lhs function */


                char *create_INSERT(db_node,temp,list_ptr, req_list_ptr)

/* This function generates an ABDL "INSERT" statement from the tokens
   within the DAPLEX LET statement. When finished insert points to:
   [INSERT(<TEMP,temp>,<attrib_name1,attrib_value1>**)]

   **the attrib name and value are found in the list_ptr linked list.
     there can be any number of name and value pairs.
*/
  dap_dbid_node* db_node;
  char *temp;
  linked_list_node *list_ptr;
  struct ab_req_info *req_list_ptr;
  {
  struct ab_req_info *new_node;
  struct ab_req_info* ab_req_info_alloc();
  static char insert[MAX_LEN_ABDL];
  char filler[MaxPathLen];

#ifdef EnExFlag
    printf("Into the Insert from Let Function.\n");
    fflush(stdout);
#endif

  sprintf(insert,"[INSERT(<TEMP, %s",get_addr(db_node,'E',temp));
  while(list_ptr)
  {
    if (strlen(insert) < (MAX_LEN_ABDL - 4))
    {
      strcat(insert,">, <");
      strcat(insert,get_addr(db_node,'A',temp,list_ptr->attrib_name));
      strcat(insert,", ");
      strcpy(filler,list_ptr->attrib_value);
      abdl_form(filler);
      strcat(insert,filler);
      list_ptr = list_ptr->next;
    }
```

```
  else
  {
  /*print new insert line */
  create_INSERT(db_node,temp,list_ptr, req_list_ptr);
  break;
  }
 }

 strcat(insert,">)]\0");

 new_node = ab_req_info_alloc();
 new_node->ari_req = (char *) malloc (sizeof(char)*(strlen(insert)+1));
 new_node->ari_next_req = NULL;
 strcpy(new_node->ari_req,insert);
 new_node->ari_rel_op = ExecInsReq;
 new_node->ari_next_req = req_list_ptr->ari_next_req;
 req_list_ptr->ari_next_req = new_node;

#ifdef EnExFlag
   printf("Exit the Insert from Let Function.\n");
   fflush(stdout);
#endif

 return insert;

 } /* end create_insert */


    char * create_RETRIEVE(db_node,temp,  retrieve_name, list_ptr, operation,
                                                    entity_type)

/*This function generates an ABDL "RETRIEVE" statement.
*/
 dap_dbid_node *db_node;
 char *temp;
 linked_list_node *list_ptr;
 char* retrieve_name;
 int operation;
 int entity_type;
 {
 static char retrieve[MAX_LEN_ABDL];
 char addr_name[MAX_LEN];
 char filler[MAX_LEN];
```

```
 linked_list_node *remove_ptr;
 int Compound_Retrieve;
#ifdef EnExFlag
   printf("Into the Create retrieve from Function.\n");
   fflush(stdout);
#endif

 if(list_ptr)
 {
   Compound_Retrieve = TRUE;
   strcpy(retrieve," RETRIEVE((TEMP="); /*compond (i.e. 'and') */
 }
 else
 {
   Compound_Retrieve = FALSE;
   strcpy(retrieve," RETRIEVE(TEMP="); /*simple stmt*/
 }

 /* if entity_type temp already is correct addr */
 if (entity_type) strcpy(addr_name,temp);
 else strcpy(addr_name,get_addr(db_node,'E',temp));

 if (!strcmp(addr_name,NOT_FOUND))
 {
   ERROR("Unknown entity",temp);
   return(" ");
 }

 strcat(retrieve,addr_name);

 while(list_ptr)
 {
  if(strlen(retrieve)>MAX_LEN_ABDL)
    {
    printf("RETRIEVE Statement excxeeds maximum length. \n");
    printf("Choose Smaller Entity Names\n");
    printf("%s\n",retrieve);
    return(retrieve);
    }

  strcat(retrieve,")");
  strcat(retrieve,"and(");
  strcpy(addr_name,get_addr(db_node,'A',temp,list_ptr->attrib_name));
```

```
 if (!strcmp(addr_name,NOT_FOUND))
 {
  ERROR("Unknown element",temp);
  return(" ");
 }

 strcat(retrieve,addr_name);

 strcat(retrieve,"=");
 strcpy(filler,list_ptr->attrib_value);
 abdl_form(filler);
 strcat(retrieve,filler);
 list_ptr=list_ptr->next;
} /* end while list_ptr */

if(strlen(retrieve)>MAX_LEN_ABDL)
 {
 printf("RETRIEVE Statement exceeds maximum length. \n");
 printf("Choose Smaller Entity Names\n");
 printf("%s\n",retrieve);
 return(retrieve);
 }

if(Compound_Retrieve)
  strcat(retrieve,")) ("); /* double '))' */
else
  strcat(retrieve,") ("); /* simple only 1 ')' */

strcat(retrieve,retrieve_name);
strcat(retrieve,")");

if(operation==ExecRetReq) /*add the BY clause for Retrieve*/
{
 strcat(retrieve,"BY ");
 if(strstr(retrieve_name,","))
  strncat(retrieve,retrieve_name,strcspn(retrieve_name,","));
 else
  strcat(retrieve,retrieve_name);

 /* put in beginning and ending '[ ]' for RETRIEVE stmt */
 retrieve[0] = '[';
 strcat(retrieve, "]");
}
```

```
 /* free up the list nodes */
 while(list_ptr)
 {
  remove_ptr = list_ptr;
  list_ptr = list_ptr->next;
  free(remove_ptr->attrib_name);
  free(remove_ptr->attrib_value);
  free(remove_ptr);
 }

#ifdef EnExFlag
   printf("Exit the Create retrieve from Function.\n");
   fflush(stdout);
#endif
 return retrieve;
 }
```

char attrib_type_search(tok1, tok2, first_entity)

```
/* This routine checks to see if the attribute is in the schema. If
   not in schema an error message is output. Also checks that tok1
   (the function) is an attribute of tok2.
*/
 char *tok1;
 char *tok2;
 dap_db_entity_node  *first_entity;

{

 dap_db_entity_node  *entity;
 dap_db_attrib_node  *attrib;

#ifdef EnExFlag
   printf("Into the Attribute Type Function.\n");
   fflush(stdout);
#endif

entity = first_entity;

while(entity)
{
```

```
 if(!strcmp(tok2, entity->dap_entity_name))
 {
  attrib = entity->first_attrib;
  while(attrib)
  {
   if(!strcmp(tok1, attrib->dap_attrib_name))
   {
#ifdef EnExFlag
  printf("Exit the Attribute Type Function.\n");
  fflush(stdout);
#endif

    return attrib->dap_attrib_type;
    }

    attrib = attrib->next_attrib;
  }/* end while attrib not null */

  /* tok2 was found in schema but it didn't have a tok1 attribute */
  ERROR("Following attribute not found in schema",tok1);
  return (0);

 } /* end if tok2=entity name */

 entity = entity->next_entity;
} /*end while entity not null*/

/* should have found a match must not be in out db */
ERROR("Following entity not found in schema",tok2);
    return (0);

} /* end attrib_type_search function */


        char * process_such(tok1,tok2,db_node,req_hd_ptr,dap_info_ptr)

/* function to process DAPLEX RETRIEVE attribute(entity) with the
  SUCH THAT clause.
*/
 char *tok1;
 char *tok2;
 dap_dbid_node *db_node;
 struct ab_req_info *req_hd_ptr;
```

```
 struct dap_info *dap_info_ptr;
{
 struct temp_str_info *temp_str_info_alloc();
 struct temp_str_info *temp_dap_query;
 char token[MAX_LEN];
 char token1[MAX_LEN];
 char token2[MAX_LEN];
 char token3[MAX_LEN];
 char token4[MAX_LEN];
 char temp[MAX_LEN];
 char comm_retr_stmt[MAX_LEN];
 char *retrieve_stmt;
 int entity_type;
 linked_list_node *retrieve_hd_ptr;
 struct ab_req_info *cnt_req_ptr;

#ifdef EnExFlag
   printf("Into the Process Such (RETRIEVE)  Function.\n");
   fflush(stdout);
#endif

 dap_info_ptr->dap_operation = ExecRetCReq; /*retrieve common to be made */
 strcpy(token, get_token(NULL,ONE));
 if(!(strcmp(token,"THAT")))
 {
   strcpy(temp,get_token(NULL,REST));
   parse_lhs(temp, token1, token2);
   entity_type=parse_rhs(temp, token1,token2,token3,token4,db_node,
                         req_hd_ptr,RETRIEVE,dap_info_ptr);
   if(entity_type)
   {
     retrieve_hd_ptr = NULL;
     retrieve_stmt = create_RETRIEVE(db_node,token4,ID(tok2),
                     retrieve_hd_ptr,ExecRetCReq,entity_type);
     /* insert the retrieve at the beginning of common retrieve stmt */
     strcpy(comm_retr_stmt, retrieve_stmt);
     strcpy(temp,get_token(NULL,REST));
     parse_lhs(temp, token1, token2);
     strcpy(token4, "\0");
     parse_rhs(temp, token1,token2,token3,token4,db_node,req_hd_ptr,
           RETRIEVE,dap_info_ptr);

     sprintf(temp,"COMMON(%s,%s)%s\0",ID(token2),ID(token2),comm_retr_stmt);
```

```
 strcpy(comm_retr_stmt,temp);
 retrieve_hd_ptr=NULL;
 retrieve_hd_ptr=add_linked_list(token1,token3,retrieve_hd_ptr);
 retrieve_stmt = create_RETRIEVE(db_node,token2,ID(token2),
                  retrieve_hd_ptr,ExecRetCReq,FALSE);

 strcat(retrieve_stmt,comm_retr_stmt);
 strcpy(comm_retr_stmt, retrieve_stmt);
 /* add "[ ]" around retrieve common stmt */
 comm_retr_stmt[0] = '[';
 strcat(comm_retr_stmt,"]");


 /*save retrieve items til after exec retrieve comm*/
 temp_dap_query = dap_info_ptr->dap_query;
 dap_info_ptr->dap_query = temp_str_info_alloc();
 strcpy(dap_info_ptr->dap_query->tsi_str,ID(tok2));

 /*execute retrieve common*/
 if(short_term_get(dap_info_ptr, comm_retr_stmt) == -1)
 {
  ERROR("No data matches request "," ");
  return NULL; /* ERROR -- No data fm short_term_get */
 }

 dap_info_ptr->dap_query = temp_dap_query; /*reset  retrieve items */
 /* tok1 &2 where assigned values in main */
 create_RETRIEVE_with_OR(dap_info_ptr,tok2,ID(tok2),tok1);
} /*end if entity_type */

else /* only a single retrieve needs to be made */
{
 if(!strcmp(token2,tok2))
 {
  retrieve_hd_ptr=NULL;
  retrieve_hd_ptr=add_linked_list(token1,token3,retrieve_hd_ptr);
  retrieve_stmt = create_RETRIEVE(db_node,token2,tok1,
                  retrieve_hd_ptr,ExecRetReq,entity_type);
  load_request (dap_info_ptr,retrieve_stmt, ExecRetReq);
  }

} /*end else */
```

```
 } /* end if "THAT" */
 else /* SUCH without THAT */
 {
 ERROR("ONLY 'SUCH' FOUND. Need 'SUCH THAT'",temp);
 dap_info_ptr->dap_operation = ExecNoReq;
 return NULL;
 }

#ifdef EnExFlag
   printf("Exit the Process Such (RETRIEVE)  Function.\n");
   fflush(stdout);
#endif

    return comm_retr_stmt;

} /* End process_such function */



                    struct temp_str_info *combine_lists(list1,list2)

/*  procedure to combine the results of two retrieve common stmts.
   i.e. it ANDS the retrieve commons. A pointer to the list of common
   results is returned.
*/
struct temp_str_info *list1;
struct temp_str_info *list2;
{
 struct temp_str_info *common_list,*last_entry, *remove_node;

#ifdef EnExFlag
   printf("Enter the Combine Lists  Function.\n");
   fflush(stdout);
#endif

 if(!list2)
 {
  ERROR("No data matches request "," ");
  return NULL;
 }


 common_list = temp_str_info_alloc(); /*make header node */
 last_entry = common_list;
```

```
 while(list1)
 {
  if(check_list(list2,list1->tsi_str)) /* list1 element in list2? */
  {
   /* yes in both, put in common list */
   last_entry->tsi_next = list1;
   last_entry = list1;
   list1 = list1->tsi_next;
  }
  else
  {
   /* no, remove list1 node */
   remove_node = list1;
   list1 = list1->tsi_next;
   free(remove_node);
  }
 } /* end while list1 not null */

 while(list2) /* free list2 linked list */
 {
  remove_node = list2:
  list2 = list2->tsi_next;
  free(remove_node);
 } /* end while list2 not null */

 /* REMOVE HEADER NODE */
 remove_node = common_list;
 common_list = common_list->tsi_next;
 free(remove_node);

#ifdef EnExFlag
   printf("Exit the Combine Lists  Function.\n");
   fflush(stdout);
#endif

 return common_list;

} /* end combine list function */
```

process_where(tok1, tok2,db_node,req_hd_ptr, dap_info_ptr)

```
/* function to process DAPLEX RETRIEVE attribute(entity) with the
  WHERE clause.
*/
char *tok1;
 char *tok2;
 dap_dbid_node *db_node;
 struct ab_req_info *req_hd_ptr;
 struct dap_info *dap_info_ptr;
{

 static char temp[MAX_LEN];
 char retr_val[MAX_LEN];
 char token[MAX_LEN];
 char token1[MAX_LEN];
 char token2[MAX_LEN];
 char token3[MAX_LEN];
 char token4[MAX_LEN];
 char comm[MAX_LEN];
 char retr_comm_stmt[MAX_LEN];
 char *retrieve_stmt, *function_retr;
 int entity_type;
 int first_retrieve_common = TRUE;
 struct temp_str_info *common_list;
 linked_list_node *retrieve_hd_ptr, *remove_ptr;


#ifdef EnExFlag
   printf("Into the Process Where (RETRIEVE)  Function.\n");
   fflush(stdout);
#endif

 strcpy(token,get_token(NULL,ONE));
 if(strcmp(token,"BEGIN"))
 {
   ERROR("WHERE must be followed by a BEGIN - END BLOCK",token);
   dap_info_ptr->dap_operation = ExecNoReq;
   return NULL;
 }

  /* process attribute_name(entity_name) = entity_name stmt */
  strcpy(temp,get_token(NULL,REST));
  retrieve_hd_ptr = NULL;
```

```
 parse_lhs(temp,token1,token2);
 entity_type=parse_rhs(temp, token1,token2,token3,token4,db_node,
                req_hd_ptr,RETRIEVE,dap_info_ptr);
 if(!entity_type)
 {
   ERROR("rhs must = entity name for stmt immediately after BEGIN ",temp);
  dap_info_ptr->dap_operation = ExecNoReq;
  return NULL;
 }

 /*function_retr contains retrieve statement and COMMON clause for
   retrieve common to be created*/
 function_retr = create_RETRIEVE(db_node,token4,ID(tok2),
                  retrieve_hd_ptr,ExecRetCReq,entity_type);
 /*assign the part of the retrieve common that won't change to comm*/
 sprintf(comm,"COMMON(%s,%s)%s\0",ID(token3),ID(token3),function_retr);
 /* prime dap_info with variable name to be retrieved fm short_term get*/
 dap_info_ptr->dap_query = temp_str_info_alloc();
 strcpy(dap_info_ptr->dap_query->tsi_str,ID(tok2));
 strcpy(temp,get_token(NULL,REST)); /*get 1st stmt after 'BEGIN' */

while(strcmp(temp,"END "))
{
 if(strpbrk(temp,"$@"))
 {
   ERROR("END - Not Found for WHERE statement"," ");
   return (0);
 }
 parse_lhs(temp,token1,token2);
 entity_type=parse_rhs(temp, token1,token2,token3,token4,db_node,
                    req_hd_ptr,RETRIEVE,dap_info_ptr);
 retrieve_hd_ptr = NULL;
 retrieve_hd_ptr=add_linked_list(token1,token3,retrieve_hd_ptr);

 retrieve_stmt = create_RETRIEVE(db_node,token2,ID(token2),retrieve_hd_ptr
          ,ExecRetCReq,entity_type);
 sprintf(retr_comm_stmt,"[%s %s]\0",retrieve_stmt,comm);

 /*execute retrieve common*/
if(short_term_get(dap_info_ptr, retr_comm_stmt) == -1)
{
 ERROR("No data matches request "," ");
 return NULL; /* ERROR -- No data fm short_term_get */
```

```
  }

  if(first_retrieve_common)
  {
    /* first time thru no lists to combine */
    first_retrieve_common = FALSE;
    common_list = dap_info_ptr->dpi_kc_data;
  }
  else
  {
    common_list = combine_lists(common_list, dap_info_ptr->dpi_kc_data);
  }

  strcpy(temp,get_token(NULL,REST));

  } /* end while temp <> "END" */


 /* tok1 &2 where assigned values in main */
 create_RETRIEVE_with_OR(dap_info_ptr,tok2,ID(tok2),tok1);
 sprintf(dap_info_ptr->dap_query->tsi_str,"%s(%s)\0",tok1,tok2,retr_comm_stmt);



#ifdef EnExFlag
   printf("Exit the Process Where (RETRIEVE)  Function.\n");
   fflush(stdout);
#endif

} /* End process_where function */



int process_LET_stmt(str, tok1, tok2, tok3, tok4, entity_tok1, db_node,  req_hd_ptr,
                                called_by, dap_info_ptr)

/* This function processes The LET statements found in a BEGIN and END block
  found after a FOR statement. Information is parsed so that an ABDL INSERT
  statement can generated later in the program. There are two forms of LET
    I. LET entity_NAME = literal
    II. LET entity_name_1 = entity_name_2  SUCH THAT entity_name = literal
  The first form generates a single ABDL INSERT statement. THe second
  must retreieve the element number of entity_name_2 (i.e. an ABDL RETRIEVE
```

```
  stmt is generated and executed) and create an Additional ABDL INSERT
  from information after the SUCH THAT clause (i.e. 2 INSERTS created).
*/
 char* str;
 char *tok1;
 char *tok2;
 char *tok3;
 char *tok4;
 char *entity_tok1;
 dap_dbid_node *db_node;
 struct ab_req_info *req_hd_ptr;
 int called_by;
 struct dap_info *dap_info_ptr;
{
 char string[MAX_LEN];
 char substring[MAX_LEN];
 int i;

#ifdef EnExFlag
   printf("Into the Process Let Function.\n");
   fflush(stdout);
#endif

 strcpy(string,str);
 parse_lhs(str,tok1,tok2);
  if(entity_tok1[0]!='\0') /* not NULL if FOR NEW stmt */
  {
    if (strcmp(tok2,entity_tok1))
    {
      ERROR("#3 Illegal Definition format. Improper Left Hand side\n",string);
     dap_info_ptr->dap_operation = ExecNoReq;
     return (0);
    }
  }

 /* SUCH_THAT (1) is returned if type was 'E' else LET(0)  returned */
 called_by= parse_rhs(string, tok1, tok2, tok3, tok4, db_node, req_hd_ptr,
                      called_by,dap_info_ptr);

#ifdef EnExFlag
   printf("Exit the Process Let Function.\n");
   fflush(stdout);
#endif
```

```
 return called_by;
} /*end process_LET_stmt*/

        parse_rhs(string, tok1,tok2, tok3, tok4, db_node, req_hd_ptr,called_by,
                                                    dap_info_ptr)

 /* function processes the right hand side (rhs) of a statement that has
   an '=' in it.
*/
  char *string;
  char *tok1;
  char *tok2;
  char *tok3;
  char *tok4;
  dap_dbid_node *db_node;
  struct ab_req_info *req_hd_ptr;
  int called_by;
  struct dap_info *dap_info_ptr;
  {
  /*tok 21 -24 used to create additional INSERT stmt if SUCH THAT clause
    in LET stmt.
*/
  char att_type;
  char *retrieve_stmt;
  char tok21[MAX_LEN];
  char tok22[MAX_LEN];
  char tok23[MAX_LEN];
  char tok24[MAX_LEN];
  char substring[MAX_LEN];
  char elem2[MAX_LEN];
  char elem3[MAX_LEN];
  char tok22_entity_id[MAX_LEN];
  char tok2_entity_id[MAX_LEN];
  char tok3_entity_id[MAX_LEN];
  char token[MAX_LEN];
  char SUCH_THAT_string[MAX_LEN];
  int t_size;

  linked_list_node *insert_head_ptr;
  linked_list_node *retrieve_hd_ptr;

#ifdef EnExFlag
```

```
  printf("Into the process rhs Function.\n");
  fflush(stdout);
#endif

 strtok(string,"="); /* clear string out to '=' */
 strcpy(substring,strtok(NULL,"=")); /*get rest of line */
 att_type = attrib_type_search(tok1,tok2,db_node->first_entity);
 switch(att_type)
  {
  case 'S':
     /*for(;substring[0]!='"'&&substring[0]!='\0';substring++) ;*/
     strtok(substring,"\"");
     if (substring)
     {
        strcpy(tok3,strtok(NULL,"\""));
     }
     else
     {
       ERROR("#4 Illegal Definition format. Improper Left Hand side",
            string);
       dap_info_ptr->dap_operation = ExecNoReq;
       return (0);
     }
     break;

  case 'I':
     if(substring)
     {
       strcpy(tok3,strtok(substring," "));
       if (!(is_integer(tok3)))
       {
         ERROR("#5 Illegal Definition format. Improper Left Hand side",
              string);
         dap_info_ptr->dap_operation = ExecNoReq;
         return (0);
       }
     }
     else
     {
       ERROR("Incomplete definition of Let Stmt.\n",string);
       dap_info_ptr->dap_operation = ExecNoReq;
       return (0);
     }
```

```
    break;

case 'F':
    if(substring)
    {
      strcpy(tok3,strtok(substring," "));
      if (!(is_real(tok3)))
      {
        ERROR("#6 Illegal Definition format. Improper Left Hand side",
            string);
        dap_info_ptr->dap_operation = ExecNoReq;
        return (0);
      }
    }
    else
    {
      ERROR("Incomplete definition of Let Stmt.\n",string);
      dap_info_ptr->dap_operation = ExecNoReq;
      return (0);
    }
    break;

case 'C':
    /*for(;substring[0]!=\"&&substring[0]!='\0';substring++) ;*/
    strtok(substring,"");
    if (substring)
    {
       strcpy(tok3,strtok(NULL,""));
    }
    else
    {
      ERROR("#7 Illegal Definition format. Improper Left Hand side",
            string);
      dap_info_ptr->dap_operation = ExecNoReq;
      return (0);
    }
    break;

case 'E':
case 'A':

   switch(called_by)
   {
```

```
case SUCH_THAT:

  /*ERROR: Entity ('E') type after = in a Such That stmt*/
  ERROR("SUCH THAT clause: token right of '=' can't be an entity",
        string);
  dap_info_ptr->dap_operation = ExecNoReq;
  return (0);
  break;

case LET:
  get_ptr_entity_addr(db_node,substring,tok1,tok2,tok3,tok4);
  /*tok4 will be the new name of the pointer entity */

  /* Next part of LET Stmt must be SUCH THAT else error */
  strcpy(token,get_token(NULL,ONE));
  if (strcmp(token,"SUCH"))
            {
    ERROR("Illegal query format.",string);
    return (0);
  }

  strcpy(token,get_token(NULL,ONE));
  if (strcmp(token,"THAT"))
            {
    ERROR("Illegal query format.",string);
    return (0);
  }

  strcpy(SUCH_THAT_string,get_token(NULL,REST));
  process_LET_stmt(SUCH_THAT_string,tok21,tok22,tok23,tok24,
        tok3,db_node,req_hd_ptr,SUCH_THAT,dap_info_ptr);


  retrieve_hd_ptr = NULL;
  retrieve_hd_ptr=add_linked_list(tok21,tok23,retrieve_hd_ptr);
  /* get tok22 object number from database */
  retrieve_stmt = create_RETRIEVE(db_node,tok22,ID(tok22),
                  retrieve_hd_ptr,ExecRetReq,FALSE);
  /* put retrieve name into dap info for short get */
  dap_info_ptr->dap_query = temp_str_info_alloc();
  strcpy(dap_info_ptr->dap_query->tsi_str,ID(tok22));
  dap_info_ptr->dap_query->tsi_next = NULL;
```

```
        if (get_object_no(dap_info_ptr, retrieve_stmt) == -1)
          return (0); /* ERROR return empty string */

        strcpy(elem3,dap_info_ptr->dpi_kc_data->tsi_str);
        sprintf(elem2, "%-d", db_node->object_counter);
        insert_head_ptr = NULL;
        insert_head_ptr=add_linked_list(ID(tok2), elem2,insert_head_ptr);
        insert_head_ptr=add_linked_list(ID(tok3), elem3,insert_head_ptr);
        sprintf(elem2, "%-d", db_node->object_counter);
        insert_head_ptr=add_linked_list(ID(tok4), elem2, insert_head_ptr);
        /* Create and Print the additional ABDL "INSERT" stmt generatede
           by the SUCH THAT clause. */
        create_INSERT(db_node,tok4, insert_head_ptr,req_hd_ptr);
        break;

       case RETRIEVE:
        get_ptr_entity_addr(db_node,substring,tok1,tok2,tok3,tok4);
        /*tok4 is addr of the ptr_entity*/
         break;

     } /* End called_by switch */
     break; /*for case 'E' */

   default:
     ERROR("Attribute is unrecognizable",string);
     return (0);

   } /*End of switch*/

#ifdef EnExFlag
   printf("Exit the process rhs Function.\n");
   fflush(stdout);
#endif


  if(att_type=='E'||att_type=='A')
  {
    return (SUCH_THAT); /* types 'E'or 'A' return SUCH_THAT(1)*/
  }
  else
  {
    return (LET); /*all types except 'E' return LET(0) */
  }
```

```
} /* End process_rhs function */


                    get_object_no(dap_info_ptr, retrieve_stmt)

/* This function returns the entity object number assigned to the
   attribute. It generates an ABDL stmt (via create_retrieve) and
   executes the retrieve statement (via short_term_get) to get the
   desired number
*/
  struct dap_info *dap_info_ptr;
  char *retrieve_stmt;
{
char *x;
#ifdef EnExFlag
   printf("Into the get_entity_object_no Function.\n");
   fflush(stdout);
#endif
  /*execute retrieve*/
  if(short_term_get(dap_info_ptr, retrieve_stmt) == -1)
    {
    ERROR("No data to Retrieve"," ");
    return -1; /* ERROR -- No data fm short_term_get */
    }

  if(dap_info_ptr->dpi_kc_data)
  {
    if(is_integer(dap_info_ptr->dpi_kc_data->tsi_str))
    {
      if(dap_info_ptr->dpi_kc_data->tsi_next)
      {
        ERROR("Corrupt Database! More than one entity id # for ",
                                    retrieve_stmt);
        dap_info_ptr->dap_operation = ExecNoReq;
        return (0);
      }
    }
    else
    {
      ERROR("Database Corrupted Non-Numeric data in entity id # for ",
                                    retrieve_stmt);
```

```
      dap_info_ptr->dap_operation = ExecNoReq;
      return (0);
    }
  }
  else
  {
    ERROR("Database Corrupted No entity id # found for ",retrieve_stmt);
    dap_info_ptr->dap_operation = ExecNoReq;
    return (0);
  }

  /* We have a valid entity object number */
#ifdef EnExFlag
   printf("Exit get_entity_object_no Function.\n");
   fflush(stdout);
#endif


} /* End get_entity_object_NO */
```

create_RETRIEVE_with_OR(dap_info_ptr, template, conditional_item, retrieve_item)

```
/* This function generates an ABDL RETRIEVE with 1 or more OR cluases.*/

 struct dap_info *dap_info_ptr;
 char *template;
 char *conditional_item;
 char *retrieve_item;
{
 struct temp_str_info *kc_data;
 struct ab_req_info *req_node;
 char temp_clause[MAX_LEN];
 char retrieve_or_stmt[MAX_LEN_ABDL];
 struct dap_db_id_node *db_node;

#ifdef EnExFlag
   printf("Into the create_RETRIEVE_with_OR Function.\n");
   fflush(stdout);
#endif

 db_node = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;
```

```
if(!dap_info_ptr->dpi_kc_data) /*no data*/
{
  ERROR("DAPLEX query unable to find requested data"," ");
  dap_info_ptr->dap_operation = ExecNoReq;
  return (0);
}

strcpy(retrieve_or_stmt,"[RETRIEVE");
strcpy(temp_clause,"((TEMP=");
strcat(temp_clause,get_addr(db_node,'E',template));
strcat(temp_clause,")and(");
strcat(temp_clause,conditional_item);
strcat(temp_clause,"=");
kc_data = dap_info_ptr->dpi_kc_data;

/* make an OR clause for each element in kc_data linked list*/
while(kc_data)
{
 strcat(retrieve_or_stmt,temp_clause);
 strcat(retrieve_or_stmt,kc_data->tsi_str);
 strcat(retrieve_or_stmt,"))or");
 kc_data = kc_data->tsi_next;
}
 /* removes the last OR */

 retrieve_or_stmt[strlen(retrieve_or_stmt) -2] = '\0';
 strcat(retrieve_or_stmt,"(");
 strcat(retrieve_or_stmt,retrieve_item);
 strcat(retrieve_or_stmt,") BY ");
 strcat(retrieve_or_stmt,retrieve_item);
 strcat(retrieve_or_stmt,"]");

 load_request (dap_info_ptr,retrieve_or_stmt, ExecRetReq);

#ifdef EnExFlag
  printf("Exit the create_RETRIEVE_with_OR Function.\n");
  fflush(stdout);
#endif


} /* end Create_RETRIEVE_With_OR */
```

int load_request (dap_info_ptr,request, ExecReqType)

```
/* This functions loades the created ABDL statement(s) into the dap_info
  data structure. (i.e. last step before execution) */
 struct dap_info *dap_info_ptr;
  struct ab_req_info *request;
  int ExecReqType;
{
  struct ab_req_info *req_node;

#ifdef EnExFlag
   printf("Enter the load_request Function.\n");
   fflush(stdout);
#endif

  /* load a request node with RETRIEVE */
  req_node = ab_req_info_alloc();
  req_node->ari_req = (char *) malloc (sizeof(char)*strlen(request)+1);
/*  req_node->ari_next_req = NULL; */
  strcpy(req_node->ari_req,request);
  req_node->ari_rel_op = ExecReqType;

  dap_info_ptr->dap_operation = ExecReqType;
  dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req = req_node;
  dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req = req_node;
  dap_info_ptr->dpi_abdl_tran.ti_no_req = 1;


#ifdef EnExFlag
   printf("Exit the load_request Function.\n");
   fflush(stdout);
#endif

} /* end load_request */
```

int dap_to_abdl (dap_info_ptr,query_ptr)

```
/* This program is the main driver program which then calls various other procedures
within query.c*/

   struct dap_info *dap_info_ptr;
   struct temp_str_info *query_ptr;
```

```
{

 int entity_type;
 int t_size, i;
 int New_Entity;

 char token[MAX_LEN];
 char entity_token1[MAX_LEN];
 char token1[MAX_LEN];
 char token2[MAX_LEN];
 char token3[MAX_LEN];
 char token4[MAX_LEN];
 char token5[MAX_LEN];
 char temp[MAX_LEN];
 char entity_id[MAX_LEN];
 char LET_string[MAX_LEN];
 char FOR_no_NEW[MAX_LEN];
 char combo_token[MAX_LEN];
 char *retrieve_stmt;

 struct temp_str_info* temp_str_info_alloc();
 linked_list_node *insert_head_ptr, *retrieve_hd_ptr;
 struct ab_req_info *req_hd_ptr,*cnt_req_ptr, *req_node;
 struct temp_str_info *last_dap_query;
 dap_dbid_node *db_node;
 char *t;
 int first_time;

#ifdef EnExFlag
   printf("Into the dap_to_abdl Function.\n");
   fflush(stdout);
#endif

 db_node = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;
 req_hd_ptr = ab_req_info_alloc();
 req_hd_ptr->ari_req = (char *) malloc (sizeof(char)*21);
 strcpy(req_hd_ptr->ari_req,"Header Record\0");
 req_hd_ptr->ari_next_req = NULL;
 New_Entity = FALSE;
 if(!query_ptr)
 {
  ERROR("No DAPLEX query found"," ");
```

```
 dap_info_ptr->dap_operation = ExecNoReq;
 return (0);
}


strcpy(token, get_token(query_ptr,ONE));
if (!strcmp(token,"FOR"))
{
 /* the FOR statement can be of two forms:
    I.  FOR NEW entity-name
    II. FOR entity-name
   the first form is for an entity that has not been
   created yet where the second has been created but we
   are modifying the entries.
 */
 strcpy(token,get_token(NULL,ONE));
 if (!strcmp(token,"NEW"))
 {
   strcpy(entity_token1,get_token(NULL,ONE));

   insert_head_ptr = NULL;

   db_node->object_counter++;
   sprintf(temp,"%-d",db_node->object_counter);
   insert_head_ptr=add_linked_list(ID(entity_token1),temp,insert_head_ptr);
   New_Entity = TRUE;
 }
 else /*FOR without NEW*/
 {
   entity_token1[0] = '\0'; /*set it to NULL */
   strcpy(FOR_no_NEW,token);
   strcat(FOR_no_NEW,get_token(NULL,REST));
   process_LET_stmt(FOR_no_NEW,token1,token2,token3,token4,entity_token1,
                    db_node,req_hd_ptr,SUCH_THAT,dap_info_ptr);
   strcpy(entity_token1, token2);
   retrieve_hd_ptr = NULL;
   retrieve_hd_ptr=add_linked_list(token1,token3,retrieve_hd_ptr);
   retrieve_stmt = create_RETRIEVE(db_node,token2,ID(token2),
                    retrieve_hd_ptr,ExecRetReq,entity_type);
   /* put retrieve name into dap info for short get */
   dap_info_ptr->dap_query = temp_str_info_alloc();
   strcpy(dap_info_ptr->dap_query->tsi_str,ID(token2));
   dap_info_ptr->dap_query->tsi_next = NULL;
```

```
  get_object_no(dap_info_ptr, retrieve_stmt);
  New_Entity = FALSE;
}
/* The FOR stmt is always followed by a BEGIN and END block. */
strcpy(token,get_token(NULL,ONE));
if (strcmp(token,"BEGIN"))
{
  ERROR("Invalid query header", query_ptr);
  return (0);
}

strcpy(token,get_token(NULL,ONE));
while (!strcmp(token,"LET"))
{
  strcpy(LET_string,get_token(NULL,REST));
  entity_type=process_LET_stmt(LET_string,token1, token2, token3,
    token4,entity_token1, db_node,req_hd_ptr,LET,dap_info_ptr);

 if(!entity_type) /*attribute is not entity ('E') type?*/
 {
     /* Not E so add new attribute and its value to insert list */
   insert_head_ptr=add_linked_list(token1,token3,insert_head_ptr);

 }
 strcpy(token,get_token(NULL,ONE)); /*get next token*/

}/*End  while Token = "LET"*/

if (strcmp(token,"END")) /*next stmt after LET's better be END*/
{
  ERROR("Expecting END statement but following stmt found:\n",token);
  dap_info_ptr->dap_operation = ExecNoReq;
  return (0);
}

if (New_Entity)
   create_INSERT(db_node,entity_token1,insert_head_ptr,req_hd_ptr);
dap_info_ptr->dap_operation = ExecInsReq;
dap_info_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req = req_hd_ptr->ari_next_req;
dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req = req_hd_ptr->ari_next_req;
/*count # of requests */
dap_info_ptr->dpi_abdl_tran.ti_no_req = 0;
for(cnt_req_ptr=req_hd_ptr->ari_next_req;cnt_req_ptr;cnt_req_ptr=
```

```
      cnt_req_ptr->ari_next_req)dap_info_ptr->dpi_abdl_tran.ti_no_req++;

  /*copy_requests_to_dap_info(dap_info_ptr,req_hd_ptr);*/
  req_hd_ptr->ari_next_req = NULL; /* reset the ptr */
  return 1;
} /* end if FOR */
else
{
  if(!strcmp(token,"RETRIEVE"))
  {
   first_time = TRUE;
   combo_token[0] = '\0';
   do
   {
    strcpy(token,get_token(NULL,ONE));
    /*keep concatenating tokens until ')' found*/
    while(!strstr(token,")")&&!(token)) strcat(token,get_token(NULL,ONE));


    if (first_time)
    {
      first_time = FALSE;
      dap_info_ptr->dap_query = temp_str_info_alloc();
      strcpy(dap_info_ptr->dap_query->tsi_str,token);
      last_dap_query = dap_info_ptr->dap_query;
    }
    else
    {
      strcat(combo_token,",");
      last_dap_query->tsi_next = temp_str_info_alloc();
      strcpy(last_dap_query->tsi_next->tsi_str,token);
      last_dap_query = last_dap_query->tsi_next;
    }

    strcat(token, "=");
    parse_lhs(token,token1,token2);
    strcat(combo_token,token1);
    strcpy(token,get_token(NULL,ONE));
   } while ((!strcmp(token,"AND")) && (temp));

   strcpy(token1,combo_token);
   /* Null string, 'SUCH THAT' or "WHERE" clause must be next */
```

```
    if(token[0]=='\0') /*simple retrieve with no qualifiers*/
    {
     retrieve_stmt = create_RETRIEVE(db_node,token2,token1,NULL,
                                     ExecRetReq,FALSE);
     load_request (dap_info_ptr,retrieve_stmt, ExecRetReq);
    }
    else
    {
      if(!(strcmp(token,"SUCH")))
      {
       retrieve_stmt = process_such(token1,token2,db_node,req_hd_ptr,
                        dap_info_ptr);
      }
      else
      {
        if(!(strcmp(token,"WHERE")))
        {
         process_where(token1,token2,db_node,req_hd_ptr,dap_info_ptr);
        }
        else
        {
          ERROR("RETRIEVE must contain a 'SUCH THAT' or 'WHERE' clause",temp);
          dap_info_ptr->dap_operation = ExecNoReq;
          return (0);
        } /* end if WHERE */
      } /* end if SUCH */
    } /* end if NULL */
    } /* End if "RETRIEVE" */
  else
  {
    ERROR("Not a valid DAPLEX statement - Unknown first token",temp);
    dap_info_ptr->dap_operation = ExecNoReq;
    return (0);
  }
} /*end else */

#ifdef EnExFlag
   printf("Exit the dap_to_abdl Function.\n");
   fflush(stdout);
#endif
return 1;
} /* end _abdl routine */
```

```
/*
 * File Name: read_cat.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/read_cat.c
 * This file contains procedures for future growth allowing the user to
   create a functional model based on best guess info from the template
   and descriptor files.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <licommdata.h>
#include <dap_info.h>

                        f_load_catalog(dap_info_ptr)

  struct dap_info *dap_info_ptr;
{
   dap_dbid_node *db_ptr;
   int count1,count2;
   char string[MaxPathLen], temp[MaxPathLen];
   FILE *file_ptr;
   dap_db_entity_node *entity, *curr_ent;
   dap_db_attrib_node *attrib, *curr_att;

/*assign pointers and open the file*/
file_ptr = fopen(dap_info_ptr->dpi_ddl_files->ddli_temp.fi_fname,"r");
db_ptr = dap_info_ptr->dpi_curr_db.cdi_db.dn_dap;
printf("Enter the read program\n");

/*Clear out the name at the top of the file*/
strcpy(string, fgets(temp, MaxPathLen, file_ptr));

/*Get the necessary data, number of entities*/
strcpy(string, fgets(temp, MaxPathLen, file_ptr));
count1 = atoi(string);
db_ptr->number_of_entitys = count1;

/* Ensure that we aren't pointing at garbage*/
db_ptr->first_entity = NULL;

/*Loop once for each entity*/
```

```
for (;count1;count1--)
{
   entity = (dap_db_entity_node*) malloc(sizeof(dap_db_entity_node));
   entity->dap_entity_name = (char*) malloc (sizeof(char)*MaxPathLen);
   strcpy(string, fgets(temp, MaxPathLen, file_ptr));

   /* Ensure that we aren't pointing at garbage*/
   entity->first_attrib = NULL;

   count2 = atoi(string);
   entity->number_of_attribs= count2;
   strcpy(entity->dap_entity_name, fgets(temp, MaxPathLen, file_ptr));

   /*Loop once for each attribute in each entity*/
   for (;count2;count2--)
   {
     attrib = (dap_db_attrib_node*) malloc(sizeof(dap_db_attrib_node));
     attrib->dap_attrib_name = (char*) malloc (sizeof(char)*MaxPathLen);

     strcpy(string, fgets(temp, MaxPathLen, file_ptr));
     strcpy(attrib->dap_attrib_name,strtok(string," "));
     strcpy(temp,strtok(NULL," "));
     attrib->dap_attrib_type = temp[0];

              /*get ready for next attribute*/
     attrib->next_attrib=entity->first_attrib;
     entity->first_attrib=attrib;
    }

/*get ready for next entity*/
entity->next_entity = db_ptr->first_entity;
db_ptr->first_entity = entity;

}
}
```

```
/*
 * File Name: read_file.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Lil/read_file.c
 * This file contains procedures for actually reading the schema/defintion data
 *
 */

#include <stdio.h>
#include <strings.h>
#include <stdlib.h>
#include <dap_info.h>
#include <licommdata.h>

                        int  read_a_file(dap_info_ptr)

  struct dap_info *dap_info_ptr;

{
  char *input, *tmp, *result;
  int i, MAXLEN = 80;

  tmp = (char*) malloc (sizeof(char)*(MAXLEN+1));
  input = (char*) malloc (sizeof(char)*(MAXLEN+1));
/* *dap_info_ptr.dpi_file->fi_fid will point to the file or stdin */
   while((result=fgets(tmp,MAXLEN,dap_info_ptr->dpi_file.fi_fid))!=NULL)
   {
     strcpy(input,tmp);
     input[strlen(tmp)-1]='\0';
     for (i=0;i<MAXLEN;i++)
     {
       input[i]=toupper(input[i]);
                }
/* It had better at least begin with a D */
     if (strchr(input,'D'))
       ddl_parse(input,dap_info_ptr);
   }
  /* creates the .d and .t files */
   if (dap_info_ptr->dap_error != ErrCreateDB)
   create_t_file(dap_info_ptr->dpi_curr_db.cdi_db.dn_dap);
}
```

```
/*

 * File Name: readrtnes.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Lil/readrtnes.c $
 * This file contains procedures to ensure a file exists, open it and
   provide a pointer to it for future operations.
 */

#include <stdio.h>
#include <ctype.h>
#include <strings.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"

                        f_read_transaction_file(dap_info_ptr)

struct dap_info *dap_info_ptr;/* Use the same old one*/
{
                int  open_flag;  /* boolean flag */
                int  i;          /* counter     */
     char  filename[FNLength + 1];
     char  filler[MaxPathLen+FNLength+1];

                /* This routine opens a create/query file and reads the */
                /* creates/queries into the request list.          */

#ifdef EnExFlag
  printf ("Enter f_read_transaction_file\n");
#endif

 open_flag = FALSE;
 printf ("[7;7m\nWhat is the name of the CREATE/ QUERY file ---->[0;0m ");

 /* open the file */
 while (open_flag == FALSE)
  {
   gets (filename);
   strcpy(dap_info_ptr->dpi_file.fi_fname, add_path(filler,filename));
   if ((dap_info_ptr->dpi_file.fi_fid =
                fopen (dap_info_ptr->dpi_file.fi_fname, "r")) == NULL)
                {
                 printf ("\nUnable to open file %s\n", dap_info_ptr->dpi_file.fi_fname);
```

```
                /*ring_the_bell();*/
                printf ("Please reenter valid filename\n");
        printf ("[7;7m");
        printf ("\nWhat is the name of the CREATE/QUERY file --->[0;0m ");
                } /* end if */
    else
                open_flag = TRUE;

   } /* end while */

#ifdef EnExFlag
  printf ("Exit f_read_transaction_file\n");
#endif

} /* end f_read_transaction_file */


                        f_read_reciept_file(dap_info_ptr)

struct dap_info *dap_info_ptr;
{
      int  open_flag;  /* boolean flag */
      int  i,num;
      char  filename[FNLength + 1];
      char  filler[MaxPathLen+FNLength+1];

      /* This routine opens a file to allow a mass */
      /* dump of data to it.  */

#ifdef EnExFlag
  printf ("Enter f_read_reciept_file\n");
#endif

  open_flag = FALSE;
  printf ("\nWhat is the name of the receiving file ----> ");

  /* open the file */
  while (open_flag == FALSE)
    {
     gets (filename);
     strcpy(dap_info_ptr->dpi_file.fi_fname, add_path(filler,filename));
     if ((dap_info_ptr->dpi_file.fi_fid =
                fopen (dap_info_ptr->dpi_file.fi_fname, "r")) != NULL)
       {
```

```
        printf ("\nFile %s already exists. Overwrite?(Y/N)\n", dap_info_ptr-
>dpi_file.fi_fname);
        /*ring_the_bell();*/
        dap_info_ptr->dap_answer = get_ans(&num);
        if (dap_info_ptr->dap_answer=='y' || dap_info_ptr->dap_answer=='Y')
        {
          open_flag = TRUE;
        } /* end if answer */
      else
        {
          fclose(dap_info_ptr->dpi_file.fi_fid);
          printf ("Please reenter valid filename\n");
          printf ("");
          printf ("\nWhat is the name of the receiving file ---> ");
        }
       } /* end if */
      else
        open_flag = TRUE;

    } /* end while */

    dap_info_ptr->dpi_file.fi_fid =
      fopen (dap_info_ptr->dpi_file.fi_fname, "w");

#ifdef EnExFlag
  printf ("Exit f_read_reciept_file\n");
#endif

} /* end f_read_reciept_file*/
```

f_read_file(dap_info_ptr,f_tran_info_ptr)/* Use this to read our queries*/

```
 struct dap_info *dap_info_ptr;
 struct tran_info *f_tran_info_ptr;
{
    struct temp_str_info *new_t_ptr,  /* ptrs to linked list of 80 col input */
                                 *curr_t_ptr,
                *head_t_ptr,  ..
                                                *rd_temp_str_info();
    struct dap_req_info  *new_req_ptr,  /* ptrs to request list */
```

```
                                        *curr_req_ptr,
                                        *dap_req_info_alloc();
    int   i,
          first_req,    /* boolean flag */
                first_line,   /* boolean flag */
                length_so_far, /* length of a single transaction */
                EOF_flag,     /* boolean flag */
          EOR_flag;     /* boolean flag */
    char  *var_str_alloc();

                /* This routine reads a file of transactions into */
                /* the user's request list structure.  */

#ifdef EnExFlag
  printf ("Enter f_read_file\n");
#endif

  first_req = TRUE;
  EOF_flag = FALSE;

  /* create the request list from the inner loop's line list */
  while (EOF_flag == FALSE)
   {
    EOR_flag = FALSE;
    length_so_far = 0;
    first_line = TRUE;

    /* create a linked list node for each request read.  */
    /* each node represents a line of the request */
    while (EOR_flag == FALSE)
     {
                /* allocate a line */
                new_t_ptr = rd_temp_str_info (dap_info_ptr,&EOR_flag, &EOF_flag);
                if (new_t_ptr != NULL)
                 {
                  length_so_far = length_so_far + strlen (new_t_ptr->tsi_str);
                  if (first_line)
                                {
                                 /* line is the first on the list so set appropriate ptrs */
                                 head_t_ptr = new_t_ptr;
                                 curr_t_ptr = new_t_ptr;
                                 first_line = FALSE;
                                } /* end if */
```

```
                else
                        {
                         /* link line to the rest of the line list */
                         curr_t_ptr->tsi_next = new_t_ptr;
                         curr_t_ptr = new_t_ptr;
                        } /* end else */

    } /* end if */

  } /* end while */

/* check for no input situation */
if (first_line && EOF_flag )
            {
             printf ("WARNING - number of requests read = 0!\n\n");
            }
else
            {
             /* allocate a request structure */
   new_req_ptr = dap_req_info_alloc();
   /* store head_t_ptr as the input request */
   new_req_ptr->dpri_in_req = head_t_ptr;
   new_req_ptr->dpri_req = var_str_alloc(length_so_far + 1);
   new_req_ptr->dpri_req[0] = '\0';
   curr_t_ptr = head_t_ptr;
            /* concatenate line list to form a request node */
   while (curr_t_ptr != NULL)
              {
               strcat (new_req_ptr->dpri_req, curr_t_ptr->tsi_str);
               curr_t_ptr = curr_t_ptr->tsi_next;
              } /* end while */

            /* capitalize the request DEBUGGER follows*/
   /*printf("\n%s\n", new_req_ptr->dpri_req);*/
   to_caps (new_req_ptr->dpri_req);
   new_req_ptr->dpri_req_len = length_so_far;
   new_req_ptr->dpri_next_req = NULL;
   if (first_req)
              {
               /* request is the first on the list so set appropriate ptrs */
               f_tran_info_ptr->ti_first_req.ri_dap_req = new_req_ptr;
               f_tran_info_ptr->ti_curr_req.ri_dap_req  = new_req_ptr;
               curr_req_ptr = new_req_ptr;
```

```
                first_req = FALSE;
               } /* end if */
        else
               {
                /* link request to the rest of the list */
                curr_req_ptr->dpri_next_req = new_req_ptr;
                curr_req_ptr = new_req_ptr;
               } /* end else */

        ++f_tran_info_ptr->ti_no_req;
       } /* end else */

   } /* end while */

/* TEST and trouble shooting code */
/* //////// to test what is read from file before sending to KMS ////////////

f_tran_info_ptr->ti_curr_req.ri_dap_req =
                    f_tran_info_ptr->ti_first_req.ri_dap_req;
while (f_tran_info_ptr->ti_curr_req.ri_dap_req)
{
 printf("%s", f_tran_info_ptr->ti_curr_req.ri_dap_req->dpri_req);
 f_tran_info_ptr->ti_curr_req.ri_dap_req =
                f_tran_info_ptr->ti_curr_req.ri_dap_req->dpri_next_req;
}
   ////////////////////////////////////////////////////////////////// */

#ifdef EnExFlag
  printf ("Exit f_read_file\n");
#endif

} /* end f_read_file */

                              f_read_terminal(dap_info_ptr)

struct dap_info *dap_info_ptr;

{

               /* This function prompts the user to input creates/queries */
               /* from their terminal
  */

#ifdef EnExFlag
```

```
  printf ("Enter f_read_terminal\n");
#endif

 /* set input device to be the terminal */
 dap_info_ptr->dpi_file.fi_fid = stdin;
 printf ("\nPlease enter your transactions one at a time.\n");
 printf ("You may have multiple lines per transaction.\n");
 printf ("Each transaction must be separated by a line that\n");
 printf ("\tonly contains the character '@@'.\n");
 printf ("After the last transaction, the last line must consist only\n");
 printf ("\tof the '$' character to signal end-of-file.\n\n\n");
 printf ("[7;7m Input the transactions on the following lines :[0;0m\n\n");

#ifdef EnExFlag
  printf ("Exit f_read_terminal\n");
#endif

} /* end f_read_terminal */



 static struct temp_str_info *rd_temp_str_info(dap_info_ptr,EOR_flag, EOF_flag)

                struct dap_info *dap_info_ptr;
     int   *EOR_flag,    /* boolean flags */
                   *EOF_flag;

{
                struct temp_str_info *temp_str_info_alloc(),
                                                *temp_ptr;  /* ptrs to new temp_str_info
structs */
     int        i, j;                                       /* counter */

                /* This routine fills an allocated line list node */
                /* and sends back a pointer to the node.        */

#ifdef EnExFlag
  printf ("Enter rd_temp_str_info\n");
#endif

 /* set a ptr to the allocated line */
 temp_ptr = temp_str_info_alloc();
```

```
/* now read in a line of input */
readstr(dap_info_ptr->dpi_file.fi_fid, temp_ptr->tsi_str);

/* now eat the leading blanks in line */
for (i = 0;(temp_ptr->tsi_str[i] == ' ' || temp_ptr->tsi_str[i] == '\t'); i++)
  ;
if (i)
{
  for (j = 0; temp_ptr->tsi_str[j+i] != '\0'; j++)
      temp_ptr->tsi_str[j] = temp_ptr->tsi_str[j+i];
  temp_ptr->tsi_str[j] = '\0';
}

/* now add a blank before '\0' */
for (i = 0; temp_ptr->tsi_str[i] != '\0'; i++)
  ;
if (i)
{
 temp_ptr->tsi_str[i++] = ' ';
 temp_ptr->tsi_str[i] = '\0';
}
temp_ptr->tsi_next = NULL;

switch (temp_ptr->tsi_str[0])
 {
  case EORequest: /* check for end-of-request ('@@') */
            *EOR_flag = TRUE;
#ifdef EnExFlag
  printf ("Exit1 rd_temp_str_info\n");
#endif
            return NULL;
            break;
   case EOFile: /* check for end-of-file ('$') */
                  *EOR_flag = TRUE;
                  *EOF_flag = TRUE;
#ifdef EnExFlag
  printf ("Exit2 rd_temp_str_info\n");
#endif
           return NULL;
           break;
   case '\0': /* check for empty line */
#ifdef EnExFlag
  printf ("Exit3 rd_temp_str_info\n");
```

```
#endif
            return NULL;
            break;
    case '/': /* check for comment line */
#ifdef EnExFlag
  printf ("Exit4 rd_temp_str_info\n");
#endif
            return NULL;
            break;
    default: /* create or request line */
#ifdef EnExFlag
  printf ("Exit5 rd_temp_str_info\n");
#endif
            return temp_ptr;
            break;
   } /* end switch */

} /* end rd_temp_str_info */
```

```
/*
 * File Name: req_const.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/req_const.c
 * Utility for ensuring queries are in the proper format
 */

#include <stdio.h>
#include <ctype.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"


                    f_fixup_ABDL(dap_info_ptr)


struct dap_info *dap_info_ptr;
{
 /* New for TI in MDBS - added to correct request syntax */

 int  i,
      req_len;
 char *req;

#ifdef EnExFlag
 printf("Enter f_fixup_ABDL\n");
#endif

 req_len = strlen(dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_req);

 req = dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_req;

 /* First, we must fix up the template name to be of the form */
 /* first letter caps, remainder lower case.                  */
 /* Find the first occurrence of "TEMP =" or "TEMP," in the request */

 for (i=0; req[i] != '=' && req[i] != ','; i++)
   ;
 i = i + 3;
 for ( ; req[i] != ')' && req[i] != '>'; i++)
   req[i] = tolower( req[i]);
```

```
/* Now search for other occurrences (i.e., retr-common request or */
/* a retrieve, delete or update with multiple conjunctions */

for ( ; (i+3) < req_len; i++)
  if ((req[i] == 'T') && (req[i+1] == 'E') &&
                (req[i+2] == 'M') && (req[i+3] == 'P'))
    {
     i = i + 8;
     for ( ; req[i] != ')' ; i++)
       req[i] = tolower( req[i]);
    }

/* now fixup attribute values which are literals */
req = dap_info_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_req;

for (i=0; (req[i] != '\0'); i++)
  if (req[i] == '\'')
    {
     ++i;
     ++i; /* skip first char, since it's already upper case */
     for ( ; (req[i] != '\'); i++)
       req[i] = tolower(req[i]);
    }

#ifdef EnExFlag
  printf("Exit f_fixup_ABDL\n");
#endif

} /* end f_fixup_ABDL */
```

```
/*

 * File Name: req_execute.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kc/req_execute.c
 * This file contains procedures which call upon and check the
   interface with the backends.
 */

#include <stdio.h>
#include <licommdata.h>
#include <dap_info.h>
#include "flags.def"


                        dap_req_execute(dap_ptr)

   /* This procedure accomplishes the following:              */
   /* (1) Sends the request to MDBS using TI_S$TrafUnit()     */
   /*     which is defined in the Test Interface.             */
   /* (2) Calls dap_chk_responses_left() to ensure that all   */
   /*     requests have been processed.                       */
   /* (3) Calls TI_finish() for post operation processing.    */


struct dap_info *dap_ptr;
{

struct ab_req_info* temp;
int msg_type=0;

#ifdef EnExFlag
  printf("Enter dap_req_execute \n");
#endif


   switch (dap_ptr->dap_operation)
     {
     case ExecRetReq:
     case ExecRetCReq:

        /* send request to MDBS */
     while(dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req != NULL)
     {
       if (msg_type != ReqsWithErr)
```

```
        {
         TI_S$TrafUnit(dap_ptr->dpi_curr_db.cdi_dbname,
                  dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_req);

         /* wait until request is completely processed */
         dap_chk_responses_left(dap_ptr);

         TI_finish();   /* tidy things after processing is completed*/
         msg_type = TI_R$Type();
        }
        temp = dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req;
        dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req=
         dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_next_req;
        free(temp);
       }

         break;

      case ExecInsReq:

/* If there are transactions to process continue */
      while(dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req != NULL)
      {
        if (msg_type != ReqsWithErr)
        {
         TI_S$TrafUnit(dap_ptr->dpi_curr_db.cdi_dbname,
                  dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_req);

         /* wait until request is completely processed */
         dap_chk_responses_left(dap_ptr);

         TI_finish();   /* tidy things after processing is completed*/
         msg_type = TI_R$Type();
        }

         temp = dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req;
         dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req=
          dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_next_req;
         free(temp);
       }
          break;

      case ExecDelReq:
```

```
    case ExecUpdReq:
      dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req =
         dap_ptr->dpi_abdl_tran.ti_first_req.ri_ab_req;
       while (dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req)
        {
         /* send each request to MDBS */
         TI_S$TrafUnit(dap_ptr->dpi_curr_db.cdi_dbname,
               dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_req);

         /* wait until request is completely processed */
         dap_chk_responses_left(dap_ptr);

         TI_finish(); /* tidy things after processing is completed */

       temp = dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req;
       dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req=
        dap_ptr->dpi_abdl_tran.ti_curr_req.ri_ab_req->ari_next_req;
       free(temp);
        }
       break;

    default:    /* This handles any errors */
       printf ("Error in dap operation type\n");
       break;

   }  /* end switch */


#ifdef EnExFlag
 printf("Exit dap_req_execute \n");
#endif

} /* end procedure dap_req_execute */
```

```
/*
 * File Name: short_get.c
 * Source: /u/mdbs/greg/CNTRL/TI/LangIF/src/Dap/Kms/short_get.c
 * This file contains procedures for retrieving information from the
   backends, placing responses into a linked list set, and returning
   to the requestor vice proceding to the KFS.
 *
 */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <licommdata.h>
#ifndef DAPLEX_INFO
#include <dap_info.h>
#include "flags.def"
#endif

                        short_term_get(dap_info_ptr, query)

 struct dap_info *dap_info_ptr;
 char *query;

{
 struct temp_str_info  *value,
                       *temp, *temp2,
                       *temp_str_info_alloc();
 int     OddMark = TRUE;
 int     msg_type,
         done,
         i;
 char    *response;

 /* values for these two depends on the defines in tstint.def (CNTRL/TI) */
 char    request[1002],   /* Added request length */
         err_msg[200];
 char    function[InputCols];
 struct  ReqId    rid;  /* Defined in licommdata.h */


#ifdef EnExFlag
```

```
 printf("Enter short_term_get\n");
#endif

if (dap_info_ptr->dpi_kc_data != NULL)
{
  value = dap_info_ptr->dpi_kc_data->tsi_next;

/* Free up any previously used response set spaces */
  while (value != NULL)
  {
   temp = value->tsi_next;
   free(value);
   value = temp;
  }
  free(dap_info_ptr->dpi_kc_data);
}
/* Start with a clean slate */
 temp=NULL;
 value=NULL;
 TI_S$TrafUnit(dap_info_ptr->dpi_curr_db.cdi_dbname,query);
 response = dap_info_ptr->dpi_kfs_data.kfsi_dap.kdi_response;

 done = FALSE;
 while (!done)/*Not all responses for the current request have been received*/
  {
   TI_R$Message();/*receive message from MBDS*/

   msg_type = TI_R$Type(); /* get the message type of the received message*/

   switch(msg_type) /* Is the response correct or are there errors? */
   {
     case CH_ReqRes: /* The response is correct */

      TI_R$ReqRes(&rid,response); /* Receive the results */

      done = f_chk_if_last_response(dap_info_ptr); /*Are we done */
      ++response;/*Skip initial [ */

      temp2=dap_info_ptr->dap_query;
      while (temp2!=NULL)
      {
        strtok(temp2->tsi_str,"(");
        temp2=temp2->tsi_next;
```

```
        }
        temp2=dap_info_ptr->dap_query;
        strcpy(function,temp2->tsi_str);

  while(*response != CSignal && strcmp(function,response))
  {
    response+=(strlen(response)+1);
  }

        while(*response != CSignal)
        {
            if (OddMark)
            {
             if(check_list(temp2,response))
             {
              response+=(strlen(response)+1);
              if (value==NULL || !check_list(value,response))/* Don't give repeat values */
              {
                value = temp_str_info_alloc();
                value->tsi_next=temp;
                temp=value;
              }
              OddMark = FALSE;
             }/*End if check list*/
             else
/* Else we skip the attribute name AND value we don't want*/
             {
              response+=(strlen(response)+1);
              response+=(strlen(response)+1);
             }/*End not if check list*/
            }/*End id OddMark*/
            else /* Not OddMark */
            {
              if (value==NULL || !check_list(value,response))
                 strcpy(value->tsi_str,response);
              response+=(strlen(response)+1);
              OddMark = TRUE;
            }/*End not OddMark */
       }/* End while not CSignal */
      break;
      case ReqsWithErr:

       TI_R$ErrorMessage(request,err_msg);
```

```
        TI_ErrRes_output(request,err_msg);
        return 0;
        break;

      default:
        printf("Illegal msg_type = %d received.\n", msg_type);
        break;

    } /* End switch */

  } /* End while not done */

 /* Reset the External from MBDS */
 dap_info_ptr->dpi_kc_data = value;

#ifdef EnExFlag
 printf("Enter short_term_get\n");
#endif

}
```

LIST OF REFERENCES

(Bane79) Banerjee, Jayanta, David K. Hsiao, and Krishnamurthi Kannan, "DBC--A Database Computer for Very Large Databases," IEEE Transactions on Computers, vol C-28, no 6, June 1979, pp. 414-429.

(Bour93) Bourgeois, Paul; *The Instrumentation of the Multimodel/Multilingual User Interface*, Master's Thesis, Naval Postgraduate School, March 1993.

(Elma89) Elmarsi, R., Navathe, S., *Fundamentals of Database Systems*, The Benjamin/ Cummings Publishing Company, Inc., 1989.

(Hall89) Hall, James E., *Performance Evaluations of a Parallel and Expandable Database Computer - The Multi-Backend Database Computer*, Master's Thesis, Naval Postgraduate School, June 1989.

(Hsia91) Hsiao, David K. and Kamel, M.N., "The Multimodel and Multilingual Approach to Interoperability of Multidatabase Systems," *International Conference on Interoperability of Multidatabase Systems*, Kyoto, Japan, April 1991.

(Hsia92a) Hsiao, David K., "Federated Databases and Systems - Part I: A Tutorial on their Data Sharing," *VLDB* Journal, 1, 1992, pp. 127-179.

(Hsia92b) Hsiao, David K., "Federated Databases and Systems: Part II- A Tutorial on their Resource Consolidation," VLDB Journal, 2, 1992, pp. 285-310.

(Kloe85) Kloepping, Gary R. and Mack, John F. , *The Design and Implementation of a Relational Interface for the Multimodel and Multilingual Database System*, Master's Thesis, Naval Postgraduate School, June 1985.

(Karl93) Karlider Turgay and Moore, John W., *Design and Implementation of an Object-Oriented Interface for the Multi-Model/Multi-Lingual Database System*, Master's Thesis, Naval Postgraduate School, March 1993.

(Lim86) Lim, Beng H., *The Impementation of a Functional/Daplex Interface for the Multi-Lingual Database System*, Master's Thesis, Naval Postgraduate School, June 1993.

(Meek93) Meeks, Andrew P., *The Instrumentation of the Multibackend Database System*, Master's Thesis, Naval Postgraduate School, December 1986

(Ship81) Shipman, D. W., "The Functional Data Model and the Data Language DAPLEX", *ACM Transactions on Database Systems*, 1981, 6(1), pp. 140-173.

(Watk93) Watkins, Stanley H., *A Porting Methodology for Parallel Database Systems*, Master's Thesis, Naval Postgraduate School, September 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library — 2
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. Ms. Doris Mlezko — 2
   Code P22305
   NAWCWPNS
   Point Mugu, CA 93042-5001

4. Chairman, Code CS — 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Professor David K. Hsiao, Code CS/Hq — 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. Ronald J. Roland — 1
   500 Sloat Avenue
   Monterey, CA 93940

7. Commander William A. Demers, USNR — 1
   405 Flamingo Road
   Slidell, LA 70461

8. Lieutenant Jon M. Rogelstad, USNR — 1
   301 3rd Ave SE
   Pelican Rapids, MN 56572