(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF A GRAPHICAL USER INTERFACE FOR A MULTIMEDIA DATABASE MANAGEMENT SYSTEM

by

Metin Balci, Erhan Saridogan

September 1992

Thesis Co-Advisors: Neil C. Rowe , C. Thomas Wu

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

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): CS37	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)
DESIGN AND IMPLEMENTATION OF A GRAPHICAL USER INTERFACE FOR A MULTIMEDIA DATABASE MANAGEMENT SYSTEM

12. PERSONAL AUTHOR(S)
Metin BALCI, Erhan SARIDOGAN

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 09/91 TO 09/92	September 1992	136

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author 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)
Multimedia Database Management System, Graphical User Interface, Retrieval, InterViews, Retrival by Natural Language Captions

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

The purpose of this thesis was to design and implement a graphical user interface for a multimedia database management system prototype previously implemented at the Naval Postgraduate School. Because of complexity of data types and difficulty of manipulating them, it was very hard for a casual user to use the previous system. Since graphical interaction simplifies control dialogue, we designed and implemented a graphical user interface using C++ and InterViews 3.0.1 for a Sun-3 workstation under Unix X-Windows environment with mouse support. We then connected this interface to the multimedia database system prototype. Our interface supports incremental query specification using extended SQL and can be connected to database applications in several different ways. An embedded global data structure, a text file or character string can be used for connections. A second version of the interface for a Sun-4 workstation was built and connected to another database system using the character string and text file. This version demonstrated even better performance.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Neil C. ROWE, C. Thomas WU	(408) 646-2462, 646-3391	CsRp, CsWq

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

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited

DESIGN AND IMPLEMENTATION OF A GRAPHICAL USER INTERFACE FOR A MULTIMEDIA DATABASE MANAGEMENT SYSTEM

by

Metin Balci
Lieutenant JG, Turkish Navy
B.S., Turkish Naval Academy, 1986

Erhan Saridogan
Lieutenant JG, Turkish Navy
B.S., Turkish Naval Academy, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1992

Authors: _____ _____
Metin Balci *Erhan Saridogan*

Approved By: _____
Neil C. Rowe, Thesis Co-Advisor

_____
C. Thomas Wu, Thesis Co-Advisor

_____
Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

The purpose of this thesis was to design and implement a graphical user interface for a multimedia database management system prototype previously implemented at the Naval Postgraduate School. Because of complexity of data types and difficulty of manipulating them, it was very hard for a casual user to use the previous system. Since graphical interaction simplifies control dialogue, we designed and implemented a graphical user interface using C++ and InterViews 3.0.1 for a Sun-3 workstation under Unix X-Windows environment with mouse support. We then connected this interface to the multimedia database system prototype. Our interface supports incremental query specification using extended SQL and can be connected to database applications in several different ways. An embedded global data structure, a text file or character string can be used for connections. A second version of the interface for a Sun-4 workstation was built and connected to another database system using the character string and text file. This version demonstrated even better performance.

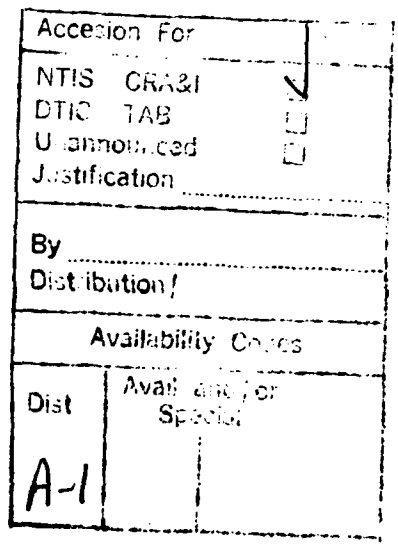

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Recently visual effects have gained much importance in both programming and interacting with the computers. With the help of rapidly improving technology, we can put highly detailed graphics on our computer screen instead of only text characters. This makes the computers definitely more user-friendly. In conventional programming, there is only a command line interface between the user and the computer, and the user has to type in words which may lead to input errors. The user must also know about the program and its flow and be aware of what will come next; this requires some training in typing the appropriate commands. However, with the help of a pointing device and a window environment, a program can be made very easy to use.

Today the expectations from computers are growing rapidly. Increasing speed and storage capabilities make the computers very useful in large database applications. Although there are some very good graphical user interfaces currently implemented for non-database applications, user interfaces of this quality do not exist for databases [PEA91]. The user interface is an important part of a database system especially when dealing with multimedia data. Due to different types of data and manipulating capabilities in a multimedia database management system, a casual system user needs an interface which provides an easy but efficient way to access the formatted and media data and to display them clearly. In this thesis, we present and implement a possible solution for a graphical user interface to make the query specification process easier.

A. BRIEF DESCRIPTION OF OUR WORK

A multimedia database management system prototype is being implemented at the Naval Postgraduate School (NPS). The existing prototype program [AYG91] was designed to work with command line interface. The user has to answer some questions, select menu

options and type in the conditions that are necessary to manipulate data. He has to remember which step he is in at any time and cannot see the whole query.

The goal of this thesis is to design and implement the retrieval function of a graphical user interface for a multimedia database management system which is easier to use for the casual users. We initially started our work by designing a partial interface for the existing prototype. We also connected our interface to another multimedia database management system [ROW92]. We added some extra features for this system and increased the reliability of the interface.

We designed our interface so that it is possible to connect the application program and the graphical user interface source code in three ways. These are:

1.) A record containing all the necessary information in order to construct a query.

2.) A C-type string containing extended SQL statements

3.) A text file containing the SQL statements.

We used the record interface to communicate with the prototype program [AYG91] and a string and text-file-connection system for the other database program [ROW92].

B. CHAPTER LAYOUT

Chapter II discusses the background and previous work in this area, briefly describing the existing multimedia database programs. We included some basic guidelines in designing user interfaces. We give examples of graphical user interfaces for database programs. Chapter III presents a detailed description of our implementation principles, design decisions and the assumptions we made. Chapter IV describes the program we wrote with the aid of the interface building tool. Chapter V has a discussion of the results of our thesis. Chapter VI provides a summary and some design recommendations with our proposal for future work.

II. PREVIOUS AND RELATED WORK

This chapter describes previous and related work in user interface design for a multimedia database management system. In order to understand the basics, first we will explain the general design principles for user interfaces and then give some examples from previous attempts which address the same subject.

A. USER INTERFACE DESIGN

User interfaces can be described as the parts of the computing systems that allow the person using the computer, access to the facilities offered by the computer. Since the facilities offered by the computer are increasing each day, the interfacing capabilities are becoming equally important. Researchers have shown that redesign of human-computer interface can make a substantial difference in learning time, performance speed, error rates, and user satisfaction. Although it seems impossible to get the best user interface at the very first time without any iterative design periods, there are some formal methods, which help the designer to implement interfaces correctly in accordance with the specifications worked out in advance, with the need for after-the-fact experimentation [THI90].

Since the purpose of computers is to benefit people, the designer should consider the system users at each phase of design and implementation. He should have a clear idea who the users will be and what their capabilities are. The goal is to run the computer as easy and effective as possible without special training of the user. Such a design will combine both psychology and computer science disciplines. Since these disciplines do have very different backgrounds, it is difficult to design a good interface which requires unification of these two perspectives.

Today's computer world has a great potential when of using a window environment and various pointing devices rather than the simple command line interface. Some operating systems are using graphical environments which make their applications more

user-friendly, and many application programs are also being written according to these graphical environments. Some examples are Macintosh, OS/2, Microsoft Windows, Unix X-Windows.

Some of the basics of user interface design have been discussed in [PEA91] and [GRP90]. We considered these design principles as guidelines in our work. We summarize these principles below:

1. Be Consistent

A consistent system is one in which the conceptual model, functionality, sequencing and hardware bindings are uniform and follow a few simple rules. The basic purpose is to allow the user to generalize knowledge about one aspect of the system to the other aspects. Consistency also helps to avoid the frustration induced when a system does not behave in an understandable and logical way.

2. Provide Feedback

The feedback is an indicator showing that the computer understands the given commands correctly. Some examples are displaying the position of the cursor and the pointing device, echoing the characters typed in, highlighting and echoing selections, displaying appropriate messages about the current status of the computer.

3. Minimize Error Possibilities

The user should not be allowed to make mistakes. Some examples for this rule are: Disabling some buttons when they should not be used, arranging the dialog boxes so that the user does not get confused.

4. Provide Error Recovery

It is proven that people are more productive if their mistakes can be readily corrected. With good error recovery, the user is free to explore unlearned system facilities, program capabilities without the fear of failure. This freedom encourages exploratory

learning. The user's trust of systems is fragile; one experience with unexpected results will undermine a person's willingness to use a system for a long time [SHN92].

5. Accommodate Multiple Skill Levels

The interface must provide enough features for novice users as well as efficient methods for experienced users. If the designer favors only one part, then the interface may not be so user-friendly for everyone.

6. Minimize Memorization

The user should not be forced to remember commands, what to do next and what has been done so far. In general, a graphical user interface is very effective for this guideline. Since every function can be implemented as a button or a menu selection, the user does not need to memorize any of the commands to run the program.

B. SIMILAR WORK

With increasing success of graphical user interfaces in today's computer applications, it was not difficult to find examples in almost every area of programming. Since multimedia database management systems are relatively new, we could not find a complete example, which would fullfil our needs. Other than commercial products, there are some previously done studies and implemented programs at NPS and other research centers. We will briefly introduce these studies.

1. Principles of Icons-Based Command Language [FRA86]

This study reviews and analyzes the usage of icons in database applications. It proposes an "iconic" language in order to provide an easier interface to the end-user who understands his own needs and wishes a direct interaction with the computer.

2. OdeView: The Graphical Interface to Ode [AGS91]

OdeView is the graphical front end for Ode, an object-oriented database system, and implemented using X-Windows and HP-Widgets on a Sun workstation running the Unix system in AT&T Bell Labs. OdeView is intended for users who do not want to write

programs in Ode's database programming language O++ to interact with Ode but instead want to use a friendlier interface to Ode.

3. A Graphical Dataflow Query Language [CLA91]

In order to find solutions to the problems of SQL, especially in the ease-of-use area, Dataflow Query Language has been implemented on an Apple Macintosh using an ORACLE relational database management system. It uses the Prograph language for the implementation.

4. A Multimedia Database Management System [GUG92]

This project has been implemented as a PHD study at N PS. It explores using captions for accessing the multimedia databases. Although the interface is not the main topic in this study, one is built to test the actual programs. It uses ProWINDOWS (an interface building program which is in Prolog) for the graphical user interface. It also uses digitization method to store the image files. This method enables having many image files in the databases.

5. Design of a Graphical User Interface for a Multimedia Database Management System [PEA91]

This study presents criteria and necessary features to evaluate and design a good graphical user interface for a Multimedia Database Management System. It has very nice features for an interface in design phase, but it requires a very powerful interface building tool which should provide an active dynamic graphical data input system. On the other hand, we thought that it was really difficult for a casual user to create a query by using this interface as it requires considerable knowledge about Entity Relationship Diagram and SQL.

6. A Friendly and Intelligent Approach to Data Retrieval in a Multimedia DBMS [KKL91]

This paper describes some design details and problems for data retrieval in a Multimedia DBMS. In order to make the query specification process easier, it also proposes a graphical user interface supporting incremental query specification and a natural way of join expression. They support special operators to define the query. We thought that their study was the closest one to our needs. Hence we used some ideas from this study, especially table and attribute name browsers, tool box, predefined join conditions and natural language editor are the important ideas that we used in our implementation. This paper was a good starting point for us in the design phase of the interface.

III. DESIGN DECISIONS

In this chapter, we will discuss our design decisions which were based on our initial considerations related to the existing multimedia database management system prototype [AYG91]. We will also discuss how we selected the working environment, which software selections were available, and how we designed for the interface.

A. SELECTION OF WORKING ENVIRONMENT

The existing prototype program [AYG91] was implemented for a Sun-3 type workstation with the Unix operating system. It was using an IBM compatible Personal Computer (PC) 286, equipped with a sound card to play the sound stored in the PC. Ethernet was used to pass the file name from the work station to the PC. The pictures were displayed in Unix SunView environment using some low level system functions. The main program was implemented in C programming language. Besides normal database functions, this program ran a Prolog program for data retrieval using natural language description in another Unix work station having the Prolog compiler loaded. During the initial study, we found that the SunView system was not suitable for a graphical user interface, as it required considerable amount of low level programming and did not have the available tools to achieve this goal. Therefore we searched for other available environments with which implement the interface.

1. Macintosh

A Macintosh computer could be used to build and run the graphical user interface and pass the required data to the work station. The pictures could be displayed in Unix work station using SunTools, and sound could be played in PC. In order to achieve this, the overall program would be running on three machines with three different operating systems. Incompatibilities such as connecting the Macintosh computer to Unix system, difficulties in communication of the user interface and database application programs and

portability problems can be listed as the disadvantages of this configuration. We could use Macintosh to store and display pictures, however since the existing code was written to store the pictures in the Unix work station, some part of it would be useless. In order to display image files on the Macintosh, we would need to use some commercial software which was not available in the department and also the picture files would take up too much space in the Macintosh secondary memory which is small when compared to the Unix system. The overall program would not be accessible from other Unix terminals. Programming in such a different environment would require extra skill in order to establish a connection between the different machines and use of different source codes without an appropriate software tool.

2. IBM or Compatible Personal Computer

Using an IBM or compatible personal computer would have the same disadvantages. Additionally, its graphics software was not as efficient as the Macintosh computer when we began our work.

3. Sun Workstations

Since the existing code was running mainly in Sun workstations, we concentrated on finding a solution which would combine every function of a multimedia database management system in a work station. We found that a special sound card could be installed into a work station to add sound capability. By using Sun workstations, we would deal with only one type of operating system so that we wouldn't have to establish a physical connection between the interface and the prototype program. We would be able to produce a single executable code which would be very helpful in debugging.

When we compare the choices, we realized that the best working environment for our work would be the Sun workstations. In addition to the advantages explained above, availability of many software packages and our familiarity of these machines led us to choose this environment.

B. SELECTION OF SOFTWARE

Several choices of software were available to us from the Computer Science Department at NPS. Brief descriptions of these follow:

1. SunTools

The existing database prototype was designed to run in the SunView environment. The picture display routines use some functions which are available only in SunView. The department did not have any convenient tool to build a graphical user interface with this environment. We would have had to implement a very low level programming using graphics primitives. As SunView does not provide a portable environment and does not have wide acceptance, we chose to look for another environment for our interface.

2. ProWINDOWS

ProWINDOWS is an interface building package which uses Prolog and also has an interface in C for applications. As indicated in Chapter II, an interface for a database system was implemented [GUG92] using this tool. ProWINDOWS does not have an interactive interface building tool, so instead the programmer must write the code to construct the graphics screen; it also not widely available.

3. TAE (Transportable Applications Environment)

TAE runs in the X-Windows environment and is available only on the Sun-4 type workstations. It could not be run on the Sun-3 workstations due to its limited portability.

4. InterViews 3.0.1

InterViews is a software system for window-based applications. It was developed at Stanford University by a group of researchers leaded by Mark A. Linton. We will explain its functions in Chapter IV. We selected this software system for the following reasons:

a. X-Window Environment

The X-Window environment has become a standard in computers, especially in Unix systems providing portability and compatibility for different types of computers. InterViews is built on top of X-Window environment.

b. Object-Oriented

InterViews is implemented with object-oriented approach. This approach makes the code generated by InterViews easier to understand and follow. Especially for implementing a graphical user interface, this method is very powerful and efficient. Each window is implemented as an object having its own methods for its functionality.

c. Graphical Object Editor

InterViews has a graphical object editor (Ibuild) which allows the user to build a graphical interface having dialog boxes, buttons or menus. Inside this builder the user can create an interface by drawing and placing the objects, then Ibuild generates the C++ source code with unimplemented member functions which are to be written according to the user's need later.

d. Ease of Connecting to an Application

After writing the appropriate code for the unimplemented functions, we could easily connect the source code generated by Ibuild to the existing C code of the prototype program with small changes.

e. Picture Display Capability

In the InterViews manual we found some examples of displaying image files. We could use these functions to display our picture files in the prototype program.

f. Availability

InterViews did not require permission to use and can be loaded and used on any machine. Our work required the InterViews to be run from a Sun-3 workstation.

Reconfiguration of a copy of the InterViews software for the Sun-3 workstations was feasible where for some other software was not.

g. Help

There was an InterViews group in InterNet news facility. We could communicate with other researchers working in the same area. Even though there was not sufficient documentation about InterViews and its general capabilities, a Reference Manual and a brief User Manual would be guiding us in building the interface.

C. IMPLEMENTATION DECISIONS

In our design an implementation of the interface, we used the basic guidelines which we explained in Chapter II.

We decided not to change the functional behaviors of the current prototype system because of complexity and integrity.

We worked only on the retrieval part of the existing code. We did not attempt to change the main algorithm or data structure of the current system. As we will propose in Chapter VI, a similar database management system could be built with less complexity by using a graphical user interface.

InterViews generates a source code in C++ and the existing code was written in C. We first tried to compile all the existing modules in C++, which is more strict in type checking. Some of the problems we encountered were type mismatches in function calls, inappropriate modularization, lots of cross dependencies of modules without having header files and many warning messages. We could not continue the process because of insufficient documentation of the existing code, so we decided to use the object codes produced by the C compiler and link them to our interface object code produced by the C++ compiler. This main restriction lead us to use a small interface module ("common.c" and its header file) which can be used by the two types of source codes.

After having the prototype program run with our interface, we rebuilt the interface for Sun-4 workstations and connected it to another database application [ROW92]. We will introduce this system in Appendix-A in detail.

D. SCREEN LAYOUT DESIGN DECISIONS

In order to make the interface more user friendly, we initially thought to put all of the functions into one big window. This would contain all functional capabilities of the program. We implemented the interface with this design on the Sun-4 workstation. We couldn't run it on the Sun-3 workstation because of hardware and operating system incompatibilities. Therefore we had to use another copy of the InterViews for Sun-3 workstations and recompile the same interface. This time, the large number of objects in the window caused system stack overflow which was impossible to correct. Then, as a major change in our design, we divided the window into smaller dialog boxes. We managed to run the system and continued the implementation.

In the existing prototype program, the command line interface was asking the user to enter data for the retrieval operation. It was processing the input data one at a time. The working principle is described in the reference [AYG91] with more detail. We did not want to establish a two-way communication between our interface and the prototype program source code, even though they were in the same environment. Instead we decided to build the whole query step by step and store the input data, when the query is completed, make a function call to the prototype code. We designed the interface so that the incremental user input is stored in the data structure, and when the query is processed, this data structure is accessed by the database system prototype functions.

As we indicated in Chapter II, it is not always possible to create the best interface at once. Ours was not an exception. We had to modify the interface several times as we faced new problems. However our screen layout was similar to the interface proposed in [KKL90]. Due to the difficulties in working environment of the Sun-3 workstations, we decided to concentrate on the second application [ROW92] and added extra features for

that. Since this interface is an improved version of the first one, we will explain it in Chapter IV with more detail.

E. ASSUMPTIONS

The following is a list of the assumptions we made in writing our program:

1. Hardware

We assumed that the overall system [AYG91] could be used with all capabilities only with an IBM compatible PC connected to the Unix system via Ethernet and a preferably colored Sun-3 workstation for the time being. Since the program was using Ingres libraries we had to use a compiler that generated Sun-3 type object code. In the second version we used a different compiler and a different database application running on the Sun-4 workstations.

2. Implemented Parts and Database

We assumed that the "Retrieval" function of a database management system is the most important part, so in our design and implementation we concentrated on only the main menu and the retrieval function, we left the other functions unimplemented. We presumed that the database already exists. We also designed our interface according to the available implemented functions in the existing prototype. We made small changes and added some additional functions in the second version.

3. User

Our interface was designed and implemented to be used by novice users. The user does not have to know a lot about database systems and was assumed that has not much experience on computers. Therefore we minimized using the keyboard which prevents incorrect entries. This also eliminates the necessity of checking each user input with the existing ones in many cases.

4. Future Work

We designed our interface so that an application program can use it with a small change in the source code. In order to use the interface with the data structure (q_rec) which is placed in a module (namely "common.h") the application program must be written in C/ C++. Other applications can use the C-type character string or the text file containing the query in extended SQL form. We assumed that future application programs will have these properties. Two of the connection types were implemented and tested in two versions of our interface.

IV. DESCRIPTION OF THE PROGRAM

In this chapter we will explain the implementation details of our program. We will cover the system requirements, the block diagram of the system, the data structures and the error checking facilities of the program, and also will give details about the components of the interface program.

Our program consists of two main parts. The first part deals with the screen layout and the data input, the other part deals with the operations to communicate with the actual database program. We used InterViews 3.0.1and C++ to implement these parts. We will separately discuss the implementation details and the usage of InterViews in this chapter.

A. SYSTEM REQUIREMENTS

Our program works in the AT&T Unix Operating System with the X-Windows environment having InterViews 3.0.1 in the file server. However, due to the properties of the X-Windows and InterViews, the program runs very slow on the Sun-3 workstations with AT&T Unix 4.0.3 operating system. We have two versions of the program for two different environments:

Workstation:	Operating System:	InterViews:	C++ Compiler:
Sun-3	Unix 4.0.3	InterViews 3.0.1	AT&T 2.0
Sun-4	Unix 4.1.1	InterViews 3.0.1	AT&T 2.1

Since the previously implemented multimedia database management system prototype [AYG91] runs only on the Sun-3 workstations with the Unix 4.0.3 operating system, we had use the same system to implement and run the user interface. During the initial tests we found that it was both very slow to run the user interface and work with it. We then divided the user interface screen layout into smaller parts so that it could run faster, but the result was still not satisfactory.

On the other hand, in order to demonstrate reusability, we tried a never version using a Sun-4 type workstation together with an updated operating system and compiler. This version was dramatically faster than the old one. We connected this interface to the Multimedia Database Management System Project which is explained in detail in reference [ROW92] and [GUG92].

B. PROGRAM INPUT AND OUTPUT

The graphical user interface we designed for the retrieval function of a multimedia database management system creates extended SQL queries and display the results. The interface can display image files and play sound in a PC as the media data.

We explain how to create a query by using this interface step by step in Appendix A. At this point if the reader is interested in how to run the program, he may skip to Appendix A and take a look at the examples. In general the user creates an extended SQL query statement either by using the mouse or by typing only the condition input in the editors. All of the information provided from the user is stored in a globai data structure named as "q_rec". This is the most important data structure of our interface for those application programs which can not handle direct SQL codes. Any program that wants to access a specific information about the query, can look up this embedded data structure and use the necessary parts for the retrieval operation. We used this data structure in order to communicate with the first application program [AYG91]. Since the latter had a command line interface, it was very useful to use the embedded global data structure to pass the input query and process it. When the user activates the query processing button, the database program which is designed for the command line input accesses the query specification from the data structure. With this method the original program looks up appropriate field of the data structure instead of asking it from the user. The results of the query are also passed from this record to the graphical user interface. When the query is processed from the interface, the function *Retrieve(MODE)* is called. After this function returns, the results are ready in the data structure, so the interface can read and display the tuples in the

"Result" window. It is also possible to display media data from this window if the related attributes are selected in the query. We chose this method in order not to change the flow of the original program.

For other kinds of applications that can manipulate direct extended SQL code as a character string or a text file we provided a file and a C-type string. We tested the string output "sql_string" with the application Multimedia Database Management System [ROW92],[GUG92]. We used text files ("sqloutput" and "pictureids") to read the results of the query and the retrieved picture file names.

The global data structure "q_rec", the file "sql_file" and the character string "sql_string" can be interpreted as the *outputs* of the query building part of the graphical user interface. The results can either be put into the data structure or into the text files. By using one of these outputs communication can be established with the actual database application program. We have tested both methods in two separate applications.

In addition to formatted data that is retrieved by a query, the user can select a line in the "Result" window and display the picture or play the sound of the selected tuple if this data is available. For test purposes we implemented both picture and sound display in the application program Multimedia Database Management System [AYG91] and only picture display in Multimedia Database Management System [ROW92]. We did not implement the sound playing in the latter as the actual database program is not capable of handling sound data at this time. We think that the latter, with the capability of handling sound, will be a very powerful and easy-to-use database management system.

C. MAIN PARTS OF THE PROGRAM

The Figure 4.1 shows the interface program divided into several parts. The boxes represent the modules, and the arrows shows the dependency between the modules. It is always possible to return to the parent module. Interaction with InterViews creates the objects and their source codes so that each module corresponds to a separate window. This will support the idea of modularization and information hiding.

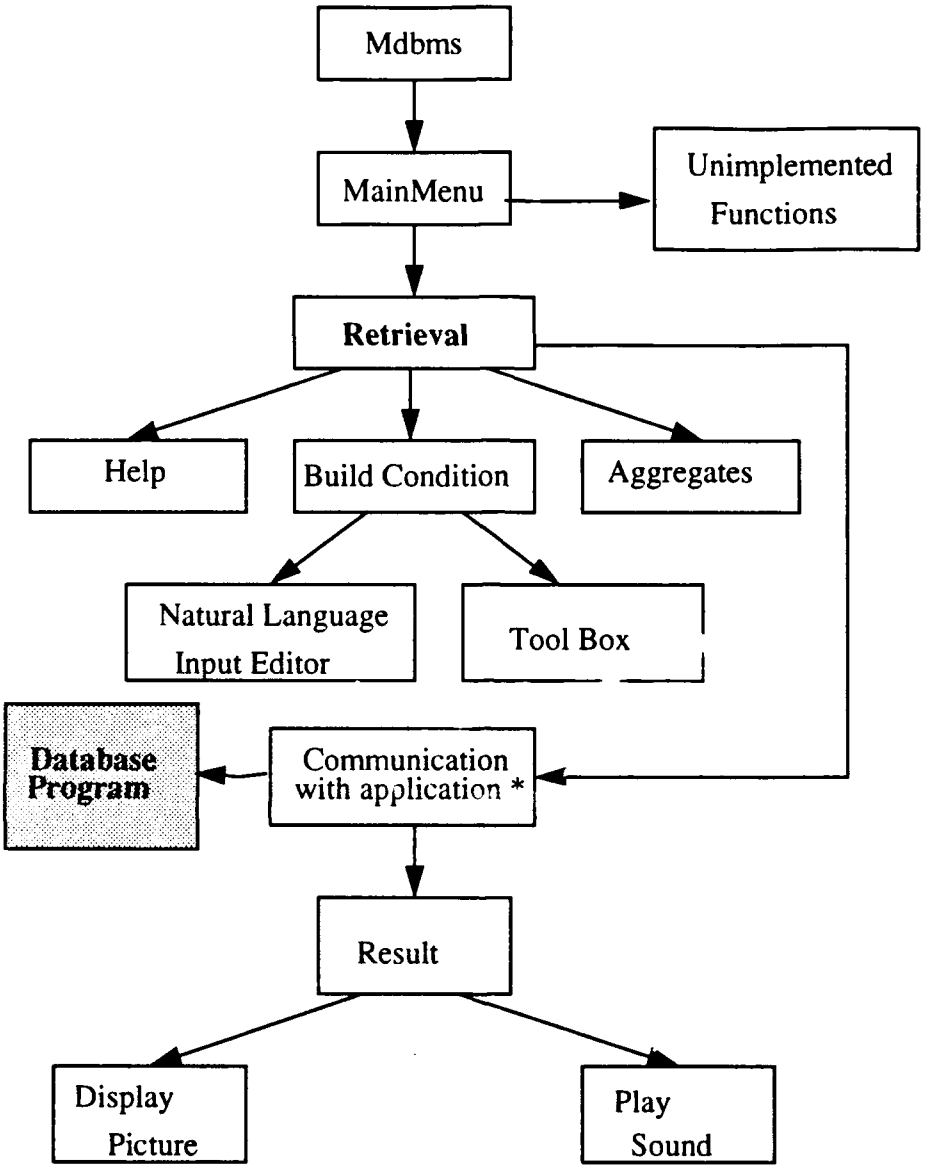

Figure 4.1 Main parts of the interface program * Application dependent

The module "Mdbms" is the main program. Most of the initializations and the database connections are performed in this module. Inside this module, the "Main Menu" dialog box which covers all of the basic database operations is created and displayed. We implemented only the "Retrieval" part of this menu. When a button other than "Quit" is

pressed the user gets a message saying this function is not implemented. The user can exit the program by pressing the "Quit" button in this menu. Figure 4.2 shows the "Main Menu" dialog box.

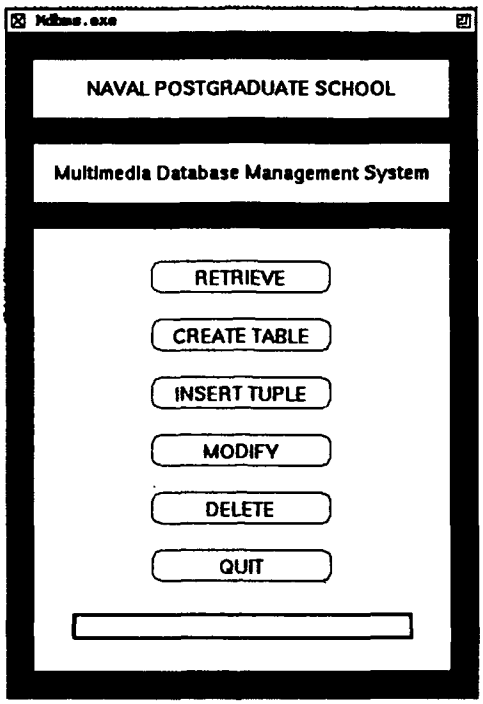

Fig. 4.2 Main Menu Dialog Box

"Retrieval" is a dialog box in which the user can select or type some values to build the query in extended SQL form and then process it. Extended SQL form is used to create queries which addresses media data as well as formatted data. Detailed explanation can be found in [AYG91]. "Retrieval" module is used to create "SELECT" and "FROM" parts of the SQL code. Additionally it contains two windows which are used to display the extended SQL query being built and the error or status messages. It is also possible to go back to the "Main Menu" from here. An on-line help is available whenever the "Help" ' utton is pressed. Figure 4.3 shows the "Retrieval" dialog box. We will explain how to use the buttons at the top and possible different combinations in Appendix A. When the

user presses the "Retrieval" button in the "Main Menu" the "Retrieval" and the "Aggregates" dialog boxes are displayed on the screen. The "Aggregates" box gives the user the choice of defining some aggregate functions before the attributes that will be selected. Figure 4.4 shows the "Aggregates" dialog box.

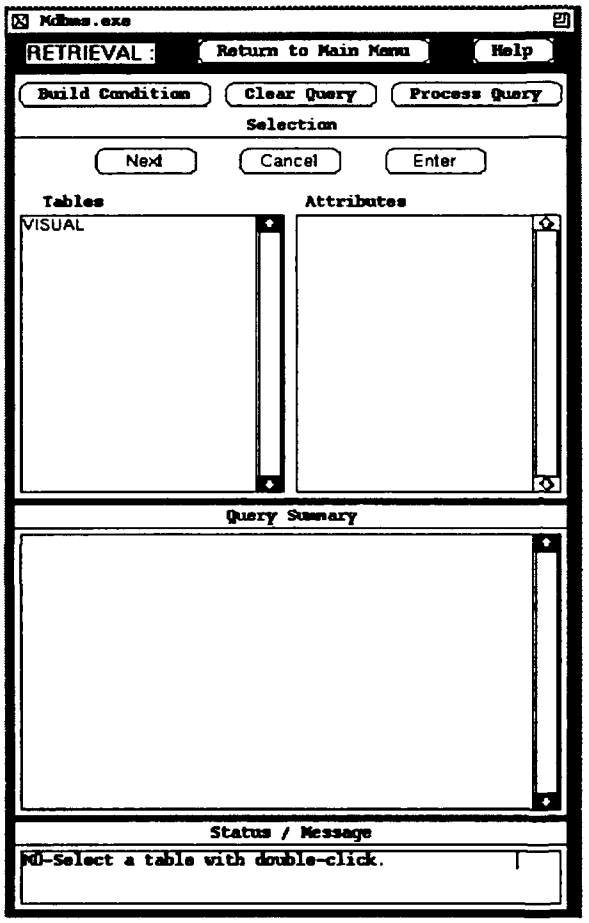

Figure4.3 Retrieval Dialog Box

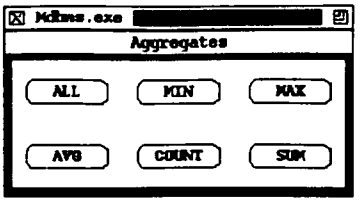

Figure 4.4 Aggregates Dialog Box

The buttons "Build Condition", "Clear Query" and "Process Query" in the "Retrieval" dialog box are related to whole interface program where the buttons "Next", "Cancel" and "Enter" belong to the "Selection" section. The functions of these buttons are explained with examples in Appendix A.

After the selection is completed, the user has the choices of either processing the query without any condition or adding a condition to the query. The "Build Condition" button displays another dialog box which we called "Build Condition" dialog box. Figure 4.5 shows this box.

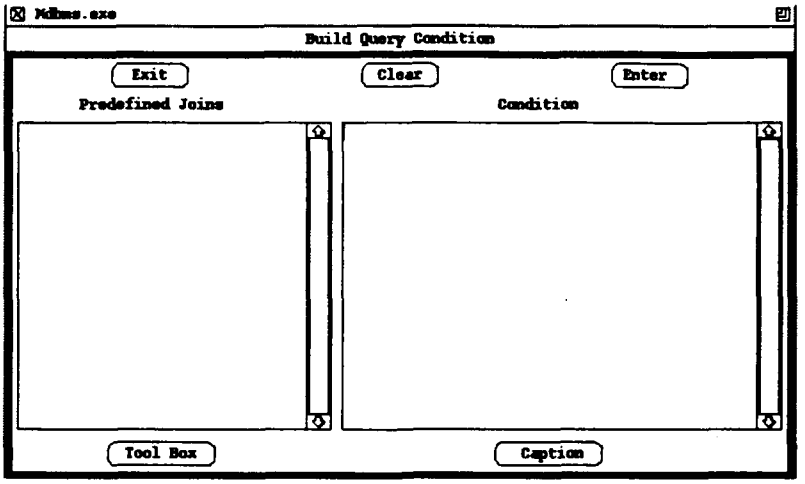

Figure4.5 Build Condition Dialog Box

This dialog box contains two main parts. The "Predefined Joins" section contains the previously defined join conditions that link the tables if there are more than one table in the database. The "Condition" section allows the user to create a condition for the query. The user can create a condition by typing it in with the editor or by selecting the tables, attributes and comparators one by one, or by some combination of both methods. Several conditions can be connected by using logical operators "AND" and "OR" together with parenthesis. The buttons on the bottom part are used to activate other dialog boxes. If the user is not sure what kind of tools are applicable, he can invoke the "Tool Box" (Figure 4.6) to see and use the available operators. This dialog box may be left on the screen until exiting from the "Build Condition" dialog box. Both of these dialog boxes may be left on the screen until returning to the main menu.

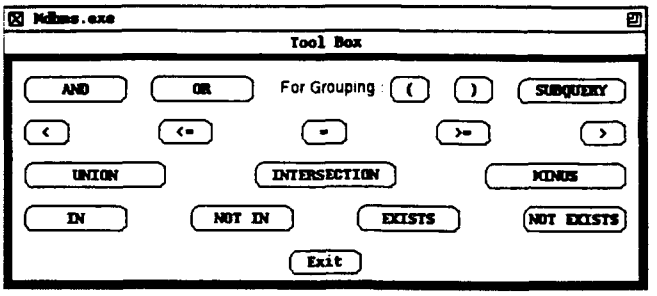

Figure 4.6 Tool Box

The "Natural Language Description" dialog box is one way to enter a natural language caption for an attribute. The user does not have to know the syntactical usage of the natural language description. This box will place the necessary keyword and quotations into the condition editor at the end of its operation. Depending on the implementation, a search type must be chosen. This button was required in the first version, but not in the second. Figure 4.7 shows the "Natural Language Description" dialog box.

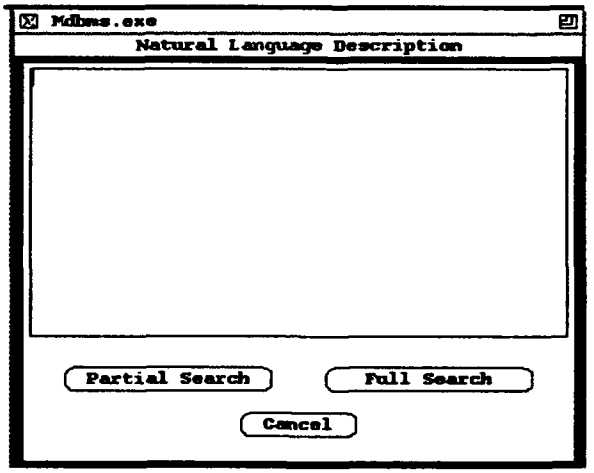

Figure 4.7 Natural Language Description Dialog Box

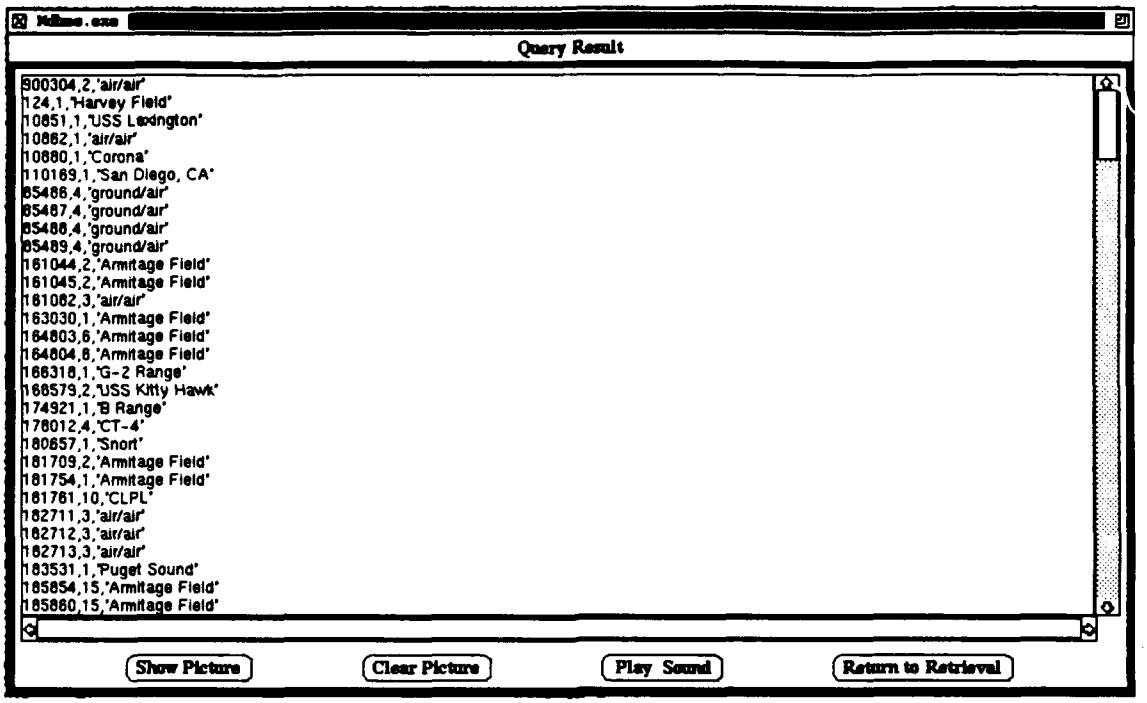

Figure 4.8 Result Window

The two Multimedia Database Management Systems [AYG91], [ROW92], previously implemented database programs, are indicated by the shaded box in Figure 4.1.

We wrote all of the member functions to perform the operations when a button is pressed and some communication modules; however, we used some other functions that we didn't write. One of them was a function which was available in InterViews 3.0.1 Reference Manual for displaying TIF-format files in the X-Window environment. We implemented the "Result" window [Figure 4.8] of our interface according to this picture display method. After displaying the pictures the user can clear them by using the "Clear Picture" button. In both systems the existing file formats were different than TIF-format and not compatible with our display method. Therefore we used another function called "qp_xloadimage(filename)" which was previously implemented by Gene Guglielmo and used in a similar way in his work [GUG92] to display the image files of various formats.

This function does not allow the user to use the picture clear button; instead the user must kill the picture window using the X-Window manager. We also used a special module implemented by Gene Guglielmo, "comm.c", which contains functions to establish communication between our interface and the latter database program.

D. DATA STRUCTURES USED IN THE PROGRAM

For modularity purposes, we put all of the global variables and data structures in a header file named as "common.h". We think that this module will help future programmers to go through our code easily.

The main data structure used in the program is "q_rec" which is a record type of QUERYRECORD. Figure 4.9 shows this data structure. As we indicated in Chapter III, we assume that the database management system for our user interface is based on the relational model. We designed our record so that it will cover all of the necessary information to build a query in the extended SQL form. We explain the fields of this record and the other data structures below:

1. The fields of the record "q_rec"

The field "q_table" stores selected table and attribute names in an array. It also has the information about the aggregate functions of the attributes. This field basically corresponds the "SELECT" and "FROM" parts of a SQL query.

The field "q_condition" stores the conditions in the query. It is possible to specify more than one conditions in this field. The condition part of the query can be divided into three logical parts. The first part is the name of the attribute, the second part is the condition tool and the third part is the condition input (right hand side of the condition tool). The fields "cond_table" and "cond_attribute" store the table name and attribute name respectively. Since we have media data in our database management systems, we added the field "cond_nat_description". This field stores the natural language caption which istyped in by the user and the type of search as either *full* or *partial* search. ererestructure

of the caption to be searched, an appropriate character must be inserted intothe field "search _type". The next field in the "q_condition" is the "cond_tool" which holds an operator. The field "cond_input" can be another attribute or a user input. The "cond_log_opr" is used to join separate conditions in the same query by using logical operators. The application program must be capable of distinguishing the logical operator precedence in these fields. The fields are filled by parsing the contents of the condition editor in the "Build Condition" dialog box.

The field "q_temp_table_name" stores the name of the temporary table name to be used in the database program for the nested query processes. In the second version of the interface, we are able to build a nested query in SQL form. If the application program can handle this extended SQL code, it will process multi-level queries. We believe that this would make a good thesis topic for future studies. We will also give some recommendations in implementing nested queries in Chapter VI.

The field "q_predef_join" stores the selected predefined join conditions in the query if there is more than one table in the database. The user can choose one predefined join between two tables to specify the relation between them. It is also possible to use more than one join condition in the case of using more than two tables.

The last field is "q_result". It stores the necessary results of the query. This field has three separate fields for formatted and unformatted (media) data. The prototype that we worked on [AYG91], can handle more than one picture or sound data related to the same tuple as the result of the query. In order to be able to perform this feature of the prototype, we designed the data structure so that more than one picture and sound can be stored for the same tuple.

2. The Other Data Structures Used in the Program

Besides the data structure "q_rec", we used some local array type data structures to temporarily store the user input before sending them to the"q_rec". In these data structures, which we called "buffer", we test the user input or parse the input string to send

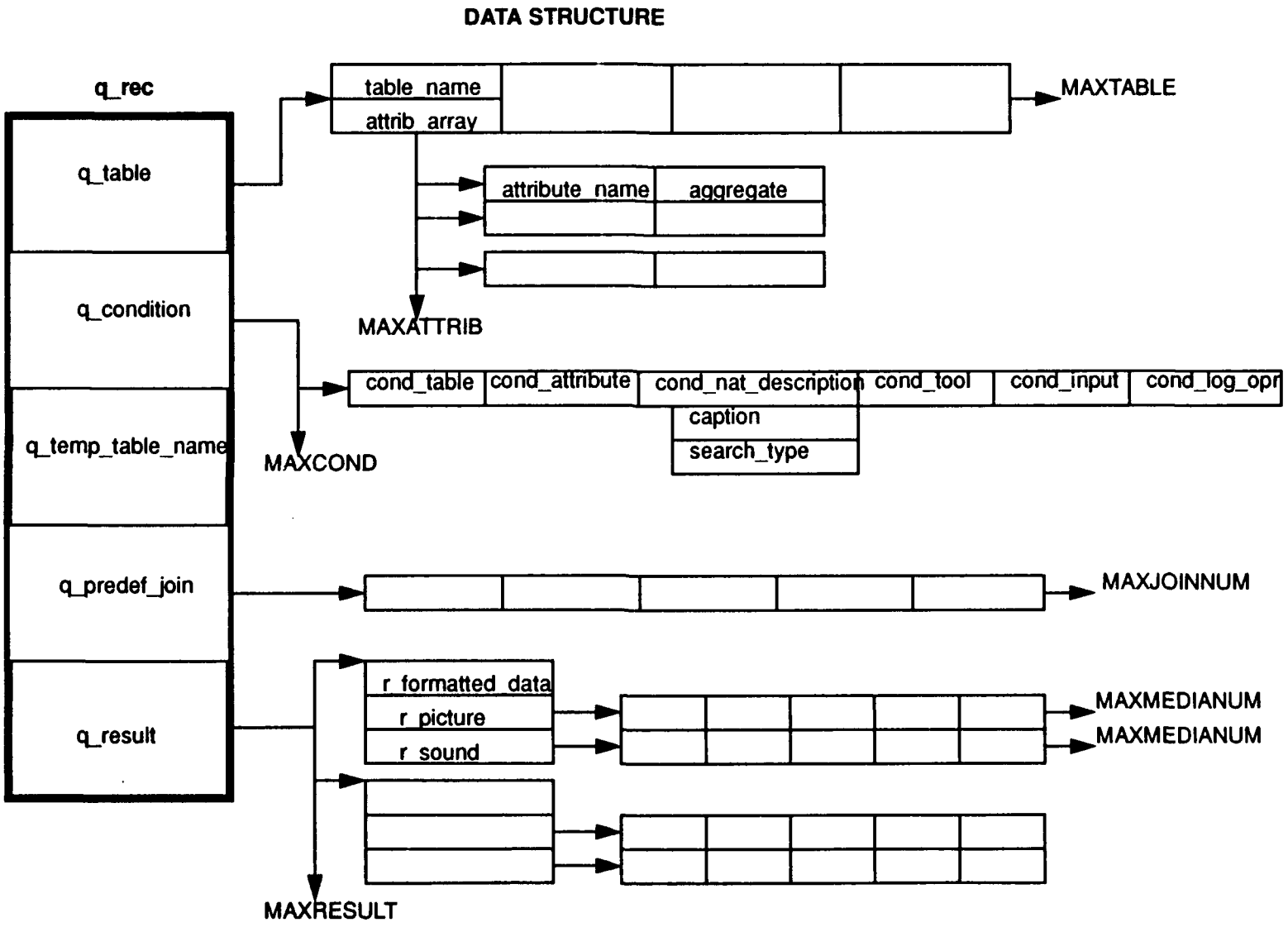

Figure 4.9 Global Data Structure "q_rec"

the information to the appropriate fields of the record "q_rec". In this way, we can keep our global record with correct values throughout the operations. If the input is invalid it is discarded and the user is informed.

Another important data structure we used is the C-type character string form of the created SQL query. Although the data structure itself is not complicated, it is very important as it is the main output of the user interface. This string can be used as an input to another database management program. We used this method in the second application program [ROW92]. The string is passed to a communication module and the module puts the string in an inter-terminal communication package and sends it to the application program. The same communication module returns a signal indicating that the search process is completed and the results are ready in the text files "sqloutput" and "pictureids". Then the interface can continue its execution by reading those result files and displaying their contents.

We used several global variables to keep track of some important values as well as some flag variables to enable or disable some operations. We used long but clear variable names in order to avoid confusion.

E. COMPONENTS OF THE PROGRAM

In this section, we will explain the components of our program one by one. It is assumed that the reader has a general idea about object-oriented programming, relational database management systems and multimedia database management systems.

1. General Issues

Learning about InterViews was the most important and time-consuming part of our work. We will briefly introduce InterViews and explain some important issues we used our implementation.

We used C++ for our implementation and C for the connection to the database programs. The first application program was written in C. Although C and C++ are supposedly compatible, we had some problems in the implementation process.

a. InterViews

InterViews is a software system for window-based applications. Like most user environments, InterViews is *object-oriented* in that components such as windows, buttons, menus, browsers or editors. The name InterViews comes from the idea of a user interface object presenting an *interactive view* of some data. It was developed in Stanford University with the support of Fujitsu America, Digital Equipment Corporation, and NASA CASIS project [IVR91]. We used the latest version, 3.0.1, which was released in October 4, 1991. The InterViews Reference Manual [IVR91] explains all the architecture, implementation techniques basic terminology and the functions of the base classes.

InterViews generates C++ code which can be compiled by any compiler that accepts the 2.0 or 2.1 revisions of the language, and can run on X11R4 or X11R5 X-Window environments. The graphical user interface consists of windows of many types and analyzes the input events of these windows in concert with the window manager. InterViews 3.0.1 contains Ibuild, a tool interactively building a user interface. Ibuild allows the user to arrange a common interactors and scenes, generate the C++ code for the interface, compile the code and execute the resulting application. The generated code defines a base class from which subclasses can be written to complete the program. this approach allows the interface to be modified later without affecting the subclasses [IVR91].

We will explain only those parts we used to build our interface and how to use InterViews. The programmer must have InterViews in the working path and appropriate environment settings must be done to have Ibuild run. It would be better to have a separate directory that contains the InterViews permanent tool files. In the working directory Ibuild is started by typing "ibuild" in the command line. The interactive building tool is displayed. The bottom part contains the basic classes so that placing it on the canvas

with the mouse creates an instance of that class. Therefore care should be taken when creating the objects. The top part contains the functions to manipulate the created objects. The properties of the objects such as size, color, class names, button state names can be changed by using these functions. The whole interface must be built in the same canvas by giving them appropriate window types. Once a window type is assigned to a box, a base class is created with that name and the related files take the same names. We used monoscene class for the Main Menu and dialog class for the other pop-up type windows. If a change is to be made to one of the objects in a window then that window must be dissolved until the level in which that object is created. After the change the window must be recombined until wrapping it with a window type. This process changes the window class names and the generated file names.

After all of the interface windows are created on the canvas, the source code can be generated from the "Files" menu. When the compilation and linkage is completed it is possible to run the resultant executable file which draws all of the interface windows on the screen. Since the functions are not implemented it is not possible to use the buttons, menus or the other interactive parts of the interface.

The generated files are based on the individual windows. Each window has a *core* file containing the implementation of the screen drawing with a header file, and a *source* file containing unimplemented member functions with its header file. These functions have the names that are given by the programmer in Ibuild during drawing. Other than these, two *makefiles* for command line compilation, *props* file containing the color description of each object, are generated. We first generated the files using a color work station in order to obtain color values in the properties files (in our work "Mdbms-props"). We then edited the files containing unimplemented functions and wrote the appropriate algorithms to perform the desired action when the related buttons are pressed. We needed to add some more member functions in order to provide reusability. We edited the Makefile (Mdbms-make) to insert "common.c". We also added the necessary lines to include the linkage of the object files of the database program generated by the C compiler.

We had to put Ingres libraries in the path defined in Mdbms-make. In order to provide automatic dependency checking, each included header file must be placed in the Makefile. Care must be taken to edit this file because it is very complicated and there is no documentation about it. There may be some unusual read errors if several changes are made at once. It is recommended that the Makefile should be tested after adding or changing a line. The command to compile the project is "make -f Mdbms-make". This will generate the executable file "Mdbms.exe".

There were some scope and visibility problems when we compiled the program, therefore, we had to edit the core and header files to make some variables visible. One of the most important changes was to declare a variable of type "TextBuffer" pointer for each text editor in the top of the related file, in global scope. This declaration is made in the body of the core file which the dialog box belongs to. Also we increased the sizes of the text buffers defined in these files. The main function was changed according to the initial window operations. The generated executable file can be run by itself from an X-Window. The InterViews libraries must be visible from where the program is launched.

We had to go through these steps each time when we made a change in the dialog box contents, such as adding another button, changing the size or changing a message. We needed to include many error handlers because InterViews does not have a default checking system.

We feel that InterViews is easy to work with once you become familiar with it; however, it still is difficult to manipulate the windows, browsers, buttons and editors. The reasons for this are that InterViews does not have a good user manual and these basic functions must be implemented by the programmer. It is also hard to make changes to an already-created interface. We wrote some comments in our source code describing the details of these operations.

b. C And C++ Compatibility Problems

When we tried to compile the C code of the first database system [AYG91] using the C++ compiler, we encountered so many errors that we had to give up. Some errors were missing parameters or type mismatches in function calls, invalid type conversion, invalid pointer type, declared but not used variables, multiply declared variables, visibility errors caused by not using header files, old type function declarations. These situations are ignored by the C compiler, but not by the C++ compiler which is more strict. We tried to correct these errors and warnings, we but couldn't solve all of them. Therefore we decided to use the object code of the prototype generated by the C compiler and call its functions from C++ programs. The functions that are written in C and defined in a C-module without a prototype must be listed as: "extern "C" { function list }". Even though we compiled "common.c" in C++ we used a C type header file "common.h" because it is used by the C program.

2. The Modules of The Interface

InterViews generates four files for each of the windows. These files, as described in the previous chapter, define the layout of the window (*window_name*-core.h and *window_name*-core.c), and their unimplemented member functions (*window_name*.h and *window_name*.c). The interface program as a whole is a combination of these windows together with the user interactions. We describe each module of the interface below:

a. Module Mdbms-main

This module contains the main function of the whole interface. It is used for calling the functions to establish the communication with the application program, displaying the "Main Menu", loading the database tables into the interface and ending the processes before quitting the program. We had only one main function in the first application, so there was only one executable file. In the second application, the user interface has its own main function where the database program has its own executable generated by Prolog compiler.

b. Module MainMenu

This module contains the functions to initiate the main database operations. We implemented only the retrieval function. When the "Retrieve" button is pressed, the "Retrieve" and "Aggregate" dialog boxes are displayed on the screen. The table names which exist in the database are listed in the "Tables" browser of the "Retrieval" dialog box. The text file containing the help information is read into a buffer in order to provide a faster display.

c. Module RetrieveDialog (_Dialog_9)

This module is the main dialog box for constructing the "SELECT" and "FROM" parts of a SQL query. Figure 4.3 shows this dialog box. There are two string browsers for "Tables" and "Attributes" of the database. Two editors are used only to display the SQL form of query summary and status/error messages respectively. There are many error checking facilities particularly to prevent incorrect button selections which will be explained later in this chapter.

The "Build Condition" button activates the "Build Condition" dialog box displaying the predefined join conditions. Once this window is activated it can be left on the screen until returning to the "Main Menu". Hence this button is disabled if the "Build Condition" dialog box is active.

The "Clear Query" button resets all of the editors, variables and data structures in order to restart building the query. The same resetting processes are performed when the user returns to this dialog box from the "Result" window.

The "RetrieveDialog" module performs the communication with the application program when the "Process Query" button is pressed. For the first application we call the "Retrieve(MODE)" function from the program. We modified this function so that it loads the results into the global record "q_rec". We then create the "Result" dialog and call the "DisplayResults()" function to display the results in this window.

For the second application, pressing the "Process Query" button creates the "sql_string" by reading the "Query Summary Display", calls the function "comm_with_db(sql_string)" to establish the communication with the database and reads the result files (sqloutput and pictureids) into the global record "q_rec" after the search operation is completed, then it displays the "Result" window showing the results.

d. Module Aggregate Dialog (_Dialog_10)

This module contains the aggregate functions for the SQL query. When a button is pressed, the related aggregate function is added to the selected attribute in the form "aggregate_function(attribute)". We did not handle these functions in the first application program. We will give an example to this feature for the second application in Appendix A.

e. Module Build Condition (_Dialog_11)

This module contains the necessary functions to build the condition part of the query. The function related to the string browser displays the predefined joins and gets the user's selection. An editor and its input handling function is provided to let the user type in the condition. This editor is also used to display the condition being built. The other functions regulates the button operations.

There are five buttons on this dialog box. The "Clear" button is used to delete the condition from the editor. When the "Exit" button is pressed the condition is discarded and the dialog box will be closed. The "Enter" button first calls the *condition parser*, which checks the syntax and sends the information to the global record "q_rec". After the parsing is completed, the contents of the editor is sent to "Query Summary Display" to form the "WHERE" part of the query. If an error is detected during the parsing operation an error message will be displayed in the "Message/Status" box.

The parser that we wrote can be improved for additional features. If the application program already has an error checking mechanism and it does not use the global data record "q_rec", then it is not necessary to use this parser. In this method, it is not possible to check and correct only the condition part. In case of an error in the SQL code

the whole query must be rebuilt. We used this method in the second application program [ROW92].

The "Tool Box" and the "Captions" buttons activate the "Tool Box" and the "Natural Language Editor" respectively. When the user presses the "Exit" button on the "Build Condition" dialog box, these boxes will also be closed. These two dialog boxes are optional to use. The user can simply type the condition in the same format into the editor.

f. Module Tool Box Dialog (_Diallog_12)

The "Tool Box" dialog box provides the opportunity of seeing and using available tools in the interface. When a button is pressed, the selected tool is inserted into the condition editor. We think that it is a very good feature particularly for the new users of the program, and the user will not have to memorize the available tools in the system. Some of the tools defined in this box are implementation dependent as indicated in the source code (_Dialog_12.c).

g. Module Natural Language Dialog (_Dialog_8)

This module creates an editor (Figure 4.7) which is used to type the natural language description of the unformatted data. The editor uses an input handling function which provides the standard editor commands. In order to accept the contents of the editor the user must press one of the buttons defining the search type. Depending on the syntax rules of the implementation some keywords are inserted into the condition editor. The "Cancel" button simply deletes this window.

h. Module Help Dialog (_Dialog_7)

This module contains the implementation of the "Exit" button and the text editor functions to display the help text reading from the buffer.

i. Module Result Dialog(Result)

This module is used to display the result of the query. When the results are displayed on the screen the user can choose one tuple by double-clicking on that line. Once

the user selects the line, the corresponding image and sound data can be displayed if any available. In this module, there are two different functions for displaying the image files. The first one (load(*filename*)) is taken from the InterViews 3.0.1 Reference Manual and it can display only image files in TIF-format. It has also some sub-functions that resize the picture. The second function is "qp_xloadimage(*filename*)". This function displays most of the common image formats. Since existing database for the first application program contains image files in "sun-ras" format, the function "load(*filename*)" is only used for test purposes. Due to the software restrictions we could not use the "qp_loadimage(*filename*)" function for this application. In the second application program, we used "qp_loadimage(*filename*)". This function requires the X-Window manager to clear the picture; therefore in this system the user can not use the "ClearPicture" button.

j. Module common (common.h and common.c)

This module contains the global variables, data structure and functions used throughout the program. Any programmer that might wish to change our code should look at this module at the very beginning of the study. The file "*common.c*" contains the implementation of global functions which are used to clear records and load the database information to the interface.

k. Module comm (comm.c and db_file)

The file "comm.c" contains the functions to establish the connection between the user interface program and the second application program. It is compiled using the C compiler and linked to the others. The file "db_file" contains the communication parameters and is located at "~guglielm/marie/etc". The detailed explanation about this communication method can be found "Quintus Prolog User Manual".

F. ERROR-CHECKING FACILITIES OF THE PROGRAM

We wrote our program as a prototype, but we included several error-checking facilities and instructions so that a novice user can use the program without crashing or

loosing important data. The user must push the buttons in the correct order. When an incorrect button is pressed, an error message is displayed in some cases to give an instruction to the user. In other cases we have disabled buttons when they are not to be used. For example, if the user presses the "Enter" button of the "Retrieval" dialog box without selecting an attribute, a message will be displayed saying "There is no selected attribute. Please select an attribute to continue.". If the user has activated the "Build Condition" dialog box, the "Build Condition" button is disabled (shaded) to prevent creating multiple dialog boxes.

Another aspect of error-checking is on the correct form of the query condition. We wrote a parser to check the syntax of the condition and warn the user in case an error is detected. If the application program has such an error-checking facility, then the parser is disabled and the whole SQL query is checked in the application program and the error messages are displayed in the "Result" window. The error and status messages that we use can be found in the module "Retrieve Dialog (_Dialog_9.c)".

V. DISCUSSION OF RESULTS

Our user interface provides an easier query construction using graphical selection methods. We can list the benefits of using our interface as follows:

1.) The tables and attributes are ready to be selected at all times. The user does not have to memorize these names in order to construct a query.

2.) The selection methods prevents the user from entering an incorrect value.

3.) Predefined join conditions provides an efficient way of using multiple tables in a query condition.

4.) The "Query Summary" displays the SQL query being constructed.

5.) The natural language editor inserts the system dependent keyword into the condition. This feature is helpful for the new users.

6.) The media data as well as formatted data can be displayed easily and clearly.

7.) The available functions and tools are displayed so that the user can simply select or activate them.

8.) The instructions and error messages help the user to navigate through the query construction and execution.

9.) The user has the flexibility of canceling a process or changing a value during the query construction.

10.) The on-line help is available at any time.

The above features are common in all versions of the interface. However there are considerable speed differences between the first and second version. We discuss the performance of our interface for each version.

A. VERSION 1

This version is connected to the existing multimedia database management system [AYG91] and runs on a Sun-3 workstation. Due to the old architecture, operating system and C++ compiler used on the Sun-3 workstations, the performance of this version was not satisfactory. Table 5.1 shows average time values.

Table 5.1 TIMING VALUES OF THE FIRST VERSION

Functions	Time
Initialization and Connection	45 seconds
Main Menu Window	2 minutes
Retrieval Window	2 minutes
On-line Help	1.5 minute
Build Condition Window	1.5 minute
Tool Box	2 minutes
Caption Editor	45 seconds
Result Window	1.5 minute
Displaying an image file (20 kilobyte, TIF format)	2 minutes
Resizing an image window (enlarge 50%)	2 minutes
InterViews code generation and compilation	18 minutes
Recompilation using Makefile	13 minutes
Linking all the modules	9 minutes

These values show that this environment was not suitable for software development because we spent a lot of time during debugging. To see the effects of a change in the source code cost sometimes 30 minutes. We could not increase this performance of the system. Table 5.2 shows the storage amounts of the files.

Table 5.2 STORAGE AMOUNTS OF THE FIRST VERSION

Files	Amount (KiloByte)
Original database program source code	520
Source code generated by InterViews	225
User interface executable (Mdbms.exe)	590

Because of low performance and lack of portability, we decided to concentrate on building a better, reusable, faster and more reliable graphical user interface for any multimedia database management system.

B. VERSION 2

We regenerated the interface source code using the InterViews for the Sun-4 workstations and connected to another database application [ROW92]. Table 5.3 shows the average starting time values for this version on a Sun-4 workstation.

Table 5.3 TIMING VALUES OF THE SECOND VERSION

Functions	Time
Initialization and Connection	45 seconds
Main Menu Window	10 seconds
Retrieval Window	20 seconds
On-line Help	30 seconds
Build Condition Window	10 seconds
Tool Box	10 seconds
Caption Editor	2 seconds
Result Window	5-30 seconds *
Displaying an image file (20 kilobyte, TIF-format)	2 seconds
Resizing an image window (enlarge 50%)	Not Applicable
InterViews code generation and compilation	6 minutes
Recompilation using Makefile	4 minutes
Linking all the modules	4 minutes

* Depends on query process time

This performance increase encouraged us to improve our interface. The additional features we added were:

(1) Another window containing aggregate functions.

(2) Modification of the condition editor parser for additional syntax rules.

(3) An appropriate error checking system to prevent crashing in case of using an incorrect button.

(4) A "C" type character string and a text file containing the query in extended SQL form to establish communication with different database programs.

(5) Instructions to help the user to build and process the query.

(6) A different picture display function which supports various image file formats.

Table 5.4 shows the storage amounts of this version

Table 5.4 STORAGE AMOUNTS OF THE SECOND VERSION

Files	Amount (KiloByte)
User interface source code	265
User interface executable	1057
Main-memory database system	1400

C. VERSION 3

This version is almost the same as the second one except it can use natural language captions to retrieve data. The overall system runs as four separate processes. The major change in this version is disabling the condition input parser. Instead the query parser of the database program written by Professor Rowe checks the SQL code. If the code is invalid then an appropriate error message can be seen on the "Result" window. The timing values are the same as the previous one only with some additional setup time to initialize three other processes before starting the user interface. If a natural language caption is used in a query, additional time will usually be needed to search and display the results in the "Result" window. Table 5.5 shows the storage amounts of this version.

Table 5.5 STORAGE AMOUNTS OF THE THIRD VERSION

Files	**Amount** (KiloByte)
User interface source code	265
User interface executable	1057
Main-memory database system	1400
Natural language parser executable	1640
Fine search executable	1040
Coarse search executable	1070
Scheduler executable	1000

VI. CONCLUSION

A. MAJOR ACHIEVEMENTS OF OUR PROGRAM

One important achievement of our thesis is the efficient method for the retrieval of multimedia data by using a graphical user interface with a pointing device. The user simply selects the necessary information from the browsers, push buttons or type a condition. The constructed query can be passed to a database program using an embedded data structure or a character string and text file.

We built a test program for the existing multimedia database management system [AYG91] which can retrieve and display image and sound. We then connected an improved version of the interface to another multimedia database system [GUG92]. This system can also retrieve data using natural language captions.

B. COMPARISON

A different interface implemented by Prof. Rowe and Gene Guglielmo [ROW92] is capable of retrieving pictures by using natural language captions. There are two major differences between this interface and ours. First our interface can be used with multiple tables and predefined join conditions using separate windows. Second our interface can be used by any database program with the three ways of communication methods.

The user interface proposed by Charles Peabody [PEA91] was to use an interactive graphical query representation which requires more graphics on the screen. We used a text-based query representation so that the implementation can easily be modified according to the new design decisions. Also our user does not have know much about database diagrams.

C. SUGGESTED FUTURE WORK

With the increasing speed of computing machines, it is getting easier to access large databases and manipulate data. The problem now is how to make the database operations

easy to use and effective. We think that such a database management system will require the graphical user interface and the database program to be designed and implemented together.

We believe that our work can be used as a base to implement the other database functions like insert, update or delete. InterViews can be used to build a similar user interface by using our work as an example. We describe the major weaknesses and unimplemented parts of our interface below. Even though these points do not affect the running of the interface, we believe that they will help future designers.

1. We included only those functions which were already implemented in the application programs. Even though we used extended SQL to create queries, we did not use all of the functions in standard SQL.

2. We included the buttons for nested queries, but we did not implement and test them with the database programs. Such a query can be constructed in SQL and if the database program is capable of handling this SQL code at once, all the query can be passed to the program using the string or text file. In this method our interface can generate any level of nested queries without using the data structure "q_rec". If the application program is designed to use this data structure, then a list of data structures of type "QUERYRECORD" should be created. Then the application program can access the fields of these records in a an appropriate order.

3. We do not allow the user to enter a table or attribute name by typing it. This would require some extra error-checking facilities.

4. We do not save a query after it is constructed and processed. Therefore if the results are not satisfactory the user must build the query from the beginning. One way of implementing this feature is to save the query in a data structure and then load it when asked. We believe that being able to save the query will increase the user-friendliness of the interface. Some of the queries could be saved as a graphical icon so that user would have a chance of using it easily.

5. We do not use any terminal sound to warn the user. Also we did not use separate pop-up windows showing the error messages.

6. The user cannot change any attribute of the screen layout (color, size, font etc.) except moving the windows or iconizing them using the X-Windows manager.

7. In the "Retrieval" dialog box the selected attributes are not left highlighted due to the properties of InterViews. Instead we display the attributes in the "Query Summary Display" section when they are selected one by one.

8. In the result window we do not display the selected attributes in columns. This feature mostly depends on the application program. The result window may be designed so that the results are displayed under their corresponding attributes.

9. The parser we used in the condition input editor can handle only certain situations. It cannot detect words or operators without spaces before and after. Depending on the type of new query-building features, this parser should be modified.

10. The arrows on the keyboard cannot be used to move the cursor in the editors; instead the mouse and its left button can be used.

11. We did not use the "Return" key to activate a default button or to enter the contents of the editors.

12. We do not store the windows as bitmaps in the memory; therefore it takes some time to draw the graphics screen each time. However, the timing is acceptable.

13. In the first version of our interface, we cannot handle group conditions. In order to have this feature either data record "q_rec" should be modified to include the parenthesis around the conditions or application database program should have a precedence among the operators. Since the record "q_rec" is not used in the second and the third versions of the interface, we can build queries with group conditions successfully.

LIST OF REFERENCES

[AGS91] Agrawal R., Gehani N. H., Srinivasan J., *"OdeView: A User Friendly Graphical Interface to Ode"*, SIGMOD'90, Volume 19, Issue 2, June 1990

[AYG91] Aygun H., *"Design And Implementation of A Multimedia DBMS: Complex Query Processing"*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, September 1991

[CLA91] Clark G.J., *"DFQL: A graphical Dataflow Query Language"*, Master's Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, September 1991

[FRA86] Frasson C., Er-radi M. *"Principles of Icon-Based Command Language"*, SIGMOD'86 Volume 15, Number 2, June 1986

[GRP90] Folley J., van Dam A., Feiner S., Huckhes J., *"Computer Graphics Principles and Practice"* Addison-Wesley, 1990

[GUG92] Guglielmo E. J., *"Intelligent Information Retrieval for a Multimedia Database Using Captions"*, PH.D. Thesis, Naval Postgraduate School, Department of Computer Science, Monterey, California, July,1992

[IVR91] Linton A. M., Calder P. R., Interrante J.A, Tang S., Vlissides J. M., *"InterViews Referance Manual Version 3.0.1"*, The Board of Truestees of the Leland Stanford Junior University, October 4,1991

[KKL91] Keim D. A., Kim K., Lum V. Y., *"A Friendly and Intelligent Approach to Data Retrieval in a Multimedia DBMS"*, Naval Postgraduate School, Monterey, CA, March 1991

[PEA91] Peabody, C., *"Design and Implementation of a Multimedia Database Management System: Graphical User Interface Design and Implementation"*, Master'sThesis, Naval Post Graduate School, Department of Computer Science, Monterey, California, September 1991

[ROW92] Rowe N. C., Guglielmo E. J., *"Exploiting Captions for Access to Multimedia Databases"*, to be published in Information Processing and Management 1992

[SHN92] Shneiderman B. *"Designing the User Interface"*, Addison Wesley,1992

[THI90] Thimbleby, H. *"User Interface Design"*, Addison Wesley, 1990

APPENDIX - A

HOW TO USE THE INTERFACE

We will explain how to use the interface for the two application programs in two separate sections. In these sections, we will give sample sessions for the database programs [AYG91] and [ROW92].

A. EXAMPLES FOR THE FIRST APPLICATION [AYG91]

The source codes and the picture files of this system are in the "~mdbms/MDBMS" directory of the "virgo" file server. To run the program the user must first connect to this file server from the X-Windows environment, change the directory and then type in "Mdbms.exe". If the user wants to play sound data then the PC must be started as explained in [ATT91] and [AYG91].

The following simple query will be used as an example for this database.

QUERY:

Retrieve the names, ranks, salaries and pictures of the officers whose salaries are greater than $5000.

SQL FORM:

```
SELECT o_name, rank, salary, o_picture
FROM   officer
WHERE  officer.salary > 5000
```

EXECUTION:

1. When the program starts it establishes the connection to the Ingres database and displays the "Main Menu" dialog box. Press "Retrieve" on this menu.

2. When the "Retrieval" dialog box is displayed with all the database tables listed in the browser, select "officer" from the "Tables" browser by double clicking on the left mouse button. This will highlight the selection and the attributes of the "officer" table will

be displayed in the "Attributes" browser and the selection will be echoed in the "Query Summary Display". Then select the attributes by double clicking on them.

3. The "Cancel" button can be used to cancel this selection operation. Since there is only one table for this query, after the selections are completed press "Enter". The display will show the SELECT and FROM parts of the query.

4. Press the "Build Condition" button to activate the "Build Condition" dialog box.

5. The condition can be typed in with the editor while the mouse cursor is inside the frame, or the "Selection" part of the "Retrieval" dialog box can be used to select the attributes. For this example select "officer" from the "Tables" and then select "salary" and press "Enter". The condition editor will display "officer.salary".

6. According to the syntax we used for the conditions, a space must be left before and after the condition tool. This tool can be chosen from the "Tool Box" or typed in. Activate the "Tool Box" and select " > " for this example.

7. Type 5000 for the condition input.

8. Figure A.1 shows the "Build Condition" dialog box at this point.

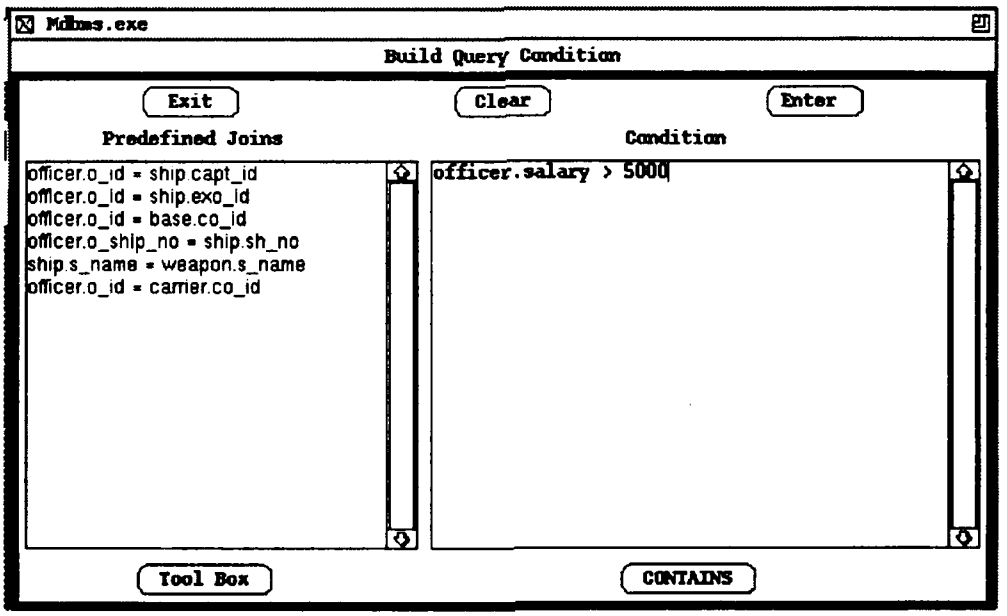

Figure A.1 "Build Condition" dialog box

9. If the condition is not correct press "Clear", otherwise continue by pressing "Enter". This will send the condition to the WHERE part of the query in the "Query Summary Display".

10. The query can be processed now. Figure A.2 shows the "Retrieval" dialog box at this stage. Check the query in the display. If it is incorrect or not satisfactory use the "Clear Query" button to delete this query and start again from the beginning.

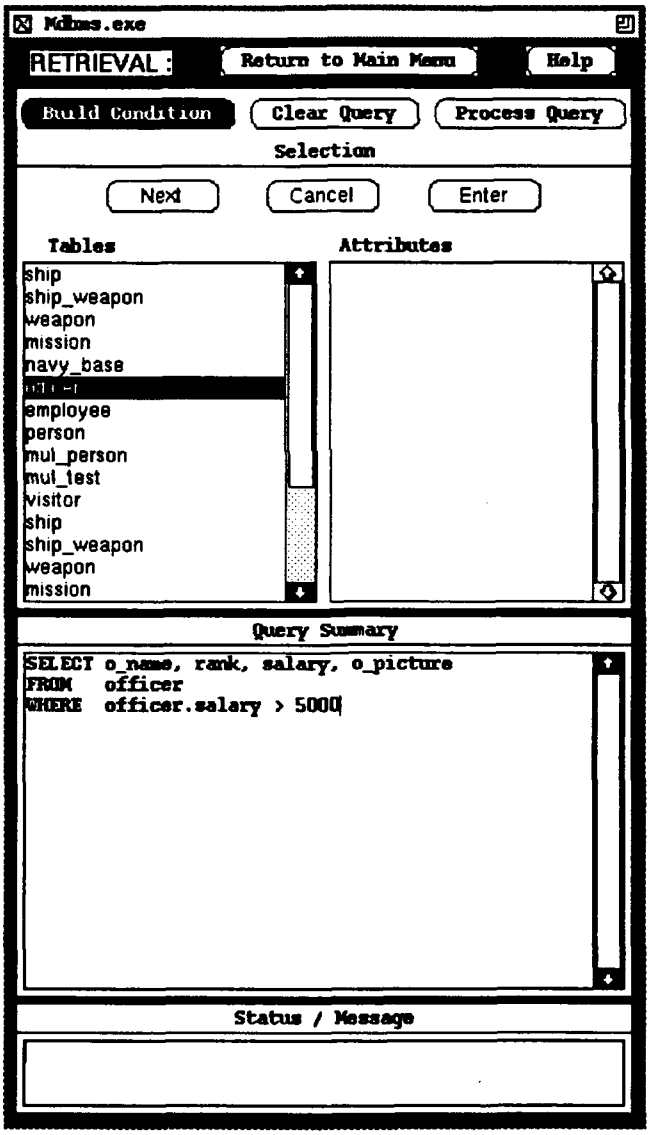

Figure A.2 The "Retrieval" dialog box

11. Press the "Process Query" button to activate the retrieve function of the database program.

12. When the "Result" window" is displayed, the selected attributes and their values except the media data will be listed.

13. Select a tuple by double clicking.

14. Press the "Show Picture" button to see the picture belonging to the selected officer. It is also possible to resize the picture by dragging the upper left corner of the window. Figure A.3 shows the "Result" window with pictures.

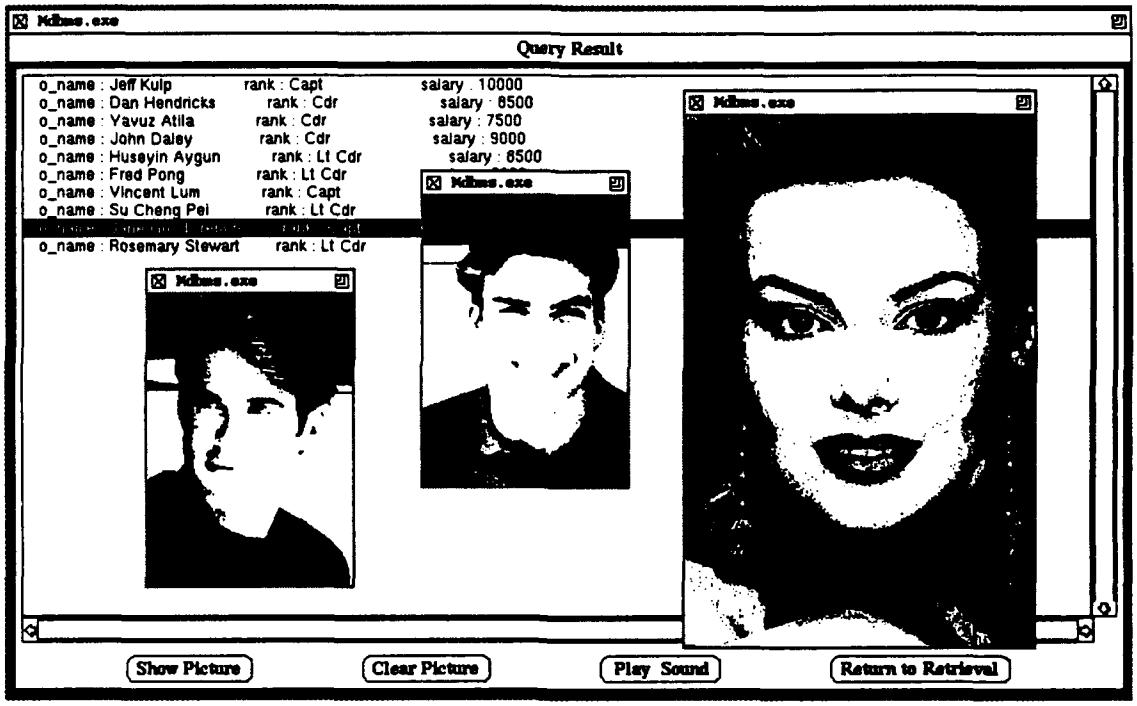

Figure A.3 The "Result" window

15. Press "Play Sound" to hear the voice.

16. Press "Clear Picture" to clear the picture.

17. Use the "Return to Retrieval" button to exit from the "Result Window. The results will be lost and the query will be deleted.

18. The system is now ready for another query.

The second example shows the usage of join condition between two tables.

QUERY:

Retrieve the name, rank, voice and picture of the ship captains together with the name and type of their ships.

SQL FORM:

SELECT o_name, rank, o_voice, o_picture, s_name, type

FROM officer, ship

WHERE officer.o_id = ship.capt_id

EXECUTION:

1. Select "officer" from the "Tables" browser, and the desired attributes in the query from the "Attributes" browser then press the "Next" button. Follow the same procedure for the "ship" table and attribute selection. When selection is completed, press the "Enter" button. The "Query Summary Display" will show the SELECT and FROM parts of the query.

2. Press the "Build Condition" button to activate the "Build Condition" dialog box.

3. Select "officer.o_id = ship.capt_id" from the "Predefined Joins" browser. The predefined join condition will be displayed as the WHERE part of the query in the "Query Summary Display". Since there is no other condition, the query is ready to be processed.

4. Check the query in the display. If it is incorrect or not satisfactory use the "Clear Query" button to delete this query and start again from the beginning.

5. Press the "Process Query" button to activate retrieve function of the database program.

6. When the "Result" window is displayed, the selected attributes and their values (except the media data) will be listed.

7. Select a tuple by double clicking.

8. Press the "Show Picture" button to see the picture belonging to the selected officer.

9. Press "Play Sound" to hear the voice.

10. Press "Clear Picture" to clear the picture.

11. Use the "Return to Retrieval" button to exit from the "Result" window. The results and the query will be deleted.

12. The system is now ready for another query.

B. EXAMPLE FOR THE SECOND APPLICATION [ROW92]

We will give a sample session covering both versions for the second application.

1. System Initiation

a. Start X-Windows in "~mdbms" on the Sun-3 or Sun-4 workstation.

b. Connect to "ai9" by typing in "rxterm proteon1".

c. In the new window created, do "mariefine &"; wait until it says "Fine search initialized".

d. Do "marienlp &"; wait until it says it is done.

e. Do "mariecoarse &".

f. Do "mariesched &"; wait until it says it is done (this will take about three minutes).

g. If "MATCHES" will be used then do "startdserver", otherwise just do "startserver" and skip steps (c) to (f).

h. In order to run the interface, do "rxterm gemini" and type in "rundb2 in the new window". Th "Main Menu" window will be displayed.

2. Example

QUERY:

Retrieve the picture ids, dates, quantities and descriptions of the pictures whose quantities are greater than or equal to five and the descriptions containing "air to air view of Sidewinder".

SQL FORM:

SELECT ID, QUANTITY, DATE_ORIG, CAPTION

FROM VISUAL

WHERE VISUAL.QUANTITY >= 5

AND

VISUAL.CAPTION MATCHES "air to air view of Sidewinder"

EXECUTION:

1. Press the "Retrieve" button on the "Main Menu".

2. Select the "VISUAL" from the "Tables" browser by double clicking on the left mouse button then select the attributes. Each selection will be echoed in the "Query Summary Display".

3. The "Cancel" button can be used to cancel this selection operation. Since there is only one table for this query, after the attribute selections are completed press "Enter". The display will show the "SELECT" and "FROM" parts of the query.

4. Press "Build Condition" button to activate the "Build Condition" dialog box.

5. The condition can be typed in with the editor while the mouse cursor is inside the frame, or the "Selection" part of the "Retrieval" dialog box can be used to select the attributes. For this example select "VISUAL" from the "Tables" and then select "QUANTITY" and press "Enter". The condition editor will display "VISUAL. QUANTITY".

6. According to the syntax we used for the conditions, a space must be left before and after the condition tool. This tool can be chosen from the "Tool Box" or typed in. Activate the "Tool Box" and select " >= ".

7. Type in 5 for the condition input.

8. For the second part of the condition, press "AND" from the "Tool Box".

9. Select "VISUAL.CAPTION".

10. Press the "CAPTION" button to activate the "Natural Language Editor".

11. Type "air to air view of Sidewinder" (no quotes) with this editor. Figure A.4 shows the screen layout at this point.

12. Press one of the search type buttons.

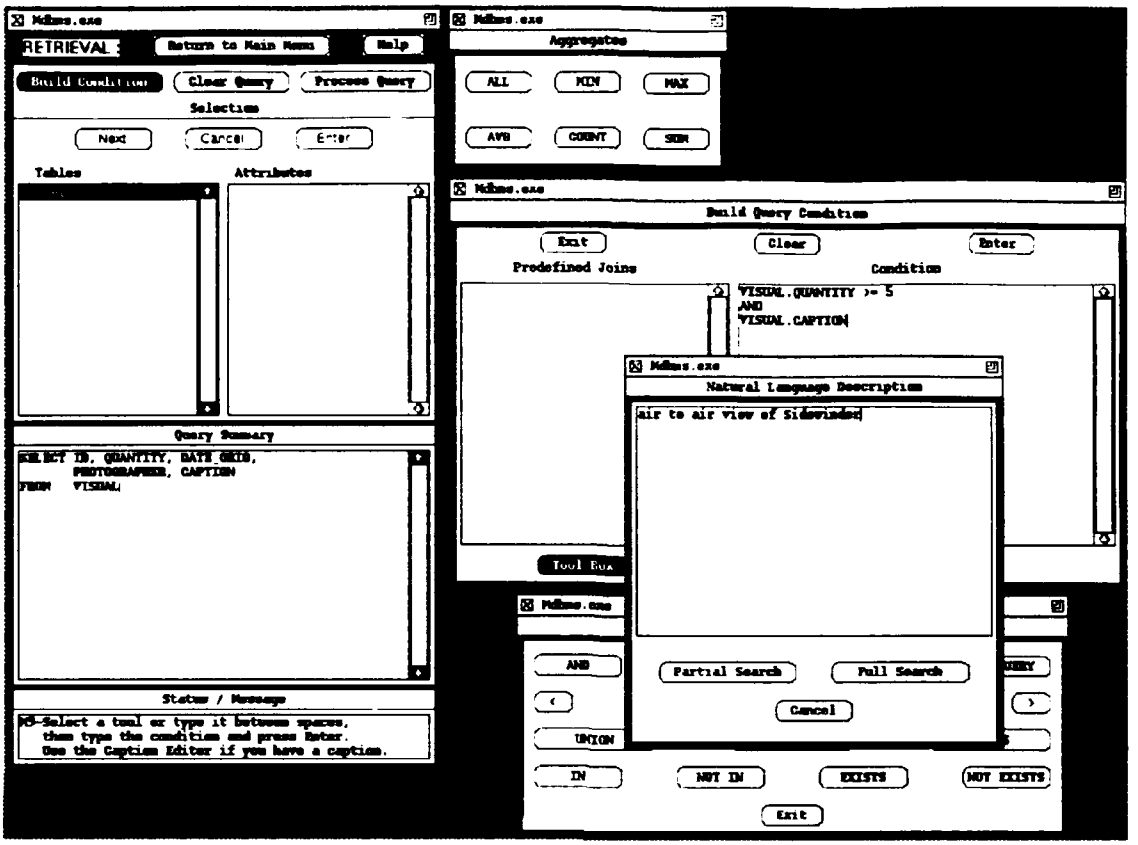

Figure A.4 The screen layout

13. The condition has been built. Press "Clear" if the condition is not correct, otherwise press "Enter". This will send the condition to the WHERE part of the query in the "Query Summary Display".

14. The query can be processed now. Check the query in the display. If it is incorrect or not satisfactory use the "Clear Query" button to delete this query and start again from the beginning.

15. Press the "Process Query" button.

16. When the "Result" window is displayed, the selected attributes and their values except the media data will be listed.

17. Select a tuple by double clicking.

18. Press the "Show Picture" button to see the picture belonging to the selected row. Figure A.5 shows the screen layout with displayed pictures.

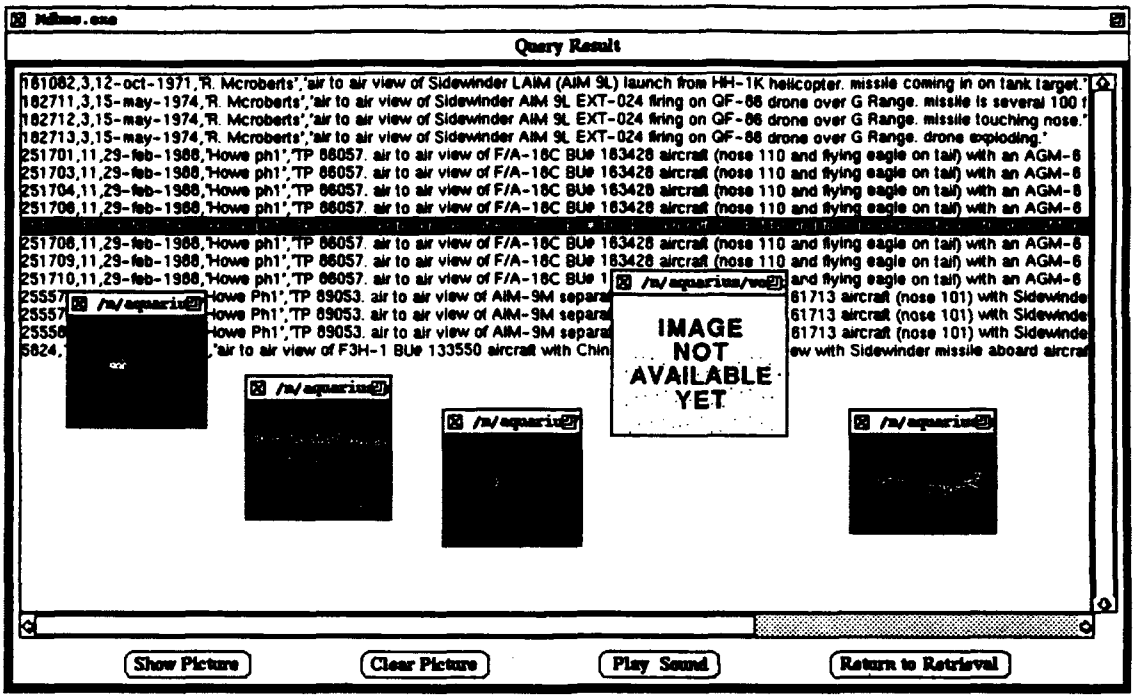

Figure A.5 The screen layout with displayed pictures

19. Clear the pictures by killing the windows using the X-Windows manager.

20. Use the "Return to Retrieval" button to exit from the "Result" window. The results will be lost and the query will be deleted.

21. The system is now ready for another query.

APPENDIX - B

WORK DIVISION

This is a joint thesis by two students. Although we both worked on the same thesis we each worked on separate subjects. We had to work on separate workstations for especially debugging. The following is a list of our work division:

A. Erhan SARIDOGAN

Designing the user interface

Drawing the interface using the Ibuild

Working on the InterViews

Implementation of some functions

Connecting the interface to the first application

Working on some of the functions of the first application

Adding more features to the interface by redrawing

Adding more error-checking facilities

B. Metin BALCI

Research on previous and related work

Designing the user interface

Working on InterViews

Implementation of the interface modules

Solving C and C++ compatibility problems

Connecting the interface to the first application program

Connecting the interface to the second application program

Working on an improved parser to be used in the condition editor

APPENDIX-C

PROGRAM SOURCE CODE

We present the source code for the third version of our interface. This version is the improved version of the first two versions. The main difference is the way of communication with the application database programs. We give a list for the source code directories below. Especially, in the first version, in order to connect the existing program to our interface, we had to modify a considerable amount of Retrieval module of that code.

The directories for the source code:

1. MDBMS Version 1 using the application program explained in [AYG91]:

   "~mdbms/MDBMS"

2. MDBMS Version 2 using the application program explained in [ROW92] and [GUG92]:

   "~mdbms/guisun4"

3. MDBMS Version 3 using the same application program with version 2, in addition to that

   capable of processing natural language descriptions:

   "~mdbms/rowedb"

Below is the listings of the programs for Version 3 of our interface.

A. common.h

```
#ifndef COMMON_H
#define COMMON_H

#define MAXNAME         20
#define MAXATTRIB       30
#define MAXTABLE        20
#define MAXCOND          20
#define MAXNATDESC      1000
#define MAXTEMP         5
#define MAXJOIN          50
#define MAXJOINNUM      30
#define MAXDBTABLES     30
```

```
#define CONDEDITLEN        500
#define MAXRESULTCHAR  200
#define MAXRESULT          500
#define MAXMEDIANUM    10
#define MAXPATH             60
#define MAXSQL              1000

typedef char TABLENAME[MAXNAME];          /* for unique table and attrib */
typedef char ATTRIBUTENAME[MAXNAME];
typedef char PREDEFJOIN[MAXJOIN];

typedef char RESULTLINE[MAXRESULTCHAR];
typedef char PICTURE[MAXPATH];
typedef char SOUND[MAXPATH];

typedef struct {
 ATTRIBUTENAME attribute_name;       /*attrib. names for a specific table */
 char aggregate[6];
 } ATTRIBUTENAMELIST;

typedef struct {
  RESULTLINE r_formatteddata;
  PICTURE r_picture[MAXMEDIANUM];
  SOUND r_sound[MAXMEDIANUM];
} RESULT;

typedef struct{
  TABLENAME  table_name;
  ATTRIBUTENAMELIST  attrib_array[MAXATTRIB];
 } TABLE;

typedef struct{
  char caption[MAXNATDESC];
  char search_type;      /* this field is application dependent */
 } NATURALDESC;


typedef struct{
  TABLENAME  cond_table;
  ATTRIBUTENAME  cond_attribute;
  NATURALDESC cond_nat_description;
  char cond_tool[10];
  char cond_input[MAXCOND];
  char cond_log_opr[4];
 } CONDITION;

typedef struct {
  TABLE  q_table[MAXTABLE];
  CONDITION  q_condition[MAXCOND];
  char  q_temp_table_name[MAXTEMP];
  PREDEFJOIN q_predef_join[MAXJOINNUM];
  RESULT q_result[MAXRESULT];
 } QUERYRECORD;

/* Buffertable:
```

```
   This structure is used to get the values from the string browsers when they are clicked;
   then this buffer is written into q_rec when Next or Enter is pressed.  */

typedef struct{
  TABLENAME  table_name_buffer;
  ATTRIBUTENAMELIST  attrib_array_buffer[MAXATTRIB];
 } BUFFERTABLE;

extern QUERYRECORD q_rec;
extern TABLE dbtables[MAXDBTABLES];
extern struct BUFFERTABLE buffertable;
extern int TableIndex;
extern int AttributeIndex;
extern int TotalAttributeIndex;
extern char*  PredefinedJoins[MAXJOINNUM];
extern char sqlstring[MAXSQL];

extern int Contains;
extern int BuildCondition;
extern int ConditionReady;
extern char clipboard[CONDEDITLEN];
extern int NextTableSelected;
extern int Subquery;
extern int Nested;
extern char clipboard[CONDEDITLEN];

void ClearRecord();
void ClearConditionRecord();
void Loaddbtables();
void LoadPredefinedJoins();

#endif COMMON_H
```

B. common.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

#define TRUE 1
#define FALSE 0

QUERYRECORD q_rec;
TABLE dbtables[MAXDBTABLES];     /* aggregates are not used */
struct BUFFERTABLE buffertable;
int TableIndex = 0;
int AttributeIndex = 0;
int TotalAttributeIndex = 0;
char sqlstring[MAXSQL];
char*  PredefinedJoins[MAXJOINNUM];
int Contains  = FALSE;
int BuildCondition = FALSE;
int ConditionReady = FALSE;
```

```
int NextTableSelected = FALSE;
int Nested = FALSE;
int Subquery = FALSE;
char clipboard[CONDEDITLEN];

/* Global functions */

void ClearRecord(void) {
  /* q_table initialization */
 for(int t = 0; t < MAXTABLE; t++) {
   for(int s = 0; s < MAXATTRIB; s++) {
    q_rec.q_table[t].attrib_array[s].attribute_name[0] = '\0';
    q_rec.q_table[t].attrib_array[s].aggregate[0] = '\0';
   }
   q_rec.q_table[t].table_name[0] = '\0'; }


  /* condition */
 for(int k = 0; k < MAXCOND; k++) {
  q_rec.q_condition[k].cond_table[0] = '\0';
  q_rec.q_condition[k].cond_attribute[0] = '\0';
  q_rec.q_condition[k].cond_nat_description.caption[0] = '\0';
  q_rec.q_condition[k].cond_nat_description.search_type = '\0';
  q_rec.q_condition[k].cond_tool[0] = '\0';
  q_rec.q_condition[k].cond_input[0] = '\0';
  q_rec.q_condition[k].cond_log_opr[0] = '\0';
 }

  /* predefined joins */
 for(int p = 0; p < MAXJOINNUM; p++)  q_rec.q_predef_join[p][0] = '\0';

  /* result array */
 for(int r = 0; r < MAXRESULT; r++){
   q_rec.q_result[r].r_formatteddata[0] = '\0';
  for (int k = 0; k < MAXMEDIANUM; k++ ) {
   q_rec.q_result[r].r_picture[k][0] = '\0';
   q_rec.q_result[r].r_sound[k][0] = '\0';
  }
 }
} /* end of function */

void ClearConditionRecord(void) {
 for(int k = 0; k < MAXCOND; k++) {
  q_rec.q_condition[k].cond_table[0] = '\0';
  q_rec.q_condition[k].cond_attribute[0] = '\0';
  q_rec.q_condition[k].cond_tool[0] = '\0';
  q_rec.q_condition[k].cond_input[0] = '\0';
  q_rec.q_condition[k].cond_log_opr[0] = '\0';
 }
}

void Loaddbtables() {
 /*database tables initialization */
 for(int t = 0; t < MAXTABLE; t++) {
  for(int s = 0; s < MAXATTRIB; s++)
```

```
            dbtables[t].attrib_array[s].attribute_name[0] = '\0';
  dbtables[t].table_name[0] = '\0';
 }

 / ************** This part is implementation dependent *********************
  For test purposes the table and attribute names are hard coded into the program */
 /* table */
 strcpy(dbtables[0].table_name, "VISUAL");
 /* attributes */
 strcpy(dbtables[0].attrib_array[0].attribute_name, "ID");
 strcpy(dbtables[0].attrib_array[1].attribute_name, "DESIGNATOR");
 strcpy(dbtables[0].attrib_array[2].attribute_name, "QUANTITY");
 strcpy(dbtables[0].attrib_array[3].attribute_name, "DATE_ORIG");
 strcpy(dbtables[0].attrib_array[4].attribute_name, "RETENTION");
 strcpy(dbtables[0].attrib_array[5].attribute_name, "MEDIUM_INFO");
 strcpy(dbtables[0].attrib_array[6].attribute_name, "PHOTOGRAPHER");
 strcpy(dbtables[0].attrib_array[7].attribute_name, "CODE");
 strcpy(dbtables[0].attrib_array[8].attribute_name, "LOCATION");
 strcpy(dbtables[0].attrib_array[9].attribute_name, "DATE_LOADED");
 strcpy(dbtables[0].attrib_array[10].attribute_name, "CAPTION");
 strcpy(dbtables[0].attrib_array[11].attribute_name, "CLASSIFICATION");
 strcpy(dbtables[0].attrib_array[12].attribute_name, "CROSS_REF");
}

void LoadPredefinedJoins() {

  /************************ implementation dependent ********************
   since there is only one table there is no join condition                    */

}
```

C. Mdbms-make

```
# ************** Makefile for Mdbms.exe  Version 3,    AUGUST 1992 ***************
# *******  comm.c must be compiled in C before using this Makefile .
# *******  common.c can be compiled using this Makefile.
# *******  All include files should be placed where they are called at the end
# *******  of this Makefile. This provides dependency checking.
# *******  Care must be taken on using End-of-line in this file
# *******  Libraries are added for TCP connection.

# ************************Generated by InterViews***********************************
# Makefile generated by imake - do not edit!
# $XConsortium: imake.c,v 1.51 89/12/12 12:37:30 jim Exp $
#
# The cpp used on this machine replaces all newlines and multiple tabs and
# spaces in a macro expansion with a single space.  Imake tries to compensate
# for this, but is not always successful.
#

# Read "InterViews/template" to understand how this Makefile was generated.
# Edit <InterViews/arch.def> to add support for a new platform.
# Edit <InterViews/iv-sun.cf> to change platform-specific parameters.
# Edit <InterViews/local.def> to change site-specific parameters.
# Edit <./Mdbms-imake> to change actions that make should perform.
```

```
# architecture:  SUN4

# platform:  $XConsortium: sun.cf,v 1.38 89/12/23 16:10:10 jim Exp $
# operating system:  SunOS 4.0.3

          SHELL = /bin/sh

          IMAKE = imake
       IMAKEFLAGS = \
-T "InterViews/template"\
-I$(CONFIGDIR) -I$(XCONFIGDIR)\
$(SPECIAL_IMAKEFLAGS)
SPECIAL_IMAKEFLAGS =   -DUseInstalled
          DEPEND = makedepend
    DEPEND_CCFLAGS = $(CCDEFINES) $(CCINCLUDES) -I/usr/include/CC
           MAKE = make
        PASSARCH = ARCH="$(ARCH)" SPECIAL_IMAKEFLAGS="$(SPECIAL_IMAKEFLAGS)"
       ARCHORCPU = $$CPU
           ARCH = $(ARCHORCPU)$(SPECIAL_ARCH)
     SPECIAL_ARCH =

        CCDRIVER = CC +p
        CCSUFFIX = c
           SRC = .
          SLASH = /
           SRCS = $(SRC)$(SLASH)*.$(CCSUFFIX)
           OBJS = *.o
           AOUT = a.out

        CCFLAGS = $(APP_CCFLAGS) $(IV_CCFLAGS) $(EXTRA_CCFLAGS)
      IV_CCFLAGS = \
\
\
$(SHARED_CCFLAGS)\
$(CCDEFINES)\
$(CCINCLUDES)
     DEBUG_CCFLAGS = -g
   OPTIMIZE_CCFLAGS = -O
    SHARED_CCFLAGS =
     EXTRA_CCFLAGS =

       CCDEFINES = $(APP_CCDEFINES) $(IV_CCDEFINES) $(EXTRA_CCDEFINES)
     IV_CCDEFINES = $(LANGUAGE_CCDEFINES) $(BACKWARD_CCDEFINES)
LANGUAGE_CCDEFINES = -Dcplusplus_2_0
BACKWARD_CCDEFINES =
   EXTRA_CCDEFINES =

      CCINCLUDES = $(APP_CCINCLUDES) $(IV_CCINCLUDES) $(EXTRA_CCINCLUDES)
    IV_CCINCLUDES = \
$(BACKWARD_CCINCLUDES)\
$(TOP_CCINCLUDES)\
$(X_CCINCLUDES)
BACKWARD_CCINCLUDES =
    TOP_CCINCLUDES = -I$(INCDIR)
```

```
    X_CCINCLUDES =
  EXTRA_CCINCLUDES =

      CCLDFLAGS = $(APP_CCLDFLAGS) $(IV_CCLDFLAGS) $(EXTRA_CCLDFLAGS)
    IV_CCLDFLAGS =\
\
\
$(NONSHARED_CCLDFLAGS)
NONSHARED_CCLDFLAGS =
  EXTRA_CCLDFLAGS =

      CCDEPLIBS = $(APP_CCDEPLIBS) $(IV_CCDEPLIBS) $(EXTRA_CCDEPLIBS)
    IV_CCDEPLIBS =\
$(DEPLIBUNIDRAW)\
$(DEPLIBGRAPHIC)\
$(DEPLIBIV)\
$(DEPLIBXEXT)\
$(DEPLIBX11)\
$(DEPLIBM)
  EXTRA_CCDEPLIBS =

      CCLDLIBS = $(APP_CCLDLIBS) $(IV_CCLDLIBS) $(EXTRA_CCLDLIBS)
    IV_CCLDLIBS =\
$(LIBDIRPATH)\
$(LDLIBUNIDRAW)\
$(LDLIBGRAPHIC)\
$(LDLIBIV)\
$(XLIBDIRPATH)\
$(LDLIBXEXT)\
$(LDLIBX11)\
$(LDLIBM)\
$(ABSLIBDIRPATH)

  EXTRA_CCLDLIBS =
  APP_CCLDLIBS = /usr/local/q3.1.1/generic/qplib3.1.1/IPC/TCP/sun4-4/tcp_c.so

      INSTALL = install
    INSTPGMFLAGS = -s
    INSTBINFLAGS = -m 0755
    INSTUIDFLAGS = -m 4755
    INSTLIBFLAGS = -m 0644
    INSTINCFLAGS = -m 0444
    INSTMANFLAGS = -m 0444
    INSTDATFLAGS = -m 0444
   INSTKMEMFLAGS = -m 4755

         AR = ar clq
         AS = as
         CP = cp
        CPP = /lib/cpp $(EXTRA_CCDEFINES)
  PREPROCESSCMD = $(CCDRIVER) -E $(EXTRA_CCDEFINES)
         LD = ld
         LN = ln -s
     MKDIRHIER = mkdirhier
         MV = mv
```

```
        RANLIB = ranlib
  RANLIBINSTFLAGS =
           RM = rm -f
       RM_CMD = $(RM) ,* .emacs_* *..c *.BAK *.CKP *.a *.bak *.ln *.o a.out core errs make.log make.out tags
TAGS
        TROFF = psroff

          TOP = .
    CURRENT_DIR = .

       BINDIR = /usr/local/iv/bin/$(ARCH)
     CONFIGDIR = /usr/local/iv/config
       INCDIR = /usr/local/iv/include
       LIBDIR = /usr/local/iv/lib/$(ARCH)
    LIBALLDIR = /usr/local/iv/lib/all
       MANDIR = /usr/local/iv/man

   ABSCONFIGDIR = /usr/local/iv/config
     ABSLIBDIR = /usr/local/iv/lib/$(ARCH)
   ABSLIBALLDIR = /usr/local/iv/lib/all
    XCONFIGDIR = /usr/lib/X11/config
      XINCDIR = /usr/include
      XLIBDIR = /usr/lib
     PSFONTDIR = /usr/lib/ps
#        TCP_INC = /usr/local/q3.1.1/generic/qplib3.1.1/IPC/TCP

all::

Makefile::
    -@if [ -f Makefile ]; then \
    $(RM) Makefile.bak; \
    $(MV) Makefile Makefile.bak; \
    else exit 0; fi
    $(IMAKE) $(IMAKEFLAGS) -DTOPDIR=$(TOP) -DCURDIR=$(CURRENT_DIR)
Makefiles::
depend::
install::

clean::
    @$(RM_CMD) "#"*

# ----------------------------------------------------------------------

# generated by ibmkmf
# *******  comm.c must be compiled in C before using this Makefile

SPECIAL_IMAKEFLAGS = \
  -f Mdbms-imake -s Mdbms-make -DUseInstalled -DTurnOptimizingOn=0

SRCS = \
  common.$(CCSUFFIX) \
  Result.$(CCSUFFIX) \
  Result-core.$(CCSUFFIX) \
  _Dialog_9.$(CCSUFFIX) \
  _Dialog_9-core.$(CCSUFFIX) \
```

```
    _Dialog_7.$(CCSUFFIX) \
    _Dialog_7-core.$(CCSUFFIX) \
    _Dialog_11.$(CCSUFFIX) \
    _Dialog_11-core.$(CCSUFFIX) \
    MainMenu.$(CCSUFFIX) \
    MainMenu-core.$(CCSUFFIX) \
    _Dialog_10.$(CCSUFFIX) \
    _Dialog_10-core.$(CCSUFFIX) \
    _Dialog_8.$(CCSUFFIX) \
    _Dialog_8-core.$(CCSUFFIX) \
    _Dialog_12.$(CCSUFFIX) \
    _Dialog_12-core.$(CCSUFFIX) \
    Mdbms-main.$(CCSUFFIX)
OBJS = \
    comm.o \
    common.o \
    Result.o \
    Result-core.o \
    _Dialog_9.o \
    _Dialog_9-core.o \
    _Dialog_7.o \
    _Dialog_7-core.o \
    _Dialog_11.o \
    _Dialog_11-core.o \
    MainMenu.o \
    MainMenu-core.o \
    _Dialog_10.o \
    _Dialog_10-core.o \
    _Dialog_8.o \
    _Dialog_8-core.o \
    _Dialog_12.o \
    _Dialog_12-core.o \
    Mdbms-main.o
AOUT = Mdbms.exe

DEPLIBUNIDRAW =
DEPLIBIV =
DEPLIBXEXT =
DEPLIBX11 =
DEPLIBM =

LIBDIRPATH = -L$(LIBDIR)
LDLIBUNIDRAW = -lUnidraw
LDLIBIV = -lIV
XLIBDIRPATH =
LDLIBXEXT = -lXext
LDLIBX11 = -lX11
LDLIBM = -lm
ABSLIBDIRPATH =

BACKWARD_CCDEFINES = -Div2_6_compatible
BACKWARD_CCINCLUDES = -I$(INCDIR)/InterViews/2.6 -I$(INCDIR)/IV-look/2.6

PROGRAM = Mdbms
```

```
all:: $(AOUT)

$(AOUT):  $(CCDEPLIBS)
	-@if [ ! -w $@ ]; then $(RM) $@; else exit 0; fi
	$(CCDRIVER) $(CCLDFLAGS) -o $@ $(OBJS) $(CCLDLIBS)

install:: $(AOUT)
	-@if [ -d $(BINDIR) ]; then exit 0;\
	else (set -x; $(MKDIRHIER) $(BINDIR)); fi
	$(INSTALL) -c $(INSTPGMFLAGS) $(INSTBINFLAGS) $(AOUT) $(BINDIR)/$(PROGRAM)
	$(RM) $(AOUT)

depend::
	$(DEPEND) -s "# DO NOT DELETE" -- $(DEPEND_CCFLAGS) -- $(SRCS)

$(AOUT): common.o
common.o: common.$(CCSUFFIX)
	@$(RM) common.o common..c
	$(CCDRIVER) $(CCFLAGS)  -c common.$(CCSUFFIX)

$(AOUT): Mdbms-main.o
Mdbms-main.o: Mdbms-main.$(CCSUFFIX)
	@$(RM) Mdbms-main.o Mdbms-main..c
	$(CCDRIVER) $(CCFLAGS)  -c Mdbms-main.$(CCSUFFIX)

$(AOUT): Result.o
Result.o: Result.$(CCSUFFIX)
	@$(RM) Result.o Result..c
	$(CCDRIVER) $(CCFLAGS)  -c Result.$(CCSUFFIX)

$(AOUT): Result-core.o
Result-core.o: Result-core.$(CCSUFFIX)
	@$(RM) Result-core.o Result-core..c
	$(CCDRIVER) $(CCFLAGS)  -c Result-core.$(CCSUFFIX)

$(AOUT): _Dialog_9.o
_Dialog_9.o: _Dialog_9.$(CCSUFFIX)
	@$(RM) _Dialog_9.o _Dialog_9..c
	$(CCDRIVER) $(CCFLAGS)  -c _Dialog_9.$(CCSUFFIX)

$(AOUT): _Dialog_9-core.o
_Dialog_9-core.o: _Dialog_9-core.$(CCSUFFIX)
	@$(RM) _Dialog_9-core.o _Dialog_9-core..c
	$(CCDRIVER) $(CCFLAGS)  -c _Dialog_9-core.$(CCSUFFIX)

$(AOUT): _Dialog_7.o
_Dialog_7.o: _Dialog_7.$(CCSUFFIX)
	@$(RM) _Dialog_7.o _Dialog_7..c
	$(CCDRIVER) $(CCFLAGS)  -c _Dialog_7.$(CCSUFFIX)

$(AOUT): _Dialog_7-core.o
_Dialog_7-core.o: _Dialog_7-core.$(CCSUFFIX)
	@$(RM) _Dialog_7-core.o _Dialog_7-core..c
	$(CCDRIVER) $(CCFLAGS)  -c _Dialog_7-core.$(CCSUFFIX)
```

```
$(AOUT): _Dialog_11.o
_Dialog_11.o: _Dialog_11.$(CCSUFFIX)
	@$(RM) _Dialog_11.o _Dialog_11..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_11.$(CCSUFFIX)

$(AOUT): _Dialog_11-core.o
_Dialog_11-core.o: _Dialog_11-core.$(CCSUFFIX)
	@$(RM) _Dialog_11-core.o _Dialog_11-core..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_11-core.$(CCSUFFIX)

$(AOUT): MainMenu.o
MainMenu.o: MainMenu.$(CCSUFFIX)
	@$(RM) MainMenu.o MainMenu..c
	$(CCDRIVER) $(CCFLAGS) -c MainMenu.$(CCSUFFIX)

$(AOUT): MainMenu-core.o
MainMenu-core.o: MainMenu-core.$(CCSUFFIX)
	@$(RM) MainMenu-core.o MainMenu-core..c
	$(CCDRIVER) $(CCFLAGS) -c MainMenu-core.$(CCSUFFIX)

$(AOUT): _Dialog_10.o
_Dialog_10.o: _Dialog_10.$(CCSUFFIX)
	@$(RM) _Dialog_10.o _Dialog_10..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_10.$(CCSUFFIX)

$(AOUT): _Dialog_10-core.o
_Dialog_10-core.o: _Dialog_10-core.$(CCSUFFIX)
	@$(RM) _Dialog_10-core.o _Dialog_10-core..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_10-core.$(CCSUFFIX)

$(AOUT): _Dialog_8.o
_Dialog_8.o: _Dialog_8.$(CCSUFFIX)
	@$(RM) _Dialog_8.o _Dialog_8..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_8.$(CCSUFFIX)

$(AOUT): _Dialog_8-core.o
_Dialog_8-core.o: _Dialog_8-core.$(CCSUFFIX)
	@$(RM) _Dialog_8-core.o _Dialog_8-core..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_8-core.$(CCSUFFIX)

$(AOUT): _Dialog_12.o
_Dialog_12.o: _Dialog_12.$(CCSUFFIX)
	@$(RM) _Dialog_12.o _Dialog_12..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_12.$(CCSUFFIX)

$(AOUT): _Dialog_12-core.o
_Dialog_12-core.o: _Dialog_12-core.$(CCSUFFIX)
	@$(RM) _Dialog_12-core.o _Dialog_12-core..c
	$(CCDRIVER) $(CCFLAGS) -c _Dialog_12-core.$(CCSUFFIX)

# DO NOT DELETE


common.o: common.h
Result.o: /usr/local/iv/include/IV-look/2.6/InterViews/strbrowser.h
```

Result.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
Result.o: /usr/local/iv/include/InterViews/enter-scope.h
Result.o: /usr/local/iv/include/InterViews/iv.h
Result.o: /usr/local/iv/include/InterViews/_defines.h
Result.o: /usr/local/iv/include/InterViews/2.6/_enter.h
Result.o: /usr/local/iv/include/InterViews/2.6/_names.h
Result.o: /usr/local/iv/include/OS/os.h
Result.o: /usr/local/iv/include/InterViews/boolean.h
Result.o: /usr/local/iv/include/InterViews/coord.h
Result.o: /usr/local/iv/include/InterViews/geometry.h
Result.o: /usr/local/iv/include/OS/math.h
Result.o: /usr/local/iv/include/OS/enter-scope.h
Result.o: /usr/local/iv/include/OS/_defines.h
Result.o: /usr/local/iv/include/InterViews/_enter.h
Result.o: /usr/local/iv/include/InterViews/_names.h
Result.o: /usr/local/iv/include/InterViews/_leave.h
Result.o: /usr/local/iv/include/InterViews/glyph.h
Result.o: /usr/local/iv/include/InterViews/resource.h
Result.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
Result.o: /usr/local/iv/include/InterViews/2.6/_leave.h
Result.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
Result.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
Result.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
Result.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h Result.h
Result.o: Result-core.h /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
Result.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
Result.o: common.h _Dialog_9.h
Result-core.o: /usr/local/iv/include/InterViews/canvas.h
Result-core.o: /usr/local/iv/include/InterViews/boolean.h
Result-core.o: /usr/local/iv/include/InterViews/enter-scope.h
Result-core.o: /usr/local/iv/include/InterViews/iv.h
Result-core.o: /usr/local/iv/include/InterViews/_defines.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
Result-core.o: /usr/local/iv/include/OS/os.h
Result-core.o: /usr/local/iv/include/InterViews/coord.h
Result-core.o: /usr/local/iv/include/InterViews/_enter.h
Result-core.o: /usr/local/iv/include/InterViews/_names.h
Result-core.o: /usr/local/iv/include/InterViews/_leave.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
Result-core.o: /usr/local/iv/include/InterViews/resource.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
Result-core.o: /usr/local/iv/include/InterViews/sensor.h
Result-core.o: /usr/local/iv/include/InterViews/event.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
Result-core.o: /usr/local/iv/include/InterViews/geometry.h
Result-core.o: /usr/local/iv/include/OS/math.h
Result-core.o: /usr/local/iv/include/OS/enter-scope.h
Result-core.o: /usr/local/iv/include/OS/_defines.h
Result-core.o: /usr/local/iv/include/InterViews/glyph.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h

Result-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h
Result-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/strbrowser.h
Result-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
Result-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/adjuster.h
Result-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/scroller.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/glue.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
Result-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
Result-core.o: Result.h Result-core.h
Result-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_9.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_9.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_9.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_9.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_9.o: /usr/local/iv/include/OS/os.h
_Dialog_9.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_9.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_9.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_9.o: /usr/local/iv/include/OS/math.h
_Dialog_9.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_9.o: /usr/local/iv/include/OS/_defines.h
_Dialog_9.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_9.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_9.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_9.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_9.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_9.o: /usr/local/iv/include/IV-look/2.6/InterViews/strbrowser.h
_Dialog_9.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
_Dialog_9.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_9.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_9.o: _Dialog_9.h _Dialog_9-core.h Result.h common.h MainMenu.h
_Dialog_9.o: _Dialog_7.h _Dialog_8.h _Dialog_10.h _Dialog_12.h _Dialog_11.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/canvas.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_9-core.o: /usr/local/iv/include/OS/os.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/_leave.h

_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/sensor.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/event.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_9-core.o: /usr/local/iv/include/OS/math.h
_Dialog_9-core.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_9-core.o: /usr/local/iv/include/OS/_defines.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/glue.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/border.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/strbrowser.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/adjuster.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/scroller.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
_Dialog_9-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
_Dialog_9-core.o: _Dialog_9.h _Dialog_9-core.h
_Dialog_9-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_7.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_7.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_7.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_7.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_7.o: /usr/local/iv/include/OS/os.h
_Dialog_7.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_7.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_7.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_7.o: /usr/local/iv/include/OS/math.h
_Dialog_7.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_7.o: /usr/local/iv/include/OS/_defines.h
_Dialog_7.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_7.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_7.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_7.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_7.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h

_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
_Dialog_7.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_7.o: _Dialog_7.h _Dialog_7-core.h
_Dialog_7.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_7.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/canvas.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_7-core.o: /usr/local/iv/include/OS/os.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/sensor.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/event.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_7-core.o: /usr/local/iv/include/OS/math.h
_Dialog_7-core.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_7-core.o: /usr/local/iv/include/OS/_defines.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_7-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/glue.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h
_Dialog_7-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
_Dialog_7-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/border.h
_Dialog_7-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/adjuster.h
_Dialog_7-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/scroller.h
_Dialog_7-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_7-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
_Dialog_7-core.o: _Dialog_7.h _Dialog_7-core.h
_Dialog_7-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_11.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_11.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_11.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_11.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/_enter.h

_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_11.o: /usr/local/iv/include/OS/os.h
_Dialog_11.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_11.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_11.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_11.o: /usr/local/iv/include/OS/math.h
_Dialog_11.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_11.o: /usr/local/iv/include/OS/_defines.h
_Dialog_11.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_11.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_11.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_11.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_11.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_11.o: /usr/local/iv/include/IV-look/2.6/InterViews/strbrowser.h
_Dialog_11.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
_Dialog_11.o: _Dialog_11.h _Dialog_11-core.h
_Dialog_11.o: _Dialog_8.h _Dialog_9.h _Dialog_12.h common.h MainMenu.h Result.h
_Dialog_11.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_11.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/canvas.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_11-core.o: /usr/local/iv/include/OS/os.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/sensor.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/event.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_11-core.o: /usr/local/iv/include/OS/math.h
_Dialog_11-core.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_11-core.o: /usr/local/iv/include/OS/_defines.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h

_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/glue.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/strbrowser.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/border.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/adjuster.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/scroller.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
_Dialog_11-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
_Dialog_11-core.o: _Dialog_11.h _Dialog_11-core.h
_Dialog_11-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
MainMenu.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
MainMenu.o: /usr/local/iv/include/InterViews/enter-scope.h
MainMenu.o: /usr/local/iv/include/InterViews/iv.h
MainMenu.o: /usr/local/iv/include/InterViews/_defines.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/_enter.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/_names.h
MainMenu.o: /usr/local/iv/include/OS/os.h
MainMenu.o: /usr/local/iv/include/InterViews/boolean.h
MainMenu.o: /usr/local/iv/include/InterViews/coord.h
MainMenu.o: /usr/local/iv/include/InterViews/geometry.h
MainMenu.o: /usr/local/iv/include/OS/math.h
MainMenu.o: /usr/local/iv/include/OS/enter-scope.h
MainMenu.o: /usr/local/iv/include/OS/_defines.h
MainMenu.o: /usr/local/iv/include/InterViews/_enter.h
MainMenu.o: /usr/local/iv/include/InterViews/_names.h
MainMenu.o: /usr/local/iv/include/InterViews/_leave.h
MainMenu.o: /usr/local/iv/include/InterViews/glyph.h
MainMenu.o: /usr/local/iv/include/InterViews/resource.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/_leave.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
MainMenu.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
MainMenu.o: MainMenu.h MainMenu-core.h
MainMenu.o: _Dialog_9.h _Dialog_10.h
MainMenu.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
MainMenu.o: /usr/local/iv/include/InterViews/window.h
MainMenu.o: /usr/local/iv/include/InterViews/canvas.h
MainMenu-core.o: /usr/local/iv/include/InterViews/canvas.h
MainMenu-core.o: /usr/local/iv/include/InterViews/boolean.h
MainMenu-core.o: /usr/local/iv/include/InterViews/enter-scope.h
MainMenu-core.o: /usr/local/iv/include/InterViews/iv.h
MainMenu-core.o: /usr/local/iv/include/InterViews/_defines.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
MainMenu-core.o: /usr/local/iv/include/OS/os.h
MainMenu-core.o: /usr/local/iv/include/InterViews/coord.h

MainMenu-core.o: /usr/local/iv/include/InterViews/_enter.h
MainMenu-core.o: /usr/local/iv/include/InterViews/_names.h
MainMenu-core.o: /usr/local/iv/include/InterViews/_leave.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
MainMenu-core.o: /usr/local/iv/include/InterViews/resource.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
MainMenu-core.o: /usr/local/iv/include/InterViews/sensor.h
MainMenu-core.o: /usr/local/iv/include/InterViews/event.h
MainMenu-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
MainMenu-core.o: /usr/local/iv/include/InterViews/geometry.h
MainMenu-core.o: /usr/local/iv/include/OS/math.h
MainMenu-core.o: /usr/local/iv/include/OS/enter-scope.h
MainMenu-core.o: /usr/local/iv/include/OS/_defines.h
MainMenu-core.o: /usr/local/iv/include/InterViews/glyph.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/glue.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
MainMenu-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
MainMenu-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
MainMenu-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
MainMenu-core.o: MainMenu.h MainMenu-core.h
MainMenu-core.o: /usr/local/iv/include/InterViews/window.h
_Dialog_10.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_10.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_10.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_10.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_10.o: /usr/local/iv/include/OS/os.h
_Dialog_10.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_10.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_10.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_10.o: /usr/local/iv/include/OS/math.h
_Dialog_10.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_10.o: /usr/local/iv/include/OS/_defines.h
_Dialog_10.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_10.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_10.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_10.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_10.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h

_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_10.o: _Dialog_10.h _Dialog_10-core.h _Dialog_9.h
_Dialog_10.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_10.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/canvas.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_10-core.o: /usr/local/iv/include/OS/os.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/sensor.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/event.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_10-core.o: /usr/local/iv/include/OS/math.h
_Dialog_10-core.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_10-core.o: /usr/local/iv/include/OS/_defines.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_10-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_10-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/glue.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_10-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
_Dialog_10-core.o: _Dialog_10.h _Dialog_10-core.h
_Dialog_10-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_8.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_8.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_8.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_8.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_8.o: /usr/local/iv/include/OS/os.h
_Dialog_8.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_8.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_8.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_8.o: /usr/local/iv/include/OS/math.h

_Dialog_8.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_8.o: /usr/local/iv/include/OS/_defines.h
_Dialog_8.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_8.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_8.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_8.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_8.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/InterViews/textbuffer.h
_Dialog_8.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_8.o: _Dialog_8.h _Dialog_8-core.h
_Dialog_8.o: _Dialog_9.h _Dialog_11.h MainMenu.h common.h
_Dialog_8.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_8.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/canvas.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_8-core.o: /usr/local/iv/include/OS/os.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_8-core.o: /usr/local/iv/includ /InterViews/2.6/InterViews/painter.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/sensor.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/event.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_8-core.o: /usr/local/iv/include/OS/math.h
_Dialog_8-core.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_8-core.o: /usr/local/iv/include/OS/_defines.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_8-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h
_Dialog_8-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/texteditor.h
_Dialog_8 core.o: /usr/local/iv/include/Inte Views/2.6/Inter Views/textbuffer.h
_Dialog_8-core.o: /usr/local/iv/include/Inter Views/2.6/Inter Views/scene.h
_Dialog_8-core.o: /usr/local/iv/include/Inter Views/2.6/Inter Views/shape.h
_Dialog_8-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_8-core.o: /usr/local/iv/include/Inte rViews/2.6/InterViews/glue.h

_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_8-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
_Dialog_8-core.o: _Dialog_8.h _Dialog_8-core.h
_Dialog_8-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_12.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_12.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_12.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_12.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_12.o: /usr/local/iv/include/OS/os.h
_Dialog_12.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_12.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_12.o: /usr/local/iv/include/InterViews/geometry.h
_Dialog_12.o: /usr/local/iv/include/OS/math.h
_Dialog_12.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_12.o: /usr/local/iv/include/OS/_defines.h
_Dialog_12.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_12.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_12.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_12.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_12.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_12.o: _Dialog_12.h _Dialog_12-core.h
_Dialog_12.o: _Dialog_9.h _Dialog_11.h common.h
_ og_12.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
_Dialog_12.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/canvas.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/boolean.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/enter-scope.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/iv.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/_defines.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/_enter.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/_names.h
_Dialog_12-core.o: /usr/local/iv/include/OS/os.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/coord.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/_enter.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/_names.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/_leave.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/resource.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/_leave.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/sensor.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/event.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/box.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/geometry.h

_Dialog_12-core.o: /usr/local/iv/include/OS/math.h
_Dialog_12-core.o: /usr/local/iv/include/OS/enter-scope.h
_Dialog_12-core.o: /usr/local/iv/include/OS/_defines.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/glyph.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
_Dialog_12-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/frame.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/message.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_12-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/button.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/subject.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/glue.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/shape.h
_Dialog_12-core.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
_Dialog_12-core.o: _Dialog_12.h _Dialog_12-core.h
_Dialog_12-core.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/catalog.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/classes.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/globals.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/defs.h
Mdbms-main.o: /usr/local/iv/include/InterViews/enter-scope.h
Mdbms-main.o: /usr/local/iv/include/InterViews/iv.h
Mdbms-main.o: /usr/local/iv/include/InterViews/_defines.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/_enter.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/_names.h
Mdbms-main.o: /usr/local/iv/include/OS/os.h
Mdbms-main.o: /usr/local/iv/include/InterViews/boolean.h
Mdbms-main.o: /usr/local/iv/include/InterViews/coord.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/alignment.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/_leave.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/textstyle.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/minmax.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/uformat.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/uhash.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/umap.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/uarray.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/unidraw.h
Mdbms-main.o: /usr/local/iv/include/Unidraw/creator.h
Mdbms-main.o: /usr/local/iv/include/InterViews/canvas.h
Mdbms-main.o: /usr/local/iv/include/InterViews/_enter.h
Mdbms-main.o: /usr/local/iv/include/InterViews/_names.h
Mdbms-main.o: /usr/local/iv/include/InterViews/_leave.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/painter.h
Mdbms-main.o: /usr/local/iv/include/InterViews/resource.h
Mdbms-main.o: /usr/local/iv/include/InterViews/sensor.h
Mdbms-main.o: /usr/local/iv/include/InterViews/event.h
Mdbms-main.o: /usr/local/iv/include/InterViews/world.h
Mdbms-main.o: /usr/local/iv/include/InterViews/session.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/perspective.h
Mdbms-main.o: Result.h Result-core.h
Mdbms-main.o: /usr/local/iv/include/IV-look/2.6/InterViews/dialog.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/interactor.h
Mdbms-main.o: /usr/local/iv/include/InterViews/geometry.h

```
Mdbms-main.o: /usr/local/iv/include/OS/math.h
Mdbms-main.o: /usr/local/iv/include/OS/enter-scope.h
Mdbms-main.o: /usr/local/iv/include/OS/_defines.h
Mdbms-main.o: /usr/local/iv/include/InterViews/glyph.h _Dialog_9.h
Mdbms-main.o: _Dialog_9-core.h _Dialog_7.h _Dialog_7-core.h _Dialog_11.h
Mdbms-main.o: _Dialog_11-core.h MainMenu.h MainMenu-core.h
Mdbms-main.o: /usr/local/iv/include/InterViews/2.6/InterViews/scene.h
Mdbms-main.o: /usr/local/iv/include/InterViews/window.h _Dialog_10.h
Mdbms-main.o: _Dialog_10-core.h _Dialog_8.h _Dialog_8-core.h _Dialog_12.h
Mdbms-main.o: _Dialog_12-core.h Mdbms-props
Mdbms-main.o: Result.h Result-core.h _Dialog_11.h _Dialog_11-core.h
Mdbms-main.o: _Dialog_9.h _Dialog_9-core.h _Dialog_7.h _Dialog_7-core.h
Mdbms-main.o: _Dialog_11-core.h MainMenu.h MainMenu-core.h common.h
```

D. Mdbms-main.c

```
// Module Name : Mdbms-main.c
// Author      : Erhan SARIDOGAN, Metin BALCI
// Date        : August 1992
// The main program for Mdbms.exe
// The MainMenu window is inserted as an X-Window application.
// Communication is established.
// Database tables are loaded into the interface

#include <Unidraw/catalog.h>
#include <Unidraw/unidraw.h>
#include <Unidraw/creator.h>
#include <InterViews/canvas.h>
#include <InterViews/painter.h>
#include <InterViews/sensor.h>
#include <InterViews/world.h>
#include <InterViews/perspective.h>
#include <InterViews/2.6/_enter.h>
#include "_Dialog_7.h"
#include "_Dialog_8.h"
#include "_Dialog_9.h"
#include "_Dialog_10.h"
#include "_Dialog_11.h"
#include "_Dialog_12.h"
#include "MainMenu.h"
#include "Result.h"
#include "common.h"
#include <stdio.h>
#include <string.h>

//These are put by InterViews
static PropertyData properties[] = {
#include "Mdbms-props"
  { nil }
};

static OptionDesc options[] = {
  { nil }
```

```
};

MainMenu* mainmenu;

extern "C" {
 init_comm(char*);
 done_comm();
}

int main (int argc, char** argv) {
 // This function is declared in comm.c. It establishes the connection with
 // the other program ( implementation dependent )
 init_comm(argv[1]);

 Creator creator;
 Unidraw* unidraw = new Unidraw(
     new Catalog("/****/", &creator), argc, argv, options, properties );
 World* w = unidraw->GetWorld();

 // load the database table and attribute names into q_rec
 Loaddbtables();

 mainmenu = new MainMenu("_instance_1526");

 // Insert the application into the X-window environment
 w->InsertApplication(mainmenu);
 unidraw->Run();
 delete unidraw;

 // Disconnection
 done_comm();
 return 0;
}
```

E. MainMenu.h

```
// Module Name : MainMenu.h                    MAINMENU
// Author      : Erhan SARIDOGAN, Metin BALCI
// Date        : August 1992

#ifndef MainMenu_h
#define MainMenu_h

#include "MainMenu-core.h"

class MainMenu : public MainMenu_core {
public:
   MainMenu(const char*);

   virtual void retrievepressed();
   virtual void createtablepressed();
   virtual void inserttuplepressed();
   virtual void modifypressed();
   virtual void deletepressed();
```

```
  virtual void quitpressed();
};

#endif
```

F. MainMenu.c

```
// Module Name : MainMenu.c                    MAINMENU
// Author      : Erhan SARIDOGAN, Metin BALCI
// Date        : August 1992
// This module contains the button implementations. Only the Retrieve button is
// available for this version.

#include <InterViews/button.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/world.h>
#include <stream.h>
#include <stdio.h>
#include <stdlib.h>
#include "MainMenu.h"
#include "_Dialog_9.h"
#include "_Dialog_10.h"

_Dialog_9* retrievedialog;
_Dialog_10* aggregatedialog;

MainMenu::MainMenu(const char* name) : MainMenu_core(name) { }

char helpbuffer[20000];  // hepfile.txt is read into this buffer at the beginning

void MainMenu::retrievepressed() {
 // Two windows are inserted as applications. their positions on the screen are
 // relative to the center of the MainMenu window.
 World* ret_world = GetWorld();
 World* aggregate_world = GetWorld();
 int value = 0;
 retrieveBS->GetValue(value);
 if (value != 0) {
  if (retrievedialog == nil )
        retrievedialog = new _Dialog_9("RetrieveDialog");
  if (aggregatedialog == nil )
        aggregatedialog = new _Dialog_10("AggregateDialog");
  mainstatuseditor->SelectAll();
  mainstatuseditor->DeleteSelection();
  mainstatuseditor->InsertText("Retrieve is running...", 23);

  Coord x,y ;
  Align(Center,0,0,x,y);
  GetRelative(x,y,ret_world);
  ret_world->InsertTransient(retrievedialog,this,x,y,Center);
  aggregate_world->InsertTransient(
                aggregatedialog ,this,x+335,y+275,Center);
  retrievedialog->DisplayTables();
```

```
  //read the helpfile into the buffer in order to display faster
  FILE *fptr;
  fptr = fopen("helpfile.txt", "r");
  int i = 0;
  char ch = getc(fptr);
  while ( ch != EOF ) {
  // the length and size of the buffer must be sufficient
  // it can be adjusted in _Dialog_7-core.c line 112 , helpeditor
   helpbuffer[i] = ch = getc(fptr);
   i++;
  }
  fclose(fptr);
  retrievedialog->DisplayMessage(0);
  retrieveBS->SetValue(0);
 }
}

// These buttons are not implemented
void MainMenu::createtablepressed() {
 int value = 0;
 createtableBS->GetValue(value);
 if (value != 0)  {
  mainstatuseditor->SelectAll();
  mainstatuseditor->DeleteSelection();
  mainstatuseditor->InsertText("* Create Table not implemented *", 32);
  createtableBS->SetValue(0);
 }
}

void MainMenu::inserttuplepressed() {
 int value = 0;
 inserttupleBS->GetValue(value);
 if (value != 0) {
  mainstatuseditor->SelectAll();
  mainstatuseditor->DeleteSelection();
  mainstatuseditor->InsertText("* Insert Tuple not implemented *", 32);
  inserttupleBS->SetValue(0);
 }
}

void MainMenu::modifypressed() {
 int value = 0;
 modifyBS->GetValue(value);
 if (value != 0) {
  mainstatuseditor->SelectAll();
  mainstatuseditor->DeleteSelection();
  mainstatuseditor->InsertText("* Modify not implemented *", 26);
  modifyBS->SetValue(0);
 }
}

void MainMenu::deletepressed() {
 int value = 0;
 deleteBS->GetValue(value);
```

```
 if (value != 0) {
  mainstatuseditor->SelectAll();
  mainstatuseditor->DeleteSelection();
  mainstatuseditor->InsertText("* Delete not implemented *", 26);
  deleteBS->SetValue(0);
 }
}

void MainMenu::quitpressed() {
 int value = 0;
 quitBS->GetValue(value);
 if (value != 0) {
  cout << "Thank You for Using MDBMS.\n";
  cout.flush();
  quitBS->SetValue(0);
  exit(1);   // terminate the program
 }
}
```

G. _Dialog_7.h

```
// Module Name : _Dialog_7.h        HELP DIALOG
// Author     : Erhan SARIDOGAN, Metin BALCI
// Date       : August 1992
// Header file for help dialog.

#ifndef _Dialog_7_h
#define _Dialog_7_h

#include "_Dialog_7-core.h"

class _Dialog_7 : public _Dialog_7_core {
public:
   _Dialog_7(const char*);

   virtual void helpexitpressed();
   virtual void InsertChar(char);
   virtual void DeleteText();
};

#endif
```

H. _Dilaog_7.c

```
// Module Name : _Dialog_7.c       HELPDIALOG
// Author     : Erhan SARIDOGAN, Metin BALCI
// Date       : August 1992
// This module contains the implementations of displaying help window by reading
// the help text from a buffer.

#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/button.h>
#include <InterViews/2.6/_enter.h>
```

```
#include "_Dialog_7.h"

_Dialog_7::_Dialog_7(const char* name) : _Dialog_7_core(name) { }

void _Dialog_7::helpexitpressed() {
 int value = 0;
 helpexitBS->GetValue(value);
 if (value != 0) {
  _BS_11->SetValue(1); // BS -- Button State
                // This BS# is in _Dialog_7-core.h
  helpexitBS->SetValue(0); }
}

// Inserts a character into the text editor at the position of the cursor
void _Dialog_7::InsertChar (char c) {
 helpeditor->DeleteSelection();
 helpeditor->InsertText(&c, 1);
 helpeditor->ScrollToSelection();
}

//This function clears the text editor to avoid appending
void _Dialog_7::DeleteText () {
 helpeditor->SelectAll();
 helpeditor->DeleteSelection();
}
```

I. _Dialog_8.h

```
// Module Name : _Dialog_8.h        NATURAL LANGUAGE EDITOR DIALOG
// Author     : Erhan SARIDOGAN, Metin BALCI
// Date       : August 1992
// Header file for natural language dialog. Some functions are added.

#ifndef _Dialog_8_h
#define _Dialog_8_h

#include "_Dialog_8-core.h"
#include <InterViews/event.h>

class _Dialog_8 : public _Dialog_8_core {
public:
  _Dialog_8(const char*);

  virtual void partialsearchpressed();
  virtual void fullsearchpressed();
  virtual void descripcancelpressed();

//These functions are added for the editor
  virtual void NatLanEditorInsertChar (char);
  virtual void Handle(Event&);
};

#endif
```

J. _Dialog_8.c

```
// Module Name : _Dialog_8.c      NATURALLANGUAGEDIALOG
// Author     : Erhan SARIDOGAN, Metin BALCI
// Date       : August 1992
// This module contains the implementations of naturallanguageeditor.
// When this window is displayed, the user can type the description and
// press one of the search types to enter as a condition after the
// keyword MATCHES. When the condition is entered, this text is sent
// to q_rec. The application can access the field of q_rec or the
// sqlstring (or file) . The user can exit the window without writing
// anything by pressing the Cancel button.

#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/button.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/world.h>
#include <InterViews/event.h>
#include <ctype.h>
#include <string.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"
#include "MainMenu.h"
#include "_Dialog_8.h"
#include "_Dialog_9.h"
#include "_Dialog_11.h"

#define TRUE  1
#define FALSE 0

//make some variables visible
extern _Dialog_9*  retrievedialog;
extern _Dialog_11* buildconditiondialog;
extern _Dialog_8*  naturallanguagedialog;
extern TextBuffer* naturallanguagebuffer; //declared in _Dialog_8-core.c
                          //It is used for using the function,  "Text" to get the selected text
extern char searchtypebuf;       //declared in retrievedialog

//This declaration is put by InterViews
_Dialog_8::_Dialog_8(const char* name) : _Dialog_8_core(name) { }

// The window is exitted when this button is pressed
void _Dialog_8::descripcancelpressed(){
 int value = 0;
 descripcancelBS->GetValue(value);
 if (value != 0) {
  descripcancelBS->SetValue(0);              //Reset the Cancel button
  buildconditiondialog->containsBS->SetValue(0); //Reset the Caption button
  _BS_40->SetValue(1);               // This BS# is in _Dialog_8-core.h
  GetWorld()->Remove(naturallanguagedialog);
 }
}
```

```
void _Dialog_8::fullsearchpressed() {
 int value = 0;
 fullsearchBS->GetValue(value);
 if (value != 0) {
  //put in 'f' for the search type.
  //It can be changed later depending upon the application program
  searchtypebuf = 'f';
  // send the contents to conditioninput editor
  char nlb[MAXNATDESC];     //create a buffer
  int len = naturallanguageeditor->Dot();     //find the end of text
  strncpy(nlb, naturallanguagebuffer->Text(0,len), len);  //copy text
  nlb[len] = '\0';   // add the null byte
  int i = 0;
  // insert the keyword "MATCHES" into the condition editor
  buildconditiondialog->conditioninputeditor->
                    InsertText(" MATCHES\n\"",10 );
  while (nlb[i] != '\0') {
   buildconditiondialog->conditioninputeditor->InsertText(&nlb[i],1);
   i++;
  }
  //insert the last quote
  buildconditiondialog->conditioninputeditor->InsertText("\"",1);
  fullsearchBS->SetValue(0);      //reset buttons
  buildconditiondialog->containsBS->SetValue(0);
  _BS_40->SetValue(1);          // This BS# is in _Dialog_8-core.h
  GetWorld()->Remove(naturallanguagedialog);
 }
}

void _Dialog_8::partialsearchpressed() {
 int value = 0;
 partialsearchBS->GetValue(value);
 if (value != 0) {
  searchtypebuf = 'p';
  // send the contents to conditioninput editor
  char nlb[MAXNATDESC];   //create a buffer
  int len = naturallanguageeditor->Dot();
  strncpy(nlb, naturallanguagebuffer->Text(0,len), len);
  nlb[len] = '\0';
  int i = 0;
  buildconditiondialog->conditioninputeditor->InsertText(" MATCHES\n\"",10 );
  while (nlb[i] != '\0') {
   buildconditiondialog->conditioninputeditor->InsertText(&nlb[i],1);
   i++;
  }
  //insert the last quote
  buildconditiondialog->conditioninputeditor->InsertText("\"",1);
  partialsearchBS->SetValue(0);
  buildconditiondialog->containsBS->SetValue(0);
  _BS_40->SetValue(1); // This BS# is in _Dialog_8-core.h
  GetWorld()->Remove(naturallanguagedialog);
 }
}
```

```
//This function provides handling the user keyboard input
//The window mus bu active(the mouse cursor must be on the this window)
//The fuction can eable the user to insert or delete a character,and
//position the cursor by using the mouse

void _Dialog_8::Handle(Event& e) {
 if (e.eventType == KeyEvent) {
  if (e.len != 0) {
   char c = e.keystring[0];
   switch (c) {
   case '\010':
   case '\177':
    if (naturallanguagedialog->naturallanguageeditor->Dot() !=
          naturallanguagedialog->naturallanguageeditor->Mark()) {
      naturallanguagedialog->naturallanguageeditor->DeleteSelection();
    } else naturallanguagedialog->naturallanguageeditor->DeleteText(-1);
    break;
   case '\015':
    naturallanguagedialog->NatLanEditorInsertChar('\n');
    break;
   default:
    if (!iscntrl(c)) naturallanguagedialog->NatLanEditorInsertChar(c);
    break;
   } //switch c
  }  // if (e.len )
 }   // if (e.event..)
 else if (e.eventType == DownEvent) {
  GetRelative(e.x, e.y, naturallanguagedialog->naturallanguageeditor );
  naturallanguagedialog->naturallanguageeditor->
   Select(naturallanguagedialog->naturallanguageeditor->Locate(e.x, e.y));
  do {
   naturallanguagedialog->naturallanguageeditor->ScrollToView(e.x, e.y);
   naturallanguagedialog-> naturallanguageeditor->
   SelectMore(naturallanguagedialog->
       naturallanguageeditor->Locate(e.x, e.y));
   Poll(e);
   GetRelative(e.x, e.y, naturallanguagedialog->naturallanguageeditor);
  } while (e.leftmouse);
 } //else if
}  // Handle function

//Used by Handle to insert a character into the editor
void _Dialog_8::NatLanEditorInsertChar (char c) {
 naturallanguagedialog->naturallanguageeditor->DeleteSelection();
 naturallanguagedialog->naturallanguageeditor->InsertText(&c, 1);
 naturallanguagedialog->naturallanguageeditor->ScrollToSelection();
}
```

K. _Dialog_9.h

```
// Module Name : _Dialog_9.h      RETRIEVE DIALOG
// Author      : Erhan SARIDOGAN, Metin BALCI
// Date        : August 1992
// Header file for retrieve dialog. Some member functions were added.
```

```
#ifndef _Dialog_9_h
#define _Dialog_9_h

#include "_Dialog_9-core.h"

class _Dialog_9 : public _Dialog_9_core {
public:
   _Dialog_9(const char*);

   virtual void returntomainmenupressed();
   virtual void helppressed();
   virtual void buildconditionentered();
   virtual void clearquerypressed();
   virtual void processquerypressed();
   virtual void nextpressed();
   virtual void cancelpressed();
   virtual void enterpressed();
   virtual void tableselected();
   virtual void attributeselected();

// These member functions were added
   virtual void DisplayAttributes(int);
   virtual void DisplayTables();
   virtual void DisplayMessage(int);
   virtual void ClearBuffer();
   virtual void CreateSqlFile();
   virtual void SetProcessQueryButton(int value);
};

#endif
```

L. _Dialog_9.c

```
// Module Name : _Dialog_9.c      RETRIEVEDIALOG
// Author      : Erhan SARIDOGAN, Metin BALCI
// Date        : August 1992
// This module contains the implementations of buttons, browsers and editors.
// The query is constructed using this window by mouse selections.
// The condition dialog box is activated by pressing the Build Condition button.
// The Query Summary display is a passive editor like the message editor.

#include <InterViews/button.h>
#include <InterViews/strbrowser.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/world.h>
#include <ctype.h>
#include <string.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include "_Dialog_7.h"
#include "_Dialog_8.h"
```

```
#include "_Dialog_9.h"
#include "_Dialog_10.h"
#include "_Dialog_11.h"
#include "_Dialog_12.h"
#include "MainMenu.h"
#include "Result.h"
#include "common.h"

#define TRUE  1
#define FALSE 0

//Make some global variables visible
extern MainMenu* mainmenu;
extern _Dialog_8* naturallanguagedialog;
extern _Dialog_9* retrievedialog;
extern _Dialog_10* aggregatedialog;
extern _Dialog_12* toolboxdialog;
extern MainMenu* mainmenu;
extern TextBuffer* querydisplaybuffer;  //declared in _Dialog_9-core.c
extern int Whereinserted;
extern char helpbuffer[20000];          //declared in MainMenu.c

extern "C" comm_with_db(char*);    //This implementation dependent C function
                                   //establishes the TCP connection

//Declare some variables in global scope
_Dialog_11* buildconditiondialog;
Result* resultdialog;
char aggregatebuffer[6];      //temporarily holds the selected aggregate
int buildconditionactive = FALSE;
int attributeselectionOK = FALSE;  //used to detect if at least one
                                   //attribute is selected

//local variable,used for accessing the attributes of the selected table
static int selectedtableindex = 0;

//These function are local and used only in this module
void DisplayResults();
void ReadResultFile();


_Dialog_9::_Dialog_9(const char* name) : _Dialog_9_core(name) { }

//When this button is pressed, first toolbox dialog is removed, then Build
//Condition dialog is cleared and removed. Some other variables are set to
//allow another query operation.

void _Dialog_9::returntomainmenupressed() {
 int value = 0;
 returntomainmenuBS->GetValue(value);
 if (value != 0) {
  if (toolboxdialog != nil)  GetWorld()->Remove(toolboxdialog);
  if (buildconditiondialog != nil) {
   buildconditiondialog->conditioninputeditor->SelectAll();
   buildconditiondialog->conditioninputeditor->DeleteSelection();
```

```
  GetWorld()->Remove(buildconditiondialog);
 //normalize the Build Condition button
  buildconditionBS->SetValue(0);
  BuildCondition = FALSE;
 }
 //remove aggregate dialog box
 if (aggregatedialog != nil) GetWorld()->Remove(aggregatedialog);
 //reset all variables
 TableIndex = 0;              //for table browser
 AttributeIndex = 0;          //for attribute browser
 TotalAttributeIndex = 0;   //total number of selected attributes
 Contains = FALSE;          //enable caption editor
 Whereinserted = FALSE;     //for checking the keyword insertion
 Nested = FALSE;              //any of the nested keyword are used or not
 Subquery = FALSE;            //subquery button is pressed
 ConditionReady = FALSE;    //to enable Process Query button
 NextTableSelected = FALSE;  //to use next button
 //reset sqlstring
 for (int p = 0; p < MAXSQL; p++) sqlstring[p] = '\0';
 //reset aggregate buffer
 aggregatebuffer[0] = '\0';
 //clear the condition record
 ClearRecord();
 //clear the table/attribute selection buffer
 ClearBuffer();
 //clear the table browser
 tablesbrowser->Clear();
 //clear Query Summary Display
 querydisplayeditor->SelectAll();
 querydisplayeditor->DeleteSelection();
 // clear the message on the main menu box
 mainmenu->mainstatuseditor->SelectAll();
 mainmenu->mainstatuseditor->DeleteSelection();
 //set the button state to the default value
 returntomainmenuBS->SetValue(0);
 //remove itself
 GetWorld()->Remove(retrievedialog);
 }
}

//When this button is pressed, the help text is read into the editor from the
//help buffer. The user can scroll through this text and use the Exit button.

void _Dialog_9::helppressed() {
 static _Dialog_7* helpdialog ;
 World* help_world = GetWorld();
 int value = 0;
 helpBS->GetValue(value);
 if (value != 0) {
  if (helpdialog == nil) helpdialog = new _Dialog_7("HelpDialog");
  Coord x,y;
  Align(Center,0,0,x,y);
  GetRelative(x,y,help_world);
  help_world->InsertTransient(helpdialog,this,x+100,y,Center);
```

```
  // display helpbuffer in helpeditor
   int len = strlen(helpbuffer);
   for (int i = 0; i < len; i++) helpdialog->InsertChar(helpbuffer[i]);
   helpBS->SetValue(0);
   boolean accept = helpdialog->Accept();  //get the exit button signal
   helpdialog->DeleteText();       //otherwise it appends to the previous one
   GetWorld()->Remove(helpdialog);
  }
}

//This button activates the build condition dialog and places it on the right side of
//the retrieval dialog box. It loads predefined joins into the q_rec and displays them.
//The button state value is not reset to 0 here in order to disable the button
//against a second use while the window is already active.

void _Dialog_9::buildconditionentered() {
 int value = 0;
 buildconditionBS->GetValue(value);
 if (value != 0) {
  //this variable is used in redirection of the output of the selection
  //enter button and disabling the next button
  BuildCondition = TRUE;
  //solve the problem when the user presses enter before build condition
  //to send the selections to the build condition editor
  buildconditionactive = TRUE;
  //a message about what to do next
  DisplayMessage(11);
  //create and insert the window
  World* bcw =GetWorld();
  buildconditiondialog = new _Dialog_11("BuildConditionDialog");
  //this provides constant location on screen relative to the retrieve dialog
  Coord x,y;
  Align(Center,0,0,x,y);
  GetRelative(x,y,bcw);
  bcw->InsertTransient(buildconditiondialog,this,x+514,y+25,Center);
  LoadPredefinedJoins();
  buildconditiondialog->DisplayJoins();
  boolean accept = buildconditiondialog->Accept();
 }
}

//When this button is pressed, all variables, records and buffers are reset.
//The query has to be built from the beginning.

void _Dialog_9::clearquerypressed() {
 int value = 0;
 clearqueryBS->GetValue(value);
 if (value != 0) {
 // reset variables
  TableIndex = 0;
  AttributeIndex = 0;
  TotalAttributeIndex = 0;
  BuildCondition = FALSE;
  Contains = FALSE;
  Whereinserted = FALSE;
```

```
  Nested = FALSE;
  Subquery = FALSE;
  ConditionReady = FALSE; //disable process query button
 // set the variable to activate the joinsbrowser
  NextTableSelected = FALSE;
 //reset sqlstring
  for (int p = 0; p < MAXSQL; p++)  sqlstring[p] = '\0';
 //reset aggregate buffer
  aggregatebuffer[0] = '\0';
 //reset records
  ClearRecord();
  ClearBuffer();
 //clear condition input editor if it is used
  if (buildconditiondialog != nil) {    //if the window is active
   buildconditiondialog->conditioninputeditor->SelectAll();
   buildconditiondialog->conditioninputeditor->DeleteSelection();
  }
 //clear attribute browser
  attributesbrowser->Clear();
 //clear Query Summary Display
  querydisplayeditor->SelectAll();
  querydisplayeditor->DeleteSelection();
  clearqueryBS->SetValue(0);
  DisplayMessage(10);
 }
}

//This button creates the result window, reads the query display editor and
//writes the contents into the sqlstring and sqlfile,then calls an
//implementation dependent function to process the query, reads the results
//and displays them in the result window.
//In order to prevent proceesing another query while the result window is active,
//the button state is reset in Result dialog.

void _Dialog_9::processquerypressed() {
 World* result_world = GetWorld();
 int value = 0;
 processqueryBS->GetValue(value);
 if (value != 0) {
  // These are for the test purposes. Default output window is used.
  cout<<"Table Name = "<< q_rec.q_table[0].table_name <<"\n";
  cout.flush();
      //the first selected attribute
  cout<<"Attrib. Name = "<<q_rec.q_table[0].attrib_array[0].attribute_name<<"\n";
  cout.flush();
  cout<<"Condition Table = "<<q_rec.q_condition[0].cond_table << "\n";
  cout.flush();
  cout<<"Condition Attribute = "<<q_rec.q_condition[0].cond_attribute<<"\n";
  cout.flush();
  cout<<"Condition Tool = "<<q_rec.q_condition[0].cond_tool << "\n";
  cout.flush();
  cout<<"Condition input = "<<q_rec.q_condition[0].cond_input <<"\n";
  cout.flush();
  cout<<"Condition log_opr = "<<q_rec.q_condition[0].cond_log_opr << "\n";
  cout.flush();
```

```
   querydisplayeditor->EndOfText();        //put the cursor to the end

   CreateSqlFile();  //write the query into a file

// ***************************************************************************
// The Retrieve(MODE) function of the first application is called here for the
// first version.
   comm_with_db(sqlstring);          //implementation dependent
   ReadResultFile();                  //implementation dependent
// ***************************************************************************

   //insert the result dialog box
   if (resultdialog == nil) resultdialog = new Result("ResultDialog");
   Coord x,y;
   Align(Center,0,0,x,y);
   GetRelative(x,y,result_world);
   result_world->InsertTransient(resultdialog,this,x+250,y,Center);

   NextTableSelected = FALSE;     //enable the next button again
   ConditionReady = FALSE;        //disable process query button
   DisplayResults();       //read the results from q_rec
   // if there are tuples to display then display a message
   if (q_rec.q_result[0].r_formatteddata[0] != '\0') DisplayMessage(6);
  }
}

//This button is used to select more than one table during selection process.
//It is disabled after the enter button pressed.
//It is enabled again when the condition is entered, or query is processed,
//or a nested query is to be built.

void _Dialog_9::nextpressed() {
  int value = 0;
  nextBS->GetValue(value);
  if (value != 0) {
    DisplayMessage(1);
    //If at least one attribute is selected allow to use the button
    //There are restrictions to use this button
      //check if it is pressed before selecting one attribute
    if (TotalAttributeIndex > 0) {
       //check if it is pressed during condition building
     if (!BuildCondition) {
       TableIndex++;
          //copy tables,attributes and aggregates into q_rec.
       strcpy(q_rec.q_table[TableIndex-1].table_name,
                              buffertable.table_name_buffer);
       for ( int j = 0; j < AttributeIndex; j++ ) {
         strcpy(q_rec.q_table[TableIndex-1].attrib_array[j].attribute_name,
                    buffertable.attrib_array_buffer[j].attribute_name);
         strcpy(q_rec.q_table[TableIndex-1].attrib_array[j].aggregate,
                    buffertable.attrib_array_buffer[j].aggregate);
       }
     // set the variable to activate the joinsbrowser
       NextTableSelected = TRUE;
```

```
    // reset buffers
      ClearBuffer();
      aggregatebuffer[0] = '\0';
      AttributeIndex = 0;
      attributesbrowser->Clear();
     } else {
      DisplayMessage(13);     //Next button is not available
     }
    } else {
     DisplayMessage(13);    //Next button is not available
    }
    nextBS->SetValue(0);
   }
}

//This button clears the selection part only. The user can select the tables
//and attributes again. The whole query is not changed.

void _Dialog_9::cancelpressed() {
 int value = 0;
 cancelBS->GetValue(value);
 if (value != 0) {
  DisplayMessage(3);
  // reset buffers
  ClearBuffer();
  aggregatebuffer[0] = '\0';
  AttributeIndex = 0;
  TableIndex = 0;
  attributesbrowser->Clear();
  cancelBS->SetValue(0);
 }
}

//This button copies the selection buffer into q_rec and resets indexes;
//then it writes to Query Summary Display from the q_rec when it is first
//pressed. It echos the selected tables and attributes in the query
//summary display for the first selection process..
//When it is used for condition input editor no echo can be seen.
//Only one attribute at a time for each table can be entered into the condition editor.

void _Dialog_9::enterpressed() {
 int value = 0;
 char t[20], a[20];   //buffers to hold only one table and attribute
 int n = 0,    count = 0;
 int tlen = 0,  alen = 0;
 enterBS->GetValue(value);
 if (value != 0) {
  //if the button is pressed after condition enter pressed
  //don't do anything, exit the function.
  if (ConditionReady) goto end;
  //check if at least one attribute is selected
  if (!attributeselectionOK) {
   DisplayMessage(19);
   goto end;
  }
```

```
if (!BuildCondition) {    // in order to use for condition input
 DisplayMessage(5);
 strcpy(q_rec.q_table[TableIndex].table_name,
                        buffertable.table_name_buffer);
 //if aggregate all is selected, then copy all the attributes of that
 //table into q_rec
 if (strcmp(aggregatebuffer, "*")==0) {     //all is pressed
  int k = 0;
  while (dbtables[selectedtableindex].
            attrib_array[k].attribute_name[0] != '\0') {
   strcpy(q_rec.q_table[TableIndex].attrib_array[k].attribute_name,
      dbtables[selectedtableindex].attrib_array[k].attribute_name);
   //empty the aggregates of these selected attributes
   q_rec.q_table[TableIndex].attrib_array[k].aggregate[0] = '\0';
   k++;
  }
 } else {  //the button is used to send the selections to condition editor
   for ( int j = 0; j < AttributeIndex; j++ ) {
    strcpy(q_rec.q_table[TableIndex].attrib_array[j].attribute_name,
          buffertable.attrib_array_buffer[j].attribute_name);
    strcpy(q_rec.q_table[TableIndex].attrib_array[j].aggregate,
          buffertable.attrib_array_buffer[j].aggregate);
   }
  }
  TableIndex++;
  ClearBuffer();
 //if nested query is not to be built, then clear the display editor
 //from table or attribute echos
 if ((!Nested)||(!Subquery)) {
  querydisplayeditor->SelectAll();
  querydisplayeditor->DeleteSelection();
 }
// write to Query Summary Display from q_rec

//************** "SELECT" part **************
 querydisplayeditor->InsertText("SELECT ",7);
// write attributes
 if (strcmp(aggregatebuffer, "*")==0)  //all attributes are selected
        querydisplayeditor->InsertText("*", 1);
 else {
  for (int m = 0; m < TableIndex; m++) {
 // loop until finding an empty attributename
    while (q_rec.q_table[m].attrib_array[n].attribute_name[0] != '\0') {
     strcpy(a, q_rec.q_table[m].attrib_array[n].attribute_name);
     alen = strlen(a);
    //check if there is any aggregate function
     if (q_rec.q_table[m].attrib_array[n].aggregate[0] == '\0')
                querydisplayeditor->InsertText(a,alen);
     else {
      //write aggregates in the form of " SUM(...) "
      int ln = strlen(q_rec.q_table[m].attrib_array[n].aggregate);
      querydisplayeditor->InsertText(
            q_rec.q_table[m].attrib_array[n].aggregate, ln);
      querydisplayeditor->InsertText("(", 1);
      querydisplayeditor->InsertText(a,alen);
```

```
        querydisplayeditor->InsertText(")", 1);
       } //else
       count++;
       //check the number of attributes in one line
       if (count < TotalAttributeIndex)
                querydisplayeditor->InsertText(", ",2);
       //write to the second row after 3 attributes
       if ( (count%3 == 0) && (count != TotalAttributeIndex) )
                querydisplayeditor->InsertText("\n      ",8);
       n++;
      } //end while
      n = 0;
    } // end for loop
   } //else
   TotalAttributeIndex = 0;

   //************** "FROM" part ****************
   //write the table names
   querydisplayeditor->InsertText("\nFROM   ",8);
   for ( int f = 0; f < TableIndex; f++ ) {
    strcpy(t, q_rec.q_table[f].table_name);
    tlen = strlen(t);
    querydisplayeditor->InsertText(t,tlen);
    if ( f < (TableIndex-1) ) querydisplayeditor->InsertText(", ",2);
   }
   // reset indexes
   TableIndex = 0;
   AttributeIndex = 0;
   attributesbrowser->Clear();
}
else {  // now the table/attrib browsers are to be used for condition input

   //if enter pressed second time to send the selections to condition
   //editor BEFORE build condition is pressed, then
   //don't do anything, exit the function.
   if (!buildconditionactive) {
    DisplayMessage(18);
    goto end;
   }
   DisplayMessage(9);
   TableIndex = 0;
   AttributeIndex = 0;
   buildconditiondialog->conditioninputeditor->
         InsertText(buffertable.table_name_buffer,
             strlen(buffertable.table_name_buffer));
   buildconditiondialog->conditioninputeditor->InsertText(".",1);
   buildconditiondialog->conditioninputeditor->
         InsertText(buffertable.attrib_array_buffer[0].attribute_name,
             strlen(buffertable.attrib_array_buffer[0].attribute_name));
   attributesbrowser->Clear();
} // for else

//to enable condition editor when this box is active
BuildCondition = TRUE;
attributeselectionOK = FALSE;       //for the next time
```

```
end:
  enterBS->SetValue(0);
  }
}

//When a table is selected by double-clicking,its string value is written
//into a buffer and its index is returned from the browser.
//This index is later used to find the attributes of that table and display
//in the attributes browser.

void _Dialog_9::tableselected() {
 int value = 0;
 char* t;
 tablesBS->GetValue(value);
 if (value != 0) {
  //get the chosen string
  selectedtableindex = tablesbrowser->Selection(0);  // select a tablename
  t = tablesbrowser->String(selectedtableindex);
  if (!BuildCondition) {
   if ((!Nested)||(!Subquery)) {
    //display the selected table in the query display ( echoing )
    querydisplayeditor->SelectAll();
    querydisplayeditor->DeleteSelection();
    querydisplayeditor->InsertText("FROM ", 6);
    querydisplayeditor->InsertText(t,strlen(t));
   }
  }
  strcpy(buffertable.table_name_buffer,t);  // copy to buffer
  DisplayAttributes(selectedtableindex);
  if (BuildCondition) DisplayMessage(12);
  else DisplayMessage(1);
  tablesBS->SetValue(0);
 }
}

//The selected attributes are written into the buffer. more than one attribute
//can be selected.

void _Dialog_9::attributeselected() {
 int value = 0;
 int index;
 char* at;
 attributesBS->GetValue(value);
 if (value != 0) {
  //get the chosen string
  index = attributesbrowser->Selection(0); // select an attribute
  at = attributesbrowser->String( index ); // char ptr to be sent to buffer
  //display the selected attributes one by one
  //available for only table/attribute selection, not for building condition
  if (!BuildCondition) {
   if ((!Nested)||(!Subquery)) {
    //display the selected attribute in query summary display
    querydisplayeditor->SelectAll();
    querydisplayeditor->DeleteSelection();
    querydisplayeditor->InsertText("SELECT ", 8);
```

```
    if (aggregatebuffer[0] != '\0') {  //display the aggregate function also
     querydisplayeditor->InsertText(aggregatebuffer,
                    strlen(aggregatebuffer));
     querydisplayeditor->InsertText("(", 1);
     querydisplayeditor->InsertText(at,strlen(at));
     querydisplayeditor->InsertText(")", 1);
    } else  querydisplayeditor->InsertText(at,strlen(at));
   }
  }
  if (BuildCondition) DisplayMessage(8);
  else DisplayMessage(2);
  strcpy(buffertable.attrib_array_buffer[AttributeIndex].attribute_name,at);
  //copy the current value in aggregatebuffer, even an empty string
  strcpy(buffertable.attrib_array_buffer[AttributeIndex].aggregate,
                                    aggregatebuffer);
  AttributeIndex++;  //ready for the next attrib selection
  TotalAttributeIndex++;
  //reset aggregatebuffer for the next selection
  aggregatebuffer[0] = '\0';
  attributeselectionOK = TRUE;  //at least one attribute has been selected
  attributesBS->SetValue(0);
 }
}

//This function is called when the retrieve button is pressed from the main
//menu. It displays the table names by reading from dbtables.

void _Dialog_9::DisplayTables(){
 int i = 0;
 while (dbtables[i].table_name[0] != '\0') {
  tablesbrowser->Append(dbtables[i].table_name);
  i++;
 }
}

//This function displays the attributes of a selected table.
//The input index is used to locate the table entry in the dbtables.

void _Dialog_9::DisplayAttributes(int index){
 int aind = 0;
 attributesbrowser->Clear();
 while (dbtables[index].attrib_array[aind].attribute_name[0] != '\0') {
  attributesbrowser->Append(
              dbtables[index].attrib_array[aind].attribute_name);
  aind++;
 }
}

//****************************** MESSAGES ******************************
//This is a local variable used to store the messages.
//The message window is 49 characters width.

char* messages[200] = {
 "M0-Select a table with double-click.          ",
 "M1-Select the attributes.\n  You may choose an aggregate function before\n\
```

```
 each attribute.                    ",
 "M2-You may select another attribute or\n  if you want to use another table\
press Next\n  otherwise press Enter to accept selections.              ",
 "M3-Selections are cancelled.\n  Please select a new Table.              ",
 "M4-Condition has been entered.\n  The query may be processed now.          ",
 "M5-Selections have been entered.\n  Now you can process the query or build\
a condition.          ",
 "M6-If you have media data, select the line\n  then press the Show Picture\
or Play Sound buttons.          ",
 "M7-Condition has been cleared.\n  Select a tool or type it with space \
before and after,\n  then type the condition and press Enter.          ",
 "M8-Press Enter to accept this attribute.          ",
 "M9-Select a tool or type it between spaces,\n  then type the condition\
and press Enter.\n  Use the Caption Editor if you have a caption.          ",
 "M10-Whole query has been cleared.\n  Select a table.\
          ",
 "M11-Build Condition Window is activated.\n  You can activate the Tool\
Box.\n  Select or type table and attribute names.          ",
 "M12-Select an attribute.          ",
 "M13-Next button is not available.          ",
 "M14-Too many results.\n  There are undisplayed tuples\n \
 Please use conditions to limit your query.          ",
 "M15-Result file is not found.          ",
 "M16-Condition has been cleared.\n  Select or type table and attribute\
names      ",
 "M17-Subquery is beeng built.\n  Start a query with a nesting keyword.\n\
  then select a table.          ",
 "M18-Build Condition Window is not active.\n  Please use Build Condition \
button first.          ",
 "M19-There is no selected attribute.\n  Please select an attribute\
to continue.          ",
 "E1-Syntax error in  attribute name.\n  Please use Clear Query and\
try again.      ",
 "E2-Incomplete condition\n  Please use Clear Query and try again.\
          ",
 "E3-Missing condition input\n  Please use Clear Query and try again.\
          ",
 "E4-Missing natural language description \n  Please use Clear Query\
and try again.          ",
 "E5-Invalid condition tool \n  Please use Clear Query and try again.\
          ",
};


//This function displays the message for given index.

void _Dialog_9::DisplayMessage(int index){
 messageeditor->SelectAll();
 messageeditor->DeleteSelection();
 messageeditor->InsertText(messages[index],strlen(messages[index]));
}


//This function clears the buffer to store the selected table and its attributes
```

```
void _Dialog_9::ClearBuffer() {
 buffertable.table_name_buffer[0] = '\0';
 for ( int k = 0; k < MAXATTRIB; k++ )
  buffertable.attrib_array_buffer[k].attribute_name[0] = '\0';
}

//This function reads the query summary editor and writes into a string and
//a text file. This string or file can be used to be passed to the database program
//The created filename is : sqlfile ;  string is : sqlstring

void _Dialog_9::CreateSqlFile() {
 FILE* fp;
 int length = querydisplayeditor->Dot();
 strncpy( sqlstring, querydisplaybuffer->Text(0,length), length);
 // for test purposes print the contents
cout << "\nsqlstring is:\n" << sqlstring << "\n";
cout.flush();
 //open and write into a file
 fp = fopen("sqlfile","w");
 fprintf(fp, "%s\n", sqlstring);
 fclose(fp);
}

//This function sets the proces query button when resultdialog->Returnretrieval
//button is pressed

void _Dialog_9::SetProcessQueryButton(int value) {
 processqueryBS->SetValue(value);
}

// ********************** LOCAL FUNCTIONS ****************************

//This function reads the results from the q_rec and displays them in the
//result window. It reads until the first empty record.
//It writes a message if the first record is empty.

void DisplayResults() {
 int i = 0;
 if (q_rec.q_result[i].r_formatteddata[0] == '\0') {
  resultdialog->resultbrowser->Append("No record is found");
 }else {
  while (q_rec.q_result[i].r_formatteddata[0] != '\0') {
   resultdialog->resultbrowser->Append(q_rec.q_result[i].r_formatteddata);
   i++;
  }
 }
}

//This function is implementation dependent.
//For this particular program, this function reads the query results from
//a file into the result fields of the q_rec.
//Also, the picture ids (filenames) are read from a file (pictureids).

void ReadResultFile() {
 FILE *rf, *pf;
```

```
int MaxBuf = 20000;
char buf[20000];  //it should have enough size to hold the retrieved results
char linebuf[1000];
char picbuf[3000];    //for picture id file
char piclinebuf[30];  //for one picture file name
int rlen = 0, plen = 0;  //some index variables
int rline = 0, pline = 0;
int ri = 0,   pi = 0;
int rn = 0,   pn = 0;

//read result file
rf = fopen("/n/aquarius/work/rowe/cap/demo/sqloutput", "r");
if (rf == nil) {   //if the result file is not found
 retrievedialog->DisplayMessage(15);
 goto end;
}
else {
 char rch = getc(rf);
 while (( rch != EOF ) && (rlen < MaxBuf)){
  buf[rlen] = rch;
  rch = getc(rf);
  rlen++;
 }
 fclose(rf);
 if (rlen == MaxBuf) {
  retrievedialog->DisplayMessage(14);   //message for limiting query
  goto end;
 }
 // display the contents of the result file for test purposes
cout <<"resultfile buf=\n"<<buf<<"\n";
cout.flush();
 }

 //read the results into the q_rec fields
 while( ri < rlen ) {
  linebuf[rn] = buf[ri];
  rn++;
  ri++;
  if ( buf[ri] == '\n' ) {
   linebuf[rn] = '\0';
   strcpy(q_rec.q_result[rline].r_formatteddata, linebuf);
   rline++; //skip new line
   rn = 0;
  }
 }
 //read picture file
 pf = fopen("/n/aquarius/work/rowe/cap/demo/pictureids", "r");
 if (pf == nil) {   //the picturefile is not found
  retrievedialog->DisplayMessage(15);
  goto end;
 }
 else {
  char pch = getc(pf);
  while ( pch != EOF ) {
   picbuf[plen] = pch;
```

```
    pch = getc(rf);
    plen++;
   }
   fclose(pf);
  //display the pictureids file for test
cout <<"pictureids picbuf=\n"<< picbuf <<"\n";
cout.flush();
  }
  while( pi < plen ) {
   piclinebuf[pn] = picbuf[pi];
   pn++;
   pi++;
   if ( picbuf[pi] == '\n' ) {
    piclinebuf[pn] = '\0';
      //handle only one picture per tuple for this application
    strcpy(q_rec.q_result[pline].r_picture[0], piclinebuf);
    pline++;
    pi++;  //skip the line
    pn = 0;
   }
  }
  end:
   nil;
}
```

M. _Dialog_10.h

```
// Module Name :_Dialog_10.h          AGGREGATE DIALOG
// Authors     : Metin Balci - Erhan Saridogan
// Date        : August 1992
// This module has the declarations for the functions of the Aggregate Dialog.

#ifndef _Dialog_10_h
#define _Dialog_10_h

#include "_Dialog_10-core.h"

class _Dialog_10 : public _Dialog_10_core {
public:
   _Dialog_10(const char*);

   virtual void allpressed();
   virtual void minpressed();
   virtual void maxpressed();
   virtual void averagepressed();
   virtual void countpressed();
   virtual void sumpressed();
};

#endif
```

N. _Dialog_10.c

```
// Module Name : _Dialog_10.c          AGGREGATE DIALOG
// Authors     : Metin Balci - Erhan Saridogan
```

```
// Date          : August 1992
// This module has the definitions for the functions of the Aggregate Dialog.
// When a button is pressed its keyword is copied into a buffer which is to be
// used later in retrieve dialog, when selections are entered.

#include <InterViews/button.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/strbrowser.h>
#include <string.h>
#include "_Dialog_9.h"
#include "_Dialog_10.h"

#define TRUE 1

_Dialog_10::_Dialog_10(const char* name) : _Dialog_10_core(name) { }
extern char aggregatebuffer[6];          //defined in retrievedialog
extern _Dialog_9* retrievedialog;
extern attributeselectionOK;

void _Dialog_10::allpressed() {
 int value = 0;
 allBS->GetValue(value);
 if (value != 0) {
  strcpy(aggregatebuffer,"*");
  attributeselectionOK = TRUE;
  allBS->SetValue(0);
 }
}

void _Dialog_10::minpressed() {
 int value = 0;
 minBS->GetValue(value);
 if (value != 0) {
  strcpy(aggregatebuffer,"MIN");
  minBS->SetValue(0);
 }
}

void _Dialog_10::maxpressed() {
 int value = 0;
 maxBS->GetValue(value);
 if (value != 0) {
  strcpy(aggregatebuffer,"MAX");
  maxBS->SetValue(0);
 }
}

void _Dialog_10::averagepressed() {
 int value = 0;
 averageBS->GetValue(value);
 if (value != 0) {
  strcpy(aggregatebuffer,"AVG");
  averageBS->SetValue(0);
 }
}
```

```
void _Dialog_10::countpressed() {
 int value = 0;
 countBS->GetValue(value);
 if (value != 0) {
  strcpy(aggregatebuffer,"COUNT");
  countBS->SetValue(0);
 }
}

void _Dialog_10::sumpressed() {
 int value = 0;
 sumBS->GetValue(value);
 if (value != 0) {
  strcpy(aggregatebuffer,"SUM");
  sumBS->SetValue(0);
 }
}
```

O. _Dialog_11.h

```
// Module Name : _Dialog_11.h          BUILDCONDITION DIALOG
// Authors     : Metin Balci - Erhan Saridogan
// Date        : August 1992
// Explanation : This module has the declarations for the functions of
// BuildCondition Dialog.

#ifndef _Dialog_11_h
#define _Dialog_11_h

#include "_Dialog_11-core.h"
#include <InterViews/event.h>

class _Dialog_11 : public _Dialog_11_core {
public:
  _Dialog_11(const char*);
  // Functions that are created by the Interviews due to the core
  virtual void conditionexitpressed();
  virtual void conditionclearpressed();
  virtual void conditionenterpressed();
  virtual void joinselected();
  virtual void toolboxpressed();
  virtual void containspressed();

  // Functions that are created later on to support the other functions.
  virtual void DisplayJoins();
  virtual void ConditionParser();
  virtual void Handle(Event& e); // There is a detailed
                    // example in the refence manual
  virtual void CondInputEditorInsertChar (char);
};

#endif
```

P. _Dialog_11.c

```
//Module Name : _Dialog_11.c          BUILDCONDITION DIALOG
//Authors     : Metin Balci - Erhan Saridogan
//Date        : August 1992
//Explanation : This module has the definations for the functions of
//the BuildCondition Dialog. Dialog_11.sav in ~mdbms/rowedb includes also
//the codes which was added for debugging.

// header files added by Interviews
#include <InterViews/button.h>
#include <InterViews/strbrowser.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/glue.h>
#include <InterViews/scene.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/world.h>

//application dependent header files
#include <ctype.h>
#include <string.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>

#include "MainMenu.h"
#include "_Dialog_8.h"
#include "_Dialog_9.h"
#include "_Dialog_11.h"
#include "_Dialog_12.h"
#include "Result.h"
#include "common.h"

#define TRUE  1
#define FALSE 0

extern _Dialog_9* retrievedialog;
extern TextBuffer* conditioninputbuffer;// defined in the core file.
                        // used to get the text part
                        // of the conditon input editor
extern char* PredefinedJoins[MAXJOINNUM];
extern _Dialog_11*  buildconditiondialog;
extern int Nested;
extern int buildconditionactive;      //declared in retrieve dialog

_Dialog_8*  naturallanguagedialog; // used for entering the natural
                    // language description_
_Dialog_12* toolboxdialog;     // tools which are used in query build
boolean Whereinserted = FALSE; // variable used to determine if Where
                  // statement is inserted to the SQL Query Summary
char searchtypebuf;      // to get the search type from the caption editor

_Dialog_11::_Dialog_11(const char* name) : _Dialog_11_core(name) { }
```

```
// Join selected: if there are more than one table in the data base
// then, there will be predefined joins displayed in this dialog box.
// This procedure explains the things that are to be done when one of
// these joins is selected.
// Gets  the join conditon and put it into the SQL Summary Window
void _Dialog_11::joinselected() {
 int index;
 int value = 0;
 joinsBS->GetValue(value);
 if (value != 0) {
   joinsBS->SetValue(0);
   index = joinsbrowser->Selection(0);
   char j[MAXJOIN];
   int jlen;
   strcpy( j, PredefinedJoins[index]);
   jlen = strlen(j);
   strcpy(q_rec.q_predef_join[0], j );
  // handles only one
   if (!Whereinserted){
    retrievedialog->querydisplayeditor->InsertText("\nWHERE ",8);
    Whereinserted = TRUE;
    retrievedialog->querydisplayeditor->InsertText("( ", 2);
    retrievedialog->querydisplayeditor->InsertText(j, jlen);
    retrievedialog->querydisplayeditor->InsertText(" )", 2);
   }
  }
}

//contains pressed : If there is a media data that you want to have a
//condion by using caption method, you will press contains button,
//after you specify the attribute. Another editor will pop up and
//you can enter the caption in this editor,and specify the type of the
//search in this editor. Appropriate additions are to be made to your
//query by the system when you exit the editor.
void _Dialog_11::containspressed() {
 int value = 0;
 containsBS->GetValue(value);
 if (value != 0) {
  World* nlw = GetWorld();
  naturallanguagedialog = new  _Dialog_8("NaturalLanguageDialog");
  Coord x,y;
  Align(Center,0,0,x,y);
  GetRelative(x,y,nlw);
  nlw->InsertTransient(naturallanguagedialog,this,x,y,Center);
  boolean accept = naturallanguagedialog->Accept();
 }
}


//conditionentered: The condition on the screen will be passed
// to the SQL Summary Window and to the condition part of the query
//record.
void _Dialog_11::conditionenterpressed() {
 int value = 0;
```

```
int i = 0;
int len = 0;
conditionenterBS->GetValue(value);
if (value != 0) {
 conditionenterBS->SetValue(0);
 // if a join condition already selected there is no need to
 //insert another extra WHERE
 if (!Whereinserted){
  retrievedialog->querydisplayeditor->InsertText("\nWHERE ",8);
  Whereinserted = TRUE;
 }

// if there is any predef join skip WHERE part put an AND
  if (NextTableSelected) {//if there is a next table selected, then
                // it means that there is a join selection
   retrievedialog->querydisplayeditor->InsertText("\n     AND ",12);
   retrievedialog->querydisplayeditor->InsertText("\n     (",9);
  }
  // gets the conditon and puts into the SQL Summary Window
  len =  conditioninputeditor->Dot();
  // clipboard will have the text portion of the condition input
  // in this way we have the chance of going through condition
  // input and check it.
  strncpy(clipboard,  conditioninputbuffer->Text(0,len), len);
  clipboard[len] = '\0';
  while ( clipboard[i] != '\0' ) {
   retrievedialog->querydisplayeditor->InsertText(&clipboard[i],1);
   if (clipboard[i] == '\n' )
       retrievedialog->querydisplayeditor->InsertText("     ",6);
   i++;
  }
  if (NextTableSelected)
           retrievedialog->querydisplayeditor->InsertText(" )",2);

/*****************************************************************************/
// THIS FUNCTION (ConditionParser() )IS NOT CALLED FOR VERSION 3
// ( PROF. ROWE'S DATABASE APPLICATION)
// In version 3 the checks are made by the application program.
  // This function gets the conditon input and parses it and puts
  // the appropriate parts into the query record.
  // It will display some error messages in case an incorrect condi
  // tion has been entered.
  // ConditionParser();
/*****************************************************************************/

 retrievedialog->DisplayMessage(4);    //condition is entered
                        //for this version only

 //if nested is pressed then we need to close the paranthesis
 if (Nested) retrievedialog->querydisplayeditor->InsertText(" )",2);

 BuildCondition = FALSE;
 ConditionReady = TRUE;  //to enable process query button

  //clear condition input editor
```

```
    conditioninputeditor->SelectAll();
    conditioninputeditor->DeleteSelection();
     // Ready to enter another conditon
   }
 }

// Condition Clear : In case of a mistake in building the condition
// you can clear the condition.This function will clear the editor and
// the record.
void _Dialog_11::conditionclearpressed() {
  int value = 0;
  conditionclearBS->GetValue(value);
  if (value != 0) {
    conditionclearBS->SetValue(0);
    ConditionReady = FALSE;
      //clear condition input editor
    retrievedialog->DisplayMessage(7);
    conditioninputeditor->SelectAll();
    conditioninputeditor->DeleteSelection();
      //a message
    retrievedialog->DisplayMessage(16);
      //reset condition record in q_rec
    ClearConditionRecord();
   }
 }
// Condition Exit : If you decide to exit the dialog box.
void _Dialog_11::conditionexitpressed() {
  int value = 0;
  conditionexitBS->GetValue(value);
  if (value != 0) {
    //clear condition input editor
    conditioninputeditor->SelectAll();
    conditioninputeditor->DeleteSelection();
    //reset condition record in q_rec
    ClearConditionRecord();
      //clear message editor
    retrievedialog->messageeditor->SelectAll();
    retrievedialog->messageeditor->DeleteSelection();

    BuildCondition = FALSE;      //to be able to use the button again
    buildconditionactive = FALSE;
    conditionexitBS->SetValue(0);
    retrievedialog->buildconditionBS->SetValue(0);   //enable the button
    toolboxBS->SetValue(0);      //if toolbox button is active
    _BS_37->SetValue(1);         //this number is in _Dialog_11-core.h
    GetWorld()->Remove(buildconditiondialog);
   }
 }

// Toolbox Pressed :In order to see the tools that are available in
// the system, you press this button.
void _Dialog_11::toolboxpressed() {
  int value = 0;
  toolboxBS->GetValue(value);
  if (value != 0) {
```

```
  World* tbw = GetWorld();
  toolboxdialog = new _Dialog_12("ToolBoxDialog");
  Coord x,y;
  Align(Center,0,0,x,y);
  GetRelative(x,y,tbw);
  tbw->InsertTransient(toolboxdialog,this,x,y-280,Center);
//  toolboxBS->SetValue(0);
  boolean accept = toolboxdialog->Accept();
 }
}

// Display Joins : This function is to display the predefined
// joins if there are any. Predefine Joins are defined in common.c
void _Dialog_11::DisplayJoins() {
 int i = 0;
 joinsbrowser->Clear();
 while ( PredefinedJoins[i] != nil ) {
  joinsbrowser->Append(PredefinedJoins[i]);
  i++;
 }
}

// Condition Parser : Gets the condition Input and by parsing it
// puts the appropriate parts into the query record.
// If there is any mistake in the condition displays the appropriate
// error messages. The debugging codes are left in the program
// so that future changes can be done easily.
void _Dialog_11::ConditionParser(){
 int parserstopped = FALSE;
 char cb[CONDEDITLEN];
 int parnum = 0;
 int len = 0;
 int  c = 0;       // condition array counter
 int  i = 0;
 int  j = 0;       // counter, index, indexmarker

 len =  conditioninputeditor->Dot();
 strncpy(cb,conditioninputbuffer->Text(0,len), len);
 cb[len] = '\0';

 //loop till the end of buffer array
 while (i < len) {
  // go till the first character to find beginning
  while (((cb[i] == ' ') || (cb[i] == '\n')) && (i < len) ) i++;
   if ((c > 0) && (i == len)) {
    retrievedialog->DisplayMessage(21); // Nothing after AND or OR etc.
    parserstopped = TRUE;              // Incomplete condition
    goto final;
   }

   // if the first char is "(" skip it/them
   while (cb[i] == '(') {             // when a group begins
    parnum++;
    i++;
   }
```

```
// skip blanks if there is any
while (cb[i] == ' ') i++;
//check if there is any other paranthesis
while (cb[i] == '(') {
 parnum++;
 i++;
}
//find beginning of tablename and read until dot
while (cb[i] != '.') {
 q_rec.q_condition[c].cond_table[j] = cb[i];
 i++;
  j++;
}
q_rec.q_condition[c].cond_table[j] = '\0'; // last char of tablename
j = 0;    // to be used later again
i++;      // now attribname started
if (cb[i]==' ') {
 retrievedialog->DisplayMessage(20); // Syntax error in attr.name
 parserstopped = TRUE;
 goto final;
}
while (cb[i] != ' ') {       //read attribname until blank
 q_rec.q_condition[c].cond_attribute[j] = cb[i];
 i++;
 j++;
}
q_rec.q_condition[c].cond_attribute[j] = '\0'; //last char of attname
j = 0;
char opr_buffer[11];  //to temporarily hold the cond. tool

for (int m=0; m<11; m++ ) opr_buffer[m] = '\0';
while ((cb[i] == ' ') && (i< len)) i++;  //find beginning of operator
if (i==len){
 retrievedialog->DisplayMessage(21); //Incomplete condition No operator
 parserstopped = TRUE;
 goto final;
}
while ((cb[i] != ' ') && (cb[i] != '\n')) {  //find end of operator
   opr_buffer[j] = cb[i];
  i++;
  j++;
 opr_buffer[j] = '\0';   // last char of tool

 //test for the appropriate tool
if ( (strcmp(opr_buffer,"MATCHES") == 0) ||
    (strcmp(opr_buffer,"LIKE") == 0) ||
    (strcmp(opr_buffer,"=") == 0) ||
    (strcmp(opr_buffer,"<") == 0) ||
    (strcmp(opr_buffer,">") == 0) ||
    (strcmp(opr_buffer,"<=") == 0) ||
    (strcmp(opr_buffer,"=>") == 0) )
     { cout << "in the first part of the operator clause" <<"\n";
      cout.flush(); }
else {
 retrievedialog->DisplayMessage(24);  // invalid Condition Tool
```

```
  parserstopped = TRUE;
  goto final;
 }
 if ( (strcmp(opr_buffer,"MATCHES")) != 0) {
  //copy operator to tool
  strcpy(q_rec.q_condition[c].cond_tool,opr_buffer);
  j = 0;
   while ((cb[i] == ' ')&& (i < len))  i++;   //find beginning of condition input
   if (i == len ) {
     retrievedialog-> DisplayMessage(22); // Missing conditon input
     parserstopped = TRUE;
     goto final;
   }
   if (cb[i]== '"') {
    i++;
    while (cb[i] != '"' ) {
     q_rec.q_condition[c].cond_input[j] = cb[i];
     i++;
     j++;
    }
    i++;
   }
   else {
   //read cond input until blank ,null or quote
     while ((cb[i] != 32) && (cb[i] != 0) && (cb[i] != '"')) {
      q_rec.q_condition[c].cond_input[j] = cb[i];
      i++;
      j++;
     }
   }
   q_rec.q_condition[c].cond_input[j] = '\0';

    // last char of cond input.  here all input is copied !!!
  while ((cb[i] == ' ') && (i < len)) i++;   //find end of group
  while (cb[i] == ')' ) {
   i++;
   parnum--;
  }
  while ((cb[i] == ' ') && (i < len)) i++;   //find end of group
  while (cb[i] == ')' ) {
   i++;
  parnum--;
  }

}
 // this is for CONTAINS or MATCHES
 //indicating that the condition contains natural language description
else {
 strcpy(q_rec.q_condition[c].cond_tool,"MATCHES");

 while ((cb[i] != '"') && ( i < len )) i++;
    //find the beginning of the natural language description

 if ( i == len ){
  retrievedialog->DisplayMessage(23);  // Missing Natural Language Description
```

```
      parserstopped = TRUE;
      goto final;
     }
     if (cb[i]== '"') {
      i++;
      j = 0;
      while (cb[i] != '"') {
       q_rec.q_condition[c].cond_nat_description.caption[j] = cb[i];
       i++;
       j++;
      }
      q_rec.q_condition[c].cond_nat_description.caption[j] = '\0';
      q_rec.q_condition[c].cond_nat_description.search_type = searchtypebuf;;
     }
     i++; // for the end of the string (last quote)
    }
     j = 0;

    //find beginning of logical operator or find the end
    while (((cb[i] == ' ') || (cb[i] == '\n') ) && (i < len)) i++;
     if ( i < len) {              // we found s.t. in 5 step
      j = 0 ;
      //start reading the condition log operator until blank or EOL
      while ((cb[i] != ' ') && (cb[i] != '\n')) {  //read logical oper until blank
       if (i < len) {
        q_rec.q_condition[c].cond_log_opr[j] = cb[i];
        i++;
        j++;
       }
      }
    // last char of cond log opr
    q_rec.q_condition[c].cond_log_opr[j] = '\0';
    j = 0;
    c++;      // increment condition array counter for grouping
    } // if (i < len)
   }
final:
if (parserstopped){
i = len+1;} // in order to get out of the first loop
else retrievedialog->DisplayMessage(4);
}

// Handle : This function is used to handle the editor functions
// of the conditioninputeditor. The mouse should be inside the editor.
//and this function will look for the keyboard inputs and when they
// are detected, will act accordingly.

void _Dialog_11::Handle (Event& e) {
   if (e.eventType == KeyEvent) {
    if (e.len != 0) {
     char c = e.keystring[0];
      switch (c) {
      case '\010':
      case '\177':
       if (conditioninputeditor->Dot() != conditioninputeditor->Mark()) {
```

```
            conditioninputeditor->DeleteSelection();
    } else conditioninputeditor->DeleteText(-1);
    break;
        case '\015':
          CondInputEditorInsertChar('\n');
          break;
        default:
          if (!iscntrl(c)) CondInputEditorInsertChar(c);
          break;
  }
     }
    } else if (e.eventType == DownEvent) {
      GetRelative(e.x, e.y, conditioninputeditor);
      conditioninputeditor->Select(conditioninputeditor->Locate(e.x, e.y));
      do {
        conditioninputeditor->ScrollToView(e.x, e.y);
        conditioninputeditor->SelectMore(conditioninputeditor->
        Locate(e.x, e.y));
        Poll(e);
        GetRelative(e.x, e.y, conditioninputeditor);
      } while (e.leftmouse);
    }
}

// if a character is keyed from the keyboard then this function will
// insert it into the conditioninputeditor.

void _Dialog_11::CondInputEditorInsertChar (char c) {
 conditioninputeditor->DeleteSelection();
 conditioninputeditor->InsertText(&c, 1);
 conditioninputeditor->ScrollToSelection();
}
```

Q. _Dialog_12.c

```
// Module Name : _Dialog_12.h        TOOL BOX DIALOG
// Author      : Erhan SARIDOGAN, Metin BALCI
// Date        : August 1992
// Header file for tool box dialog.

#ifndef _Dialog_12_h
#define _Dialog_12_h

#include "_Dialog_12-core.h"

class _Dialog_12 : public _Dialog_12_core {
public:
   _Dialog_12(const char*);

   virtual void andpressed();
   virtual void orpressed();
   virtual void openparanpressed();
   virtual void closeparanpressed();
   virtual void subquerypressed();
   virtual void lessthanpressed();
```

```
    virtual void lessorequalpressed();
    virtual void equalpressed();
    virtual void greaterorequalpressed();
    virtual void greaterthanpressed();
    virtual void unionpressed();
    virtual void intersectionpressed();
    virtual void minusentered();
    virtual void inpressed();
    virtual void notinpressed();
    virtual void existspressed();
    virtual void notexistspressed();
    virtual void toolexitpressed();
};

#endif
```

R. _Dialog_12.c

```
// Module Name : _Dialog_12.c       TOOLBOX DIALOG
// Author     : Erhan SARIDOGAN, Metin BALCI
// Date       : August 1992
// This module contains the implementation of tool buttons. When a button
// is pressed its keyword or symbol appears in the condition editor .
// The Nested and Subquery buttons are implementation dependent. For this version
// they create a nested query only in the query summary display, thereby only
// as a SQL string (and file).

#include <InterViews/button.h>
#include <InterViews/scene.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/world.h>
#include "_Dialog_9.h"
#include "_Dialog_11.h"
#include "_Dialog_12.h"
#include "common.h"

#define TRUE  1
#define FALSE 0

extern _Dialog_9* retrievedialog;
extern _Dialog_11* buildconditiondialog;
extern int Whereinserted;
extern _Dialog_12* toolboxdialog;

_Dialog_12::_Dialog_12(const char* name) : _Dialog_12_core(name) { }

void _Dialog_12::andpressed() {
  int value = 0;
  andBS->GetValue(value);
  if (value != 0) {
    buildconditiondialog->conditioninputeditor->InsertText("\nAND\n", 7);
    andBS->SetValue(0);
    }
```

```
}

void _Dialog_12::orpressed() {
 int value = 0;
 orBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" \nOR \n", 6);
  orBS->SetValue(0);
 }
}

void _Dialog_12::openparanpressed() {
 int value = 0;
 openparanBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" ( ", 3);
  openparanBS->SetValue(0);
 }
}

void _Dialog_12::closeparanpressed() {
 int value = 0;
 closeparanBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" ) ", 3);
  closeparanBS->SetValue(0);
 }
}

// This function is implementation dependent.

void _Dialog_12::subquerypressed() {
 int value = 0;
 subqueryBS->GetValue(value);
 if (value != 0) {
  Subquery = TRUE;
  retrievedialog->querydisplayeditor->InsertText(" )", 2);  //last one
  retrievedialog->querydisplayeditor->BeginningOfText();
  retrievedialog->querydisplayeditor->InsertText("\n", 1);
  retrievedialog->querydisplayeditor->BeginningOfText();
  Whereinserted = FALSE;
  BuildCondition = FALSE;       //use the selection tables
  subqueryBS->SetValue(0);
  retrievedialog->DisplayMessage(17);
 }
}

void _Dialog_12::lessthanpressed() {
 int value = 0;
 lessthanBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" < ", 3);
  lessthanBS->SetValue(0);
 }
}
```

```
void _Dialog_12::lessorequalpressed() {
 int value = 0;
 lessorequalBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" <= ", 4);
  lessorequalBS->SetValue(0);
 }
}

void _Dialog_12::equalpressed() {
 int value = 0;
 equalBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" = ", 3);
  equalBS->SetValue(0);
 }
}

void _Dialog_12::greaterorequalpressed() {
 int value = 0;
 greaterorequalBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" >= ", 4);
  greaterorequalBS->SetValue(0);
 }
}

void _Dialog_12::greaterthanpressed() {
 int value = 0;
 greaterthanBS->GetValue(value);
 if (value != 0) {
  buildconditiondialog->conditioninputeditor->InsertText(" > ", 3);
  greaterthanBS->SetValue(0);
 }
}

void _Dialog_12::unionpressed() {
 int value = 0;
 unionBS->GetValue(value);
 if (value != 0) {
  if (Subquery) {
   retrievedialog->querydisplayeditor->InsertText(" \nUNION (\n", 10);
   retrievedialog->querydisplayeditor->BeginningOfText();
  }else {
   retrievedialog->querydisplayeditor->InsertText(" \nUNION (\n", 10);
  }
  Nested = TRUE;
  BuildCondition = FALSE;        //use the selection tables
  Whereinserted = FALSE;
  unionBS->SetValue(0);
 }
}

void _Dialog_12::intersectionpressed() {
```

```
 int value = 0;
 intersectionBS->GetValue(value);
 if (value != 0) {
  if (Subquery) {
   retrievedialog->querydisplayeditor->InsertText("\nINTERSECTION (\n", 17);
   retrievedialog->querydisplayeditor->BeginningOfText();
  }
  else {
   retrievedialog->querydisplayeditor->InsertText("\nINTERSECTION (\n", 17);
  }
  Nested = TRUE;
  BuildCondition = FALSE;        //use the selection tables
  Whereinserted = FALSE;
  intersectionBS->SetValue(0);
 }
}

void _Dialog_12::minusentered() {
 int value = 0;
 minusBS->GetValue(value);
 if (value != 0) {
  if (Subquery) {
   retrievedialog->querydisplayeditor->InsertText("\nMINUS (\n", 10);
   retrievedialog->querydisplayeditor->BeginningOfText();
  } else {
   retrievedialog->querydisplayeditor->InsertText("\nMINUS (\n", 10);
  }
  Nested = TRUE;
  BuildCondition = FALSE;        //use the selection tables
  Whereinserted = FALSE;
  minusBS->SetValue(0);
 }
}

// This function is implementation dependent

void _Dialog_12::inpressed() {
 int value = 0;
 inBS->GetValue(value);
 if (value != 0) {
  if (Subquery) {
   retrievedialog->querydisplayeditor->InsertText("\nIN (\n", 7);
   retrievedialog->querydisplayeditor->BeginningOfText();
  } else {
   retrievedialog->querydisplayeditor->InsertText("\nIN (\n", 7);
  }
  Nested = TRUE;
  BuildCondition = FALSE;        //use the selection tables
  Whereinserted = FALSE;
  inBS->SetValue(0);
 }
}

// This function is implementation dependent
```

```
void _Dialog_12::notinpressed() {
 int value = 0;
 notinBS->GetValue(value);
 if (value != 0) {
  if (Subquery) {
   retrievedialog->querydisplayeditor->InsertText("\nNOT IN (\n", 11);
   retrievedialog->querydisplayeditor->BeginningOfText();
  } else {
   retrievedialog->querydisplayeditor->InsertText("\nNOT IN (\n", 11);
  }
  Nested = TRUE;
  BuildCondition = FALSE;        //use the selection tables
  Whereinserted = FALSE;
  notinBS->SetValue(0);
 }
}


void _Dialog_12::existspressed() {
 int value = 0;
 existsBS->GetValue(value);
 if (value != 0) {
  if (Subquery) {
   retrievedialog->querydisplayeditor->InsertText("\nEXISTS (\n", 11);
   retrievedialog->querydisplayeditor->BeginningOfText();
  } else {
   retrievedialog->querydisplayeditor->InsertText("\nEXISTS (\n", 11);
  }
  Nested = TRUE;
  BuildCondition = FALSE;        //use the selection tables
  Whereinserted = FALSE;
  existsBS->SetValue(0);
 }
}


void _Dialog_12::notexistspressed() {
 int value = 0;
 notexistsBS->GetValue(value);
 if (value != 0) {
  if (Subquery) {
   retrievedialog->querydisplayeditor->InsertText("\nNOT EXISTS (\n", 15);
   retrievedialog->querydisplayeditor->BeginningOfText();
  } else {
   retrievedialog->querydisplayeditor->InsertText("\nNOT EXISTS (\n", 15);
  }
  Nested = TRUE;
  BuildCondition = FALSE;        //use the selection tables
  Whereinserted = FALSE;
  notexistsBS->SetValue(0);
 }
}

void _Dialog_12::toolexitpressed() {
 int value = 0;
```

```
 toolexitBS->GetValue(value);
 if (value != 0) {
  toolexitBS->SetValue(0);
  buildconditiondialog->toolboxBS->SetValue(0);   //enable button
  GetWorld()->Remove(toolboxdialog);
 }
}
```

S. Result.h

```
// Module Name : Result.h      RESULT DIALOG
// Author     : Erhan SARIDOGAN, Metin BALCI
// Date       : August 1992
// This module contains the implementations of buttons, string browser
// and media display functions.

#ifndef Result_h
#define Result_h

#include "Result-core.h"

class Result : public Result_core {
public:
  Result(const char*);

  virtual void resultpressed();
  virtual void showpicturepressed();
  virtual void clearpicturepressed();
  virtual void playsoundpressed();
  virtual void returnretrievalpressed();
};

#endif
```

T. Result.c

```
// Module Name : Result.c      RESULT DIALOG
// Author     : Erhan SARIDOGAN, Metin BALCI
// Date       : August 1992
// This module contains the implementations of buttons, string browser and
// media display functions for the query results.
// The function xloadimage(char*) belongs to Gene Guglielmo.
// The implementation of tif-file display class is in InterViews Ref.Manual

#include <InterViews/strbrowser.h>
#include <InterViews/button.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/world.h>
#include <Unidraw/catalog.h>
#include <Unidraw/unidraw.h>
```

```
#include <Unidraw/creator.h>
#include <InterViews/canvas.h>
#include <InterViews/painter.h>
#include <InterViews/sensor.h>
#include <InterViews/display.h>
#include <InterViews/image.h>
#include <InterViews/monoglyph.h>
#include <InterViews/raster.h>
#include <InterViews/tiff.h>
#include <InterViews/session.h>
#include <InterViews/style.h>
#include <InterViews/transformer.h>
#include <InterViews/window.h>
#include <InterViews/event.h>
#include <stdio.h>
#include <stdlib.h>
#include <stream.h>
#include <string.h>
#include "Result.h"
#include "common.h"
#include "_Dialog_9.h"

#define TRUE 1
#define FALSE 0

Result::Result(const char* name) : Result_core(name) { }

extern Result* resultdialog;
extern _Dialog_9* retrievedialog;
extern char aggregatebuffer[11];
extern int Whereinserted;

Window* picturewindow[10]; //for multiple picture display
int selectedresult;        //index for browser
int picindex = 0;  //to display more than one picture at the same time

extern "C" {
 fork();
 chdir(char*),
}

//local function
void qp_xloadimage(char*);

// This class is local and used for displaying tiff files
class ScaleToFit : public MonoGlyph {
public:
  ScaleToFit(Glyph*);
  virtual void request(Requisition&)const;
  virtual void allocate(Canvas*, const Allocation&, Extension&);
  virtual void draw(Canvas*, const Allocation&)const;
private:
  Allocation allocation_;
  Transformer matrix_;
  Glyph* body_;
```

```
};


// These functions are public

void Result::resultpressed() {
 int value = 0;
 resultBS->GetValue(value);
 if (value != 0){
  resultBS->SetValue(0);
  selectedresult = resultbrowser->Selection(0);
 }
}

void Result::showpicturepressed() {
 int value = 0;
 showpictureBS->GetValue(value);
 if (value != 0) {
  char picfilename[100];
  strcpy(picfilename,"/n/aquarius/work/rowe/marie/media/data/image/");
  strcat(picfilename,q_rec.q_result[selectedresult].r_picture[0]);
  qp_xloadimage(picfilename);

/*************** this part is used when InterViews display function is used
 The necessary declarations must be made.

  Creator creator;
  Raster* raster[10];
  if (picfile[0] != '\0' ) {
   raster[picindex] = TIFFRaster::load(picfile);
   picturewindow[picindex] = new ApplicationWindow(
                    new ScaleToFit( new Image(raster[picindex])));
   picturewindow[picindex]->map();
   picindex++;
  }
****************/
  showpictureBS->SetValue(0);
 }
}

// This function deletes the displayed pictures only if the InterViews
// picture displaying functions are used.

void Result::clearpicturepressed() {
  int value = 0;
  clearpictureBS->GetValue(value);
  if(value != 0) {
   int m = 0;
   while (picturewindow[m] != nil) {
    picturewindow[m]->unmap();
    picturewindow[m] = nil;
    m++;
   }
   picindex = 0;
   clearpictureBS->SetValue(0);
```

```
  }
}


void Result::playsoundpressed() {
  int value = 0;
  char filename[20];
  playsoundBS->GetValue(value);
  if (value != 0) {
   if (q_rec.q_result[selectedresult].r_sound[0] != '\0') {
    strcpy(filename,q_rec.q_result[selectedresult].r_sound[0]);
//display the file name for test purposes
cout<<" sound filename : "<< filename;
cout.flush();
    playsoundBS->SetValue(0);
   }
  }
}

void Result::returnretrievalpressed() {
 int value = 0;
 returnretrievalBS->GetValue(value);
 if (value != 0) {
  resultbrowser->Clear();     //clear the result browser before quitting
/******** if there is any picture kill them before leaving
        when InterViews display functions are used
  int i = 0;
  while (picturewindow[i] != nil) {
   picturewindow[i]->unmap();
   picturewindow[i] = nil;
   i++;
  }
  picindex = 0;
*********/
  resultBS->SetValue(1);     // This BS name is in Result-core.h

  // to allow another query, reset the appropriate variables
  TableIndex = 0;
  AttributeIndex = 0;
  TotalAttributeIndex = 0;
  BuildCondition = FALSE;
  Contains = FALSE;
  Whereinserted = FALSE;
  Nested = FALSE;
  Subquery = FALSE;
  ConditionReady = FALSE; //disable process query button
  NextTableSelected = FALSE;
  ClearRecord();
  for (int n = 0; n < MAXSQL; n++) sqlstring[n] = '\0';
  aggregatebuffer[0] = '\0';
  retrievedialog->ClearBuffer();
  retrievedialog->DisplayMessage(11);
  ClearConditionRecord();
  //clear query display
  retrievedialog->querydisplayeditor->SelectAll();
```

```
   retrievedialog->querydisplayeditor->DeleteSelection();
   returnretrievalBS->SetValue(0);
   retrievedialog->SetProcessQueryButton(0);   //reset the button
   GetWorld()->Remove(resultdialog);
  }
}


//************** These are used to display picture files of tif format
ScaleToFit::ScaleToFit(Glyph* g ) : MonoGlyph(g) { }

void ScaleToFit::request(Requisition& r) const{
 MonoGlyph::request(r);
 ScaleToFit* s = (ScaleToFit*)this;
 Requirement& rx = r.requirement(Dimension_X);
 rx.stretch(fil);
 rx.shrink(fil);
 Coord xsize = (int)rx.natural();
 float xalign = rx.alignment();
 Allotment ax(xalign* xsize, xsize,xalign);
 s->allocation_.allot(Dimension_X, ax);
 Requirement& ry = r.requirement(Dimension_Y);
 ry.stretch(fil);
 ry.shrink(fil);
 Coord ysize = (int)ry.natural();
 float yalign = ry.alignment();
 Allotment ay(yalign* ysize,ysize,yalign);
 s->allocation_.allot(Dimension_Y,ay);
}

void ScaleToFit::allocate(Canvas* c, const Allocation& a, Extension& ext) {
 matrix_ = Transformer();
 matrix_.scale(
  a.allotment(Dimension_X).span()/
  allocation_.allotment(Dimension_X).span(),
  a.allotment(Dimension_Y).span()/
  allocation_.allotment(Dimension_Y).span()
 );
 c->push_transform();
 c->transform(matrix_);
 MonoGlyph::allocate(c,allocation_,ext);
 c->pop_transform();
}


void ScaleToFit::draw(Canvas* c, const Allocation&)const {
 c->push_transform();
 c->transform(matrix_);
 MonoGlyph::draw(c,allocation_);
 c->pop_transform();
}

//**************

// Another picture file display routine written by Gene Guglielmo
```

```
#define NARGS  10
#define FAIL   1

char *malloc();
void qp_xloadimage(char *ifile)
{
   char   *argv[NARGS];
   char   *pgm = "xloadimage";
   int    i;
   int    pid;
   char   buf[100];
   /* separate path from file name */
   strcpy(buf, ifile);

   /* set up argv for executing xloadimage */
   argv[0] = pgm;
   //   argv[1] = "-display"   no need this
   argv[1] = "-gamma";
   argv[2] = "2.5";
   argv[3] = buf;

   for (i = 4; i < NARGS; i++)  argv[i] = NULL;
   pid = fork();
   if (pid == -1) {
      fprintf(stderr, "SYSERR - Could not fork %s process\n", pgm);
      exit(FAIL);
   }
   else if (pid == 0) {
    if (execvp(pgm, argv) == -1) {
        fprintf(stderr, "SYSERR - Could not exec %s process\n", pgm);
       exit(FAIL);
    }
   }
   return;
}
```

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
Chairman, Code CS Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5100	2
Professor Neil C. Rowe Code CsRp Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5100	1
Professor C. Thomas Wu Code CsWq Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5100	1
Deniz Kuvvetleri Komutanligi Personel Daire Baskanligi Bakanliklar, Ankara / TURKEY	1
Golcuk Tersanesi Komutanligi Golcuk, Kocaeli / TURKEY	2
Deniz Harp Okulu Komutanligi 81704 Tuzla, Istanbul / TURKEY	2
Taskizak Tersanesi Komutanligi Kasimpasa, Istanbul / TURKEY	2

Erhan Saridogan 1
Florya Asfalti No: 42/5
Senlikkoy 34810
Istanbul / TURKEY

Metin Balci 1
Toprakyol, Beyaz Villa Cikmaz Sok.
Yavuz Apt. No: 18 / 8
Kartal, Istanbul / TURKEY