INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road. Ann Arbor. MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9227340

A multimedia database management system for use by educators and students in grades K through 12

Musser, Dale Roy, Ph.D.

The Ohio State University, 1992

Copyright ©1992 by Musser, Dale Roy. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A MULTIMEDIA DATABASE MANAGEMENT

SYSTEM FOR USE BY EDUCATORS AND

STUDENTS IN GRADES K THROUGH 12

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the Graduate

School of The Ohio State University

By

Dale Roy Musser, B.S.Ed., M.A.

* * * * *

The Ohio State University

1992

Dissertation Committee:

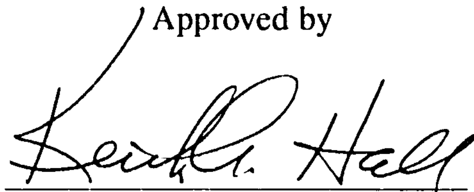
Keith A. Hall

Suzanne K. Damarin

Bradley D. Clymer

Dallas H. Miller

Approved by



Adviser, College of Education,
Educational Policy & Leadership

Copyright by

Dale R. Musser

1992

...inventions are frequently born out of a few believers' struggle with those who have something to lose from change, set in a background of indifference. Because these inventions have a certain inevitability about them, the real contribution lies in making them work. You have to believe in the idea passionately enough to carry on the struggle, until it is firmly rooted in the world and has a life of its own.

-Federico Faggin (1992)

To all those whose struggles have given me this opportunity.

ACKNOWLEDGMENTS

My sincerest appreciation goes to Thomas W. Greaves for his vision and support. Without it, this research would not have been possible.

To Keith Hall, a mentor and a friend, I am appreciative of the guidance and support he has given me. I thank him for recognizing my gifts and driving me to pursue my interests. He has contributed greatly to my professional and personal growth. I will miss our conversations.

To Suzanne Damarin, I thank her for giving me many ways to look at the world. Her views have been insightful, and she has inspired me to pursue creative solutions to problems.

To Bradley Clymer, a mentor and model for me in all aspects of my life, I owe him a great debt for his involvement in my personal growth and social development. His friendship has been most valuable.

To Dallas Miller, I am grateful for the support he has given me. His involvement with the project has gone far and above the call of duty. His input has been most helpful. His knowledge, expertise, and accomplishments have been an inspiration.

To all those at IBM and Novell who have supported and helped me during this project, especially Rick Bingham, Ed Cummins, Bob Fauver, Tom Flor, Jim Lay, and Brock Stanton. I am grateful.

VITA

October 29, 1964 . . . . Born - Ephrata, Pennsylvania

1982 - 1984 . . . . Electrical Engineering
Department of Electrical Engineering
Pennsylvania State University
State College, Pennsylvania

1984 . . . . Software and Hardware Developer
Computer Based Undergraduate Physics Labs
Department of Physics
Temple University
Philadelphia, Pennsylvania

1986 . . . . Bachelor of Science in Education
Physics and Mathematics Education
Department of Education
Shippensburg University
Shippensburg, Pennsylvania

1986 - 1988 . . . . Physics and Electronics Teacher
Hanover Public School District
Hanover, Pennsylvania

1988-1990 . . . . Graduate Teaching Associate
Department of Mathematics
The Ohio State University
Columbus, Ohio

1989 - 1990 . . . . Editor, PILOT SIG Newsletter
Association for the Development of
Computer-Based Instructional Systems
Bellingham, Washington

1989 .... Krakatau Expedition
Botanical/Zoological Research
The Ohio State University and
Oxford University, England
Krakatau Islands, Indonesia

1989 .... Master of Arts
Instructional Design & Technology
Department of Educational Policy & Leadership
The Ohio State University
Columbus, Ohio

1990 .... IBM Internship
IBM Educational Systems Division
Atlanta, Georgia

1990 - 1991 .... Graduate Research Associate
Department of Educational Policy & Leadership
The Ohio State University
Columbus, Ohio

1990 - Present .... IBM Education Instruction Specialist
IBM Education Systems Division
Atlanta, Georgia

1991 .... Consultant
BellSouth Mobility
Atlanta, Georgia

PUBLICATIONS

Hall, K. A., Musser, D. R., & Lear, G. M. (1989). Tools for developing CBI research treatments and gathering data. Proceedings of the 31st International ADCIS Conference, 261-266.

Musser, D., & Bush, M. B. (1990). Plant sampling for AIDS and cancer research. In R. J. Whittaker, N. M. Asquith, M. B. Bush, & T. Partomihardjo (Eds.), Krakatau Research Project 1989 expedition report (p. 56). Oxford, England: Krakatau Research Project, School of Geography, University of Oxford.

FIELDS OF STUDY

Major Field: • Education

(Program Area: Instructional Design and Technology)

Studies in: • Instructional Systems Design and Evaluation

Professors John C. Belland, Marjorie A. Cambre, Suzanne K. Damarin, Keith A. Hall, Richard D. Howell, and William D. Taylor

• Educational Simulations

Professor Keith A. Hall

• Research Methodology

Professors Kirby Barrick, Aage Clausen, N. L. McCaslin, David McCracken, Larry Miller, and Emma Lou Van Tilburg

• Electrical Engineering

Professors Fred Garber, Lee Hamilton, and Hooshang Hemami

TABLE OF CONTENTS

DEDICATION ____ ii

ACKNOWLEDGMENTS ____ iii

VITA ____ iv

LIST OF FIGURES ____ xi

CHAPTER PAGE

I. INTRODUCTION ____ 1

The Evolution of Computer Based Information ____ 1
The Information Processing Capabilities of Computers Today ____ 2
The Evolution from Computation Machine to Information Processor ____ 3
Multimedia ____ 4
Multimedia Database Management Systems ____ 5
Multimedia and Education ____ 6
The Need for a Multimedia Database System for Use in Educational Environments ____ 8
Source of the Need for a Multimedia Database System ____ 9
Sources of Multimedia Objects ____ 10
A Tool for a Convivial Society ____ 10
Software Research: Goals and Activities ____ 12

II. BACKGROUND ____ 17

Databases, Database Mangement Systems, and Database Systems ____ 17
Networks ____ 21
Object Orientation ____ 24
Multimedia Objects ____ 28

III. SURVEY OF EXISTING MULTIMEDIA DATABASE SYSTEMS ___ 31

Muse ___ 31
Bell Communications Research ___ 33
Domino ___ 34
The Multimedia Document Manager ___ 35
Other Systems ___ 36
Conclusions ___ 37

IV. POTENTIAL USES OF A MULTIMEDIA DATABASE SYSTEM IN A K-12 EDUCATION ENVIRONMENT ___ 38

An Exploration and Research Tool ___ 38
A Presentation Tool ___ 40
A Development Tool ___ 42
A Resource for Educational Software ___ 43
Conclusion ___ 45

V. THE DEVELOPMENT ENVIRONMENT ___ 46

Network Configuration ___ 46
Client Workstation Configuration ___ 47
Automatic Folder Creator Workstation Configuration ___ 48
Development Tools ___ 50

VI. THE AUTOMATIC FOLDER CREATOR ___ 51

Introduction ___ 51
Automatic Folder Creator Hardware Configuration ___ 52
The Automatic Folder Creation Process ___ 54
AFC Create Types ___ 57
Summary ___ 61

VII. THE DATA MODEL ___ 62

The Database Object ___ 62
The Object Handle ___ 62
The Object Referent ___ 64
Status Information ___ 66
Catalog Information ___ 67
Classifying Multimedia Objects ___ 68
Classification Criteria ___ 69
The Multimedia Object Class Hierarchy ___ 71
Exception Word, Topic, and Password Objects ___ 80
Database System Operations ___ 81

Creating an Instance of a Database 81
Connecting to and Disconnecting from a Database 82
Setting and Getting the Password 82
Adding and Removing Exception Words 82
Adding, Removing, and Getting a List of Topics 82
Getting a List of Classes Supported by the MMDBMS 83
Adding Multimedia Objects to the Database 83
Adding a Videodisc Information File 84
Searching for Objects 85
Getting Information About an Object 86
Experiencing an Object 87
Getting a Copy of an Object 87
Exporting an Object 87
Updating an Object 87
Removing an Object from the Database 88
Setting Use Locks to Zero 88
Operations on Objects of Type Videodisc Segments 88
Obtaining Error Messages 89
Summary 89

VIII. THE MULTIMEDIA DATABASE MANAGEMENT SYSTEM ARCHITECTURE 92

The Three Layer Architecture 92
The Access Method 95
The Communication Interface 98
The User Interface 100
Summary 103

IX. COMMUNICATION INTERFACE FUNCTION ALGORITHMS 104

Creating an Instance of a Database 104
Connecting to and Disconnecting from a Database 105
Setting and Getting the Password 105
Adding and Removing Exception Words 106
Adding, Removing, and Getting a List of Topics 106
Getting a List of Classes Supported by the MMDBMS 107
Adding Multimedia Objects to the Database 108
Adding a Videodisc Information File 111
Searching for Objects 111
Getting Information About an Object 115
Experiencing an Object 115
Getting a Copy of an Object 116
Exporting an Object 117
Updating an Object 117

Removing an Object from the Database 118
Setting Use Locks to Zero 118
Operations on Objects of Type Videodisc Segments 118
Obtaining Error Messages 120

X. THE USER INTERFACE: A SAMPLE APPLICATION 121

Starting MAX 121
MAX Main Functions 123
Accessing and Manipulating Objects 126
Summary 142

XI. SUMMARY AND FUTURE RESEARCH 143

Summary 143
The Data Model 144
The Automatic Folder Creator 145
The Multimedia Database System Architecture 146
The Multimedia Archive Explorer 147
Future Research and Development 147
Applications and User Interfaces 148
Cataloging and Indexing 149
Alternative Architectures 149
Examining the User's Experience 149
Expanding Access 150

APPENDICES

A. AFC Request File Formats 152

B. Descriptions of Multimedia Object Types Supported by the Multimedia Database Management System 158

C. Videodisc Information File (VIF) Format 165

D. Relational Table and Index Specification 171

E. Multimedia Database Management System Application Programming Interface 177

LIST OF REFERENCES 241

LIST OF FIGURES

FIGURE		PAGE
1.	A database system is the database management system and the database combined.	18
2.	The database management system as defined by Bordogna (et al., 1990) is structured using three layers called the access method, the communication interface, and the user interface.	20
3.	The topology of a client/server network is hierarchical.	22
4.	The toplogy of a peer to peer network is flat.	23
5.	The bird object has a state and behaviors.	25
6.	The figure shows an example class hierarchy.	26
7.	A multimedia database system can be used as an exploration and research tool.	39
8.	A multimedia database system can be used as a presentation tool.	41
9.	A multimedia database system can be used as a development tool.	43
10.	A multimedia database management system can be used as a resource for educational software.	44
11.	The network used in the development environment contained a Netware fileserver and DOS client workstations.	47
12.	The figure shows the minimum client workstation configuration.	48

13. The figure shows the configuration for the Automatic Folder Creator. ____ 49

14. The figure shows the Automatic Folder Creator hardware configuration. ____ 53

15. The Automatic Folder Creator is a client workstation on the local area network. ____ 54

16. The figure shows the process used by the Automatic Folder Creator. ____ 55

17. The diagram shows how the Automatic Folder Creator is integrated into the multimedia database system. ____ 56

18. The figure shows a snapshot of a digital video with sound (LMS) folder created by the Automatic Folder Creator. ____ 58

19. The figure shows a snapshot of a digital video and sound (LMS) folder with the folder information window displayed. ____ 58

20. The figure shows a snapshot of a slide show (LSB) folder created by the Automatic Folder Creator. ____ 59

21. The figure shows a snapshot of a slide show (LSB) folder with the folder information window displayed. ____ 59

22. The figure shows a snapshot of a sound (LAD) folder created by the Automatic Folder Creator. ____ 60

23. The figure shows a snapshot of a sound (LAD) folder with the folder information window displayed. ____ 60

24. A database object is a multimedia object plus the pieces of information associated with the object when it is stored in the database. ____ 63

25 An object handle is an eight digit base thirty-six number associated with a multimedia object stored in the database. ____ 64

26. An object referent is the way the experience method refers to a multimedia object. ____ 65

27. Status information is associated with each multimedia object stored in the database. ____ 66

28. Catalog information is associated with each object in the database. ________ 68

29. The figure shows a class hierarchy for images based on format differences and hardware dependencies. ________ 70

30. The figure shows a class hierarchy for a subset of multimedia object types based on format differences and hardware dependencies. ____ 70

31. The figure shows the **Multimedia Objects** class hierarchy for object types supported by the multimedia database management system. ________ 71

32. The figure shows the class hierarchy for **Images.** ________ 72

33. The figure shows the class hierarchy for **LinkWay Images.** ______ 73

34. The figure shows the class hierarchy for **MacPaint Images.** ________ 73

35. The figure shows the class hierarchy for **Storyboard Images.** ________ 74

36. The figure shows the class hierarchy for **Text.** ________ 74

37. The figure shows the class hierarchy for **LinkWay Pieces.** ________ 75

38. The figure shows the class hierarchy for **Sounds.** ________ 75

39. The figure shows the class hierarchy for **Digital Video.** ________ 76

40. The figure shows the videodisc object information. ________ 77

41. The figure shows the class hierarchy for **Videodisc Segments.** ________ 79

42. The figure shows the class hierarchy for **Applications.** ________ 79

43. The figure shows the class hierarchy for **Generic.** ________ 80

44. The database management system is structured using three layers called the access method, the communication interface, and the user interface. ____ 93

45. The access method is comprised of a relational record manager and operating system and network file services. ____ 96

46. The figure shows the database directory structure. ____ 96

47. The communication interface is a set of C functions constructed using functions available in the access method. ____ 98

48. The user interface is built using functions available in the communication interface layer. ____ 101

49. The Multimedia Archive Explorer (MAX) can be started from the Instructional Classroom LAN Administration System (ICLAS) menu. ____ 122

50. The Multimedia Archive Explorer (MAX) can be started from the LinkWay Tools menu. ____ 122

51. The figure shows the main MAX menu for a public user. ____ 123

52. The figure shows the main MAX menu for an administrative user. ____ 124

53. The main MAX menu is shown with the search option highlighted. Note the description of the option at the bottom of the screen. ____ 127

54. The MAX search options menu is shown. The options are used to set search criteria and to activate the search. ____ 128

55. The topic list menu is displayed after choosing to add topics to the search criteria. ____ 129

56. The class list menu is displayed after choosing to add object types to the search criteria. Multiple classes may be selected. ____ 130

57. The status list menu is displayed after choosing to add object statuses to the search criteria. Multiple statuses may be selected. The status list shown is for the public mode. ____ 131

58. The status list for an administrative user is shown. The status list is longer for an administrative user than for a public user. ____ 132

59. A search word prompt is displayed after choosing to add search words to the search criteria. Multiple search words may be entered. ________ 133

60. After search criteria have been established, the search is performed by selecting "Perform the Search" from the search options menu. ________ 134

61. The profile for the first "found" object is displayed. According to the status line at the bottom of the screen, 142 objects have been found and the search is continuing. ___ 135

62. The profile for the twenty-fifth "found" object is displayed. According to the status line at the bottom of the screen, 179 objects have been found and the search is complete. ______ 136

63. The options menu for a public user is displayed for an object. __________ 137

64. The options menu for an administrative user is displayed for an object. ________ 138

65. If a copy of an object is requested, the user is prompted for the location to put the copy of the object. _________ 139

66. The options menu is displayed for a videodisc object. Since the object is on videodisc, a request must be placed to digitize the object. ___________ 140

67. After a videodisc object is selected to be created, the user chooses a create type from a list of choices. _______ 141

CHAPTER I

INTRODUCTION

The types of information that computers can store and process have changed over time. The use of computers in education has reflected this change. In the not too distant past, computers were only capable of performing computations and storing simple forms of information. Today computers manipulate more complex forms of information including images, animations, digital video, sounds, voice, music, text, and their combinations. The term *multimedia* has gained popular usage in describing computers and software that can handle these more complex forms of information.

Multimedia has begun to find its way into educational environments because of its dynamic nature, richness, and interactivity. Missing from the multimedia computing environment in education and elsewhere is the ability to store, organize, and retrieve multimedia objects. The multimedia database management system (MMDBMS) created in this research project provides the missing functionality.

The Evolution of Computer Based Information

Early computers were computational machines. Today's computers are processors of many types of information. This section starts with a synopsis of the capabilities that the information processing computers of today possess. A historical overview of the evolution of computers follows, along with the definition of *multimedia*.

The Information Processing Capabilities of Computers Today

A student in an art class creates a piece of artwork using a computer. The work, a collage of digitized images from magazines, newspapers, and television, expresses the student's fear of AIDS. Across the room four students develop an animation that depicts the group's frustration with a school policy that confines them to study hall when they have no class . . . even when the study hall is the last period of the day. Two of the students, seated in front of a computer displaying a script for the animation, debate some of the scene transitions. A third student, using a computer, plots paths for characters and objects for one of the scenes. The fourth student, using a drawing program, paints cells for the animation.

Down the hall, in a math class, students are plotting equations of the form $Y = aX^2 + bX + c$ using a computer program called *Math Exploration Toolkit*. For each plot they vary the parameters a, b, and c and compare the graphs to determine the relationship between the parameters and the shape and direction of the parabolas that are graphed. One renegade student varies the power of the first X term and notes some interesting results.

In the library a student searches a CD-ROM of mammals for a picture of a lemur to include in a term paper on primates. For a moment the student abandons the search to listen to the chatter of some monkeys.

In the music wing a student composes a piece of music using an electronic keyboard attached to a computer via a MIDI port. The score appears on the screen as the notes are played.

In a hallway nearby, a student stands in front of a computer kiosk running an application called *Modern Solutions*. The student interacts with the computer to learn the implications of drug and alcohol consumption by watching video clips depicting the results of decisions made by the student.

The Evolution from Computation Machine to Information Processor

A few years ago computers could not perform the tasks illustrated in the previous section. The word *computer* points to the original purpose of machines by that name. Computers were developed to compute . . . to perform calculations. Computers stored and manipulated numbers.

The first step in computer evolution occurred when letters and other characters were assigned numeric values. The numbers stored in the computer became references to text that allowed computers to generate reports and display system messages such as errors. These early machines could only display upper case letters.

The capability to manipulate characters with computers led to software for this purpose: text editors, word processors, and instructional software. These applications, and others, demonstrated the need for both upper case and lower case characters and the capability was added by extending the character set.

Large amounts of numeric and textual information led to the need for its storage, organization, and retrieval. Database management systems were developed for storing the types of information that computers could process: numeric and textual.

It didn't take too long sitting at an editor or word processor to see that by placing the right characters at the right places one could draw simple pictures, diagrams, and graphs. The computer's ability to store graphical information became evident. Graphics provided an important and efficient means of relating information to users. For example, teaching trigonometry via computer was rather difficult without a graphical representation of a triangle.

Using characters to draw pictures, diagrams, and graphs was limiting because the representations were crude and the process of creating these graphic objects was difficult and time consuming. Since all information stored in a computer is comprised of numbers, a scheme was devised to convert a picture to a sequence of numbers . . . to

digitize it. In the style of the painter Seurat, the image was reduced to a series of dots; the image was sampled. If a dot was light, a one was stored in the computer, if the dot was dark, a zero was stored in the computer. The number of dots determined how well defined the image was . . . its resolution. Later enhancements added shades of gray or colors by using different numbers to represent each color or shade of gray at the sample points. Hardware was also developed to display the images stored in the computer.

If pictures could be digitized and placed in computers, so could sounds. The process was essentially the same for sounds as it was for pictures, except the samples were taken over time instead of over an area. A sound could be converted to a current or voltage that varied. The level of the current or voltage was sampled at some regular time interval and turned into a sequence of numbers that could be stored in a computer. Hardware was developed to play back the sounds stored in the computer. Techniques were developed for storing and playing back musical scores, displaying vector based graphics, generating animations, synthesizing voices and sound, and storing and playing back digital video.

The historical progression from computation machines to machines capable of processing many types of information is neither complete nor entirely accurate concerning the order with which inventions and discoveries were made. However, the intent of this discourse is to show the trend in the development of computer based information technologies.

Multimedia

Developing means by which computers can store, manipulate, and "play back" sensory experiences has been the trend. Of the five senses, sight and hearing have been the main focus. The other senses, touch, smell, and taste, have lagged because of technical constraints. The word *computer* seems somehow deficient in describing the machine with these new capabilities.

Multimedia is a word used as both an adjective and a noun to describe the capability of computers and software to handle multiple types of digital media: images, animations, digital video, sounds, voice, music, text, and their combinations. A **multimedia computer** (adjective form) is a computer capable of storing, manipulating and playing back a variety of types of digital media. **Multimedia software** (adjective form) is software that incorporates or manipulates a variety of types of digital media. ***Multimedia*** (noun form) usually refers to either software or a combination of hardware and software that uses a variety of types of digital media. Pieces of software, each incorporating and/or manipulating a single digital medium can be used in conjunction with one another to enable multimedia capabilities.

Multimedia Database Management Systems

Database management systems (DBMSs), developed before the arrival of multimedia, function as systems for storing, organizing, and retrieving simple types of computer based information: numeric, character, and character string. Many DBMSs are available today: DB2, SQL/DS, OS/2 Extended Edition, Oracle, Ingres, Sybase, SQL Server, SQLBase, and dBase IV (Groff & Weinberg, 1990). There are also DBMS products with more limited capabilities, such as Apple Works-Database, Microsoft Works-Database, and Notebook II.

The arrival of multimedia has created the need for a new type of database management system. The American National Standards Institute and International Standards Organization (ANSI/ISO) structured query language (SQL) database standard supports fixed-length character strings, integers, decimal numbers, and floating point numbers (Groff & Weinberg, 1990). Although most commercial SQL products offer an extended set of data types including variable-length character strings, money, dates and

times, boolean data, long text, unstructured byte streams, and Asian characters (Groff & Weinberg, 1990), none of them directly accept multimedia objects.

A multimedia database management system (MMDBMS) is a container for multimedia objects. An MMDBMS must supply the ability to store, organize, and retrieve a variety of types of multimedia objects in much the same way that traditional DBMSs store, organize, and retrieve numeric and character based information.

The nature of multimedia objects, along with the proposed use of an MMDBMS, must be considered in designing an MMDBMS. Among the issues that must be examined are the variety of formats for multimedia objects, the size of multimedia objects, the means by which the objects will be placed and found in the database, the compatibility of the objects with the hardware being used, and the means by which the objects will be experienced by the user once they are placed in the database.

Multimedia and Education

Because multimedia is still very young and has not been studied in any depth, there is no conclusive evidence that the use of multimedia improves instruction. The lack of evidence is also related to the difficulty of studying the question "Does multimedia improve instruction?" Instructional research takes time and of necessity lags behind innovations in technology. Many studies will be required before any notion of improvement (or undesirable effects) will be available.

Even though little empirical evidence is available to answer the question of multimedia's effectiveness in education, attributes of multimedia can be related to accepted practice and research in education. Also, the acceptance of the technology by schools, teachers, and students can lead to an understanding of multimedia's potential. The evidence of multimedia's effectiveness may be anecdotal, but it serves to fill the void until research yields some answers.

The following excerpt from Electronic Learning (September 1991, p. 22) lists some benefits of multimedia:

> **Why use multimedia technology instead of more traditional classroom tools? Based on interviews with dozens of educators and industry people, here are some of the benefits - to both student and teacher - of using multiple, integrated technologies in the classroom.**
>
> Multimedia . . .
> -**Reaches** all the senses, which enhances learning. Multimedia can be tailored to the learning styles of individuals, whether they are visual, verbal, auditory, or physical learners.
> -**Encourages** and validates self-expression. By allowing students to decide how they want to create a project - through words, images, sound, etc. - teachers are saying it's OK that students have more control and more of a voice in their own learning process.
> -**Gives** a sense of ownership to the user. Students actually create what they learn, and there is often physical evidence, such as a portfolio of work, of that learning.
> -**Creates** an active rather than passive atmosphere because it forces the student to participate and think about what they are learning. "Engages the disengaged," says one educator.
> -**Fosters** communication. The use of multimedia starts conversations between the students and teachers and allows ideas to flow in ways that may not always be possible through words alone. When a student creates what he's learning, he will probably feel more comfortable with it and want to discuss what he's done with those around him.
> -**Makes** sense. Technology is already built into the live's of today's students (television, radio, phones, computers), so it is something they feel comfortable with.
> -**Is** a lot of fun!

In 1989 Electronic Learning (June, p. 26) proclaimed, "Multimedia is taking the educational technology community by storm." A number of schools visited during the development of the multimedia database management system gave a similar impression. Although the integration of multimedia technology in these schools has not been extensive, the teachers and students were excited about multimedia technology. The showroom floor of a recent IBM Executive Conference (January 1992) displayed the wares of many educational software companies. Multimedia proliferated at this show.

That companies are willing to bring multimedia educational products to market shows that there must be buyers willing to vote with dollars. The actual procurement and use of multimedia by educators is a grand experiment whose results will in part be determined by continued support.

The Need for a Multimedia Database Management System for Use in Educational Environments

A number of database and multimedia researchers have written about the need for advances in database technology to support multimedia (Barker & Yeates, 1981; Beiser, 1990; Gano, 1988; Irven, Nilson, Judd, Patterson, & Shibata, 1988; Navathe & Elmasri, 1989). As the number of multimedia objects available to teachers and students increases, the need felt by these individuals to store, organize, and retrieve multimedia objects will increase.

Teachers and administrators in schools using multimedia have voiced their desire to IBM to acquire a database system to manage multimedia. Early prototypes developed for this project and demonstrated by IBM Educational Systems Division in Atlanta and at IBM Executive Conferences during 1991 and 1992 were met with great excitement and enthusiasm by educators and administrators. Several schools to whom the prototype software was demonstrated have volunteered as test sites. These sites include Orangeburg School District in Orangeburg, SC, Fairfax Public Schools in Fairfax, VA, and Bergen Vo-Tech, in Bergen, NJ. Response from these schools has been positive and requests have been made by additional schools to install the software. Educators at installation sites, and educators at conferences, have voiced the sentiment, "This is what we've been looking for!"

Source of the Need for a Multimedia Database System

Why are educators and administrators voicing a need for a multimedia database system? The answer lies in the difficulty of managing hundreds, thousands, tens of thousands, and in some cases hundreds of thousands of files containing multimedia objects that proliferate when multimedia is used in an educational environment.

One solution to the management of multimedia objects is to place them on disks, which are appropriately labelled, indexed, and stored on shelves. For many reasons this is neither practical nor advantageous. In this circumstance, personnel are required for maintenance and distribution of the diskettes, students' access is severely limited, and casual browsing becomes almost impossible. Add to this the fact that repeated use and handling tends to wear disks out requiring that a set of backups be maintained. And like books in the library, one copy cannot be shared by multiple users at the same time. The problems encountered are similar to those encountered by early non-networked computer labs that had to provide users with copies of application software on diskette or put a copy of each piece of application software on hard disk drives in each workstation. And, in many cases a single multimedia object is larger than a single diskette, prohibiting this method of management altogether.

If multimedia objects are stored on a fileserver on a network, but not in a multimedia database, the problem of multiple users accessing a single object at the same time is circumvented. And, to some degree, the severity of the maintenance problem is lessened. A major concern becomes one of access. How do educators and students find and access the objects? Unless the user is technically proficient, the computer inhibits rather than enables. And, with large numbers of objects even the person comfortable with computer technology will become frustrated. A multimedia database system can make the storage, organization, and retrieval of multimedia objects user friendly by hiding many of the technical details and turning the computer into a tool crafted for the task.

Sources of Multimedia Objects

Where do these thousands of multimedia objects found in educational environments come from? Some multimedia objects are purchased commercially; others result from students' and teachers' creative endeavors. One commercially available collection of multimedia objects is Mammals: A Multimedia Encyclopedia (National Geographic Society, 1990). The encyclopedia contains over 1000 multimedia objects that can be used by teachers and students in applications they create.

With appropriate hardware and software, students and teachers can digitize pictures, movies, and sound. Then, using appropriate software these objects can be recombined to create additional objects. The world is full of objects and sounds awaiting digitization. Drawing programs and animation software can be used to create original works and music software can be used to develop musical scores.

A Tool for a Convivial Society

The development of a multimedia database system creates a capability that provides options and opportunities (not requirements) for educators and students. Engelbart's concept of an augmentation system (documented by Hooper) is useful in describing how a multimedia database system creates a capability.

> The components of an augmentation system are the bundle of all things that can be added to what a human is genetically endowed with, the purpose of which is to augment these basic human capabilities in order to maximize the capabilities that a human or human organization can apply to the problems and goals of human society. (Engelbart & Hooper, 1988)

> . . . an augmentation system framework must consider two types of system contributions. It must involve both human system contributions -- the social and cultural frameworks that have evolved at any time in history to support human activities, as well as the basic human capabilities and their possible extensions through training -- and the tool system contributions --

> the capabilities that are provided to enable human activities. (Engelbart & Hooper, 1988)

A hammer is a tool system that contributes to the capabilities of a human to drive a nail into a piece of wood. A blackboard is a tool system, a pencil and paper are a tool system, and a computer is a tool system. Each can be used in conjunction with existing human capabilities to provide the human with a capability that is not available without the tool system component. A blackboard, paper and pencil, and a computer can be used to communicate. A human without these tool systems has the ability to communicate, but alternate means of communication are provided that may be more desirable in certain circumstances . . . providing a level of freedom which is not possible without the alternatives.

A multimedia database system is a tool system that facilitates the examination, study, and communication of information about the universe by managing multimedia objects which encapsulate representations of the universe. The database system provides the ability for a human to experience more of the universe than would be possible through direct experience and the multimedia nature of the objects in the database provides a greater clarity of representation than is possible through the written word alone.

Illich describes a special class of tool systems called convivial tools. Illich (1973) states, "Convivial tools are those which give each person who uses them the greatest opportunity to enrich the environment with the fruits of his or her vision." Gano (1988) relates convivial tools to multimedia and multimedia database systems by stating:

> With tools for appropriating information in various rich forms (images, sound, computer data) from various sources (on-demand data bases, correspondence from friends, trusted free-lance information hunters, broadcasts), an individual can be an active participant in information gathering . . . With tools for recording and redistributing multimedia information, an individual can achieve a voice in the electronic marketplace of ideas, and help sustain the flow of information that keeps a convivial society alive and spiritually prosperous.

Teachers and students are members of a convivial society. A multimedia database system provides the source and destination of the materials generated by members of that society.

Software Research: Goals and Activities

What does it mean to do software research?

> In a [software] research project, *the goal of a project evolves in the work.* That is, the programmer starts with a broad idea, but cannot specify the detailed behavior of the program until it is written. He or she starts with a partial understanding, attempts to program some better understood corner of the project, then interacts with the resulting program to see in what direction to proceed. (Harvey, 1991)

If the detailed behavior of a piece of software is known, then there is no reason to do research. The detailed behavior would form a design specification that could be used to engage directly in programming the software in a top down fashion.

Since the goal of a software research project evolves, initial goals and activities define a starting point. Only the most general goals will survive the project. To that end, the initiating goals and activities for this project are supplied in the following section.

The following two goals provide a framework for the research leading to a multimedia database management system for use by educators and students in grades K through 12:

Goal 1: Create a detailed design specification that describes the operation of a system to be used by educators and students in grades K through 12 for storing, organizing, and retrieving multimedia objects on a single local area network.

Goal 2: Provide a detailed description of the methods used to bring about the operation described in the design specification.

The research project evolves as a result of new insights and information. Based on current knowledge and information, the following activities provide a starting point for the project:

Activity 1: Investigate potential uses of a multimedia database system by educators and students in grades K through 12.

The types of databases that users create and use impact the development of a database management system (DBMS). Navathe and Elmasri (1989) state, "A database . . . has an intended group of users and some preconceived applications in which these users are interested . . ." Even though a DBMS is meant to be general purpose, its generality is limited. A DBMS must be built to support an intended set of uses.

Activity 2: Determine hardware constraints and requirements for a multimedia database system to be used in a K-12 education environment.

Certain types of hardware are available for use in K-12 education environments. The available hardware must be used as a constraint in the development of the multimedia DBMS. It makes no sense to develop a DBMS for use in a K-12 environment that utilizes Unix workstations if Unix workstations are not available. The objects stored in a multimedia database and the capabilities and needs of the user also impose constraints on the hardware that can be used to manipulate those objects.

Activity 3: Investigate current database technology that can be modified or can be incorporated within a multimedia database system.

Three alternatives exist for the design of a multimedia database system: update the traditional text oriented database model to accept multimedia objects, use pointers recorded in a traditional database to access multimedia objects stored in files, or start "from scratch" and build a new database system for multimedia objects (Yoon, Suzuki, Ishikawa, & Makinouchi, 1987). These alternatives must be explored to arrive at a system that meets the goals of this research.

Activity 4: Investigate the storage, retrieval, maintenance, and execution requirements for multimedia objects.

Objects stored in a multimedia database have storage, retrieval, maintenance, and execution requirements. These requirements have to be known in order to manage and manipulate them.

Activity 5: Design a data model and an architecture for a database management system to support the operation of a multimedia database system.

"A **data model** is a set of concepts that can be used to describe the structure of a database . . ." A model includes, "data types, relationships, and constraints that should hold on the data . . . Most data models also include a set of **operations** for specifying retrievals and updates on the database." (Navathe and Elmasri, 1989)

The data model directly effects the architecture of a DBMS. A data model must be developed to handle multimedia objects.

Activity 6: Investigate the integration of hardware and software systems for incorporating sources of analog video and audio into a multimedia database system.

Traditionally the term *multimedia* has included both digital and analog objects. Videodiscs, for example, contain multimedia objects which, when placed in a videodisc player, provide analog information. Analog information cannot be

stored in a multimedia database nor transferred over a digital network. Analog information must be digitized before storage and transfer can take place.

Videodisc objects can be "stored" in a multimedia database if hardware and software systems are developed for on demand digitization. When a user requests a videodisc object, a workstation attached to a videodisc player on the network can digitize the appropriate material and move it to the database. This digitized material can then be transferred to the user's workstation over the digital network from the database. The database serves as cache for large amounts of information originating from an analog source.

Activity 7: Investigate strategies for indexing and searching for objects in a multimedia database.

Locating objects in a multimedia database requires strategies different from those used to locate information in a traditional database. In a traditional database "Smith" can be located by searching the name field for the string "Smith." The informational content of a multimedia object cannot be used to perform a search. If a person searches for pictures of birds, the computer has no way to know from the data contained in a picture object that it represents a bird.

Textual information which can be searched must be associated with multimedia objects. The information associated with objects determine the types of searches that can be performed.

Activity 8: Develop an application programming interface (API) to allow a variety of user interfaces to be created for a multimedia database (e.g., library catalog system interface, interface for elementary students, interface for special needs students).

To implement a variety of user interfaces in a database system and to provide specialized applications with access to a database, a programming interface must

be provided. This programming interface must supply a generalized set of functions that can be called to perform database operations. As an example, a programming interface would allow library catalog system software to access a multimedia database.

Activity 9: Develop a prototype database application that allows access to objects in the multimedia database.

A prototype application is required to test the operation of the multimedia database management system. This prototype application represents one of many applications that could be developed to access a multimedia database. To serve as an adequate test application, all the functions available to perform database operations must be incorporated.

Activity 10: Develop strategies and software for maintenance of a multimedia database.

A database that has been used over time requires maintenance. A multimedia database creates a set of maintenance problems uniquely different from those associated with traditional databases. The strategies and software created for maintenance must be developed to ensure continued successful use of a multimedia database.

CHAPTER II

BACKGROUND

Chapter II provides background information that is needed to understand the multimedia database system developed in this research project. The first section defines terms and concepts relating to database technology; the second section provides background information on computer networks which are an important element for sharing databases; and, the third section presents concepts belonging to the object orientation (OO) paradigm which were used in the development of the multimedia database system. *Multimedia object* is defined in the final section.

Databases, Database Management Systems, and Database Systems

Many terms are used in discussing database technology, and it is important to understand and differentiate between these terms. The terminology used here is derived from language used by Elmasri and Navathe (1989).

The terms *database*, *database management system*, and *database system* are distinct and should not be confused. A **database** is an organized collection of data. A **database management system** (DBMS) is the software that allows a database to be created, manipulated, and maintained. A **database system** is the database and the database management system combined (Figure 1).

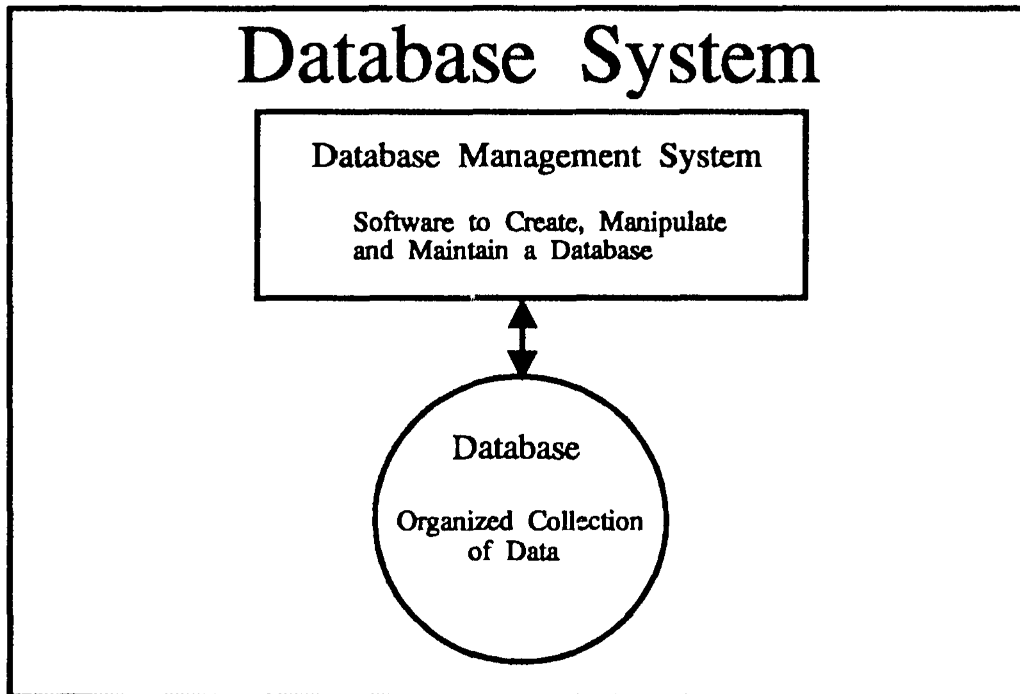


Figure 1. A database system is the database management system and the database combined.

A **multimedia database** is an organized collection of multimedia objects. A **multimedia database management system** (MMDBMS) is the software that allows a multimedia database to be created, manipulated and maintained. A **multimedia database system** is the multimedia database and the multimedia database management system combined. To simplify the language in this document *database*, *database management system*, and *database system* may be used in place of *multimedia database*, *multimedia database management system*, and *multimedia database system* respectively.

Creating a database involves allocating disk storage for the database, building structures for organizing, storing, and retrieving objects in the database, and adding objects to the database. Manipulating a database involves operations to locate, retrieve, and experience objects. ***Experience*** is used to describe the act of making the object known to a user's senses -- to experience a picture is to see it and to experience a sound is

to hear it. Maintaining a database involves operations to remove objects, make changes to objects, and to ensure database integrity.

A **single user database** is a database that is accessed by only one user at a time. A **shared database** is a database that is accessed by many users at one time. A DBMS for a shared database must include **concurrency controls** to ensure that operations applied by several users do not conflict. A user requesting the deletion of an object at the same time another user is retrieving that object is an example of a conflict.

Integrity describes the condition of a database. A loss of integrity of a database occurs when part of an object is missing, when pieces of information about an object have been lost, when an object stored in the database can no longer be accessed, or when an object is corrupted. A DBMS must include controls to maintain integrity. A loss of integrity is sure to result if a DBMS does not use concurrency controls when accessing a shared database. A majority of integrity controls make sure that an action executes successfully or has no effect at all. If an operation to add or delete an object does not execute successfully, the state of the database should be the same as before an attempt of the operation.

Most DBMS software is structured using a layered design which usually contains three layers. Elmasri and Navathe (1989) call this the three-schema architecture and label the layers the internal schema, the conceptual schema, and the external schema. Bordogna (et al., 1990) describes the structure of a DBMS as being composed of three layers called the access method, the communication interface, and the user interface (Figure 2). This is equivalent to the structure described by Navathe and Elmasri, but is more descriptive. Bordogna's terminology will be used in this document.

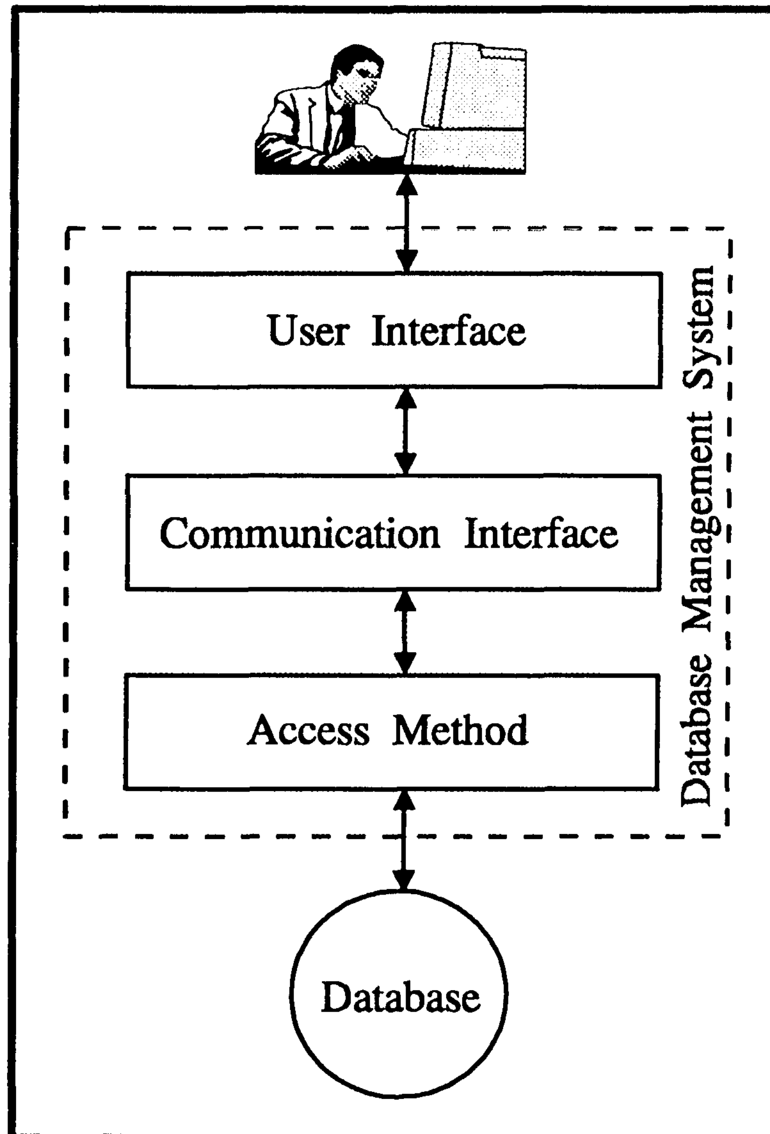


Figure 2. The database management system as defined by Bordogna (et al., 1990) is structured using three layers called the access method, the communication interface, and the user interface.

The **access method** is the method used by the computer to access the physical data. The **user interface** accepts user requests and transmits them to the access method via the **communication interface**. The information returned by the access method is passed back to the user interface via the communication interface. The communication interface shields the programmer from having to know how the data are physically stored, and the

user interface shields the user from having to know about the operations used to process a request.

The two inner-most layers are relatively fixed entities. They represent the part of the DBMS that remains consistent from application to application. There can be many applications using different user interfaces which access the database via the communication interface. This allows a single database to be used by a variety of populations with differing needs and capabilities. User interfaces can be designed to fit particular populations such as first grade students, junior high students, handicapped students, or teachers. Interfaces could also be designed for particular styles of pedagogy. In addition, existing user interfaces could be adapted to build database applications.

Library catalog system software represents one opportunity to use an existing user interface, one that may already be familiar to students, to access objects in a database. Searches from computers running library catalog system software, adapted for multimedia database access, could return references to materials on the shelves in a library along with multimedia objects that could be viewed or downloaded on the computer.

Networks

Networks provide the ability to share programs and data by connecting computers via a communication link that allows one computer to access the resources of another. In a network environment there is no need to replicate files on each computer.

Essentially two types of network architectures exist: client/server and peer to peer. A **client/server** network is comprised of client workstations connected to a server via communication lines. The **server**, shared by all the workstations on the network, provides shared storage as well as the ability to run processes used by the workstations. In a client/server network the resources of one client workstation are not generally

available to another client workstation. The topology of a client/server network is hierarchical (Figure 3).

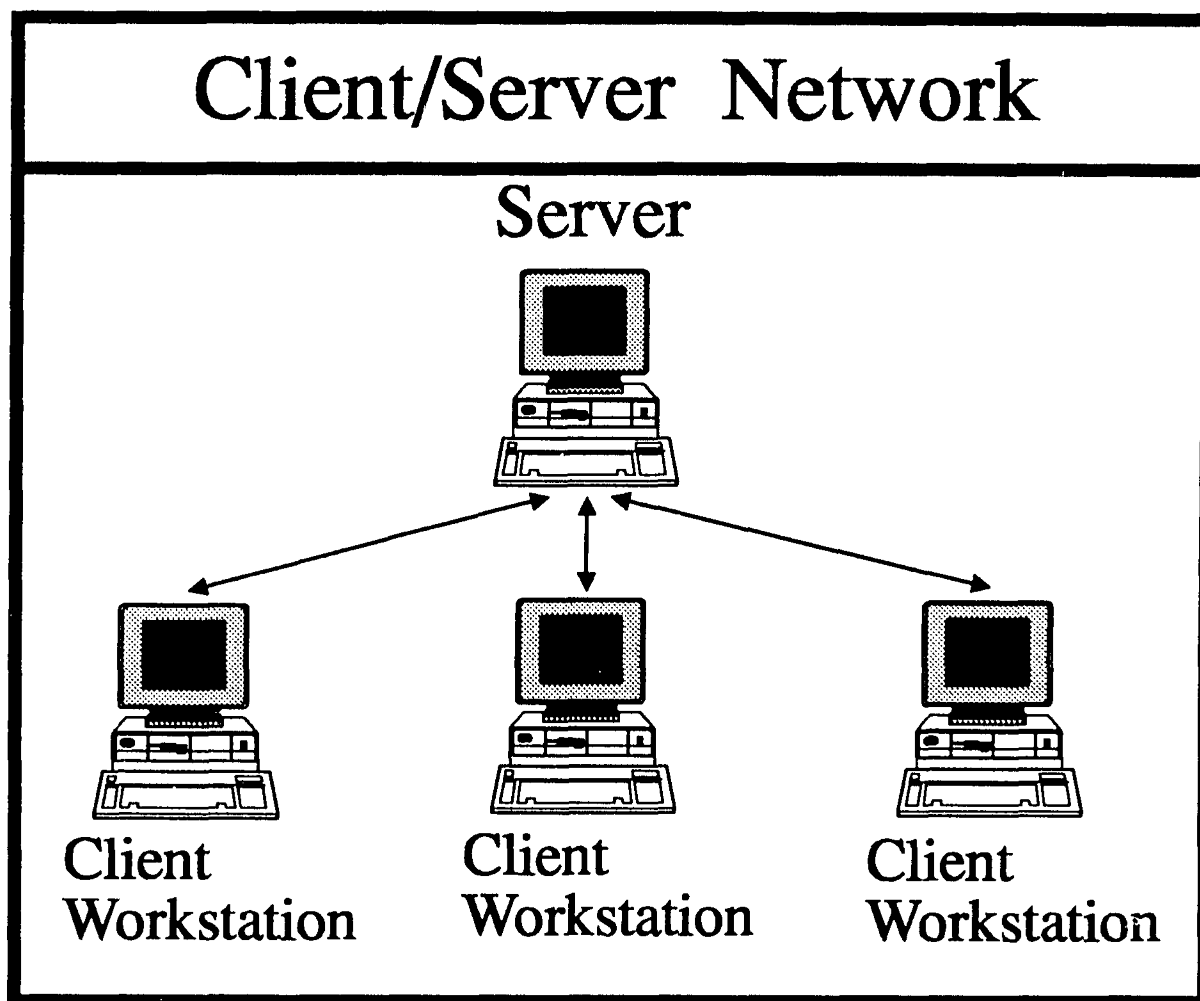


Figure 3. The topology of a client/server network is hierarchical.

In a **peer to peer** network workstations are connected to each other to provide a collective resource that is shared among the workstations. Each workstation is capable of using the resources of the other workstations making a workstation both a client and a server. The topology of a peer to peer network is flat (Figure 4) . . . there is no hierarchy.

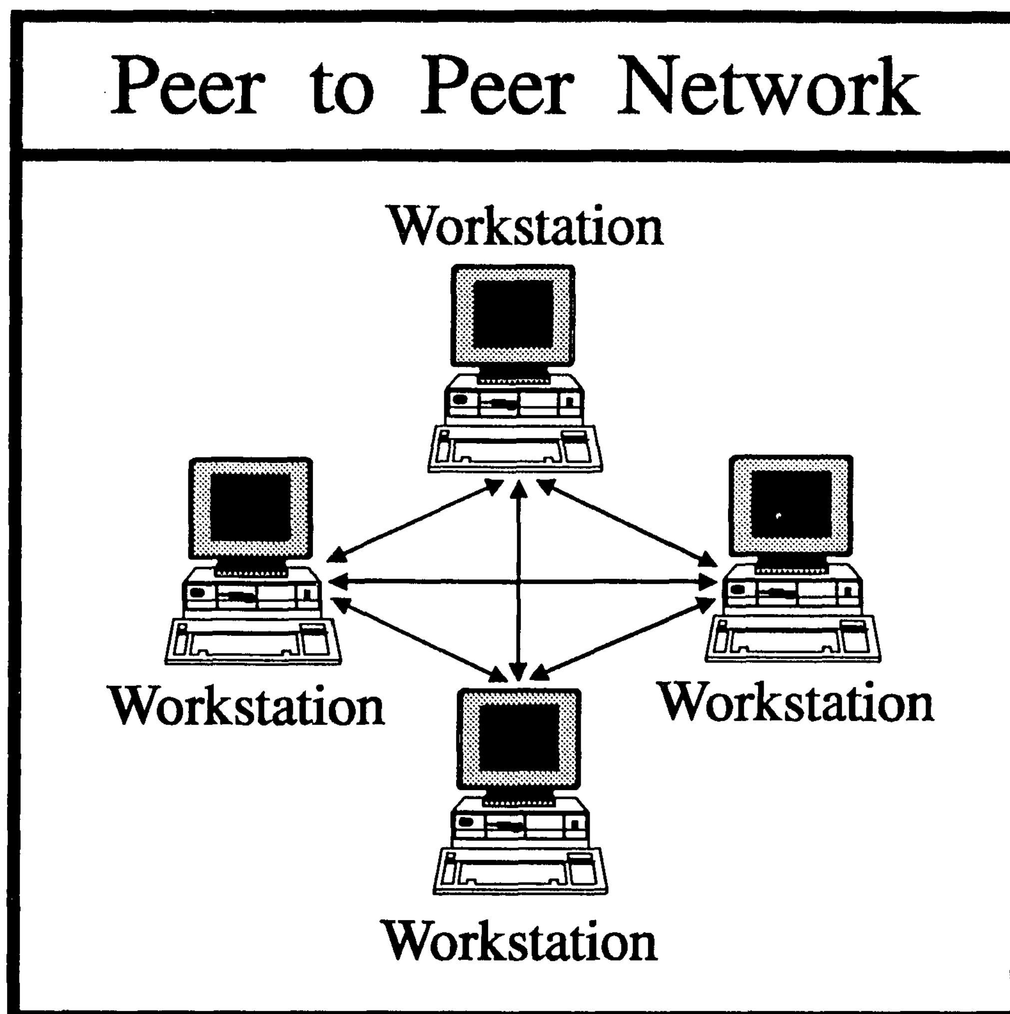


Figure 4. The topology of a peer to peer network is flat.

The client/server network architecture is used in K-12 schools to allow student (client) workstations with limited storage resources to access a wealth of programs and data located on the server. Since the server is used to store and retrieve files (programs and data), it is given the name ***fileserver***. The multimedia database management system developed in this research project works on a network using a client/server architecture. Therefore, future use of the word *network* will imply a client/server network.

A **local area network** (LAN) is comprised of a fileserver, client workstations, network software, and a cabling system that provides a communication link between the fileserver and the client workstations. A client workstation is connected to the cabling

system using a **network interface adapter**. When the appropriate software is loaded on the client workstation, the storage resources of the fileserver appear as new disk drives.

Networks are important to the development of a multimedia database management system for an educational environment because they provide the ability for many students and teachers to share a database at the same time using different computers at different locations. A network allows a single database to be built and accessed by a community of users.

Object Orientation

The object oriented programming (OOP) paradigm contains a set of concepts that were useful in the development of the multimedia database management system. This section describes and illustrates the OOP concepts that were useful.

A young child points to an object in the sky and says excitedly, "Airplane!" It is more than likely that the child has never seen this specific object before, so how did the child know to use the word *airplane*? The child has probably been shown other objects that look something like and acted like (shared similarities with) the object now in the sky that were called *airplanes*. Note that the child did not point to the sky and say, "Car!" There were not enough similarities between the object in the sky and the child's conception of the class of objects called *cars*. Also note that the child did not say, "Bird!" Even though birds and airplanes share some similarities, there were enough differences to distinguish that the object in the sky fit the class of objects called *airplanes* instead the class of objects called *birds*.

The child's parent points to a chirping flapping creature standing on a tree limb and asks, "What is that?" The child responds, "It's a bird!" "Good," says the parent. "And how are birds and airplane's the same?" The child responds after a moment of thought, "They both have wings and they both fly." "You sure are smart!," exclaims the parent.

"Can you tell me how birds and airplanes are different?" With a cocked head the child says, "The bird is an animal and the airplane is a machine. Can we play Frisbee now?"

Some key object oriented (OO) concepts are illustrated above. **Objects** have a state and behaviors. The **state** is a set of attributes that describe the object and the **behaviors** are a set of operations that operate on the state of the object. A bird is a type of object and therefore has a state and behaviors. The state of the bird object may include: eagle, has wings, twenty inches long, male, perching on branch. The behaviors of the bird object may include: flies, walks, eats, pecks, catches fish (Figure 5).

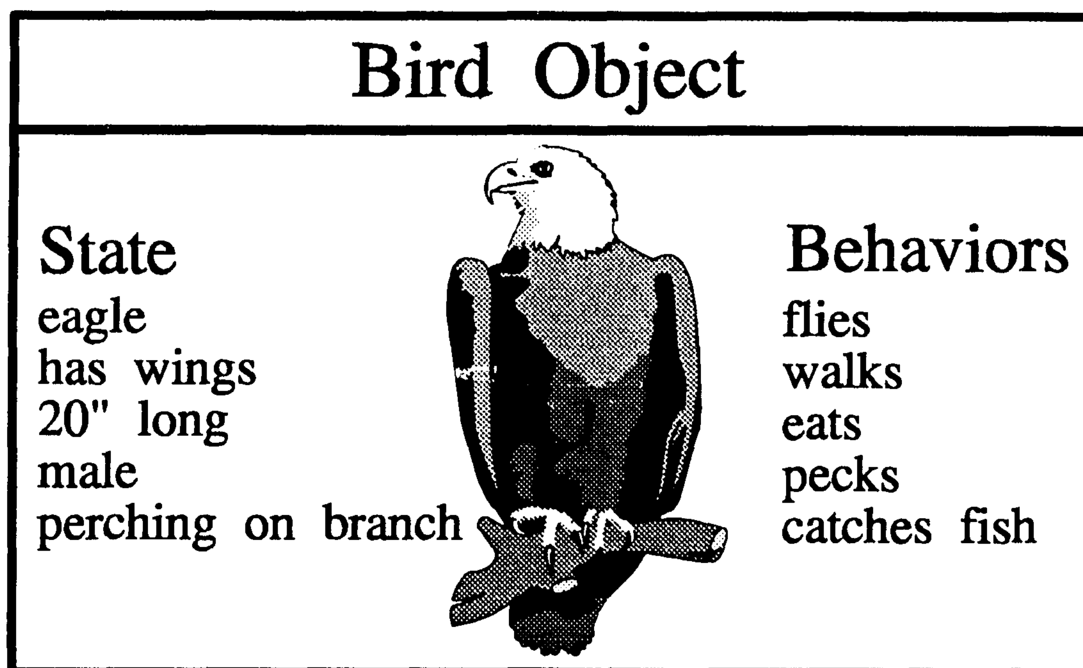


Figure 5. The bird object has a state and behaviors.

Objects, based on a set of shared similarities, can be grouped together in **classes**. Airplanes, as a class, are those objects that can fly, that are machines, and that use propellers or jets as a mechanism for propulsion. Birds, as a class, are animals that have wings, that are warm blooded, and that fly (forgive me for not accounting for the ostrich, the emu, and the penguin). Classes of objects that share commonalties are subclasses of

a common superclass. Both airplanes and birds can fly so they are subclasses of the superclass called *things which can fly.* A **superclass** is a generalization of a group of classes that share similarities. A superclass contains fewer defining attributes and behaviors than a subclass to it. A **subclass** is a specialization of a superclass. A subclass contains a larger set of defining attributes and behaviors.

Classes are organized in a hierarchy called a **class hierarchy** (Figure 6). The top of the class hierarchy contains the most generalized classes and the bottom of the hierarchy contains the most specialized classes. A class inherits all the attributes and behaviors from its ancestors in the class hierarchy. This property is called ***inheritance***. A subclass can inherit attributes and behaviors from more than one superclass. This is called ***multiple inheritance***. The class of objects called *birds* inherits attributes and behaviors from the class called *things which can fly* and the class called *animals.*

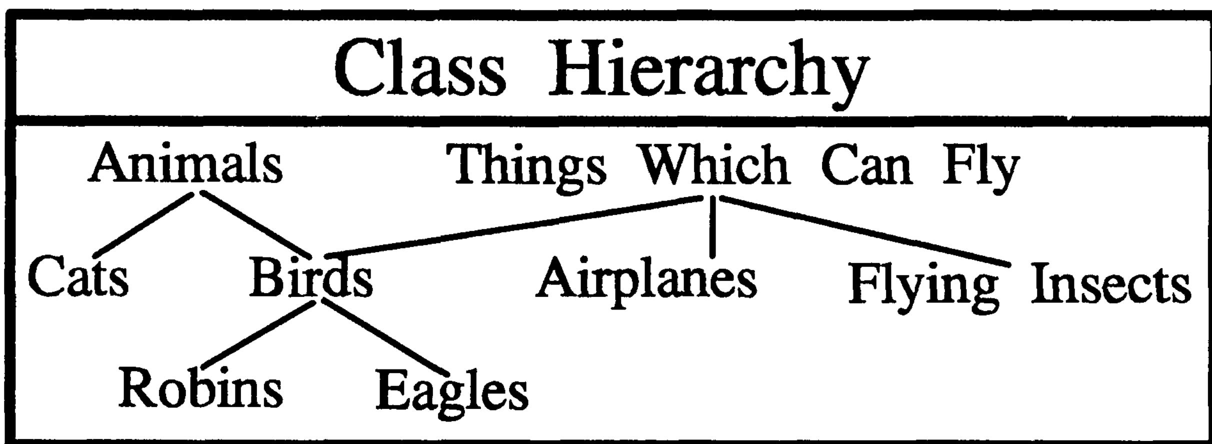


Figure 6. The figure shows an example class hierarchy.

An **instance** of a class is an object. If you look out the window and see a robin hopping across the lawn, you are looking at an instance of the bird class. The instance you are observing has, at any given moment, a state. Behaviors assigned to the bird class can be applied to the state of the object to change its state.

A state is a composite of variables and constants. Some variables associated with the birds class are location, hunger level, age, wing span, and alive or dead. Some constants associated with the birds class are gender and breed. Constants are set at the time an instance of the bird class is created. For a real bird the constants are set when the egg is laid or the bird is hatched.

Behaviors are a collection of **methods** (operations, instructions, software, programs) that operate on the state of an object. The methods change the values stored in the variables assigned to an object or use the state to perform some task. Methods cannot change the value of constants. Therefore, no method could be devised to change a bird from one type to another. A robin can't be changed into an eagle.

Polymorphism is the ability to apply a single behavior to different objects. By definition, *polymorphism* means the quality of having more than one form. A behavior is polymorphic if it can be applied to objects from different classes. Polymorphism simplifies performing operations on a collection of objects from different classes. One type of behavior can be applied to objects whose attributes are very different. For example, both birds and airplanes fly. *Flies* is a behavior that has been inherited from the superclass called *things which can fly* by both the birds class and the airplanes class. Even though the behavior *flies* requires a different set of operations to be performed on the state of a bird object than on an airplane object -- an airplane's propeller must go around and a bird's wings must flap -- the single behavior *flies* can be applied to both birds and airplanes.

A multimedia object also has a state and behaviors, where the behaviors are dependent on the methods associated with the object. For example, an image's state may include the name of the image, the image data, the format of the image data, the size of the image, and the hardware requirement for displaying the image. Methods to display the image, re-size the image, store the image, and retrieve the image may be included in

the image's behaviors. The behaviors of a multimedia object are polymorphic if they can be applied to more than one type of multimedia object. For example, if store and retrieve can be applied to sounds as well as images, then store and retrieve are polymorphic.

The presentation of OO and OOP has been very limited in scope. It is suggested that the interested reader refer to the wealth of OO literature for more information (Barkakati, 1991; Cattell, 1991; Gupta, et al., 1991; Holzner, et al., 1991; Hughes, 1991; Kim, 1991; Kim, 1990; Kim, et al., 1989; Merrill & Tennyson, 1977; Tello, 1989.)

Multimedia Objects

Multimedia objects of all types -- images, animations, digital video, sounds, voice, music, text, and their combinations -- are made up of a sequence of numbers; their digital form is a sequence of bytes. The numbers making up a multimedia object can exist in the memory of the computer or in a file or series of files on a disk. Many multimedia objects are often too big to put in memory all at once and therefore can only exist in their entirety in a disk file or in a group of disk files. For an object to persist (exist longer than the time the computer is on) it must be stored on disk.

Simple objects can be combined to form complex objects using a multimedia authoring program like IBM's LinkWay or Apple's Hypercard or by writing a program in a traditional programming language such as C, Pascal, or Basic. For example, a student may create a multimedia report about birds by combining images of different species of birds, movies of birds in the wild and in zoos, sounds made by birds, and text written about birds. Complex objects usually exist as several files in a directory or in a group of directories. The program that glues simple objects together is part of the complex object.

Some multimedia objects contain their methods; other multimedia objects are stored separate from their methods. An object that contains both data and the code needed to manipulate the object is **bound** with its methods. An object that is stored separately

from its methods contains only data and is called a ***data object***. Since a data object does not include methods, methods stored separately from the object must be used at the time the object is manipulated. **Late binding** is the process of using a method to manipulate a data object. **Early binding** is the process of placing an object's data and methods together. It is possible for an object to be bound with some, but not all, of it methods. In this case some methods are bound early and some are bound late.

A program that displays an image is a method for **experiencing** an image object; a program that plays a sound is a method for experiencing a sound object; and, a program that displays a digital video clip is a method for experiencing a digital video object. Generally, picture objects, sound objects, and digital video objects, and most other multimedia objects, are data objects. When a program is used to experience these objects, late binding is occurring. Most commercially available programs require the objects to be located in files to experience them, which leads to the fact that most multimedia objects are **file based**.

Multimedia applications are complex objects and, in general, do not require external methods to experience them. The methods for experiencing the various simple objects that make up the complex object are bound with the application. The methods bound with the multimedia application usually expect to find the component multimedia objects in files. A multimedia application object is a collection of files, which also makes it file based.

For the purpose of this document a multimedia object is considered to be a file or a collection of files. A multimedia database management system is, therefore, a type of file system. This view is based on the existence of a large number of multimedia applications and methods which require multimedia objects to exist as files. Any other view would require that standard practice in writing and developing multimedia applications and methods be changed. Also, a great deal of effort has been expended by

many people in developing the existing multimedia applications and methods. A file based approach, in the development of a multimedia database, is required to preserve this work.

CHAPTER III

SURVEY OF EXISTING MULTIMEDIA DATABASE SYSTEMS

This chapter provides an overview of existing multimedia database systems. The presentation focuses mainly on the purpose, capabilities, and structure of the database systems.

Muse

Muse, an experimental multimedia document filing system for use in office environments, is implemented on a Digital Equipment Corporation VAX 11/780, a Masscomp MC-500 workstation, and Sun-2 workstations connected by ethernet. Muse is based on a distributed architecture involving servers and clients connected via a local area network (LAN). A server is a computer that supplies computing resources and information to other computers on the network. A client is a workstation that makes requests for resources or information from a server. (Gibbs, Tsichritzis, Fitas, Konstantas, & Yeorgaroudakis, 1987)

The objects handled by the Muse document system are integers, fixed length character strings, variable length character strings, raster bitmaps, graphical data, and audio data. Content based document retrieval is supported by queries from client workstations. (Gibbs, et al., 1987)

A Muse server functions as a storage facility for multimedia documents. Requests for documents, which are sent over the LAN by a client workstation, are fulfilled by the server. While a server is in operation, it broadcasts notifications of its status over the

LAN. Server notifications of status are monitored by clients. In this way, clients are able to determine which servers are available to be queried, and multiple servers can function on one network. (Gibbs, et al., 1987)

The display on a client workstation is divided into six windows providing several locations for entering queries and commands, displaying messages, and viewing documents. As a user enters a query on a client, a structured query language (SQL) statement is generated for transmission to a server. After the issuance of a query the user has the opportunity to scan documents and/or save documents to the local workspace. The workspace on the client also serves as a cache for materials returned in response to a query. (Gibbs, et al., 1987)

A Muse server supports six categories of requests (Gibbs, et al., 1987):

1. INSERT <document>
2. FIND <query>
3. GET <physical document identifier>
4. DEFINE-TYPE <document type specification>
5. GET-TYPE < document type name>
6. LIST-TYPE <>

A type is a classification of the document based on its structure. All interaction between a client and a server is based on the six categories of requests listed above.

A useful method is employed in Muse to quickly access documents based on their titles using document signatures. Instead of searching for the title of a document by comparing the desired title to a series of titles in the database, a search is performed on signatures derived from those titles. (Gibbs, et al., 1987)

A signature is a mapping of a variable length string of characters to a fixed length byte sequence. A title, which has been entered by a user for the purpose of a search, is mapped to a signature. The resulting signature is compared to signatures stored in the

database. Since the same signature can result from a number of different character strings, a post hoc comparison is made between the actual titles to select correct titles from the returned query. Even with the need for a post hoc comparison of titles, signatures make the searching process faster. According to Gibbs (et al., 1987), "Signatures are typically one tenth the size of the original text, making it considerably faster to scan signatures than to scan text."

Muse uses a three layer architecture. The user interface is implemented on the client workstation, where SQL statements are generated based on input from the user, and the communication interface and access method are located on the server, where the data is physically stored. Messages are passed between the client and the server using network communication.

Bell Communications Research

Bell Communications Research (Bellcore) is defining the requirements for achieving a goal called *Universal Communications* "where everyone has easy and immediate access to widely distributed information sources in many media . . ." (Irven, Nilson, Judd, Patterson, & Shibata, 1988).

The Bellcore system includes four components: vendors, information services management, workstations, and a high speed network. The vendors are computers on the network that supply information; information services management is a directory of information services; workstations allow users to access materials stored on the vendors via some applicable user interface; and, the high speed network provides communication between the vendors, the workstations, and the information services management.

The user interface for multimedia information systems is one focus of Bellcore research. A three stage model is proposed by the Bellcore researchers in describing the progression through an information access session (Irven, et al., 1988):

1. Query stage - the user defines a topic and limits the scope of a search.
2. Browsing stage - the user chooses material of interest returned from a query.
3. Follow-up stage - the user does something with the selected material.

Three browsing methods are described by the Bellcore researchers for pictorial information. The first method uses a hierarchical video menu system. The user makes selections from a menu of pictures. Selection of a picture leads to another menu, or to material to be viewed. The second method uses rapid scanning of a collection of pictures that have been reduced in size. Pictures are displayed in an array on the screen that is filled with new pictures in a cyclical fashion. The set of pictures being scanned is refined by interpreting user input. A user can stop the scanning at any time and zoom in on a particular picture. In the third method, pictures are presented one at a time using the entire screen. Users can save items for later viewing by clicking on a picture when it is displayed. No findings are available to determine where and when a particular method is most useful. (Irven, et al., 1988)

Domino

Domino is an information retrieval system for documents comprised of text and graphics implemented on a wide range of computers, including mainframes and personal computers. The architecture of Domino is divided into three parts: the access method, the communication interface, and the user interface. (Bordogna, et al., 1990)

Domino uses pattern recognition to index pictorial information. An expert is used to define the objects that are of interest in pictures to be indexed, then an automated indexing process is used to locate the objects and store indexing information. Words contained in the documents are also used as textual indexes for the multimedia documents. The indexes, both pictorial and textual, form a document description that is used to locate the documents when searches are performed.

Queries are expressed "using textual terms or pictorial index terms associated with structures which have been recognized in the pictorial part of the document" (Bordogna, et al., 1990). The query language has the ability to be expanded and tailored for specific applications.

The Multimedia Document Manager

The multimedia document manager created by Blumberg and Walters (1989) was developed to operate on IBM personal computers. According to Blumberg and Walters (1989), "...the Multimedia Document Manager provides three sets of services: 1) document preparation and control; 2) library access; and 3) document distribution."

The three services are provided by at least three computers connected via a communication network: a multimedia equipped development workstation provides the ability to perform document preparation, a user workstation equipped to display or print documents is used to access documents, and, a third computer is used as a library for the multimedia documents (Blumberg & Walters, 1989).

Communication between computers takes place using an electronic mail system. Queries, documents, and messages are sent as electronic mail messages to the appropriate stations.

The user interface supports the following functions (Blumberg & Walters, 1989):

1) browsing through document titles
2) searching based on title, author, and keywords
3) modifying document abstracts
4) retrieving a single document
5) distributing information to inform users of new material

A multimedia document, in Blumberg and Walters system, is "defined by a profile containing the attributes of the document and a set of files containing the actual document" (Blumberg & Walters, 1989). The document profile contains the title of the document, the name of the author, and a set of keywords that are used to locate objects during a search.

Other Systems

REMINDS is an image and text database system created by Mehrotra and Grosky (1985). REMINDS is an extension of the architecture used by existing text database systems. According to Mehrotra and Grosky (1985), the system includes an "...**Image Understanding System** [which] handles image storage, processing, feature extraction, decomposition, and matching." Image components and interrelationships are used to create a relational description of objects stored in the database.

Yoon, Suzuki, Ishikawa, and Makinouchi (1987) describe a multimedia database management system (MMDBMS) that uses an object oriented approach in its design. Like previously described systems, this object oriented MMDBMS is comprised of three layers. Yoon (et al., 1987) calls the three layers the media oriented layer, the object oriented layer, and the application oriented layer. These layers compare to the access method, the communication interface, and the user interface, presented earlier. The media oriented layer has access to the physical storage of the objects. The application oriented layer provides the user interface. And, the object oriented layer communicates between the media oriented layer and the application oriented layer.

Masunaga (1987) also uses an object oriented approach in describing a framework for multimedia database systems. Masunaga's approach avoids the user of elements of traditional database management systems in favor of creating a new approach for multimedia database systems.

Christodoulakis (et al., 1984) presents a framework for a multimedia message system. Messages are retrieved based on their content, which includes text, images, and voice.

Conclusions

A consistent theme in reviewing the literature on multimedia database systems, and database systems in general, is the three layer architecture of these systems. At the data level, there is a method for accessing the objects stored in the database. At the user level, there is an interface that allows the user to communicate operations to be performed on the database and to locate objects. And, between the data level and the user level is a layer that provides communication between the two. Different methods exist for implementing this three layer architecture, but each method still works on the basic premise that there are three layers.

None of the database systems which have been located in the literature were designed for use in educational environments. Most were developed for use in office environments to manage multimedia information.

CHAPTER IV

POTENTIAL USES OF A MULTIMEDIA DATABASE SYSTEM IN A K-12 EDUCATION ENVIRONMENT

> Imagine a classroom with a window on all the world's knowledge. Imagine a teacher with the capability to bring to life any image, any sound, any event. Imagine a student with the power to visit any place on earth at any time in history. Imagine a screen that can display in vivid color the inner workings of a cell, the births and deaths of stars, the clashes of armies, and the triumphs of art . . . a new breed of multimedia storytellers, enchanters and guides will build us pathways and superhighways through banks of information and libraries of sights and sounds. (Sculley, 1988)

This chapter describes four ways a multimedia database system (MMDBS) can be used in a K-12 education environment: as an exploration and research tool, as a presentation tool, as a development tool, and as a resource for educational software. The uses for an MMDBMS are based on synthesis and on input from teachers, students, administrators, and developers.

An Exploration and Research Tool

As an exploration and research tool, an MMDBS functions much like an encyclopedia or a library (Figure 7). The contents of the database can be browsed for interesting pieces of information, with one piece of information leading to the next, or the contents can be searched to find specific pieces of information or to answer specific questions.

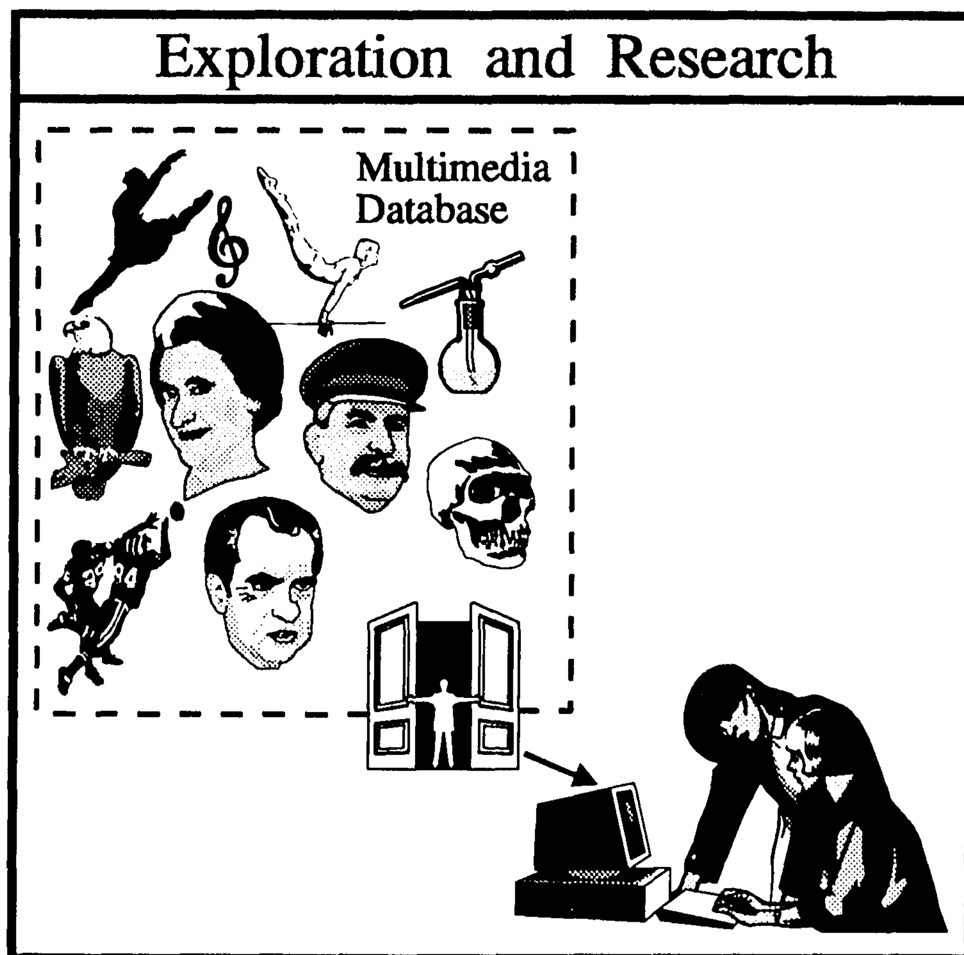


Figure 7. A multimedia database system can be used as an exploration and research tool.

As an exploration and research tool, the MMDBS helps the student and the teacher locate information to answer questions and to satisfy curiosity. It can also be used to pique curiosity. Two scenarios follow that provide examples of how an MMDBS can be used as an exploration and research tool.

This first scenerio relates how the MMDBS could be used as a research tool. Sarah receives an assignment to develop a report she is to give to her history class on John F. Kennedy. She goes to the library and finds a book written about Kennedy, locates a reference in an encyclopedia, and finds some old newspaper articles located on microfiche. Then she goes to one of the school's computer labs and searches the

multimedia database for information about Kennedy. In the database she finds several pictures, many video and sound clips of speeches given by Kennedy, and two multimedia reports written by other students in the school. The speeches refer to other people whom she also looks up in the database. Sarah thinks the video clips of the speeches are great because they let her see and hear the man. The book, the encyclopedia article, and the newspaper articles only showed posed still images of Kennedy, and even though they contained his words, she couldn't hear the way he said them.

This second scenerio relates how the MMDBS can serve as an exploration tool. Sally loves to study about astronauts and space. Almost every day she goes to the computer lab during her free periods and searches the multimedia database for anything relating to her favorite topics. She finds pictures of rockets blasting off and video clips of astronauts floating in their ships. She studies pictures of the moon and contemplates the landscapes of other planets. Sally learns that space travel involves physics, chemistry, mathematics, and human physiology. She also discovers how important politics can be to space travel, when she listens to the speeches of politicians and the words of astronauts. Sometimes Sally strays far from the topic of space when something interesting gains her attention, but mostly she is interested in space because someday she hopes to blast off into space and to walk on the surface of another planet.

A Presentation Tool

An MMDBS can be used as a tool to organize multimedia material used in a presentation, or can be used to access information important to a presentation at a moment's notice (Figure 8). A student may raise a question during the presentation of a lesson, which can be answered by accessing information located in a multimedia database. The multimedia information stored in the database can bring to life concepts that are difficult to relate using only textbooks and a chalkboard.

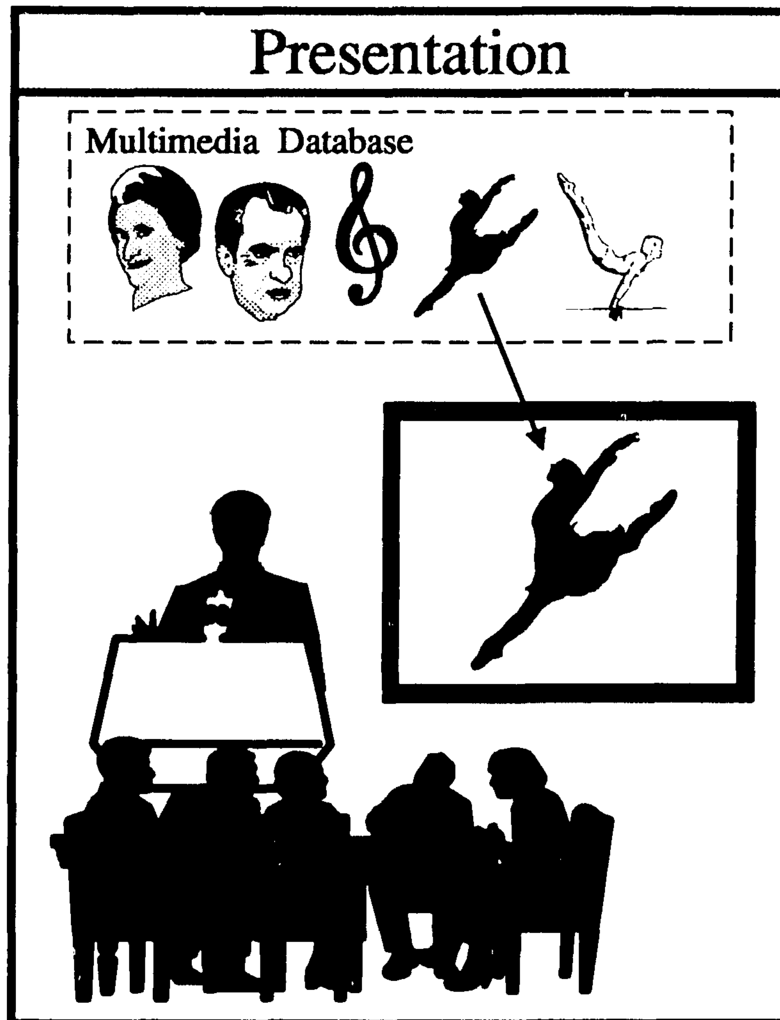


Figure 8. A multimedia database management can be used as a presentation tool.

The following scenerio relates how an MMDBS can be used as a presentation tool. John is teaching about conservation of momentum in his physics class. He begins his presentation with a review of vectors. Then, he writes the law of conservation of momentum on the blackboard and works a couple of problems for the students to see. The students don't seem particularly interested. One student, Tom, raises his hand and says, "I don't see how this stuff can be used." John uses his computer and searches for a reference to the explosion of the space shuttle Challenger. He finds a video clip of the liftoff and subsequent explosion. The video clip shows the shuttle travelling in a straight

line with a vapor trail showing the motion of the spacecraft. Immediately after the explosion, the straight vapor trail divides into two vapor trails forming a Y showing the paths of the two pieces of the shuttle. John freezes the video clip with the Y shaped vapor trail on the screen and proceeds to draw a set of vectors over the image, using a computer tablet, to show how momentum before the explosion and momentum after the explosion are the same, and is the reason for the Y shaped vapor trail. The class appears interested to learn more.

A Development Tool

Developing multimedia applications requires multimedia objects. A multimedia database serves as a source of objects that can be pieced together to create multimedia applications (Figure 9).

The term paper can be redefined to include multimedia applications created by students that are focused on a specific topic. Greenfeld (1991) mentions the video term paper as a possible alternative to the traditional written term paper. A multimedia term paper can go beyond the video term paper to include any type of digital media. Using multimedia, students have a new way to communicate which may be especially useful for students who have difficulties with expressing ideas in writing (Ambron, 1988). Teachers can also create applications to communicate ideas and to teach lessons.

As a development tool, the MMDBS should be tightly integrated with a multimedia authoring system. The Multimedia Archive Explorer (MAX) database application, described in Chapter X, can be used in conjunction with IBM LinkWay to serve as a development tool that provides access to objects stored in a multimedia database from within the LinkWay environment.

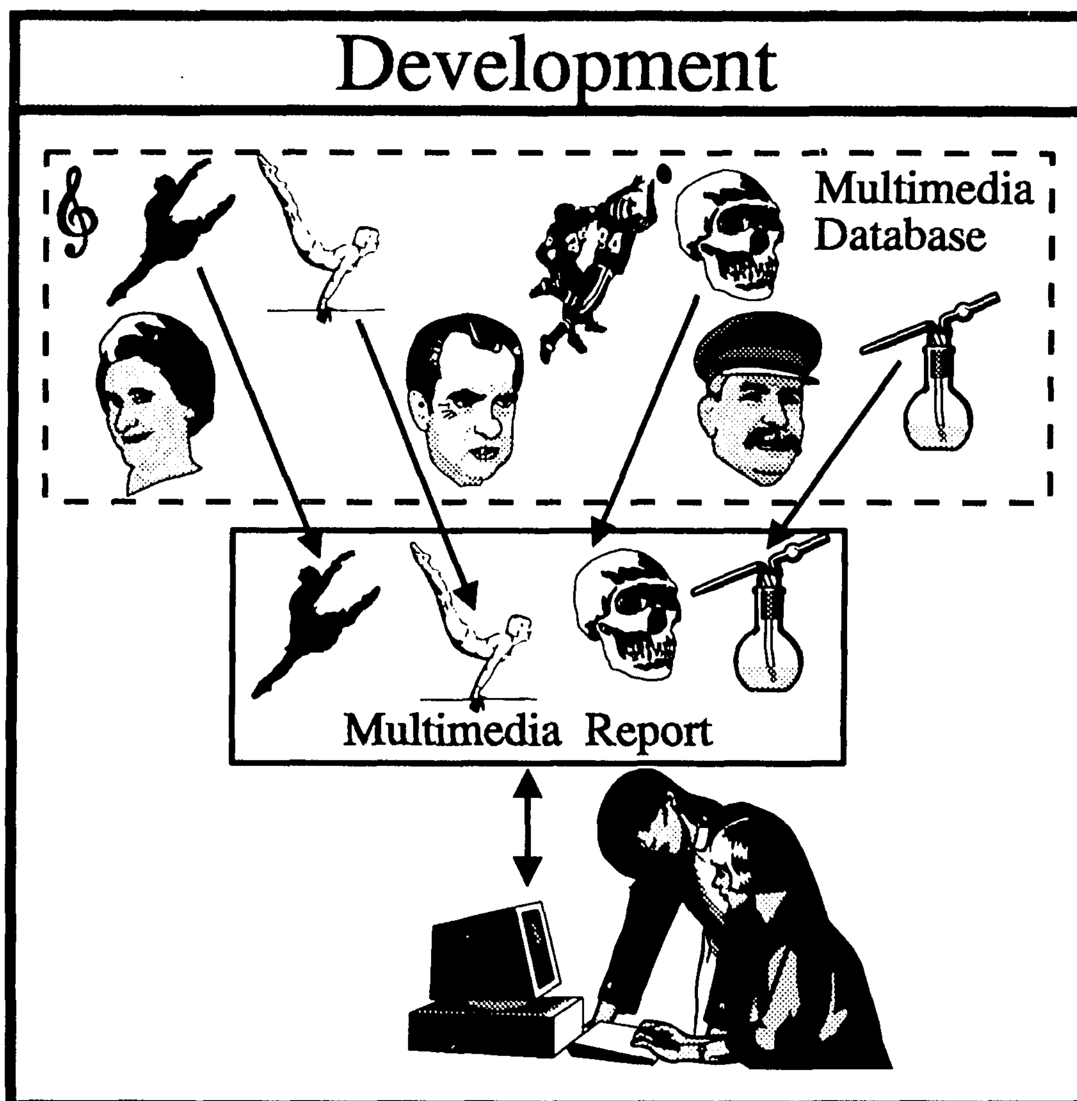


Figure 9. A multimedia database system can be used as a development tool.

A Resource for Educational Software

The MMDBS can be used as a resource that is accessed by instructional applications that require a large number of multimedia objects or share objects with other instructional applications (Figure 10).

For example, a chemistry application may need to access a large number of molecule animations, pictures of experimental apparatus, diagrams, and video clips of chemical reactions. A history application may need to access a large number of images of historic

places and people, video clips of historic events, and documents of historic importance. Furthermore, an art application may need to access hundreds or thousands of images of paintings and sculptures. The objects used by these applications could be stored in a multimedia database, and the database management system code could be embedded in the applications to access the objects. The user would be totally unaware that a multimedia database is being accessed by the application.

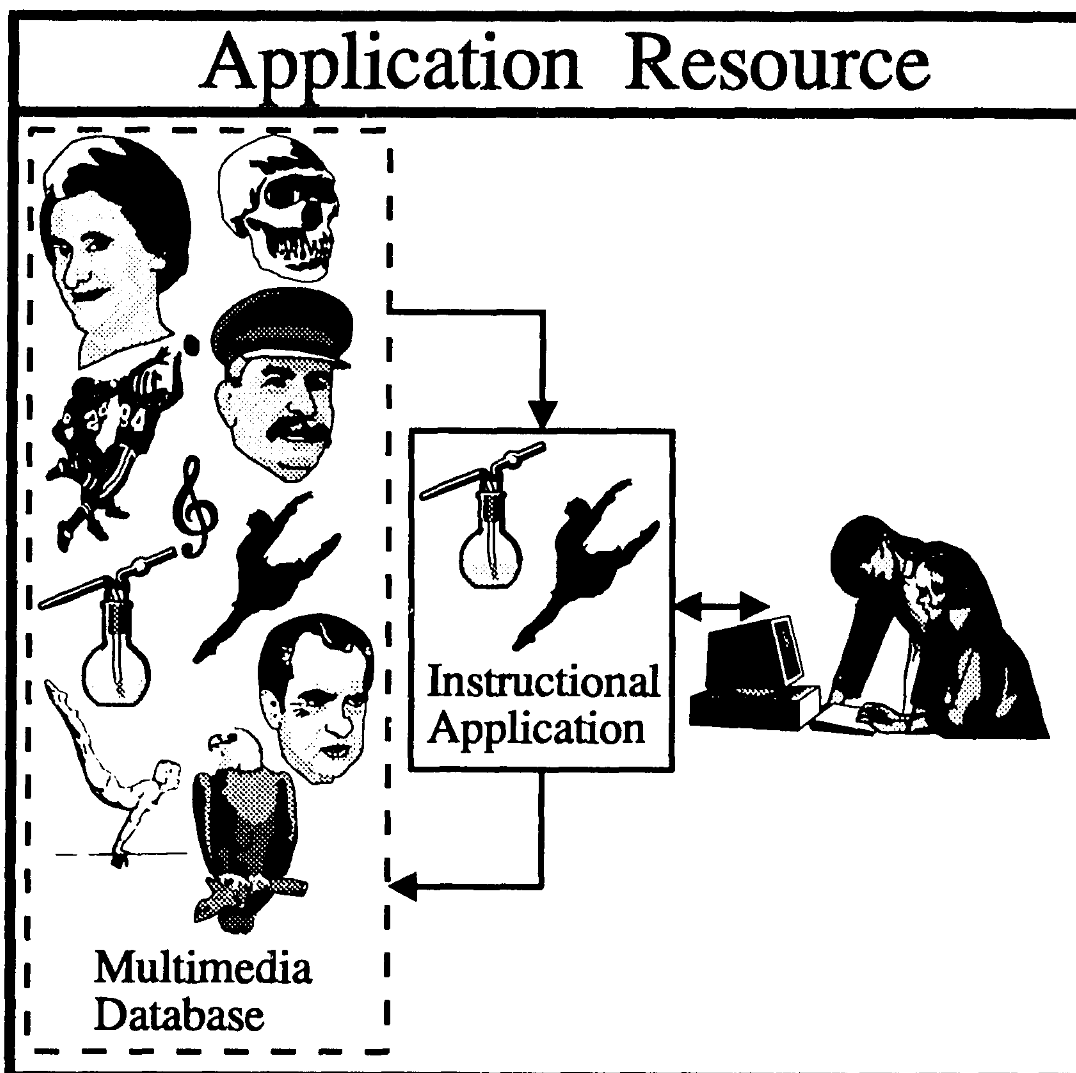


Figure 10. A multimedia database system can be used as a resource for educational software.

Conclusion

The four ways that a multimedia database management system can be used in a K-12 education environment are as an exploration and research tool, as a presentation tool, as a development tool, and as a resource for educational applications. Each of these uses may require a different user interface, but they can all use the code in the communication interface and the access method and they can share a common database.

Chapter V

THE DEVELOPMENT ENVIRONMENT

This chapter describes the types and configuration of the hardware and software that were used during development and represents the environment required to utilize the multimedia database management system (MMDBMS). The environment was selected because it contained hardware and software available in K-12 schools and because it provided sufficient functionality to achieve the goals of the development.

Network Configuration

The network contained a fileserver and DOS based client workstations. The workstations were connected to the fileserver using token ring, ethernet, and baseband connections forming a local area network (Figure 11). The network control program loaded on the fileserver was Novell Netware. Two servers were configured for development and testing; one server used Netware 2.2 and the other used Netware 3.11.

The IBM Classroom LAN Administration System (ICLAS) version 1.40 was installed on the fileservers to organize fileserver resources and to provide a user interface for students, teachers, and administrators to applications and fileserver resources. ICLAS is a shell that is loaded on the fileserver and runs on client workstations that "helps network administrators and teachers control access to the educational software, keep records on lesson performance, and track and record the activities of students" (IBM, 1991). The network was configured and Netware was installed according to instructions provided in the IBM Classroom LAN Administration Installation Instructions guide (IBM, 1991).

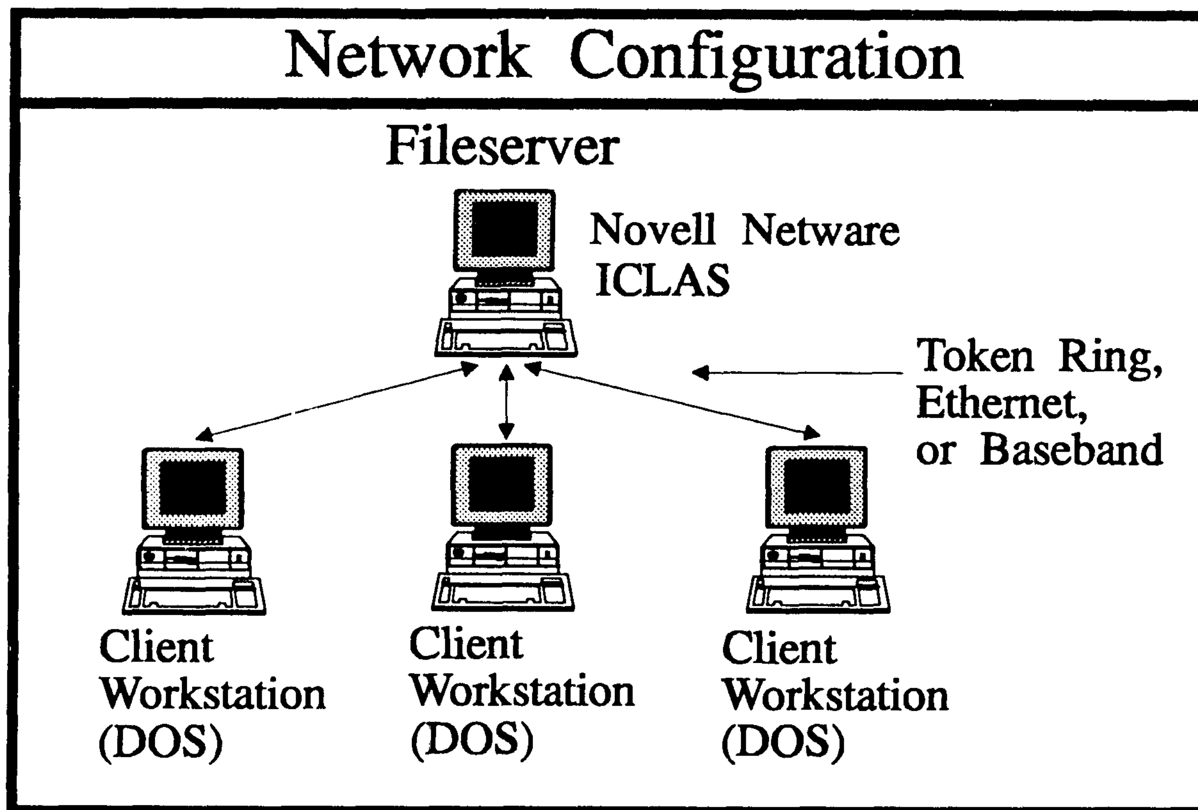


Figure 11. The network used in the development environment contained a Netware fileserver and DOS client workstations.

Client Workstation Configuration

Client workstations were IBM Personal System/2's (PS/2s) using IBM DOS 5.00. The PS/2s had the following minimum configuration: 640 kilobytes (Kb) of RAM, an MCGA graphics display adapter, one 720 Kb diskette drive, a mouse, a Digispeech speech adapter, and a network interface adapter (Figure 12). Several workstations included additional multimedia hardware and other enhancements to the minimum configuration.

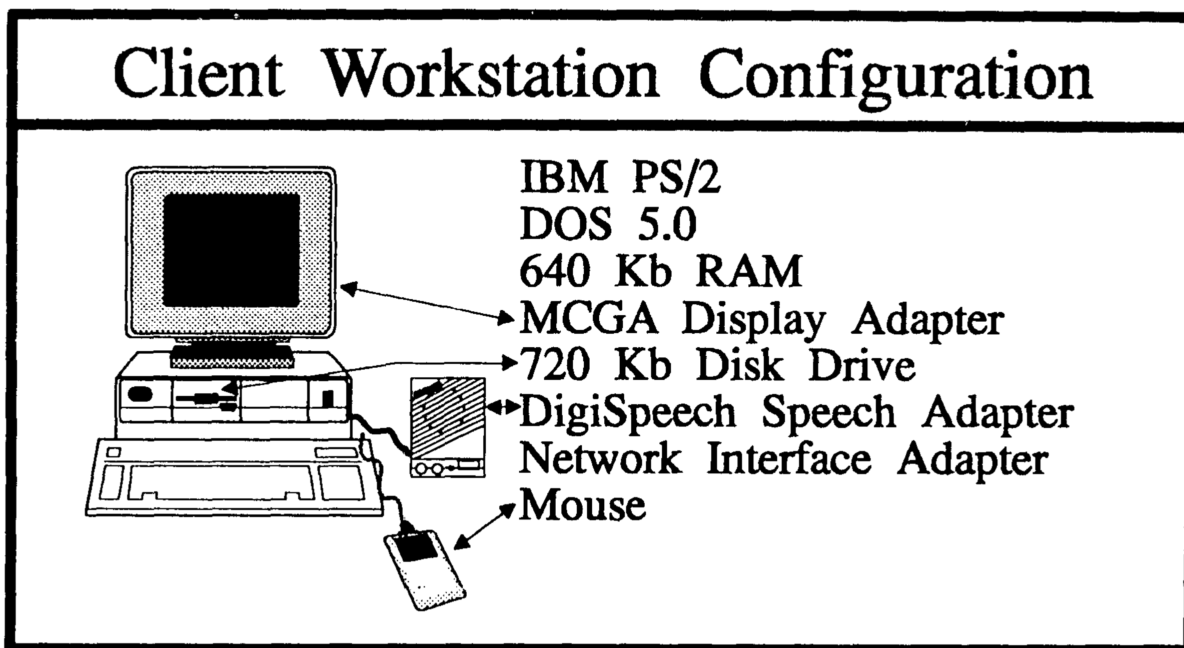


Figure 12. The figure shows the minimum client workstation configuration.

Automatic Folder Creator Workstation Configuration

The workstation used to digitize video and audio from videodiscs is called the *Automatic Folder Creator* (AFC). The AFC was a microchannel based PS/2 configured with 8 megabytes (Mb) of RAM, 30 Mb hard disk drive, an IBM Video Capture Adapter (VCA), an IBM Dual Async Adapter, a network interface adapter, a Digispeech speech adapter, and a Pioneer LC-V330 Multidisc Autochanger or a Pioneer LD-V4200 (or LD-V2200) single disc player (Figure 13). The AFC workstation used IBM DOS version 5.00.

The VCA was used to digitize video still frames and motion sequences and the speech adapter was used to digitize audio. The Pioneer LC-V330 allows up to 72 videodiscs (both sides) to be accessed by the workstation. The single disc player allows one side of one videodisc to be accessed by the workstation. The videodisc player was attached to the workstation's primary serial port and the speech adapter was connected to a serial port

provided on the dual async adapter. The network interface adapter connected the AFC workstation to the fileserver as a client in the client/server topology.

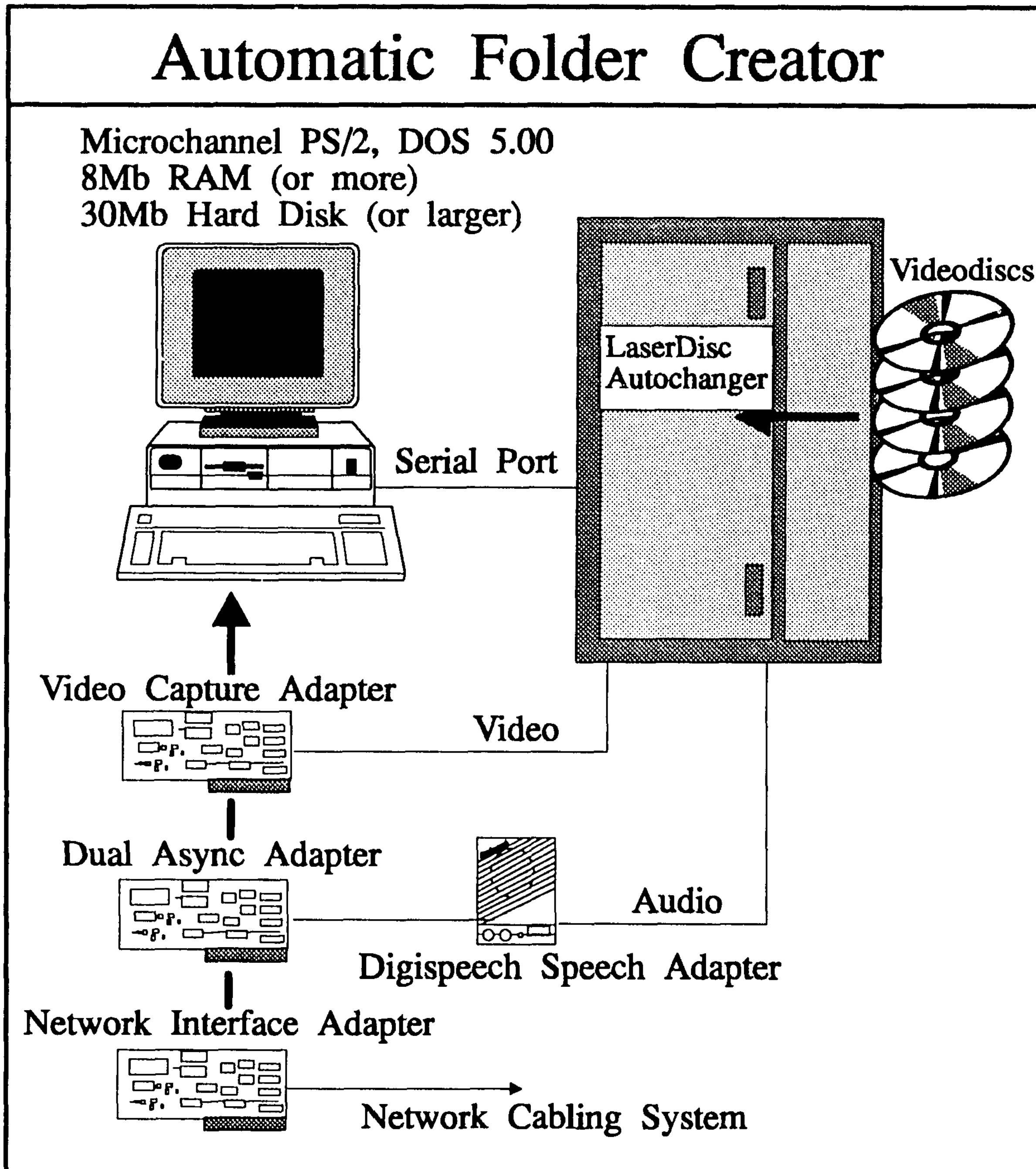


Figure 13. The figure shows the configuration for the Automatic Folder Creator.

Development Tools

A large number of software development tools were used throughout the research project. Many were abandoned when paths taken were determined to be untenable. The list of software tools provided below include only those which were finally used in the development of the multimedia database management system.

List of Development Tools:

- Microsoft C (Version 6.0)
- Hold Everything from South Mountain Software (Version 1.05)
- Novell Netware C Interface for DOS (Version 1.2)
- Paradox Engine from Borland (Version 2.0)
- C Utility Library from South Mountain Software (Version 5.0)
- IBM LinkWay (Version 2.01)
- LinkWay Developer's Toolkit from Washington Computer Services
- Video Capture Adapter High Level API from IBM

Chapter VI

THE AUTOMATIC FOLDER CREATOR

The Automatic Folder Creator is a resource that is integrated into the multimedia database system to provide access to videodisc objects. Since videodisc objects are analog, the Automatic Folder Creator digitizes the objects to put them in a form that allows them to be stored in a database and allows them to be transferred over a digital network. This chapter presents the design and operation of the AFC and shows how the AFC is integrated into a multimedia database system.

Introduction

The Automatic Folder Creator provides the means to place videodisc information in a multimedia database and to share that information on a local area network (LAN). A LAN is used to transmit digital information between computers. This prohibits the use of a LAN to distribute undigitized audio and video played from a videodisc player to workstations. Videodiscs have traditionally been included in the definition of multimedia, but since the information played from a videodisc player is analog, the ability to store videodisc material in a multimedia database, for access by many users, is impossible unless a method is implemented to digitize the information.

An interest in providing shared access to information stored on videodiscs exists because a wealth of information is available on videodiscs (Greenfeld, 1991). The Video Encyclopedia of the 20th Century, from CEL Communications, is a set of 42 discs that contain over 83 hours of historical material (Greenfeld, 1991). The Videodisc

Compendium (Pollak, 1990) provides a listing of several hundred videodiscs ranging over many topics including art, business, drama, foreign language, history, mathematics, music, psychology, science and social studies. All of this material can be placed in a multimedia database using the AFC.

The AFC solves the problem of limited digital storage (hard disk space). The number of objects that can be placed in a database is limited to the amount of available disk storage, which can be quite small in many schools due to the cost. Using the AFC, objects located on videodiscs are only digitized and placed in the database when they are requested. A least recently used (LRU) algorithm discards old digital copies of videodisc objects when database space is low and new objects need to be added. The database serves as a persistent cache for digital versions of the videodisc objects. The caching mechanism keeps digital copies of videodisc objects in database storage as long as they are being used or there is sufficient space, otherwise the digital copies are discarded. This is done since another request to digitize the material can always be made.

Automatic Folder Creator Hardware Configuration

Figure 14 shows the hardware contained in the AFC system. The central processing unit (CPU) controls the laserdisc player, processes the digital information from the digitizers, and manages network communication. Local disk storage is used to temporarily hold the digitized object before it is transferred over the network to the database. The AFC is placed on the local area network as a client workstation (Figure 15). The LAN connection is made to the AFC using the network interface shown in Figure 14.

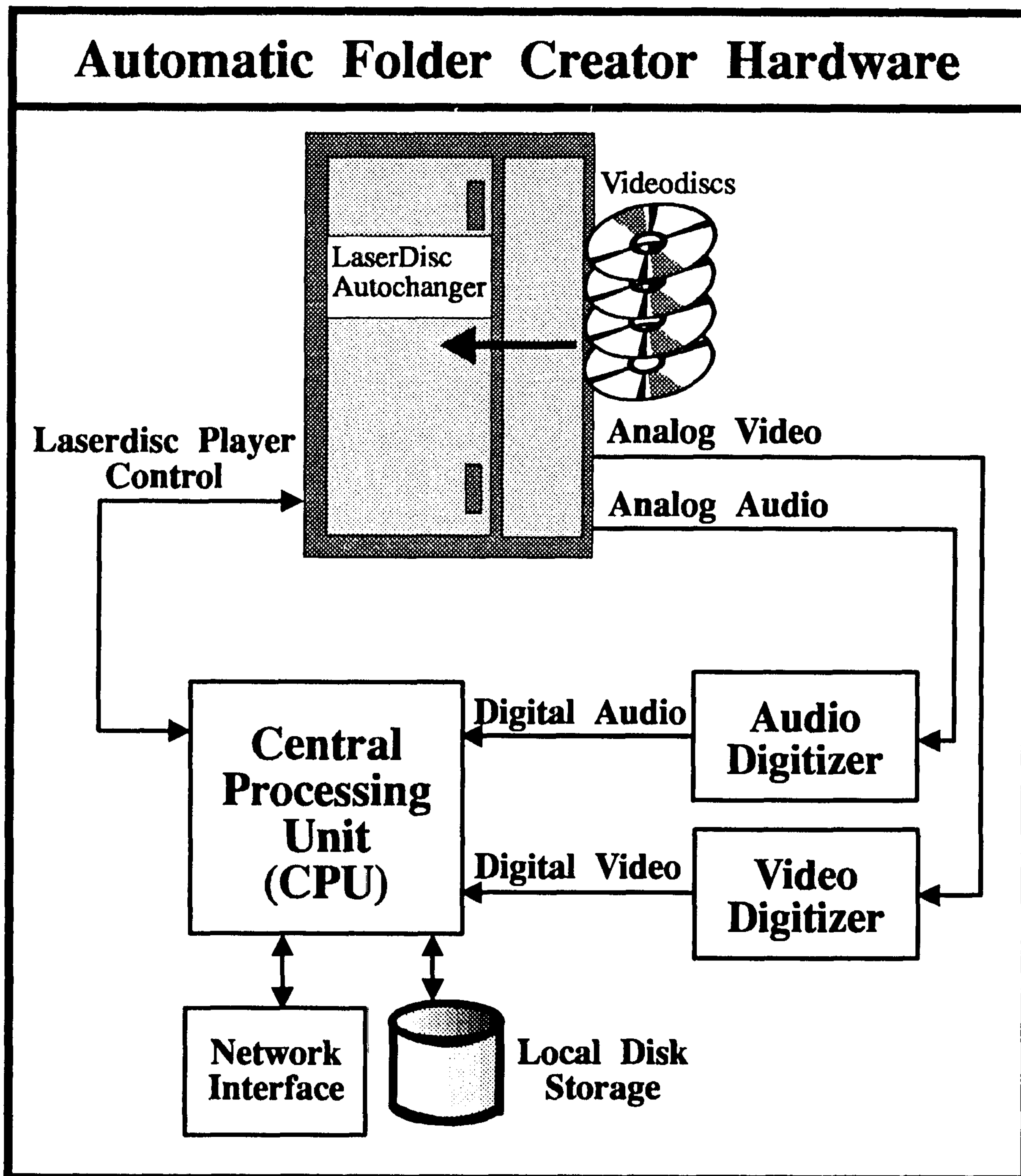


Figure 14. The figure shows the Automatic Folder Creator hardware configuration.

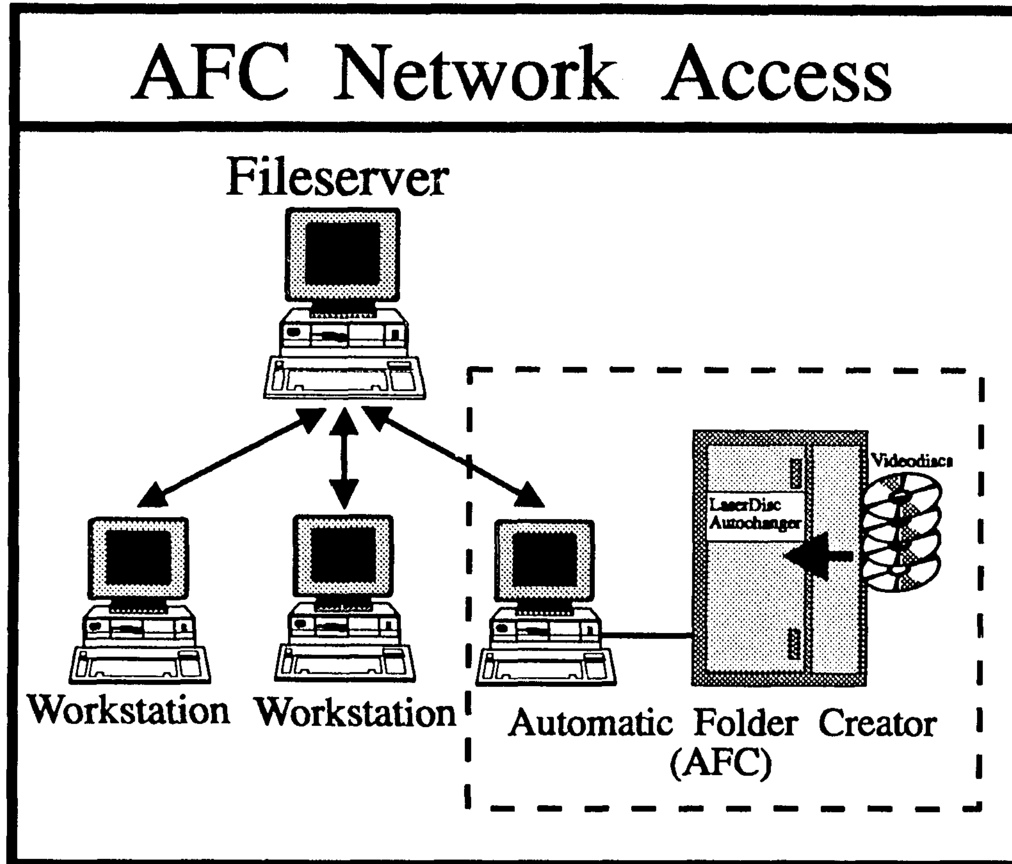


Figure 15. The Automatic Folder Creator is a client workstation on the local area network.

The Automatic Folder Creation Process

The process implemented in the AFC software is shown in Figure 16. A request queue, located on the fileserver, holds requests for videodisc material to be digitized. The AFC gets requests from the request queue in a first in first out order. A request is a message that contains all the information the AFC needs to digitize the requested videodisc material: type of object to create, title, videodisc name, videodisc side, audio control information, color information, segment start frame/time, segment end frame/time, and if required a frame/time list.

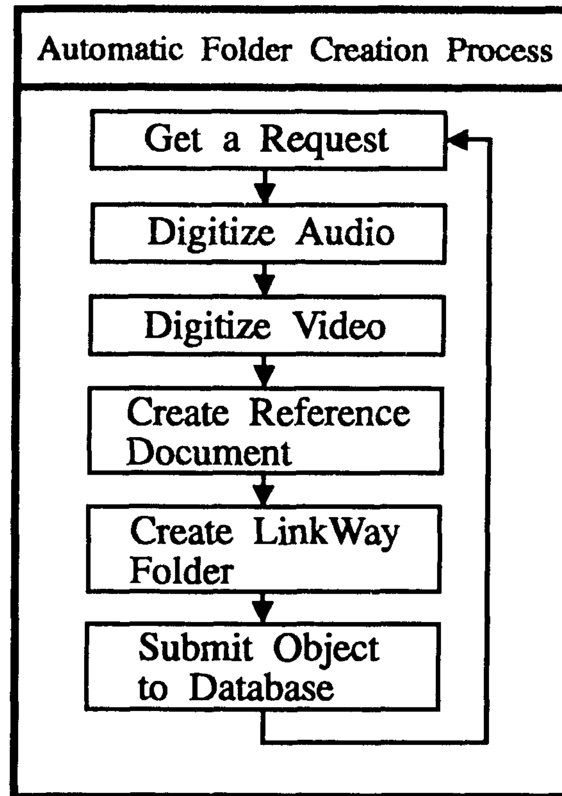


Figure 16. The figure shows the process used by the Automatic Folder Creator.

Commands are sent to the videodisc player to play the appropriate segments, based on the information provided in the request file. Because the software runs on a DOS workstation, the audio and video are digitized in two separate passes. The first pass is used to digitize the audio and the second pass is used to digitize the video.

After the video and audio are digitized a reference document and a LinkWay folder are created. The reference document describes the material that has been digitized and the LinkWay folder provides the method for experiencing the digitized videodisc material and for viewing the reference document.

The final step in the process is submission of the object to the multimedia database. Placing the object in the database is called *realizing* the object. A method called *REALIZE* is used to place the object in the database using multimedia database management system functions. REALIZE contains a least recently used (LRU)

algorithm that supports removal of the least recently used videodisc objects from the database if room is needed to make a submission. After a request has been processed, the AFC checks the request queue for another message.

The Automatic Folder Creator is an integral component of the multimedia database system. The way in which the AFC is integrated into the multimedia database system is shown in Figure 17. A user may have a request for an object fulfilled by the multimedia database management system (MMDBMS) in one of two ways. If the object exists in digital form in the multimedia database, it can be transferred to the user immediately. If the object resides on a videodisc, and is not in the database in digital form, the MMDBMS places a request in the AFC request queue to signal the object needs to be digitized. (Formats for AFC requests are provided in Appendix A.) When the object has been digitized and placed in the database, the MMDBMS returns the object to the user. A time delay is introduced if the object must be digitized before it can be returned to the user.

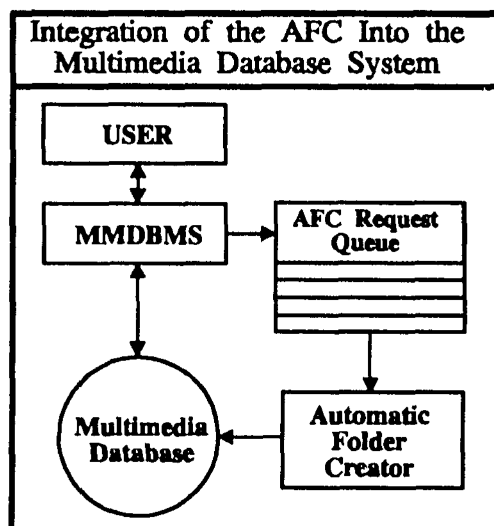


Figure 17. The diagram shows how the Automatic Folder Creator is integrated into the multimedia database system.

AFC Create Types

The Automatic Folder Creator can digitize the information on a videodisc, and combine the resulting digital information, in five different ways. Each of the five ways of digitizing, and combining the digital information, is called a *create type.* Each create type provides a different means of representing the information contained on a segment on the videodisc. The following create types are supported by the AFC:

1. LinkWay movie with sound (LMS): the motion video and sound are digitized. The digital video and sound are combined using a LinkWay folder to create a movie.
2. LinkWay slide show type A (LSA): still images are digitized at 10 second intervals and all the audio is digitized. The images and audio are combined using a LinkWay folder to create an automated slide show.
3. LinkWay slide show type B (LSB): still images are digitized according to a frame list and all the audio is digitized. The images and audio are combined using a LinkWay folder to create an automated slide show.
4. LinkWay audio (LAD): only the sound from a segment is digitized. A LinkWay application is used to playback the audio.
5. LinkWay MCGA 320 x 200 x 256 color image (FRM): a single image is digitized from a specified frame/time location on the videodisc.

Figure 18 shows a snapshot of a digital video with sound (LMS) folder created by the AFC. The digital video is played at the center of the screen and the audio is played on an attached audio adapter. Figure 19 shows the LMS folder with the folder information window displayed. The folder information window displays a description of the folder.

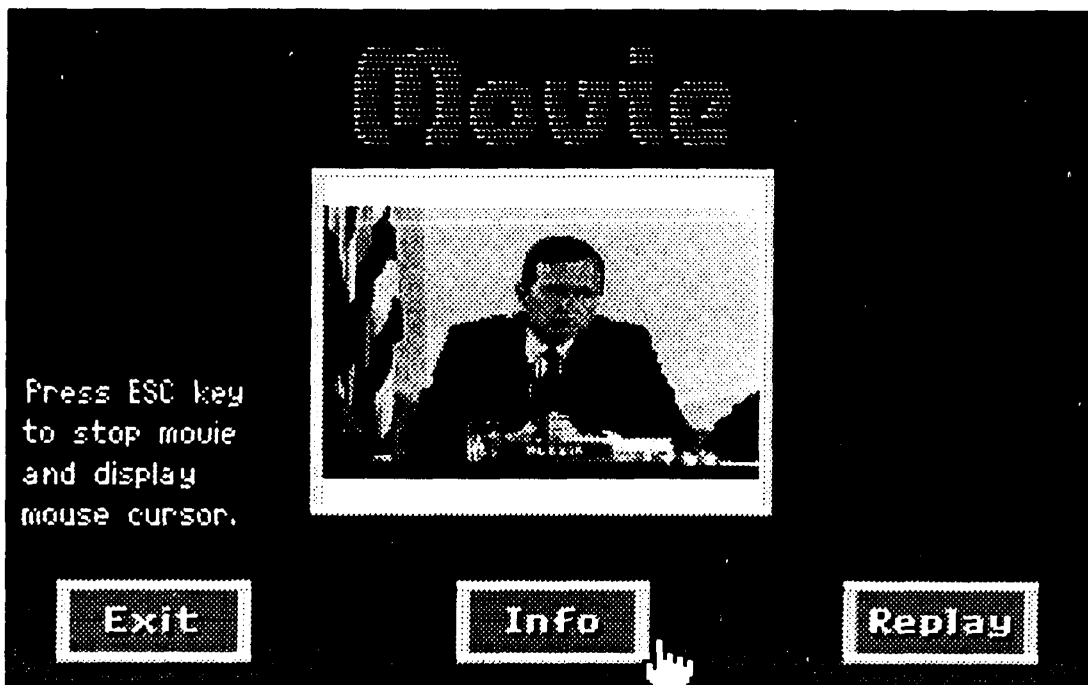


Figure 18. The figure shows a snapshot of a digital video with sound (LMS) folder created by the Automatic Folder Creator.

Figure 19. The figure shows a snapshot of a digital video and sound (LMS) folder with the folder information window displayed.

Figure 20 shows a snapshot of a slide show with sound folder where the images are based on a frame list (LSB). An LSA folder looks the same. The picture displayed is changed at 10 second intervals while the audio plays on an attached audio adapter. Several options are provide to control the slide show including rewind audio, scan forward, scan backward, pause, resume, and restart. Figure 21 shows the LSB folder with the folder information window displayed.

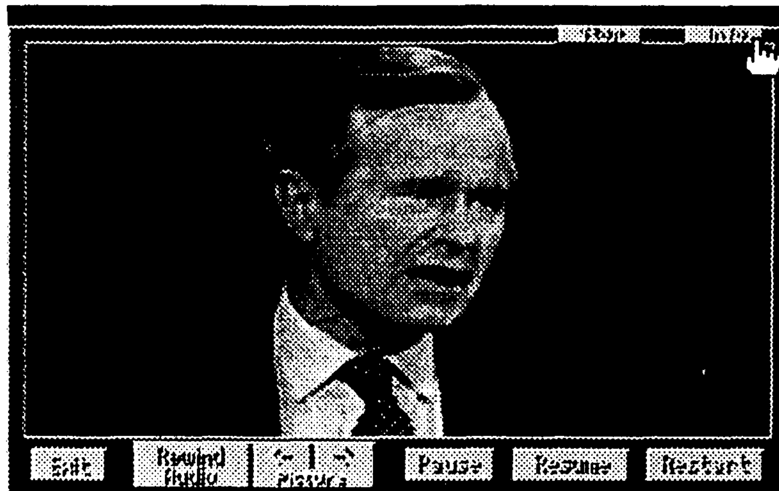


Figure 20. The figure shows a snapshot of a slide show (LSB) folder created by the Automatic Folder Creator.

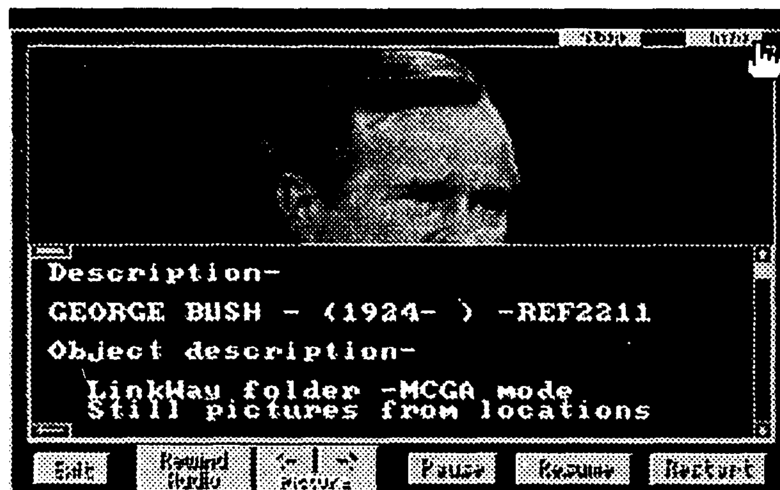


Figure 21. The figure shows a snapshot of a slide show (LSB) folder with the folder information window displayed.

Figure 22 shows a snapshot of a sound only (LAD) folder created by the AFC. The user can control the playback of the audio using controls on the screen and can display the folder information in a window as shown in Figure 23.

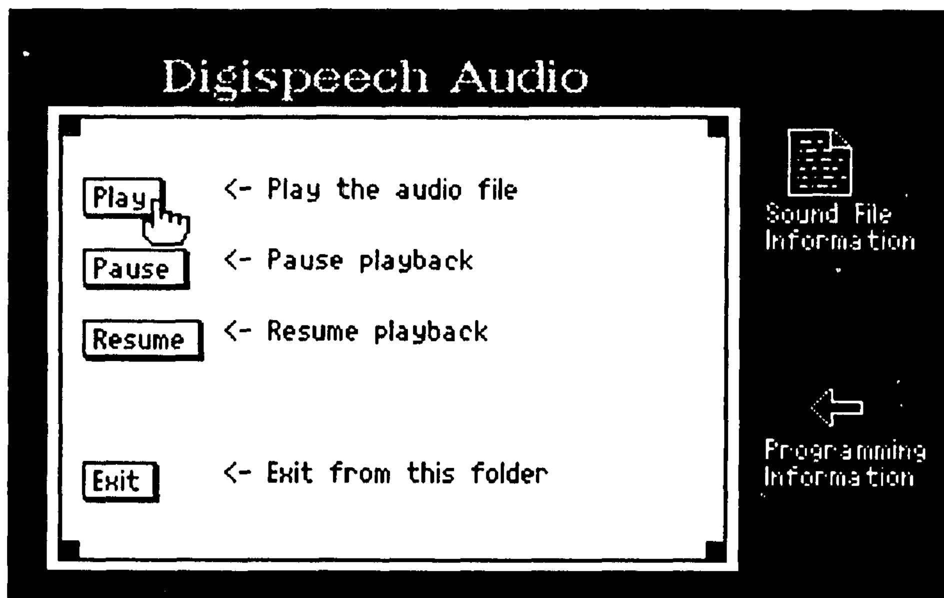


Figure 22. The figure shows a snapshot of a sound (LAD) folder created by the Automatic Folder Creator.

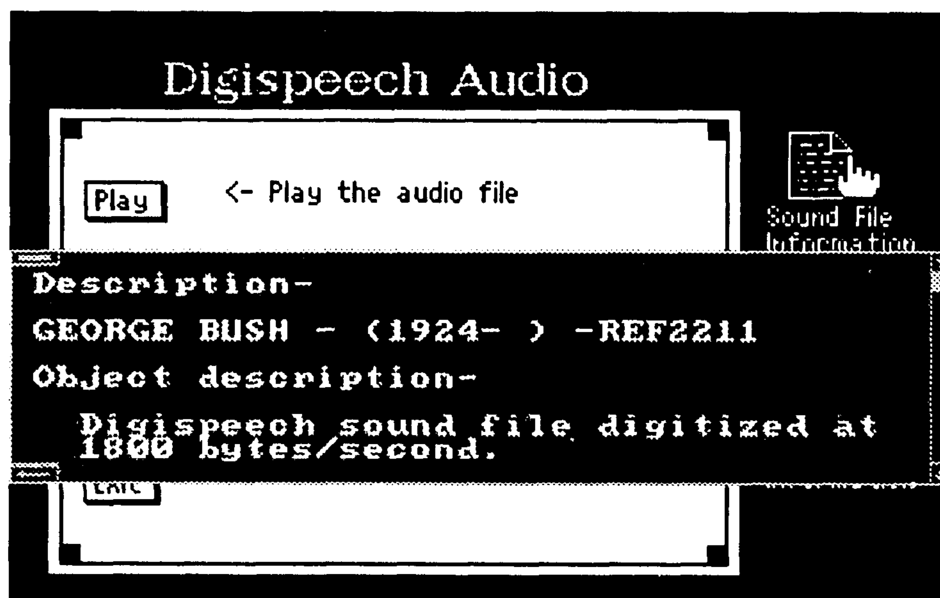


Figure 23. The figure shows a snapshot of a sound (LAD) folder with the folder information window displayed.

Summary

The Automatic Folder Creator is a resource that is integrated into the multimedia database system to provide access to videodisc objects. Since videodisc objects are analog, the Automatic Folder Creator digitizes the objects to put them in a form that allows them to be stored in a database and allows them to be transferred over a digital network.

The Automatic Folder Creator processes requests placed in a request queue. A request contains information that describes the object to be digitized. The creation process includes reading a request, digitizing audio, digitizing video, creating a reference document, creating a LinkWay folder and submitting the object to a database. The digitization process is accomplished using a video digitizer and an audio digitizer.

A method called *REALIZE* is used to place digital versions of videodisc objects in the database. In the database videodisc objects have either of two forms: a reference to a videodisc segment or a digital copy of the videodisc object. A least recently used algorithm, located in the REALIZE method, removes digital copies of videodisc objects from the database when insufficient space is available in the database to store a new digitized videodisc object. When a digital copy of a videodisc object is removed from the database, the object's reference information is left in the database so the object can be re-digitized if it is requested. The database serves as a cache for videodisc objects. Recently digitized videodisc objects replace digital versions of videodisc objects that were not recently used when free space in the database is low.

A videodisc object can be digitized and packaged in five different ways. These five ways result in a digital movie, two different types of slide shows, a sound only application, or a still image. The way a videodisc object is digitized and packaged depends on the type of information in the videodisc object reference and the request made by the user.

CHAPTER VII

THE DATA MODEL

A data model defines the objects that can be stored in the database and the operations that can be performed on the objects. Four types of objects are defined: the database object, the exception word object, the topic object, and the password object. A database object is a multimedia object plus associated pieces of information that have been stored in the database; an exception word object is a word with little or no meaning that is removed from word index lists and titles when indexes are built to speed searches and reduce storage; a topic object is a pre-defined domain of information within which an object can be classified; and, a password object is a character string used to implement database security.

The Database Object

A database object is a multimedia object plus pieces of information that are associated with the multimedia object when it is added to the database. A multimedia object is a collection of files; and, the associated pieces of information include the object handle, the referent, status information, and catalog information (Figure 24).

The Object Handle

An object handle is a base thirty-six number associated with a multimedia object that provides a reference to the object (Figure 25); object handles are similar to card catalog

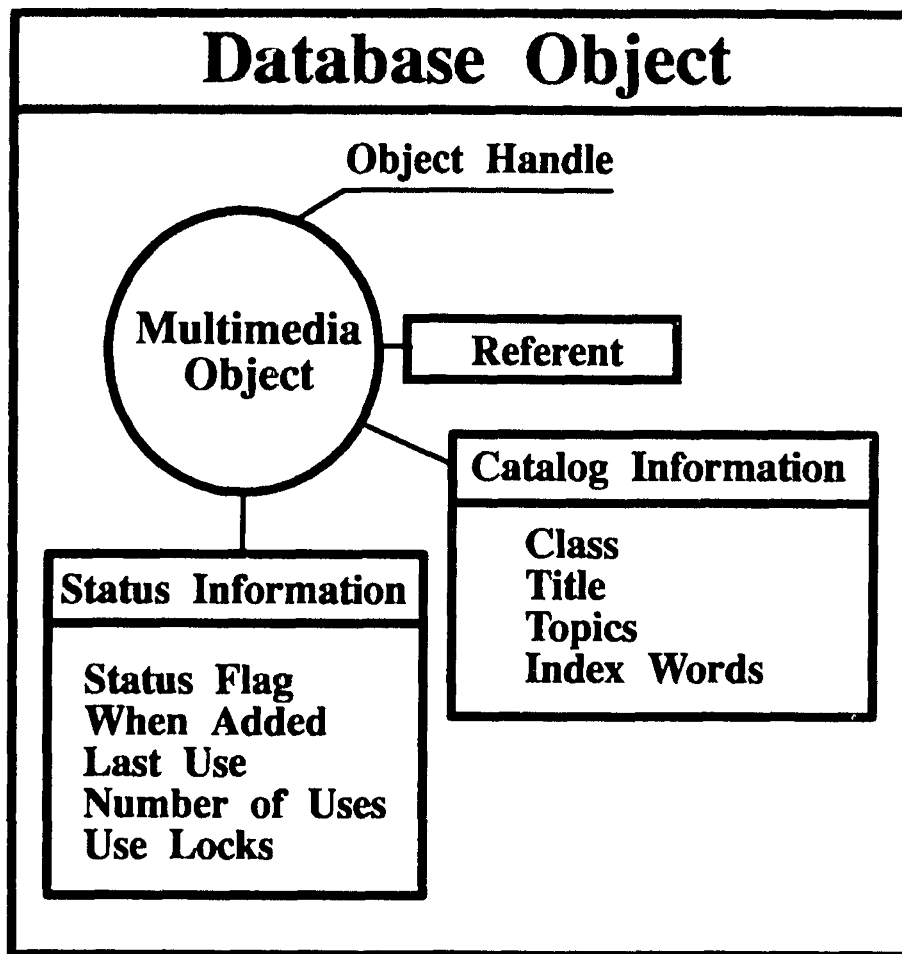


Figure 24. A database object is a multimedia object plus the pieces of information associated with the multimedia object when it is stored in the database.

numbers for books in a library. Each multimedia object stored in a multimedia database is given a unique object handle that is not reused -- even after the object has been deleted. An object handle is a permanent reference to an object, even after it is no longer part of the database.

An eight digit base thirty-six number is used as an object handle because it maps to allowable DOS directory names. Each object stored in the database is placed in a separate directory, where the name of the directory is the same as the object handle. The label placed on the object and the storage location of the object are the same.

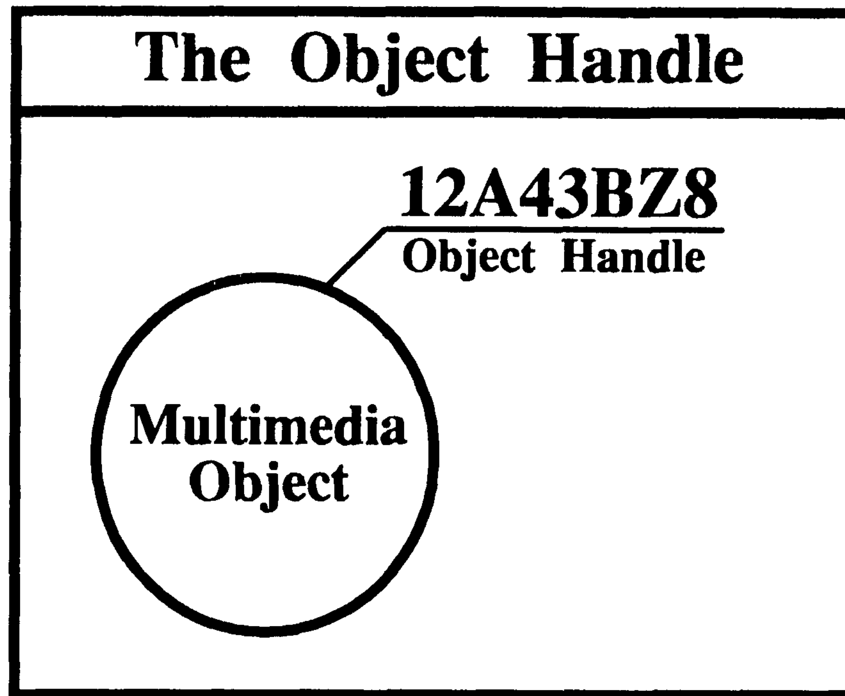


Figure 25. An object handle is an eight digit base thirty-six number associated with a multimedia object stored in the database.

The Object Referent

The object referent is a piece of information that the method for experiencing the multimedia object requires to execute. An object referent is a computer oriented way of referring to an object in the same way that a title is a human oriented way of referring to the same object. A title for an image object may be "A picture of a steam boat on the Mississippi." The referent for the same image object may be BOAT.PCM. BOAT.PCM is the filename given to the image when it is saved to disk. The filename is required for the experience method to display the image.

The operating system command interpreter is the experience method for an application and the referent for an application is its start instructions. Start instructions are passed to the operating system command interpreter to experience applications. Other multimedia objects' referents are their filenames; the filename provides sufficient information for the experience method to execute so the object can be experienced (Figure 26).

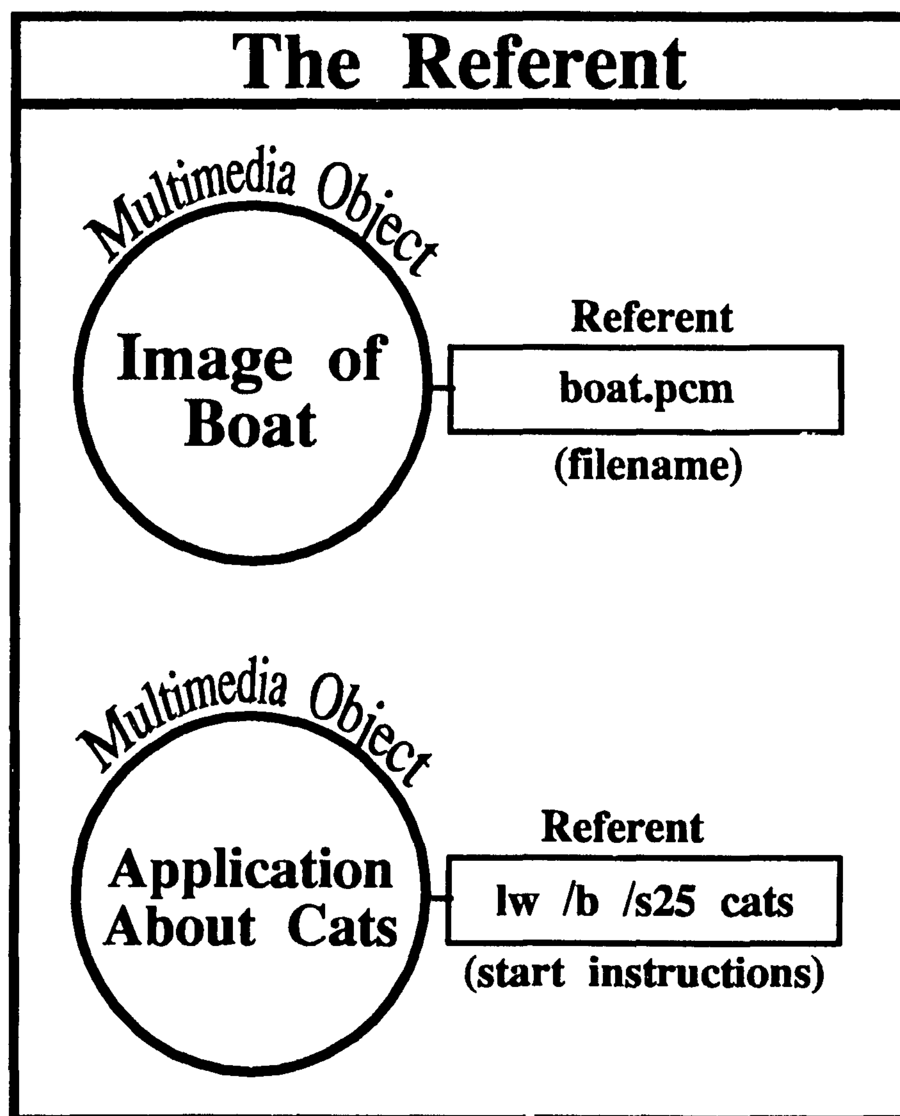


Figure 26. An object referent is the way the experience method refers to a multimedia object.

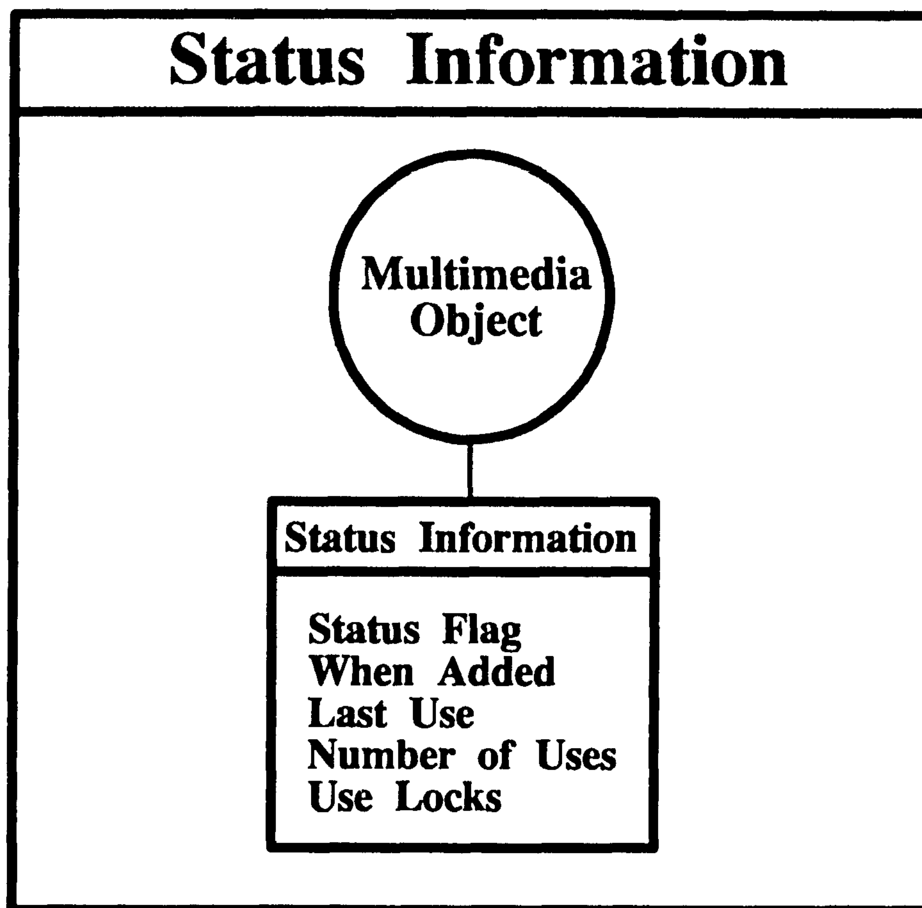


Figure 27. Status information is associated with each multimedia object stored in the database.

Status Information

Status information indicates the status of an object in the database and includes the following: status flag (STATUS), when added (WHENADD), last use (LASTUSE), number of uses (NUMUSES), and use locks (USELOCKS) (Figure 27).

STATUS is used to track the state of an object and has six values: object is on videodisc (V), object is local/available (L), object is requested from AFC machine (R), object is being added (+), object is being deleted (D), videodisc object is being reverted from L to V (M), and object is being updated (U). WHENADD is the date and time the object was added to the database. LASTUSE is the date and time the object was last

accessed by a user. NUMUSES is the number of times users have accessed the object. USELOCKS indicates the number of users currently using the object. If a use lock is applied to an object its state cannot be changed; for example, it cannot be deleted. STATUS and USELOCKS are used to manage concurrency of database use on a network.

Catalog Information

A database catalog is a descriptive list of objects in the database and is like a card catalog of books in a library. Users search the catalog to locate objects. The database catalog contains four types of information: class, title, topics, and index words (Figure 28).

The class of an object is the name of the group to which the instance of the multimedia object belongs within a class hierarchy of multimedia objects. An example of a class is *LinkWay MCGA 320 x 200 x 256 color image* (LWMCGA256). A class hierarchy for multimedia objects supported by the multimedia database management system is presented in the following section.

The title is a character string that describes the object. An example of a title is "Kennedy's Speech at the Berlin Wall." Topics (also referred to as subjects) are pre-defined domains of information. Some examples of topics are HISTORY, BIOLOGY, MATHEMATICS, POLITICAL SCIENCE, and MUSIC. And, index words are a list of words which indicate the object's content. Examples of index words are TRAGEDY, TITANIC, ICEBERG, PASSENGER, and SHIPS. Class, topics, and index words are indexed for quickly locating objects. Words used in the title and words listed as index words are lumped together as indexes for word searches.

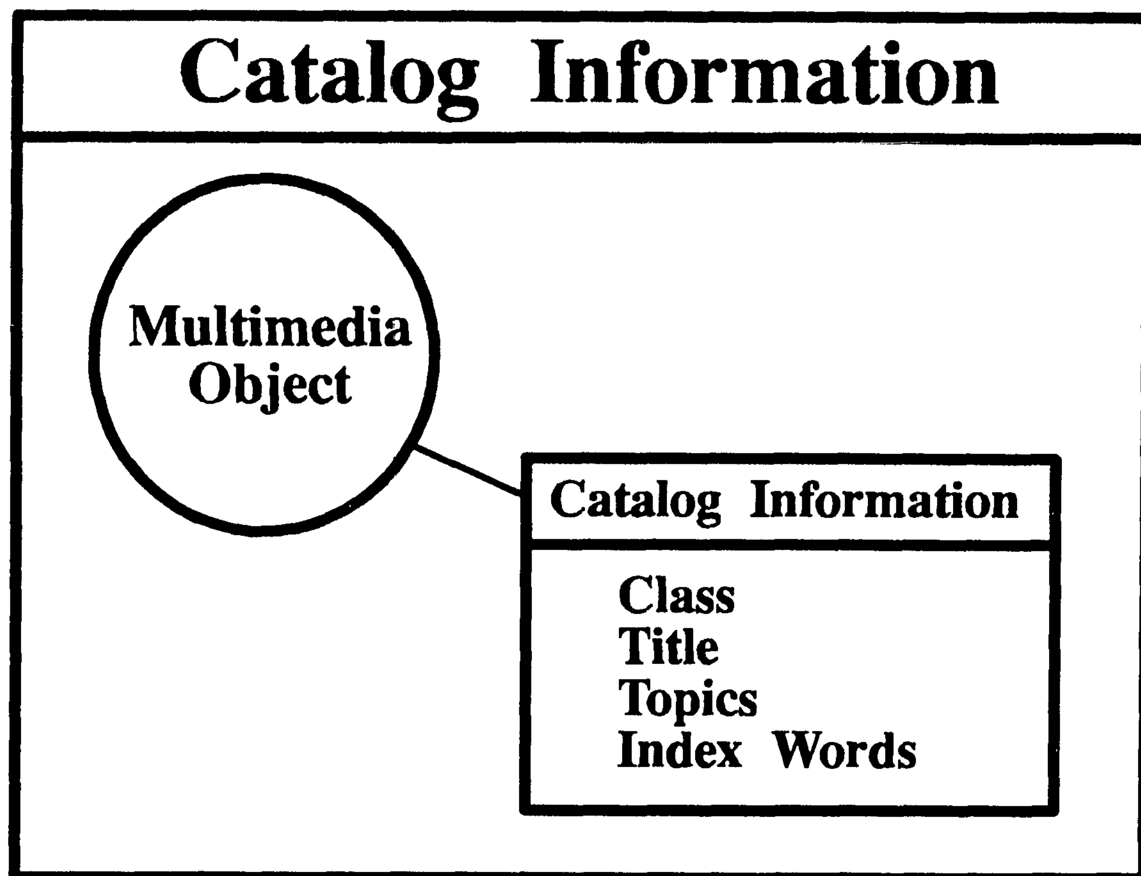


Figure 28. Catalog information is associated with each object in the database.

Classifying Multimedia Objects

Classifying objects is a process of grouping according to some systematic division into classes, which proceeds in a stepwise fashion beginning with divisions based on large differences and continuing with divisions based on smaller differences. Each step in the classification process requires finer discrimination between the objects. The result of the classification process is a rooted, directed acyclic graph called a class hierarchy (Kim, 1991).

Classification Criteria

The most general types of multimedia objects are images, animations, digital video, sound, voice, music, text, and combinations of multimedia objects called applications. Multimedia objects can be further classified based on format differences and hardware dependencies.

A format is a way of encoding the information contained in a multimedia object for storage in a file. Different formats exist for a single type of multimedia object for two reasons: different compression algorithms are used to reduce the disk space required to store an object and software developers, working independently, choose to write the object information to disk in different ways.

Multimedia objects have hardware dependencies for experiencing them. For example, an image object may require a specific screen mode to be displayed and a sound object may require a specific type of hardware to be reproduced. The differences in the hardware required for experiencing different objects must be considered when classifying objects to deal with differences in users' workstation configurations. As an example, a student using a computer equipped with a Multi-Color Graphics Adapter may not be able to display an image developed using an Enhanced Graphics Adapter.

Images, animations, digital video, sounds, voice, music, text, and applications can be divided into subclasses based on format differences. A subclass defined by a particular format can be further subdivided into subclasses based on hardware dependencies. The class hierarchy shown in Figure 29 results from classifying images based on format differences and hardware dependencies.

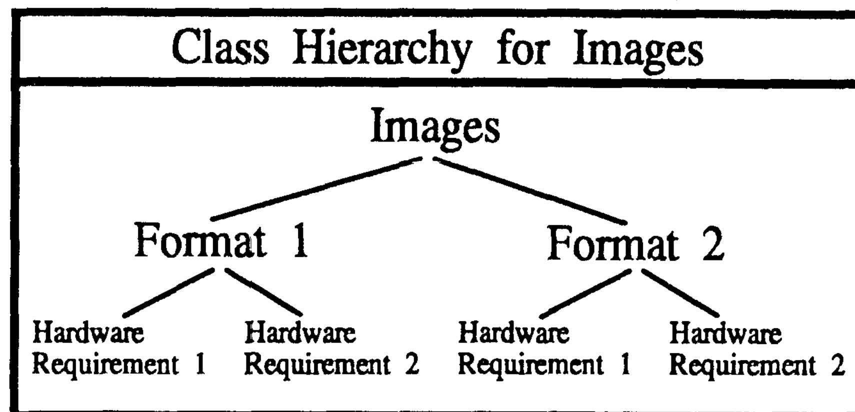


Figure 29. The figure shows a class hierarchy for images based on format differences and hardware dependencies.

To unify the organization of class hierarchies which result from classifying images, animations, digital video, sound, voice, music, text and applications, a superclass called ***Multimedia Objects*** is created (Figure 30). Most of the MMDBMS operations are defined for the **Multimedia Objects** class instead of being individually defined for specific subclasses. This allows most MMDBMS operations to be used without regard

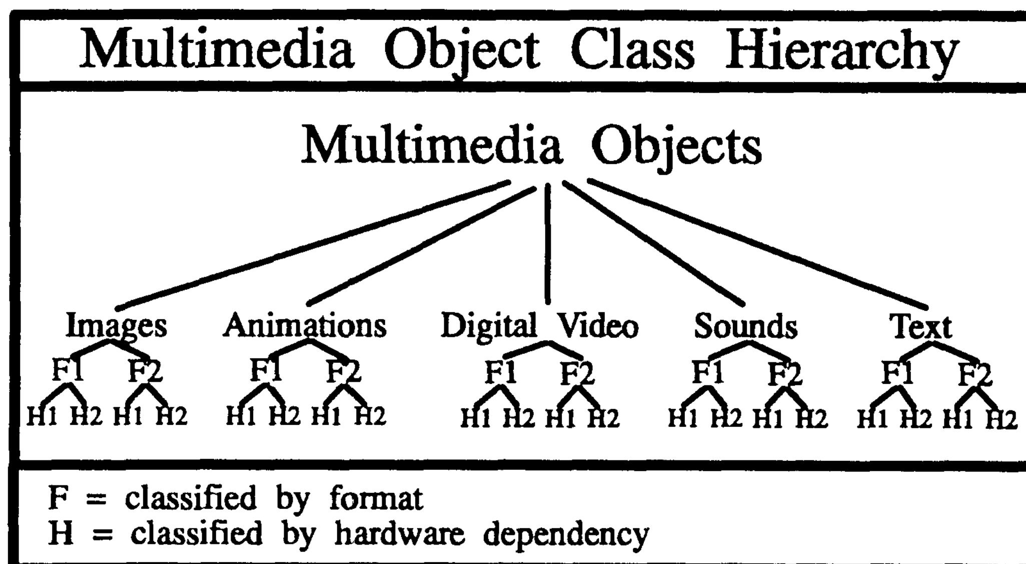


Figure 30. The figure shows a class hierarchy for a subset of multimedia objects types based on format differences and hardware dependencies.

for the specific class membership of an object on which operations are being performed because each subclass inherits functions from its ancestors. For example, the EXPERIENCE operation, which is defined for the **Multimedia Objects** class, can be used to experience an image, an animation, a digital video clip, a sound, a piece of music, a piece of text, or an application. This use of polymorphism allows the different methods required to experience each class of object to be aggregated under one operation named EXPERIENCE.

The Multimedia Object Class Hierarchy

The multimedia object class hierarchy includes a subset of the available types of multimedia objects. Even though only a subset has been classified, the hierarchy can be extended for additional object types.

The root class of the class hierarchy is the **Multimedia Objects** class, and all types of multimedia objects are subclasses of the **Multimedia Objects** class. The subclasses of the **Multimedia Objects** class are **Images, Text, LinkWay Pieces, Sounds, Digital Video, Videodisc Segments, Applications**, and **Generic** (Figure 31).

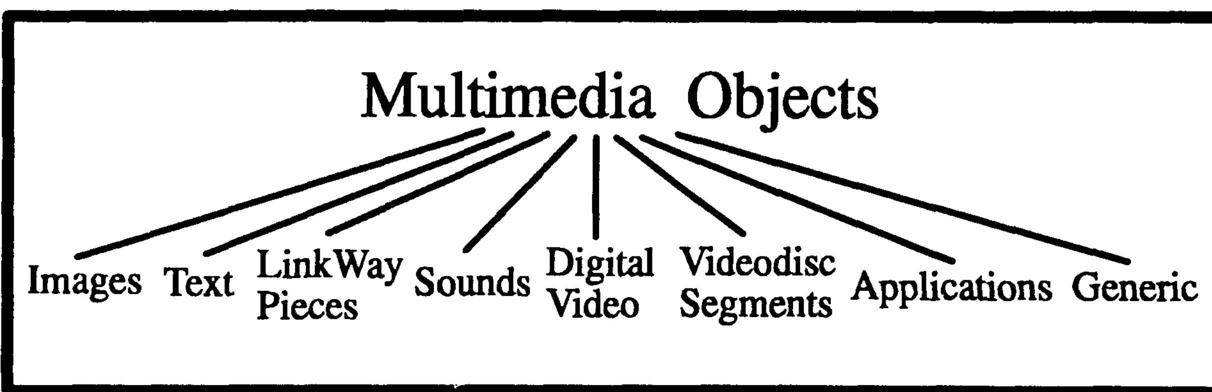


Figure 31. The figure shows the **Multimedia Objects** class hierarchy for object types supported by the multimedia database management system.

Three previously unmentioned types of multimedia objects have been included in the class hierarchy: **LinkWay Pieces, Videodisc Segments**, and **Generic. LinkWay Pieces** includes objects that are building blocks for LinkWay applications (cut objects, cut pages, bit clips, folders, and fonts) which are specific to LinkWay and are not accounted for by any of the other classes. **LinkWay Pieces** objects are used by developers of LinkWay applications. **Videodisc Segments** is a classification of objects created by the Automatic Folder Creator (AFC) which digitizes audio and video on demand. **Videodisc Segments** includes pictures, slide shows, digital video clips, and audio clips that are created using audio and video located on videodiscs. **Generic** includes any type of multimedia object that is not classified elsewhere in the class hierarchy. **Generic** provides support for unclassified multimedia objects limited to those operations that do not require class membership information. A **Generic** object cannot be experienced because the MMDBMS lacks the information required to execute the EXPERIENCE function. **Generic** objects can be stored, retrieved, and removed from the database.

The **Images** class is divided, based on format differences into **LinkWay Images, MacPaint Images**, and **Storyboard Images** (Figure 32).

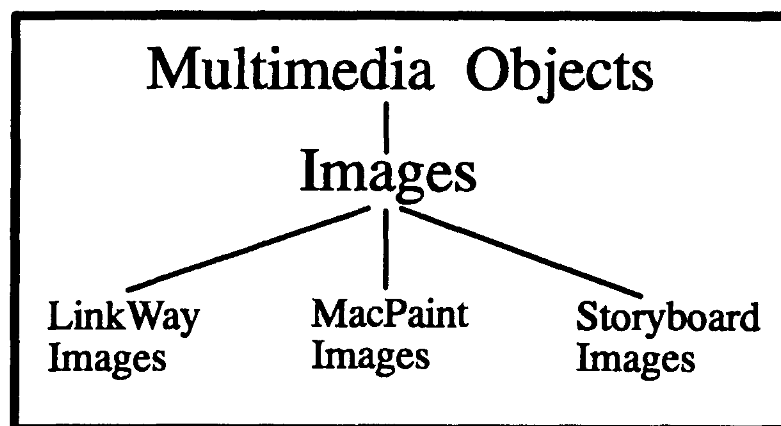


Figure 32. The figure shows the class hierarchy for **Images**.

The **LinkWay Images** class is divided, based on hardware dependencies, into the following classes: **LWCGA, LWMCGA2, LWMCGA256, LWEGA,** and **LWVGA16** (Figure 33). The five types of **LinkWay Images** require five different video modes to be experienced. A description of the five types is provided in Appendix B.

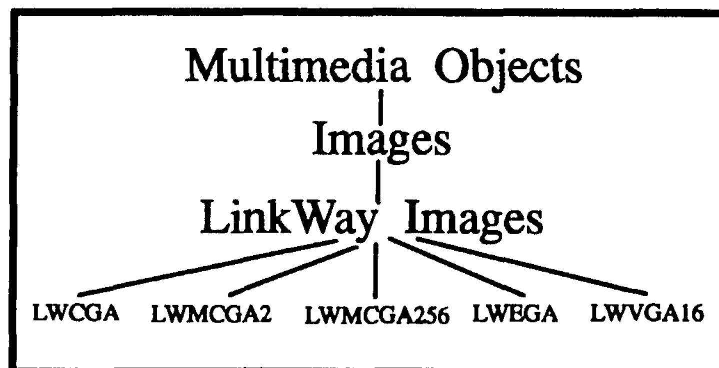


Figure 33. The figure shows the class hierarchy for **LinkWay Images.**

The **MacPaint Images** class contains only one subclass called ***MACPAINT*** (Figure 34). This subclass is not really needed because it is the only subclass associated with its parent; but, the structure supports extensibility. If several types of **MacPaint Images** become available, the class hierarchy does not need to be restructured to support the new types. The **MACPAINT** class is described in Appendix B.

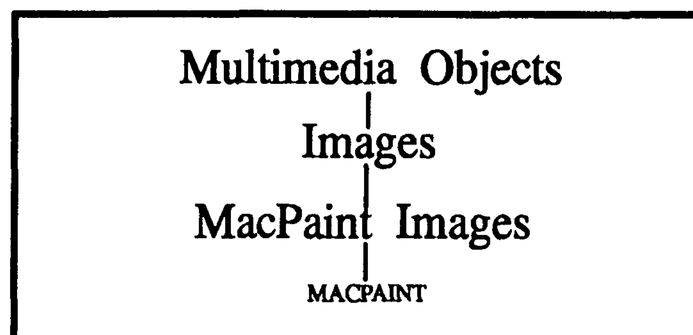


Figure 34. The figure shows the class hierarchy for **MacPaint Images.**

The **Storyboard Images** class is divided, based on hardware dependencies, into the following classes: **SBMCGA256, SB16EGA350, SBEGA200,** and **SBVGA16** (Figure 35). The four types of **Storyboard Images** require four different video modes to be experienced. A description of the four types is provided in Appendix B.

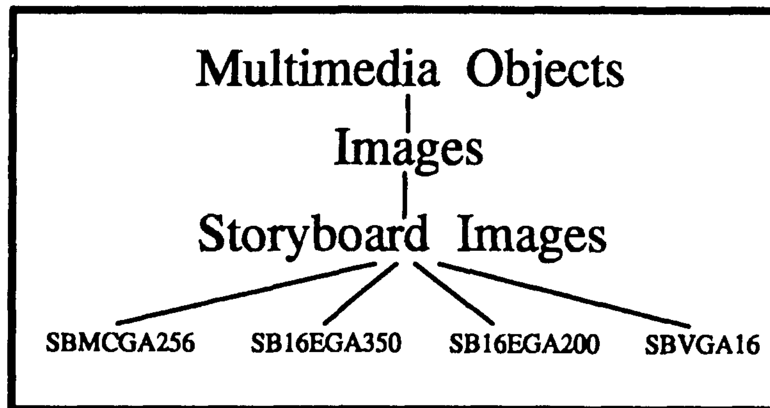


Figure 35. The figure shows the class hierarchy for **Storyboard Images.**

The **Text** class contains one subclass called ***ASCIITEXT*** (Figure 36). Like the **MACPAINT** class, **ASCIITEXT** is created as a subclass of **Text** to promote extensibility. Different text formats supported by a variety of word processors could become subclasses of the **Text** class. **ASCIITEXT** is described in Appendix B.

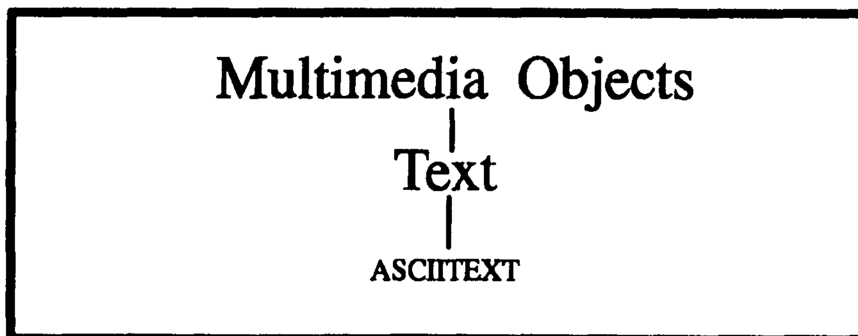


Figure 36. The figure shows the class hierarchy for **Text.**

The **LinkWay Pieces** class contains LinkWay components used to develop LinkWay applications. The subclasses of **LinkWay Pieces** are **LWOB**, **LWPG**, **LWBIT**, **LWFNT**, and **LWFOLDER** (Figure 37). **LWOB** is an object cut from a LinkWay page. **LWPG** is a page cut from a LinkWay folder. **LWBIT** is a bit map cut from a LinkWay image. **LWFNT** is a LinkWay font. And, **LWFOLDER** is a LinkWay folder. **LWOB**, **LWPG**, **LWBIT**, **LWFNT**, and **LWFOLDER** are described in Appendix B.

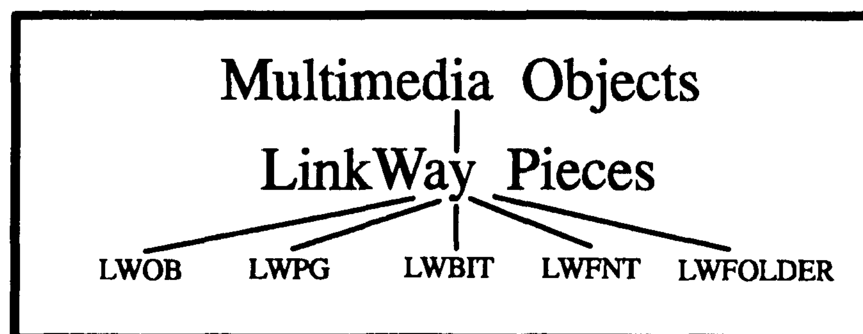


Figure 37. The figure shows the class hierarchy for **LinkWay Pieces**.

The **Sounds** class includes two subclasses: **DS201CVSD** and **ACPASOUND** (Figure 38). These classes represent two different sound file formats for two different pieces of hardware. **DS201CVSD** and **ACPASOUND** are described in Appendix B.

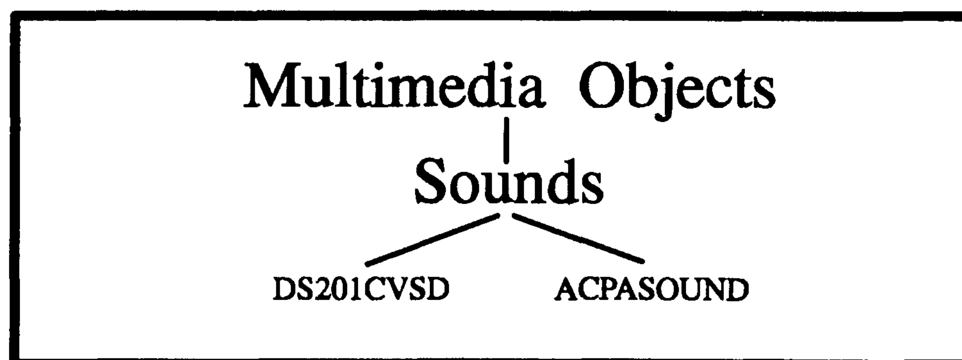


Figure 38. The figure shows the class hierarchy for **Sounds**.

The **Digital Video** class contains three subclasses: **PVI**, **DVI**, and **RTV** (Figure 39). **PVI**, **DVI**, and **RTV** are described in Appendix B.

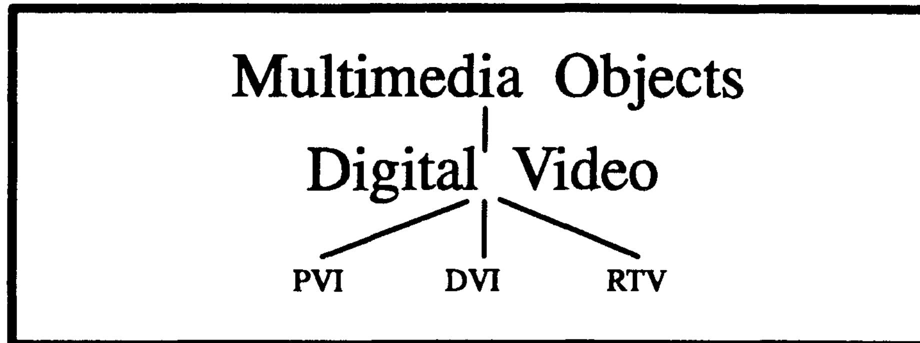


Figure 39. The figure shows the class hierarchy for **Digital Video.**

The class hierarchy for **Videodisc Segments** is constructed differently from the other class hierarchies presented in this section. The classification process for **Videodisc Segments** is provided below.

A videodisc contains either motion video clips or still frames that represent individual instances of multimedia objects. Since the analog video and audio that comes from a videodisc player cannot be placed in a multimedia database, references to the clips and frames are placed in the database. When a user requests an object located on a videodisc, the reference information stored in the database is used to instruct the Automatic Folder Creator (AFC) to digitize the analog audio and video and to place the result in the database for the user to access. A videodisc object is not realized (put into digital form) until a user requests the object. The digital object created by the AFC can take many different forms which depend on the type of reference information available for the object and the needs of the user.

The reference information for videodisc objects results from cataloging the contents of a videodisc. The information contained in a catalog entry includes type of object,

videodisc name, videodisc side, audio information, color information, start position, end position, and, if the type requires it, a frame list (Figure 40).

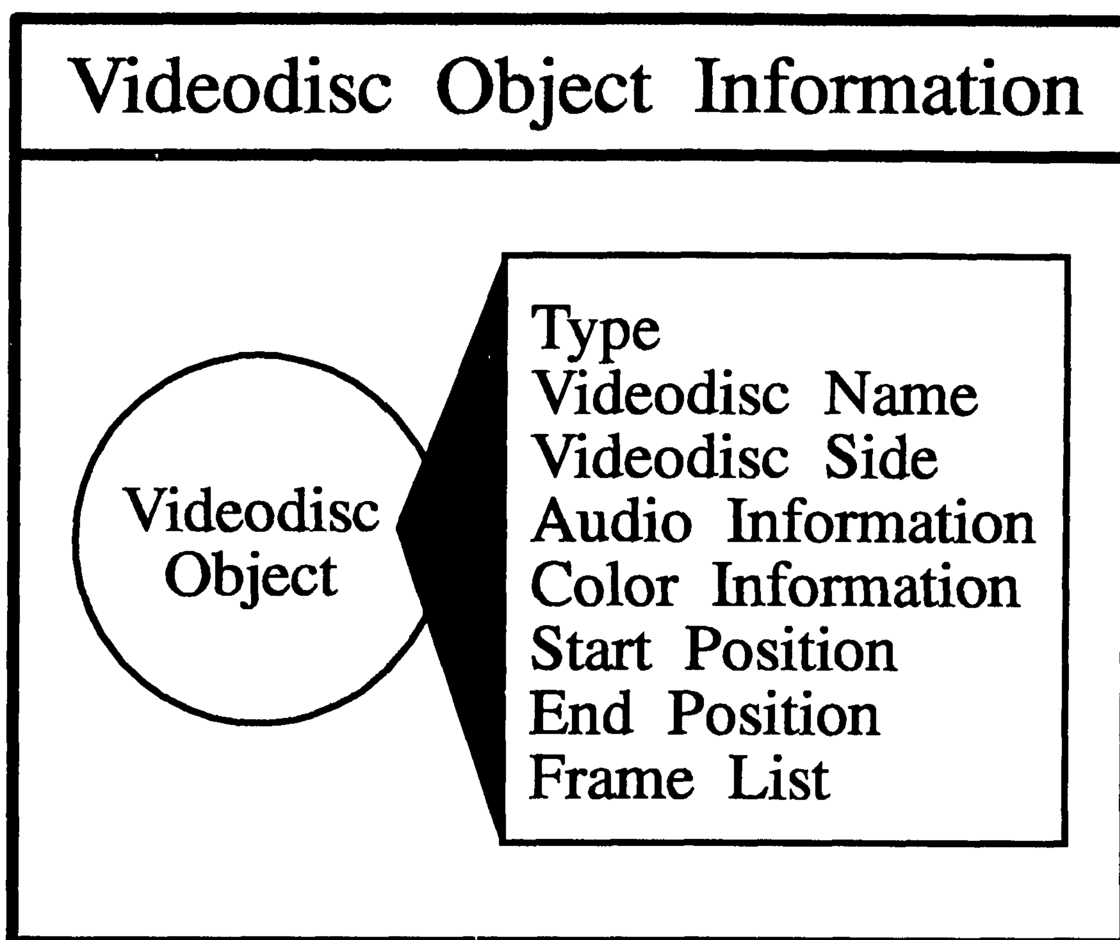


Figure 40. The figure shows the videodisc object information.

There are three types of videodisc objects: segments (**SEG**), segments with frame lists (**SFL**), and frames (**FRM**). A segment contains information for a motion video clip and is defined by a start position and an end position. A segment with frame list also contains information for a motion video clip that includes a start position and an end position, but, in addition includes a list of key frames in the segment. A frame is a single still image and is defined by a start and end position that are the same.

The Automatic Folder Creator can create five types of objects. Each type of object is a digital representation of the analog audio and video coming from the videodisc player when a segment of the videodisc is played. The first type of object is a slide show with audio, where the images in the slide show are digitized at ten second intervals. This type of object is labeled ***LSA***. The second type of object is a slide show with audio where the images in the slide show are digitized according to a list of key frames in a frame list. This type of object is labeled ***LSB***. The third type of object is a digital video clip with audio which is labeled ***LMS***. The fourth type of object is an audio clip labeled ***LAD***. And, the fifth type of object is a still image labeled ***LPM***. **LSA**, **LSB**, **LMS**, and **LAD** objects are DOS applications. **LPM** is a LWMCGA256 type object.

A **SEG** type videodisc object contains enough information for the AFC to create **LSA**, **LMS**, and **LAD** type objects. The **SFL** type videodisc object contains enough information for the AFC to create **LSA**, **LSB**, **LMS**, and **LAD** type objects. And, the **FRM** type videodisc object contains only enough information to create an **LPM** type object.

The type of object that the AFC creates is determined at the time the user makes the request for the object. Since an **SFL** type videodisc object can be realized as an **LSA**, **LSB**, **LMS**, or an **LAD** type object by the AFC, the user can choose any one of those four types. An object's type is dynamically determined based on a user's request.

The class hierarchy for **Videodisc Segments**, shown in Figure 41, demonstrates the relationship between the **SEG**, **SFL**, and **FRM** videodisc object types and the **LSA**, **LSB**, **LMS**, **LAD**, and **LPM** object types created by the AFC. Internally the MMDBMS also uses the knowledge, which is not shown in Figure 41, that **LSA**, **LSB**, **LMS** and **LAD** types are DOS applications and that the **FRM** type is an **LWMCGA256** image. **SEG**, **SFL**, **FRM**, **LSA**, **LSB**, **LMS**, **LAD**, and **LPM** types are described in Appendix B.

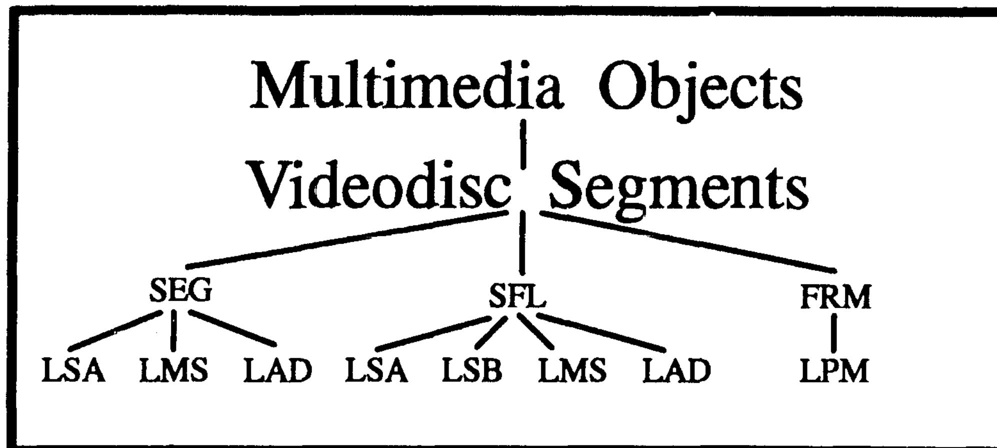


Figure 41. The figure shows the class hierarchy for **Videodisc Segments.**

The **Applications** class contains one subclass called ***DOSAPP*** (Figure 42). **DOSAPP** is an application that runs under the DOS operating system. The **Applications** class could be extended to include applications for other operating systems. **DOSAPP** is described in Appendix B.

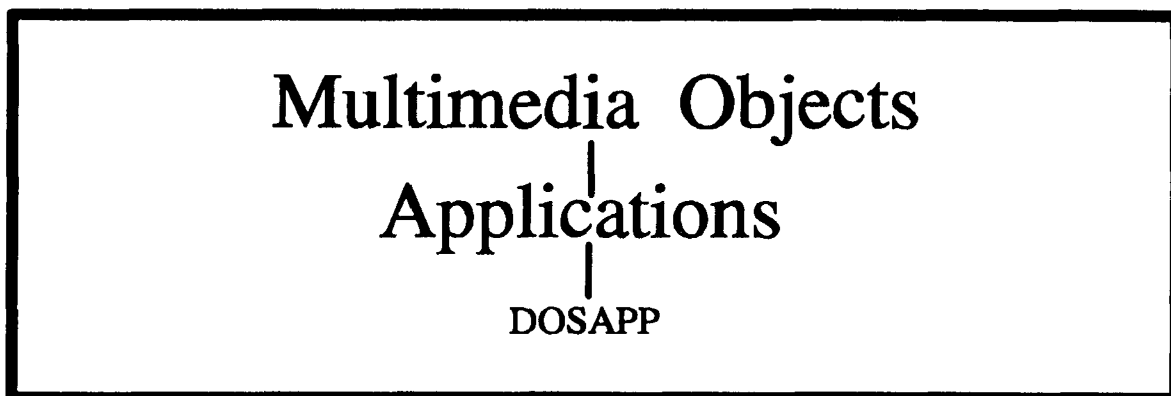


Figure 42. The figure shows the class hierarchy for **Applications.**

The **Generic** class does not contain any subclass (Figure 43). The **Generic** class is a class to support otherwise unclassified objects. The **Generic** class is described in Appendix B.

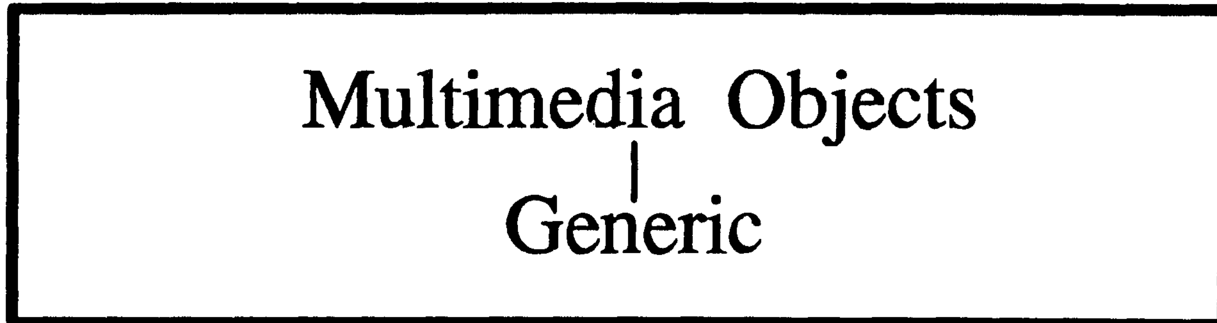


Figure 43. The figure shows the class hierarchy for **Generic**.

All multimedia objects, except for videodisc objects, share one thing in common: they are collections of files. Even videodisc objects become collections of files once they are realized by the Automatic Folder Creator. This is a property that has been inherited from the superclass called ***Multimedia Objects***. The classification of the multimedia object types is a way of typing those collections of files so their properties can be understood and used in their manipulation.

Exception Word, Topic, and Password Objects

Three additional types of objects are stored in the database to support the operation of the multimedia database management system (MMDBMS): the exception word, the topic, and the password. These three types of objects can be added to and removed from the database.

Exception words are words that are removed from titles and index word lists when indexes are built in the database. Exception words provide little or no meaning and can be removed to reduce storage space and search time. Some examples of exception words are THE, A, AN, AS, IS, AM, ARE, and WAS.

Topics are pre-defined domains of information. Some examples of topics are HISTORY, BIOLOGY, MATHEMATICS, POLITICAL SCIENCE, and MUSIC.

Topics must be added to the database before they can be referenced in the multimedia object catalog. The set of topics supported by the MMDBS is determined by the list of topics that has been added to the database. To add a topic to the database, two pieces of information must be included: a topic description and a topic pointer. A topic description is a word or phrase that can be understood by a human. A topic pointer is a short mnemonic string of characters that can be used in the database to refer to the topic description. The topic description is stored once in the database and the topic pointer is stored with each object that references the topic in its catalog.

The database supports the storage of a single password. The password can be used by the user interface software to control access to database operations. How the password is used is determined by the security features implemented in the user interface.

Database System Operations

This section defines database system operations that are used to create a database, establish a connection with the database, manipulate objects in the database, and maintain the database.

Creating an Instance of a Database

Before objects can be added to a database, the database must be created. Creating a database entails establishing a storage location for the database, defining configuration information, and building structures for the storage, retrieval, and maintenance of database objects. The operation to create an instance of a database is called **CreateMMDB**.

Connecting to and Disconnecting from a Database

Before any operations are performed on a database, a connection must be established. To establish a connection, the location of the database must be known. The operation to establish a connection with a database is called **MMDBOpen**. After a connection is no longer needed to the database, the connection must be closed. The operation used to disconnect from a database is called **MMDBClose**. A workstation can only make one connection to a database at a time.

Setting and Getting the Password

A password can be established in the database to enable some form of security defined by the user interface. The password is set using the **SetPassword** operation. To get a password, after one has been set, the **GetPassword** operation is used.

Adding and Removing Exception Words

Exception words are added to the database using the **AddExceptionWord** operation and correspondingly the **RemoveExceptionWord** operation is used to remove an exception word from the database. In addition, an operation called **RemoveAllExceptionWords** is defined to empty the exception word list stored in the database.

Adding, Removing, and Getting a List of Topics

A topic is composed of a topic pointer and a topic definition. The operation **AddTopicPtrDefinition** is used to add a topic to the database and the operation **RemoveTopicPtrDefinition** is used to remove a topic from the database. Removing a topic does not remove multimedia objects that reference the topic; only the topic object is removed. **RemoveAllTopicPtrDefs** removes all topic objects from the database.

After a long period of database activity, there may be many topic objects stored in the database that are not referenced by any multimedia object. It may be desirable to remove unreferenced topics. This is accomplished using the **RemoveUnreferencedTopPtrDefs** operation.

It is useful to know what topics are stored in the database to catalog and find objects. A copy of the current list of topics is obtained using **GetTopicDefinitionList**. When the copy of the list of topics is no longer needed, **DestroyTopicDefinitionList** is used to discard it.

Getting a List of Classes Supported by the MMDBMS

It is useful to know what types of objects can be added to or found in the database. A copy of the list of object types that the MMDBMS supports is obtained using the **GetListOfClasses** operation. When the copy of the list is no longer needed, **DestroyListOfClasses** is used to discard it.

Adding Multimedia Objects to the Database

Adding multimedia objects to the database occurs in stages. First, notification is made to the database that one or more objects is to be added. Next, one or more objects are imported or added to the database. Finally, the add is committed to the database or it is rolled back.

Notification that multimedia objects are going to be added to the database is made with the **NotifyStartOfAdd** operation. Objects can then be imported or added. An object can be imported if required database information is bound with the object. This information includes the referent and catalog information. Objects that have been previously exported from the database can be later imported. To add an object, the user must supply catalog and referent information. An object that is in import format does not

require the user to supply this information. This makes importing easier for a user to do than an add. An object is added using **AddObject** and an object is imported using **ImportObject**.

Once one or more objects have been imported or added, the operations must be committed or rolled back. A commit is used if no problems were encountered while importing or adding objects. A roll back is used to recover from an error during an import or an add. A commit is performed using the **CommitAdd** operation and a roll back is performed using the **RollBack** function.

If for some reason **RollBack** fails, a number of uncommitted objects may remain in the database. These objects must be removed to maintain database integrity. The operation **RemoveObjsWithAddStatus** is used to locate and remove uncommitted objects.

Adding a Videodisc Information File

Videodiscs can contain hundreds, or even thousands, of multimedia objects. To facilitate the easy addition of the objects located on a videodisc to the database, the operation **AddVIF** is used. The object information for videodisc based multimedia objects are defined in a file called a Videodisc Information File (VIF). (VIF file formats are provided in Appendix C.) **AddVIF** reads the object information from the VIF and adds it to the database. **AddVIF** handles all add operations including **NotifyStartOfAdd, AddObject, CommittAdd** (if no errors occurred), and **RollBackAdd** (if there was an error). All objects located in a VIF are added without error or none are added. **AddVIF** must be passed the source location of the VIF.

Searching for Objects

Before a search can be performed, a search must be opened and criteria must be established for the search. A search is opened using the **OpenSearch** operation. The criteria for the search are established using four pieces of information: topics, search words, object types (classes), and statuses. These criteria are added to the search using the following four operations: **AddTopicToSearch**, **AddSearchWordToSearch**, **AddClassToSearch**, and **AddStatusToSearch**. If N topics are to be added to a search, **AddTopicToSearch** must be used N times. The same is true for the other search criteria.

The default maximum numbers of topics, search words, classes, and statuses, that can be added to a search, are changed using the **ChangeSearchDefaults** operation. Increasing the maximum values increases the amount of memory that must be allocated for the search specification. The reverse is true for decreasing the values.

The logic used in the search specification takes the following form: ($topic_1$ OR $topic_2$ ... OR $topic_n$) AND (search $word_1$ AND search $word_2$ ... AND search $word_n$) AND ($class_1$ OR $class_2$ ... OR $class_n$) AND ($status_1$ OR $status_2$ ... OR $status_n$). Adding more topics, more classes, and more statuses causes the search to discriminate less. Adding more search words causes the search to discriminate more. If no topics are added to a search, then all topics are included by default. If no classes are added to a search, then all classes added by default. And, if no statuses are included in a search, then all statuses are included by default. If no search criteria are specified at the time the search is performed, then all objects in the database will be found.

The search logic allows users that have very little or no knowledge of formal logic to perform searches. Specifying a list of topics defines the domain of knowledge to be searched. For example, if a user includes BIOLOGY, CHEMISTRY, and PHYSIOLOGY as topics, then objects that fit any of those categories will be included in the search. If the user does not specify any topics, then the underlying assumption is that

the user doesn't care what the domain of knowledge is and all topics should be included. Specifying a list of classes defines the type of objects that are acceptable. If a user lists LinkWay and Storyboard images in the class list, then s/he is indicating objects of those types should be returned. If no class list is provided then the underlying assumption is that any object type is acceptable. The logic for statuses is the same as for class lists. When a user supplies a list of search words s/he is indicating that all those words must be associated with the returned objects. Specifying KENNEDY as a search word should return any object that has KENNEDY associated with it including JOHN KENNEDY, ROBERT KENNEDY, and KENNEDY ASSASSINATION. Specifying JOHN and KENNEDY as search words should only return JOHN KENNEDY. If no search words are specified, then the underlying assumption is that search words don't matter and should not be used to discriminate in the search.

Once the search criteria have been specified, the actual search can be performed and the search criteria cannot be changed. The search is performed using the operation **ObjectSearch**. **ObjectSearch** returns a portion of the search results on each call; therefore it may have to be called many times to return all the results. The return from **ObjectSearch** is a list of object handles.

After a search is complete, or when a search is to be ended, the operation **CloseSearch** is used to clear the search criteria and to notify the MMDBMS that the search is no longer needed.

Getting Information About an Object

Information the database contains about an object is returned by the operation **GetObjInfo**. The information returned includes the object's referent, catalog information, status information, size in bytes, and, if the object is of type Videodisc

Segment, the videodisc object information. **GetObjInfo** requires the object handle to locate the information.

Experiencing an Object

An object can be experienced using the **ExperienceObject** operation. When **ExperienceObject** is called, the appropriate experience method is selected and passed the object's referent. The object handle is passed to **ExperienceObject** to indicate the object to be experienced.

Getting a Copy of an Object

After an object has been located, a user may want a personal copy of the object. A copy of an object is obtained by using the **GetCopyOfObject** operation. **GetCopyOfObject** requires the object handle and the location where the copy is to be placed to be specified.

Exporting an Object

Exporting an object provides a copy of an object that has been bound with its referent and catalog information. Exported objects can be imported into a database, saving the user from having to catalog the object and specify the referent. An object is exported using the **ExportObject** operation. **ExportObject** requires the object handle and the location to which the object is to be exported. The object handle is not passed with the object information. A new object handle is created when the object is imported.

Updating an Object

After an object is placed in the database, it may need to be updated. **UpdateObjectFiles** updates the object when it is passed the object's handle and the

source location of the files to be used in the update. **UpdateObjectFiles** has two modes of operation. It can completely replace the current set of object files in the database or it can add the object files provided in the source location copying over any old versions of files.

Removing an Object from the Database

An object is removed from the database using the **RemoveObject** operation. **RemoveObject** must be passed the handle of the object to be removed.

Setting Use Locks to Zero

An object cannot be removed from the database if a user is currently using the object (experiencing, copying, or exporting an object). Each user using an object increases the number of use locks placed on an object. Zero use locks indicates that an object is not currently being used. It is possible for the number of use locks to be greater than zero, even if no one is using the object. This can occur if a workstation is turned off before an operation is complete or if there is a software or hardware failure. Use locks can be set to zero using the **ZeroUseLock** operation. **ZeroUseLock** must be provided with the handle of the object for which use locks are to be set to zero.

Operations on Objects of Type Videodisc Segments

The operations provided in this section are associated only with objects of type Videodisc Segments. They support the operation of the Automatic Folder Creator (AFC).

If a user finds an object that is of type Videodisc Segments, and the object has a V status indicating the object has not been digitized by the AFC, a request must be sent to the AFC to digitize the object. The operation used to send the request to the AFC is

called **CreateAFCRequest**. **CreateAFCRequest** requires two pieces of information to create the request: the object handle and the type of object to create.

When the AFC receives the request, it digitizes the appropriate video and audio and submits the resulting object to the database using the operation **RealizeAFCObject**. If **RealizeAFCObject** finds there is insufficient storage space to place the object in the database, it can call **RevertLRUAFC** to revert the least recently used object of type Videodisc Segments from local/available status (L) to videodisc status (V). Since videodisc objects can always be re-digitized, no harm is done in reverting those which have been least recently used. This operation implicitly defines the multimedia database as a cache for objects of type Videodisc Segments. **RevertLRUAFC** uses the function **RevertAFCObjToVirtual** to revert an object, specified by its object handle, from L to V status. Both **RevertLRUAFC** and **RevertAFCObjToVirtual** are available for use external to **RealizeAFCObject**.

Obtaining Error Messages

Errors returned from the MMDBMS are numbers. A character string describing an error, which can be understood by a human, is returned by the **MDErrMsg** operation when it is passed an error number. **MDErrMsg** is used to provide error information, for use by a user interfacc, to communicate an error condition.

Summary

In this chapter a data model was developed for the multimedia database management system (MMDBMS) that defines the objects that can be stored in the database and the operations that can be performed on the objects. Four types of objects were defined: the database object, the exception word object, the topic object, and the password object.

A database object is a multimedia object plus pieces of information that are associated with the multimedia object when it is added to the database. A multimedia object is a collection of files and the associated pieces of information include the object handle, the referent, status information, and catalog information. An exception word object is a word with little or no meaning that is removed from word index lists and titles when indexes are built to speed searches and reduce storage. A topic object is a pre-defined domain of information within which an object can be classified. And, a password object is a character string used to implement database security.

An object handle is a number associated with a multimedia object that provides a reference to an object; object handles are similar to card catalog numbers for books in a library. A referent is a filename or a set of start instructions that the method for experiencing a multimedia object requires to execute. Status information is used to track the state of a database object and includes five pieces of information: a status flag, when the object was added to the database, when the object was last used, the number of times the object was used, and the number of use locks placed on the object. The status flag and the number of use locks value are used to manage concurrency of database use on a network. The status flag is also used to track the state of an object. Catalog information provides a description of an object and includes four pieces of information: the class of the object, a title for the object, a set of topics within which the object is classified, and a set of index words associated with the object. A user searches the catalog to locate objects in the database.

The data model includes a classification of multimedia objects supported by the MMDBMS and is presented as a class hierarchy. The root of the class hierarchy is the class **Multimedia Objects** and the leaves of the class hierarchy are the types of multimedia objects supported by the MMDBMS. The class hierarchy is constructed

using the knowledge that different formats for multimedia objects exist and that multimedia objects have hardware dependencies when they are experienced.

The database system operations defined in the data model are used to create a database, establish a connection with the database, manipulate objects in the database, and maintain the database. Most of the operations for manipulating database objects can be used without regard for the specific class membership of an object on which operations are being performed.

CHAPTER VIII

THE MULTIMEDIA DATABASE MANAGEMENT SYSTEM ARCHITECTURE

A three layer architecture was used to construct the multimedia database management system (MMDBMS). The three layers are the user interface, the communication interface, and the access method. This chapter describes the three layers, their relationship, and their implementation.

The Three Layer Architecture

A three layer architecture provides the following advantages:

- the problem of designing the MMDBMS can be tackled in stages
- details of storage are transparent to application developers
- different types of applications and user interfaces can be designed to access a single database
- changes in one layer do not affect operation in other layers, allowing for enhancements and experimentation without complete re-development

The three layers are the user interface, the communication interface, and the access method (Figure 44). The access method is the process used to access the data that is physically stored on a disk. The communication interface is an application programming interface (API), comprised of C language functions, which enables developers to create user interfaces and applications to access a multimedia database. The user interface is a human's point of contact with the MMDBMS. It provides a place for actions to be

communicated and for results to be observed. Any number of user interfaces can be built on the functions provided by the communication interface.

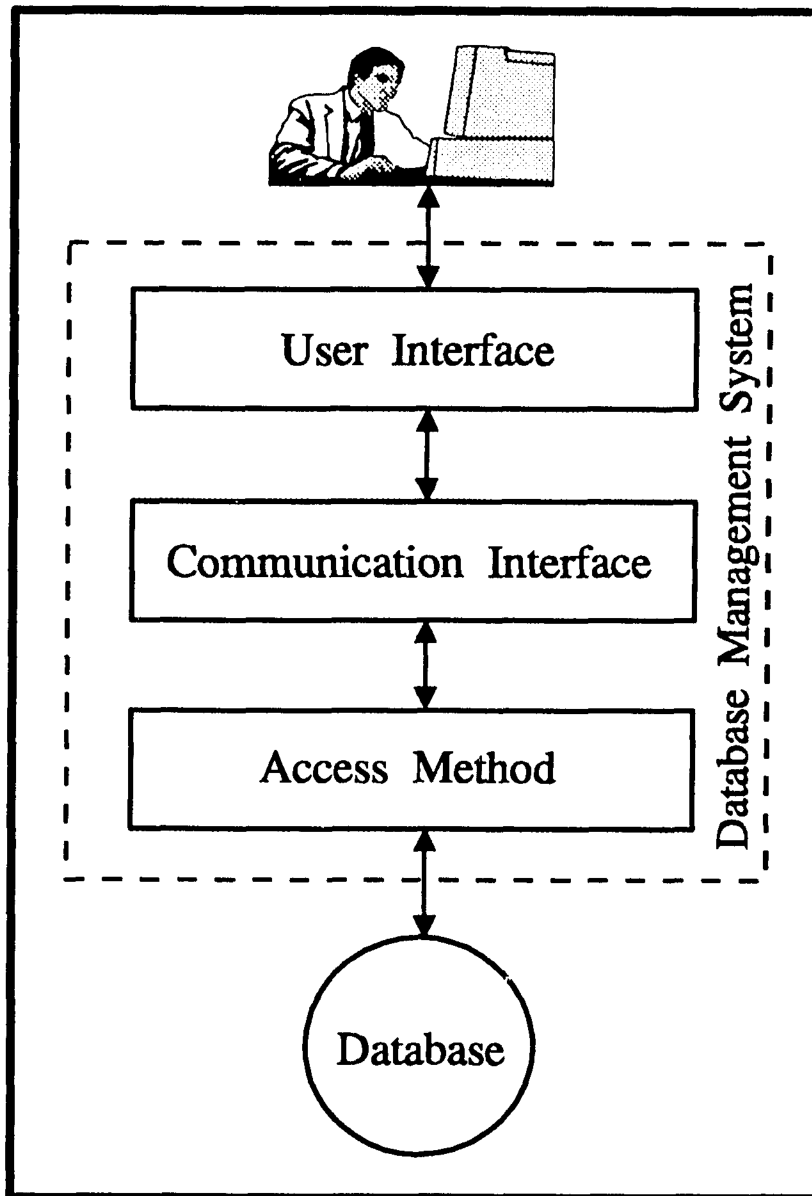


Figure 44. The database management system is structured using three layers called the access method, the communication interface, and the user interface.

The flow of information among the three layers is as follows. The user interface accepts actions from a user and transmits them to the communication interface by calling appropriate functions. The communication interface in turn uses functions provided by the access method to manipulate the database. Results are passed back to the communication interface from the access method, where they are communicated to the user via the user interface. Each layer of code provides a measure of abstraction greater than the layer beneath. The most abstract view of the database is at the user interface level.

The communication interface was pivotal in the design of the MMDBMS. The types of user interfaces and applications that could be built were determined by the operations provided by the communication interface. And, the requirements for the design of the access method were dictated by the operations that were to be implemented in the communication interface. It made sense that the design of the MMDBMS should begin with an understanding of the needs of the user. This understanding lead to a definition of the operations that had to be implemented in the communication interface.

At the same time it made sense that the physical requirements of the objects, to be stored in the database, needed to be studied and understood. The requirements resulting from an analysis of the objects dictated the design of the access method and the connection between the communication interface and the access method.

Considering the user drove a top down development and considering the objects drove a bottom up development. The two types of development met at the communication interface. The communication interface arbitrated between the needs of the user and the requirements of the objects. It was the tension generated between the needs of the user and the requirements of the objects that defined the communication interface.

The Access Method

One part of the code contained in the access method is operating system and network file services which are used to physically manage files placed on a hard disk or a server volume. Multimedia objects are collections of files and an appropriate form of isolated storage for collections of files is a directory on a hard drive or server volume. Each multimedia object can be stored in a separate directory that has the same name as the object handle associated with the object.

A relational record manager forms the second part of the code contained in the access method. A database object is a multimedia object plus associated pieces of information including the object handle, the referent, catalog information, and status information. The pieces of information associated with an object are comprised of character strings and numbers. An appropriate manager of character strings and numbers is a relational record manager. A relational record manager not only stores the information, but also provides the ability to index the information so it can be quickly located.

Together the operating system and network file services and the relational record manager form the access method (Figure 45). Operating system services are provided through DOS functions. Network services are provided through enhancements and additions that are made to the DOS functions when Novell's IPX and NETx drivers are loaded and through functions located in Novell Netware C Interface for DOS. The relational record manager functions are provided by Borland's Paradox Engine.

Figure 46 shows the disk directory structure used for database storage. When an instance of a database is created a location for the database is specified. The location of the database is the base directory. A tree of subdirectories is created beneath the base directory beginning with a directory called MMDB. Beneath the MMDB directory, four subdirectories are created called TABLES, METHODS, OBJECTS, and REQUESTS.

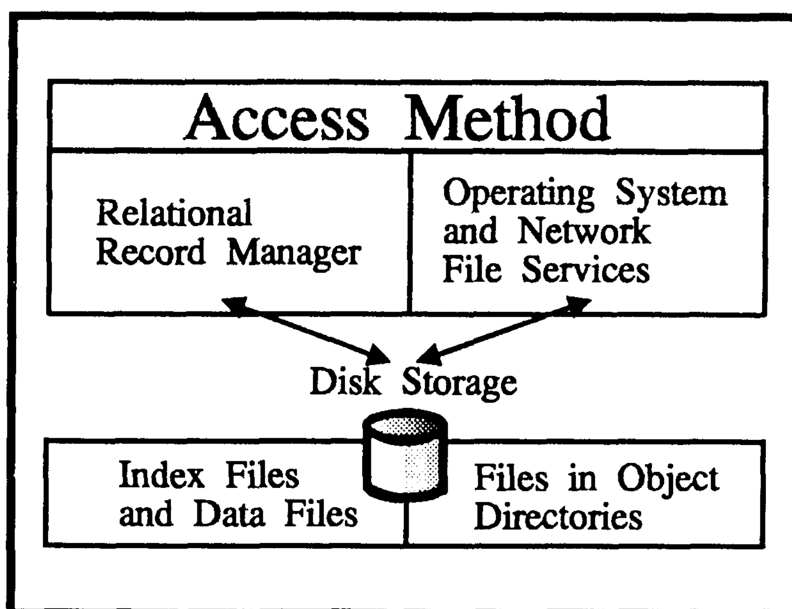


Figure 45. The access method is comprised of a relational record manager and operating system and network file services.

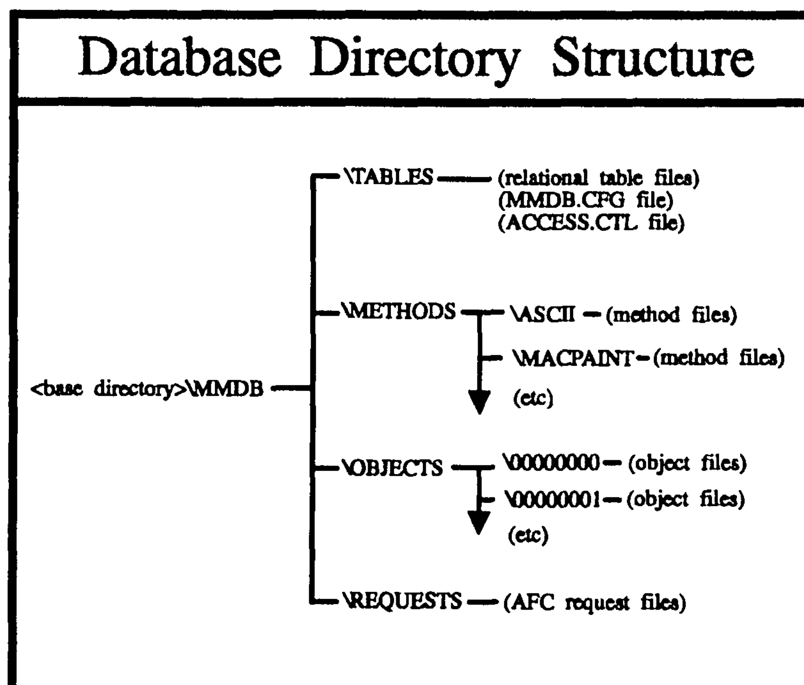


Figure 46. The figure shows the database directory structure.

The TABLES directory contains files generated and used by the relational record manager and two additional files called *MMDB.CFG* and *ACCESS.CTL*. The specifications for the relational tables and associated indices are supplied in Appendix D. MMDB.CFG contains configuration information for the database and ACCESS.CTL contains the password stored in the database. The configuration information contained in MMDB.CFG specifies a minimum level of free disk space that must be maintained by the MMDBMS. Setting a minimum for free disk space ensures that the database does not consume an entire hard disk or server volume.

The METHODS directory holds the experience methods for experiencing different classes of objects. Each experience method, which is a collection of files, is given a separate subdirectory within the METHODS directory.

The OBJECTS directory contains a subdirectory for each object stored in the database. The object handle assigned to the object is used as the subdirectory name for storing that object. When an object is removed from the database, its directory is deleted.

The REQUESTS directory holds requests generated for the Automatic Folder Creator. Each request file is given a unique filename prefix and a filename suffix of VRQ. (AFC request file formats are provided in Appendix A.) The object handle assigned to the object being requested is used as the filename prefix for the request file.

The strategy implemented in the access method places all the database management system processing on the client workstation. This strategy circumvents the need for an additional computer acting as a database server and allows the database management system to function on a non-networked workstation.

The Communication Interface

The communication interface is comprised of a set of C language functions that are constructed using functions available in the access method layer (Figure 47). The functions provided in the communication interface layer are an implementation of the operations described in the data model. The set of C functions form an application programming interface (API), which is used by programmers to develop database applications. (The communications interface function declarations and calling protocols are provided in Appendix E. Function algorithms are provided in Chapter IX.)

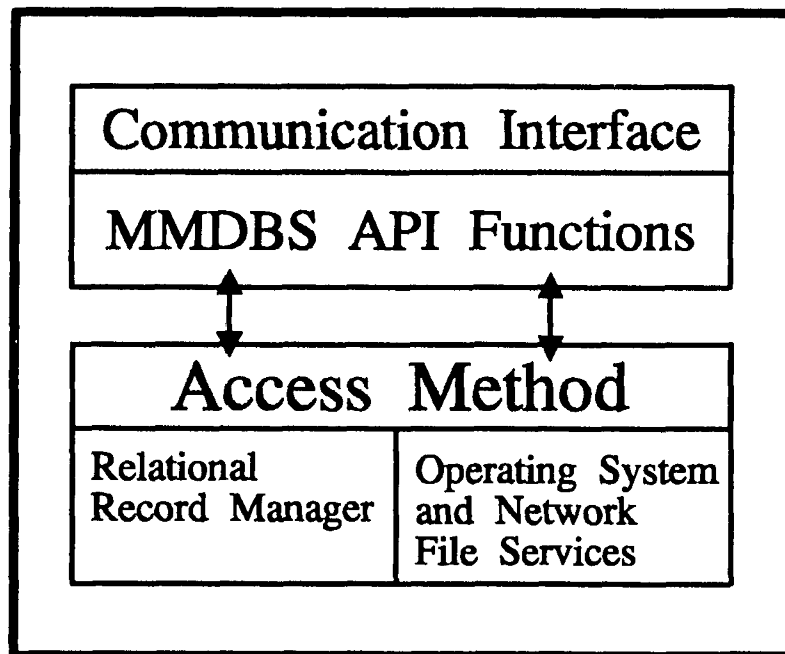


Figure 47. The communication interface is a set of C functions constructed using functions available in the access method.

The following functions are contained in the communication interface:

CreateMMDB - create an instance of a database.

MMDBOpen - open a session with a database.

MMDBClose - close a session with a database.

Communication interface functions (continued):

SetPassword - set the password.

GetPassword - get the password.

AddExceptionWord - add an exception word to the database.

RemoveExceptionWord - remove an exception word from the database.

RemoveAllExceptionWords - remove all exception words from the database.

AddTopicPtrDefinition - add a topic to the database.

RemoveTopicPtrDefinition - remove a topic from the database.

RemoveAllTopicPtrDefs - remove all topics from the database.

RemoveUnreferencedTopPtrDefs - remove unreferenced topics from the database

GetTopicDefinitionList - get a list of topics from the database.

DestroyTopicDefinitionList - free memory for a topic list.

GetListOfClasses - get a list of classes from the database.

DestroyListOfClasses - free memory for the list of classes.

NotifyStartOfAdd - make notification of the start of an add.

AddObject - add an object to the database.

ImportObject - import an object into the database.

CommitAdd - commit the added/imported objects.

RollBack - roll back the added/imported objects.

RemoveObjsWithAddStatus - remove uncommitted objects from the database.

AddVIF - add objects located in a Videodisc Information File (VIF).

OpenSearch - make notification of the start of a search.

AddTopicToSearch - add a topic to the search specification.

AddSearchWordToSearch - add a search word to the search specification.

AddClassToSearch - add a class to the search specification.

AddStatusToSearch - add a status to the search specification.

Communication interface functions (continued):

ChangeSearchDefaults - change the search parameter default values.

ObjectSearch - perform a search for objects according to the search specification.

CloseSearch - make notification that a search is to be terminated.

GetObjInfo - get information about an object.

ExperienceObject - experience an object.

GetCopyOfObject - get a copy of an object located in the database.

ExportObject - export a copy of an object located in the database.

UpdateObjectFiles - update an object's files.

RemoveObject - remove an object from the database.

ZeroUseLock - set the use locks on an object to zero.

CreateAFCRequest - place a message in the AFC request queue.

RealizeAFCObject - add a digitized videodisc object to the database.

RevertLRUAFC - revert the least recently used videodisc object from L to V status.

RevertAFCObjectToVirtual - revert a videodisc object from L to V status.

MDErrMsg - get an error message associated with an error number.

The User Interface

A database application is created when a user interface is implemented on top of the communication interface layer (Figure 48). The user interface uses C functions provided in the communication interface to create some desired functionality for the user. The presence of a database may be made obvious by the user interface design or the presence of the database may be hidden. A sample database application is described in Chapter X.

The user interface provides the most abstract view of the database of any of the three layers in the multimedia database management system (MMDBMS). The goal of a user interface design is to make it as easy and convenient as possible for users of the

application to utilize MMDBMS operations. The design of an optimal user interface depends on criteria such as the type of user and the purpose of the application. No single application serves all purposes for all people.

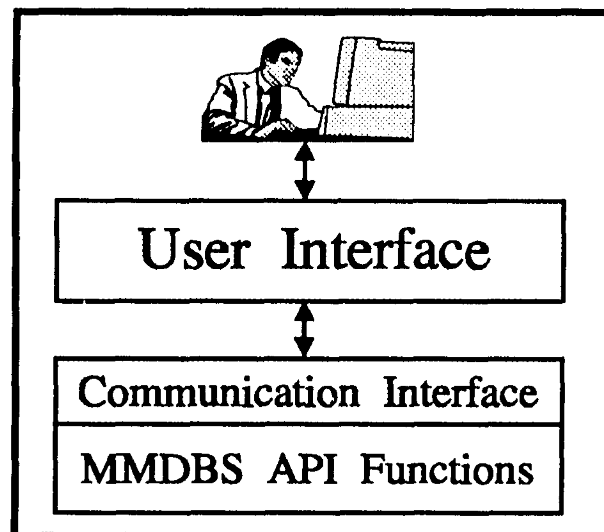


Figure 48. The user interface is built using functions available in the communication interface layer.

The functionality and ease of use of a user interface depend on the functions provided in the communication interface. Functions cannot be implemented in the user interface unless they are describable in terms of the functions provided in the communication interface.

The needs of the user were considered when the communication functions were designed. Two features of the search functions demonstrate this. First, the way search criteria are specified does not require the user to understand formal logic to define a search. Second, the **ObjectSearch** function was designed to be interruptable so that a user is always in control. While a user is browsing "found" objects, the search can continue during slack time.

The **ExperienceObject** function was designed so that the user can experience an object without understanding exactly what type of object it is. A user can see an image, watch a movie, hear a sound, read a text file, or use an application by choosing to experience it. The name of the file, the start instructions, and the object's requirements (software and hardware) do not have to be a concern for the user.

The import and export functions make it easy for users to move objects from one database to another, and to add objects. If objects are in import format, all a user has to do is notify the MMDBMS that an object is to be imported and indicate where the object is located; the MMDBMS takes care of the rest. Objects could easily be distributed to users in import format.

The Automatic Folder Creator (AFC) was designed, along with supporting functions, to make the most use out of limited storage resources, by providing access to a large number of objects, from a wide range of disciplines, without the cost of large amounts of hard disk storage or many videodisc equipped workstations. This access is provided without burdening the user with a knowledge of the location or implementation of the resource. The AFC and supporting functions tightly integrate access to videodisc objects with access to objects of other types.

The needs of another type of user, the application developer, were considered when the communication interface was designed. Functions were designed at a high enough level of abstraction so the application developer is not burdened with the need to use a large number of functions to implement a single feature in the user interface; the functions are not far removed from features that need to be implemented in a user interface. In most cases the only things the user interface has to do is take input from the user in an appropriate way, call a communication layer function, and take the return from the function and respond to the user in an appropriate way. This allows the application

developer to concentrate on an effective user interface design, instead of concentrating on MMDBMS operations.

Summary

The architecture for the multimedia database management system uses three layers: the access method, the communication interface, and the user interface. The access method contains a relational record manager and operating system and network file services. The communication interface is constructed using functions provided in the access method and is a set of C functions that form an application programming interface (API) for the multimedia database management system. The API is used by application developers to create a variety of user interfaces for applications that access the multimedia database.

The access method is designed to place all the MMDBMS processing on the client workstation. This circumvents the need for an additional computer on the network that acts as a multimedia database server and allows the MMDBMS to operate on a non-networked workstation.

The Automatic Folder Creator request queue is a directory within the database directory structure. Request messages for the AFC are in the form of files placed in the AFC request directory.

Each multimedia object stored in the database is placed in a separate directory. The objects are located by searching information associated with the objects located in files maintained by the relational record manager. Methods stored in the database are used to experience objects.

CHAPTER IX

COMMUNICATION INTERFACE FUNCTION ALGORITHMS

This chapter describes the algorithms used in the functions comprising the communication interface layer of the multimedia database management system (MMDBMS). Relational tables referenced in this chapter are described in Appendix D, and function declarations and calling protocols are presented in Appendix E.

Relational table record locking is not included in the presentation of function algorithms; although, table and record locking features built into the relational record manager were used in the design of the communication layer functions to maintain database integrity.

Creating an Instance of a Database

The function **CreateMMDB** creates an instance of a multimedia database (MMDB) at a specified location. Three pieces of information must be passed to **CreateMMDB** to create an instance of a database: the location of the database, the path to the experience methods that are to loaded into the MMDB, and the configuration information.

To create an instance of a database the following operations are performed:

- the database directory structure is created below the database location specified
- on a network, access rights are established for the directories
- the configuration information is placed in the file MMDB.CFG in \MMDB\TABLES
- the relational tables (data and index files) are created in \MMDB\TABLES

• the experience methods are copied to \MMDB\METHODS

• the ACCESS.CTL file is created in \MMDB\TABLES containing a null password

Connecting to and Disconnecting from a Database

The function **MMDBOpen** is used to connect to a database and the function **MMDBClose** is used to disconnect from a database. To connect to a database **MMDBOpen** must be passed the location of the database and a flag must be set to determine whether or not the screen will be restored after an experience method is executed. A workstation can make only one connection to one database at a time.

The following operations are performed to connect to a database:

• on a network, a drive is mapped to the location specified for the database

• configuration information is read from MMDB.CFG

• on a network, the user's ID is obtained

• memory is allocated for the session

• the task swapper is initialized (for experiencing objects)

• the relational tables are opened

The following operations are performed to disconnect from a database:

• the relational tables are closed

• memory is deallocated for the session

Setting and Getting the Password

The password is set using the function **SetPassword** and the password is read using **GetPassword**.

The following operation is performed to store a password in the database:

• the password is written into the file ACCESS.CTL, overwriting any previous password

The following operation is performed to read the password from the database:

• the password is read from the file ACCESS.CTL

Adding and Removing Exception Words

An exception words is added to the database using the function **AddExceptionWord**. Exception words are removed from the database using the function **RemoveExceptionWord** or the function **RemoveAllExceptionWords**.

The following operation is performed to add an exception word to the database:

• the exception word is inserted into the relational table called EXCEPTNS

The following operations are performed to remove an exception word from the database:

• a search is performed for the record in the EXCEPTNS table that holds the exception word to be removed

• the record is deleted

All exception words are removed by the function **RemoveAllExceptionWords** by deleting all records from the EXCEPTNS table.

Adding, Removing, and Getting a List of Topics

A topic is added to the database using the function **AddTopicPtrDefinition** and topics are removed from the database using the function **RemoveTopicPtrDefintion** or the function **RemoveAllTopicPtrDefs**.

The following operation is performed to add a topic to the database:

• a topic is comprised of a topic pointer and a topic definition. Together the pointer and the definition form a record in the TOPICS table. A record is

inserted in the TOPICS table, where the topic pointer is placed in the field TOPICPTR and the topic definition is placed in the field TOPICDEF

The following operations are performed to remove a topic from the database:

- the topic pointer portion of the topic is used to locate the record for the topic in the TOPICS table
- the record is deleted

The function **RemoveAllTopicDefs** deletes all records from the relational table called TOPICS.

The function **GetTopicDefintionList** is used to obtain a list of topics stored in the database. Both the topic pointers and the associated topic definitions are returned. The following operations are performed to obtain a list of topics:

- the number of records stored in the TOPICS table is determined
- memory is allocated to hold the list of topics
- the topics are read from the TOPICS table and placed in the allocated memory

After a topic list is no longer needed, the function **DestroyTopicDefintionList** is used to release the memory that holds the topic list.

Getting a List of Classes Supported by the MMDBMS

In the experimental MMDBMS, the list of supported classes is static. The communication interface code must be recompiled to add a new class. In future versions, this limitation will be overcome. Therefore, a function is provided to query the database for the list of supported classes to maintain continuity with future versions.

The function **GetListOfClasses** is used to obtain a list of supported classes. The following operations are performed to obtain a list of supported classes:

- the number of classes supported by the database is determined
- memory is allocated for the list

• the list of supported classes is placed in the allocated memory. Two pieces of information are associated with each class in the list: an enumeration of the class used in the catalog information for an object and a description of the class which can be understood by a human

After a class list is no longer needed, the function **DestroyClassList** is used to release the memory that holds the class list.

Adding Multimedia Objects to the Database

Adding objects to the database occurs in steps. First, a notification is posted that an add is going to occur. Second, objects are added or imported. Third, the add is committed or rolled back. If a rollback fails, uncommitted objects are left in the database. The function **RemoveObjsWithAddStatus** is used to remove uncommitted objects after a failed rollback.

To make notification that an add is going to occur, the function **NotifyStartOfAdd** is called. The following operations are performed in **NotifyStartOfAdd**:

• the number of objects added to the database is initialized to zero

• a flag is set to indicate an add has started

The function **AddObject** is used to add an object to the database, and the function **ImportObject** is used to import an object into the database. To add or import an object the following operations are performed:

• the amount of free disk space is determined

• the amount of required disk space for the object's files is determined

• if the amount of disk space required for the object's files plus the minimum free disk space set in the database configuration is more than the amount of free disk space, the function **RevertLRUAFC** is called repeatedly to attempt to create enough free disk space. If enough free disk space can be made available, the

following operations are performed; otherwise, the object is not added and an error is generated

• if the object is being added using **AddObject**, the object information is obtained from a variable passed to the function. If the object is being imported using **ImportObject**, the object information is obtained from the file named OBJECTIN.FO which is located with the object files

• a new handle for the object is obtained from the relational table named NEXTID. An incremented object handle is placed in NEXTID for the next add operation

• object information is inserted into the relational table named OBJINFO. The status field is set to + to indicate the object has not yet been committed to the database

• the list of topic pointers associated with the object is inserted into the TOPICS table

• the list of index words associated with the object is inserted into the WORDLIST table

• the title associated with the object is inserted into the TITLES table

• for each topic pointer, a record is inserted into the TOPCINDX table

• for each word in the list of index words that is not an exception word, and for each word in the title that is not an exception word, a record is inserted into the WORDINDX table

• the referent associated with the object is inserted into the REFERENT table

• if the object is a videodisc object, a record is inserted into the VDINFO table

• if the object is an SFL type videodisc object, a record is inserted into the FRAMELST table

• a directory is created in \MMDB\OBJECTS that has the same name as the object handle

• the object files are copied from the source directory into the object directory in \MMDB\OBJECTS

• the object handle is appended to the file OBHANDLE.LST located in the users workspace

• the number of added objects is incremented

The **CommitAdd** function is used to commit added or imported objects to the database. The following operations are performed to commit added or imported objects:

• for each object located in the file OBHANDLE.LST, the following steps are taken

• an object handle is read from the file OBHANDLE.LST

• the object handle is used to search for a record in the OBJINFO table

• the STATUS field of the found record is changed from + to V (for videodisc objects) or L (for all other objects)

The **RollBack** function is used to remove uncommitted objects from the database. The following operations are performed to roll back added or imported objects:

• for each object located in the file OBHANDLE.LST, the following steps are taken

• an object handle is read from the file OBHANDLE.LST

• the object, or any portion that made it into the database, referenced by the object handle is removed from the database using the function named **RemoveObject**

The following steps are performed to remove objects with an add (+) status after a failed rollback:

• a search is performed using the STATUS field in the OBJINFO table for +

• the object handle is read from the ID field

• the object, or any portion that made it into the database, referenced by the object handle is removed from the database using the function **RemoveObject**

• the steps listed above are repeated until no more objects with + status are found

Adding a Videodisc Information File

Videodisc objects can be added from a Videodisc Information File using the function named **AddVIF**. (Videodisc Information File formats are provided in Appendix C.) The following operations are performed to add videodisc objects from a VIF:

• an add is started using the function **NotifyStartOfAdd**

• object information for a videodisc object is read from the VIF

• the object is added to the database using the function **AddObject**

• objects are read from the VIF and added until there are no more objects remaining in the VIF

• if no errors occurred while adding objects, the objects are committed to the database using the function **CommitAdd**

• if an error occurred while adding objects, the objects are rolled back using the function **RollBack**

Searching for Objects

Searching for objects occurs in stages. First, a search is opened using the function **OpenSearch**. Second, search criteria are established using the functions **AddTopicToSearch**, **AddSearchWordToSearch**, **AddClassToSearch**, and **AddStatusToSearch**. The number of topics, search words, classes, and statuses that may be added to the search criteria specification can be changed using the function **ChangeSearchDefaults**. The actual search is performed by calling **ObjectSearch**

repeatedly, until all objects meeting the search criteria have been found. To terminate a search the function **CloseSearch** is used.

The following operation is performed to change the search default values:

- the new values are stored in variables that determine the number of topics, search words, classes, and statuses that may be placed in the search criteria specification

The following operations are performed to open a search:

- memory is allocated for the search criteria specification
- a flag is set to indicate that search criteria may be entered

The following operation is performed to enter the search criteria:

- the topic, search word, class, or status is placed in the memory allocated for the search criteria

The **ObjectSearch** function returns object handles of objects that fit the search criteria. The number of handles that are returned on each call to **ObjectSearch** is determined by the size of the buffer passed to **ObjectSearch** and the number of requested objects. **ObjectSearch** is called multiple times to obtain all the objects that fit the search criteria.

ObjectSearch is designed so it can be interrupted by user actions. Mouse clicks and key presses will cause **ObjectSearch** to return the objects that have been found before the interruption. Using the interruption capability of **ObjectSearch**, a search can continue during the slack time between user input events.

The **ObjectSearch** function uses an algorithm based on sources and filters. One search criterion is used as a source of objects, and the remaining criteria are used as filters to remove unwanted objects from the source. Which criterion is used as the source is dependent on the types of criteria entered into the search specification. If the search criteria include topics, then topics are the source and the search words, the classes, and

the statuses are filters. If the search criteria do not include topics, but do include search words, then the first search word is the source and the remaining search words, classes, and statuses are filters. If the search criteria do not include topics or search words, but do include classes, then classes are the source and the statuses are the filter. If no search criteria are specified, all the objects handles are returned from the database.

The source/filter algorithm works in the following way. A search is performed in the appropriate table for the first match with the criterion specified as the source. The first object handle associated with the source criterion is then "passed through" filters based on the other criteria; information associated with the object handle is read from the relational tables and compared to the filter criteria. If the information fits the search specification, the object handle is returned as a found object, otherwise the object handle is discarded. Then, a search is performed to locate the second object handle associated with the source criterion. This object handle is passed through the filters and is kept or discarded depending on whether or not it matches the filter criteria. This process is repeated until there are no more objects associated with the source criterion.

An object handle passes through a search word filter if all the search words in the filter are associated with the object handle. A match with all the specified search words is required because a logical AND is specified between search words. An object handle passes through a class filter if at least one of the classes in the search criteria is associated with the object handle. Only one match is required because a logical OR is specified between classes. An object handle passes through a status filter if at least one of the statuses in the search criteria is associated with the object handle. Only one match is required because a logical OR is specified between statuses.

The following operations are used to perform a search:

- on the first call to **ObjectSearch** a flag is set to prohibit adding additional search criteria

- a search is performed for an object handle which has information associated with it that matches the source criterion
- if no object handles can be found, **ObjectSearch** returns any previously found object handles and sets the search status to indicate the search is complete
- the object handle is "passed through" the filters
- if the object handle passes through all the filters, it is stored in the object handle buffer returned by the function; and, the number of objects found on this call to **ObjectSearch** is incremented by one. If the object handle does not pass through the filters, it is discarded
- the number of found objects is compared to the number of objects requested on this call to **ObjectSearch**. If number found equals number requested, the found object handles are returned and the search status is set to show that the search is not complete
- the keyboard buffer and the mouse buffer are checked for user input. If user input is in either buffer, **ObjectSearch** returns the found object handles, sets the search status to show the search is not complete, and sets a flag to indicate that **ObjectSearch** was interrupted by a key press or a mouse click
- if the number of found objects does not equal the number requested and if the a key press or a mouse click has not interrupted the search, the steps listed above are repeated

To end a search, the function **CloseSearch** is called. The following operations are performed to end a search:

- the memory allocated for the search criteria specification is released
- a flag is set to indicate no search is in progress

Getting Information About an Object

The function **GetObjInfo** is used to get information about an object. The pieces of information returned by **GetObjInfo** can be specified. The following operations are performed to get information about an object:

- the object handle for the object is used to search for records pertaining to the object in the relational tables
- the requested pieces of information are read from the records and are placed in a buffer
- if the size of the multimedia object is requested, DOS functions are used to determine the total amount of disk space taken by the files in the object's directory in \MMDB\OBJECTS. Then, the size information is placed in the information buffer

Experiencing an Object

The function **ExperienceObject** is used to experience an object. To make enough free memory available to execute the experience method or application, the database application must be removed from memory. An application programming interface (API) called Hold Everything, from South Mountain Software, is used to simplify the process of removing the database application from memory, executing the experience method or application, and reinstating the database application. The Hold Everything API contains a function called **holdev** which, when passed a DOS command line, removes the current application, executes the DOS command line, and reinstates the original application following the execution of the external program.

The following operations are performed to experience an object:

- a use lock is placed on the object to be experienced by incrementing the value in the USELOCKS field in the OBJINFO table

- the class of the object to be experienced is read from the OBJINFO table
- the referent for the object to be experienced is read from the REFERENT table
- based on the class of the object, the appropriate experience method is selected
- if the experience method requires it, a batch file is created in the user's temporary space
- a DOS command line is constructed to execute the experience method. The referent is passed to the experience method on the command line
- the function **holdev** is passed the DOS command line
- after the experience method has completed execution, the function **holdev** automatically restores the database application and screen
- the use lock is removed from the object by decrementing the value in the USELOCKS field in the OBJINFO table
- the value in NUMUSES field in the OBJINFO table is incremented
- the LASTUSE field in the OBJINFO table is set to the time and date of the use

Getting a Copy of an Object

The function **GetCopyOfObject** is used to make a copy of a multimedia object in a specified directory. The following operations are performed to get a copy of an object:

- a use lock is placed on the object to be copied by incrementing the value in the USELOCKS field in the OBJINFO table
- the files in the object directory in \MMDB\OBJECTS are copied to the destination directory
- the use lock is removed from the object by decrementing the value in the USELOCKS field in the OBJINFO table
- the value in the NUMUSES field in the OBJINFO table is incremented
- the LASTUSE field in the OBJINFO table is set to the time and date of the use

Exporting an Object

The function **ExportObject** is used to export a copy of a multimedia object to a specified directory. The following operations are performed to export a copy of an object:

- a use lock is placed on the object to be exported by incrementing the value in the USELOCKS field in the OBJINFO table
- the files in the object directory in \MMDB\OBJECTS are copied to the destination directory
- object information is read from the appropriate relational tables and is recorded in a file called OBJECTIN.FO in the destination directory
- the use lock is removed from the object by decrementing the value in the USELOCKS field in the OBJINFO table
- the value in the NUMUSES field in the OBJINFO table is incremented
- the LASTUSE field in the OBJINFO table is set to the time and date of the use

Updating an Object

The function **UpdateObjectFiles** is used to update the files in the object's directory in \MMDB\OBJECTS. Two options are provided: completely replace the current set of files (REPLACE) and copy the new files into the object directory overwriting any current files of the same name (UPDATE). The following operations are performed to update an object's files:

- the status of the object is changed to update by writing U in the STATUS field in the OBJINFO table
- if the REPLACE option is used, the files in the object's directory in \MMDB\OBJECTS are erased

- the files in the source directory are copied to the object's directory in \MMDB\OBJECTS

• the status of the object is changed back to its previous value by writing the value into the STATUS field in the OBJINFO table

Remove an Object from the Database

An object is removed from a database using the function **RemoveObject**. The following operations are performed to remove an object:

- the status of the object is changed to delete by writing the value D into the STATUS field in the OBJINFO table
- the object files are erased from the object's directory in \MMDB\OBJECTS
- the object's directory is removed from \MMDB\OBJECTS
- the records in the relational tables that are associated with the object are removed. The OBJINFO table entry is removed last

Setting Use Locks to Zero

The use locks applied to an object are set to zero using the function **ZeroUseLock**. The following operations are performed to set the use locks applied to an object to zero:

- the object handle is used to locate the object's record in the OBJINFO table
- a zero is written into the USELOCKS field in the OBJINFO table

Operations on Objects of Type Videodisc Segments

The function **CreateAFCRequest** is used to send a message to the AFC request queue to digitize a videodisc object. The following operations are used in **CreateAFCRequest**:

- the object handle is used to locate the object's record in the table named VDINFO
- if the object is an SFL type videodisc object, the object handle is used to locate the object's record in the FRAMELST table
- the object information read from the tables VDINFO and FRAMELST are used to create an AFC request file in \MMDB\REQUESTS. The request file has a filename prefix that is the same as the object's handle and the filename suffix is VRQ

The function **RevertAFCObjectToVirtual** is used to revert a videodisc object from a local/available (L) status to an on videodisc status (V). The following operations are used to revert a videodisc object:

- the object's handle is used to locate the object's record in the relational table named OBJINFO
- the status of the object is changed to revert by writing the value R into the STATUS field in the OBJINFO table
- the object's files are erased from the object's directory in \MMDB\OBJECTS
- the object's directory is removed from \MMDB\OBJECTS
- the status of the object is changed to "on videodisc" by writing the value V into the STATUS field in the OBJINFO table

The function **RevertLRUAFC** is used to revert the least recently used (LRU) videodisc object from a local/available (L) status to an on videodisc status (V). The following operations are performed to revert the LRU videodisc object:

- a search is performed, using the STATUS field and the LASTUSE field in the OBJINFO table, to locate the least recently used videodisc object with a status of local/available

- the function **RevertAFCObjectToVirtual** is used to revert the located LRU videodisc object

The function **RealizeAFCObject** is used to place the result of an AFC digitization of a videodisc object in the database. The following operations are performed to realize a videodisc object:

- the amount of free disk space is determined
- the amount of required disk space for the object's files is determined
- if the amount of disk space required for the object's files plus the minimum free disk space set in the database configuration is more than the amount of free disk space, the function **RevertLRUAFC** is called repeatedly to attempt to create enough free disk space. If enough free disk space can be made available, the following operations are performed; otherwise, the object's files are not added and an error is generated
- a directory is created in \MMDB\OBJECTS with the same name as the object's handle
- the object's files are copied to the object's directory in \MMDB\OBJECTS
- the status is updated to local/available by placing L in the STATUS field of the OBJINFO table

Obtaining Error Messages

The communication layer functions generate errors that are numbers. These numbers are translated to text strings, which can be understood by humans, using the function **MDErrMsg**. **MDErrMsg** returns the appropriate error string by selecting it from an error table located in memory.

CHAPTER X

THE USER INTERFACE: A SAMPLE APPLICATION

This chapter describes a multimedia database application called the Multimedia Archive Explorer (MAX). MAX is used with IBM LinkWay as a development tool, or without LinkWay as an exploration, research, and presentation tool. IBM LinkWay is an authoring system for creating multimedia applications and objects. MAX used with LinkWay provides the ability to retrieve objects from a database into the development environment and to store in a database applications and objects that were created in the development environment. The multimedia database management system (MMDBMS) application programming interface (API) along with functions from a C window and menu library were used to develop MAX.

Starting MAX

MAX can be started in two ways. The first way to start MAX is to select it from a menu in the IBM Instructional Classroom LAN Administration System (ICLAS) (Figure 49). When MAX is started from ICLAS, it is probably being used as an exploration, research, or presentation tool. The second way to start MAX is to select it from the LinkWay Tools menu (Figure 50). When MAX is started from within LinkWay, it is probably being used as a development tool. Although the functionality of MAX is the same whether it is started from ICLAS or from LinkWay, the context of its use determines the purpose for which it is being used.

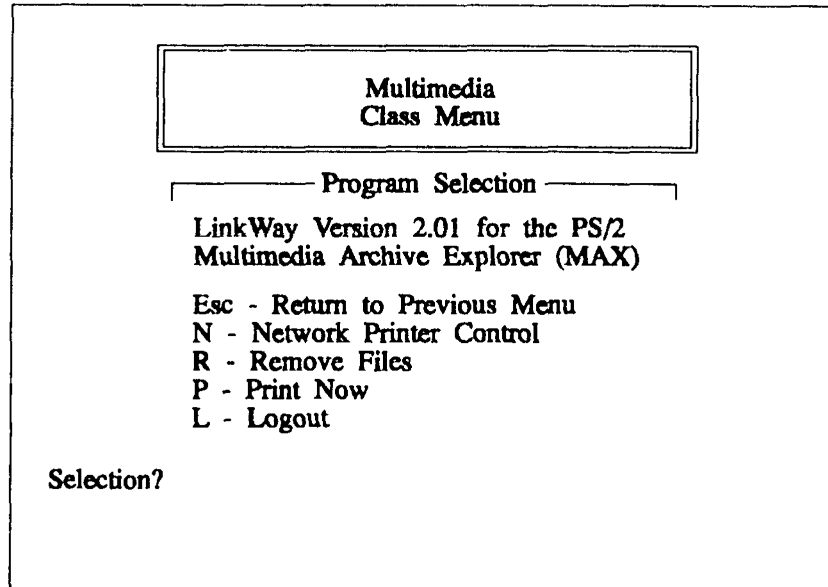


Figure 49. The Multimedia Archive Explorer (MAX) can be started from the Instructional Classroom LAN Administration System (ICLAS) menu.

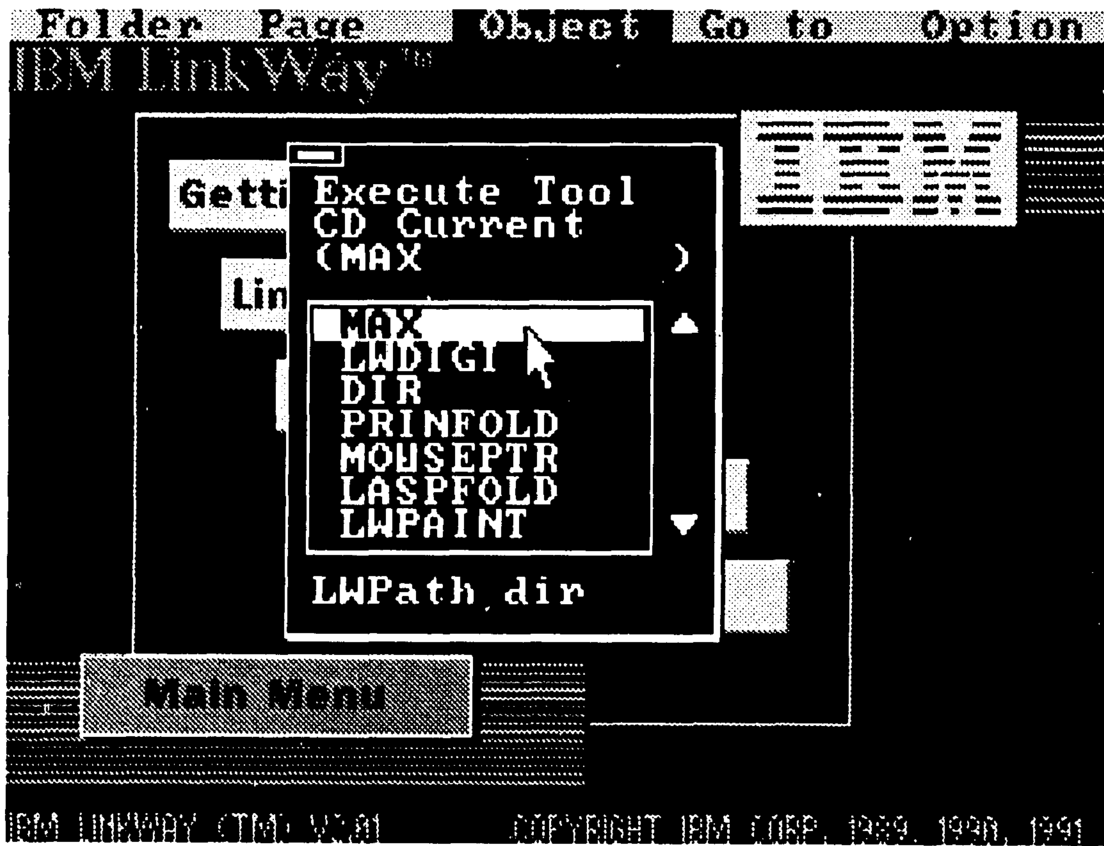


Figure 50. The Multimedia Archive Explorer (MAX) can be started from the LinkWay Tools menu.

MAX Main Functions

MAX uses the assumption that there are two general classes of users called public users and administrative users. Public users perform searches, directly access objects, experience objects, get copies of objects, and export objects. A public user cannot add or delete objects, and cannot access database maintenance functions.

When MAX is first started the main MAX menu is displayed in public mode, which allows the user to quit, to search for objects, to directly access objects, and to change to the administrative mode (Figure 51). Access to the administrative mode can be restricted, via a password, or can be available to all users who choose it. When access is

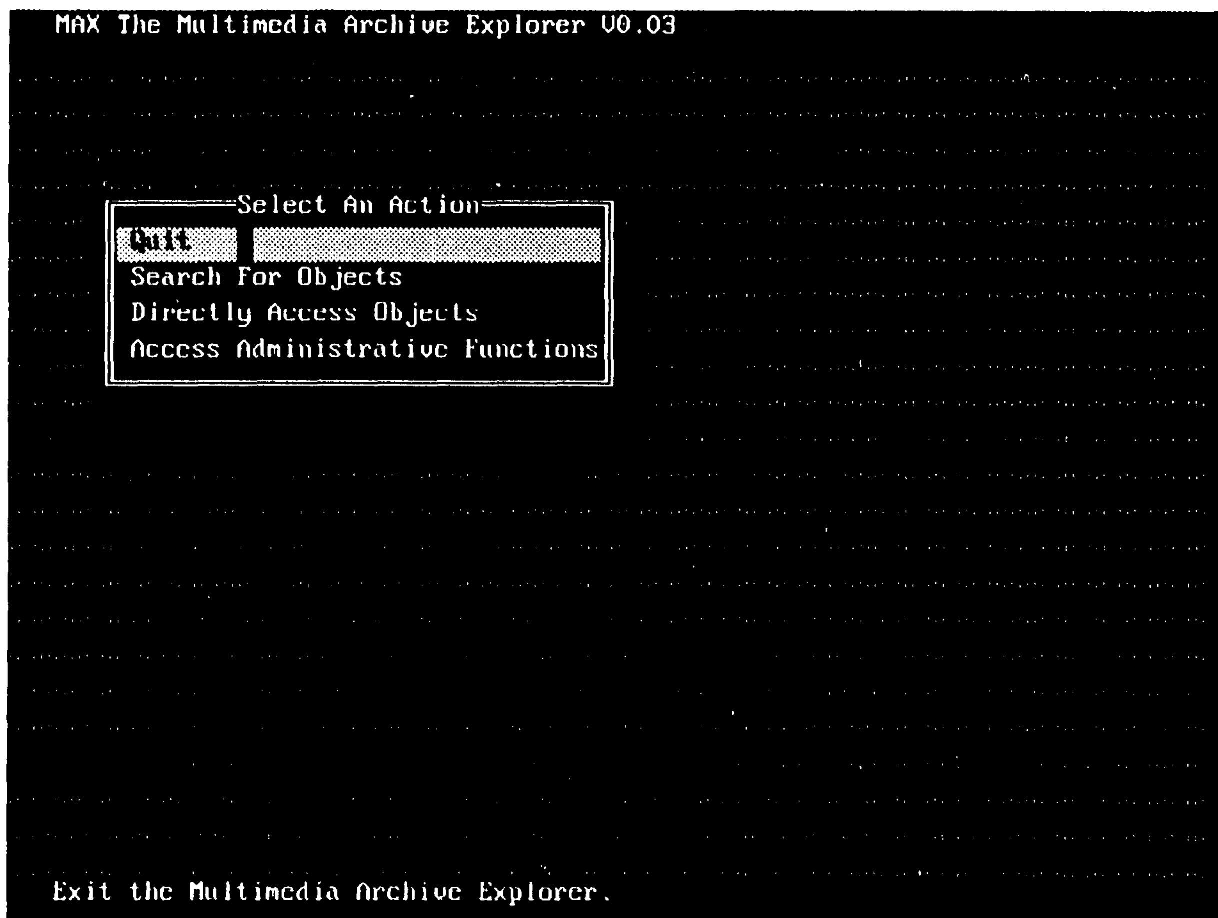


Figure 51. The figure shows the main MAX menu for a public user.

changed to the administrative mode, the main menu contains the public mode options plus a set of maintenance and add options (Figure 52).

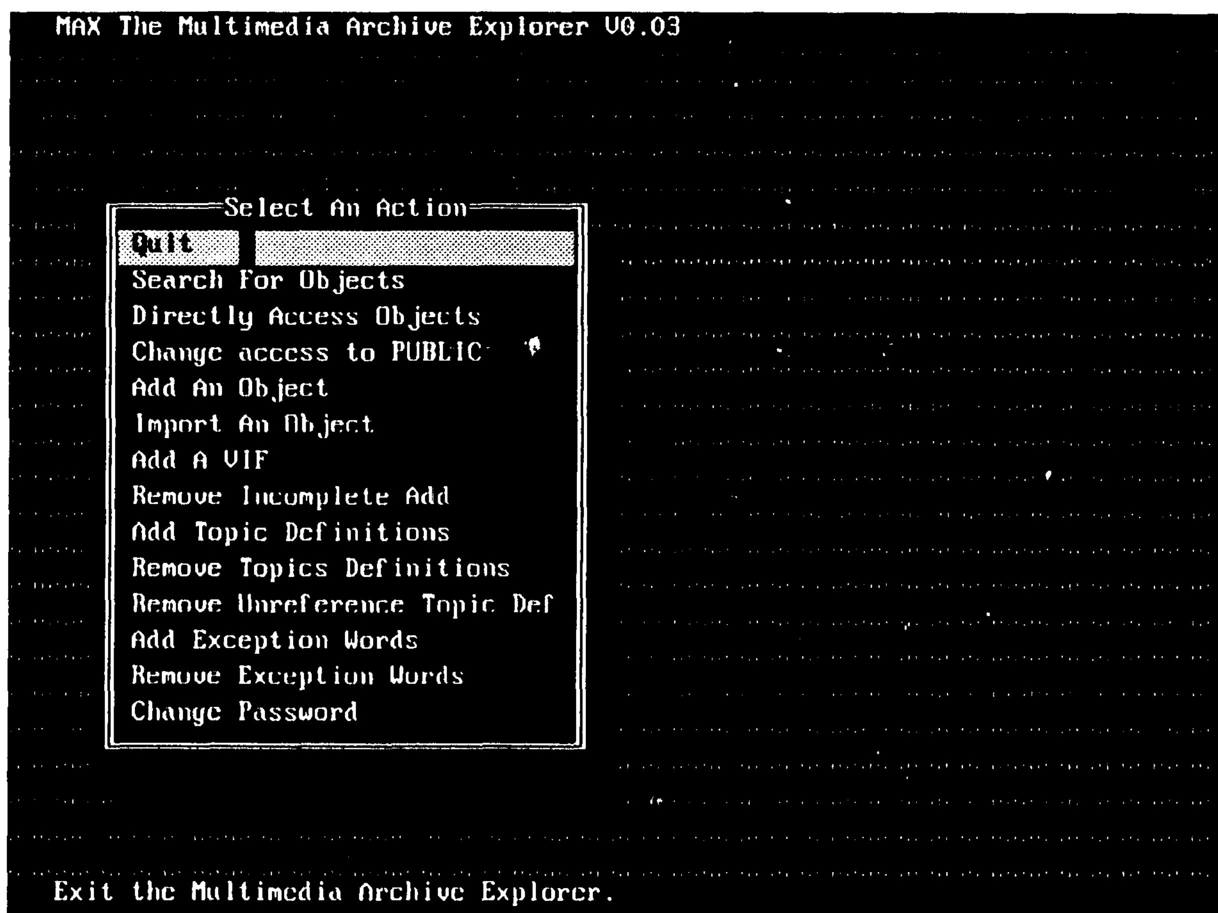


Figure 52. The figure shows the main MAX menu for an administrative user.

The first option on either the public mode or administrative mode main menus is **Quit**. **Quit** halts the operation of MAX, and returns the user to LinkWay, if MAX is started from LinkWay, or to ICLAS, if MAX is started from ICLAS.

The second and third options, on both public and administrative mode main menus, are used to gain access to objects stored in the multimedia database. The second option, **Search for Objects**, allows the user to enter search criteria to locate one or more objects in the database. The third option, **Directly Access Objects**, allows the user to enter an object handle (object ID) to gain direct access to an object stored in the database.

The fourth option on the public and administrative main menus are used to switch between modes of operation. On the public main menu, the fourth option changes to the administrative mode, and on the administrative main menu, the fourth option changes to the public mode. Going from public to administrative mode may require a password, but going from administrative to public mode does not require a password. The password is set, and changed, using the last option on the main administrative menu called **Change Password**.

The fifth, sixth, and seventh options on the administrative main menus are used to add objects to the database. The fifth option, **Add An Object**, will take a multimedia object from a specified source directory and add it to the database. The user is prompted for catalog, class, and referent information when adding an object. The sixth option, **Import An Object**, will take a multimedia object from a specified source directory, which is in import format, and add it to the database. When an object is imported, catalog, class, and referent information do not have to be supplied by the user because they are bound with the object. The seventh option, **Add A VIF**, is used to add videodisc objects located in a Videodisc Information File (VIF). The user is prompted for the location of the VIF when **Add A VIF** is selected.

Remove An Incomplete Add, the eighth option on the administrative main menu, is used to recover from an add operation that encountered an error and could not perform a roll back. The user does not have to supply any additional information when this option is selected.

The ninth, tenth, and eleventh options on the administrative main menu are used to add topics to, and remove topics from, the database. **Add Topic Definitions** adds topics to the database from a topic file. A topic file contains a list of topic definitions and associated topic pointers. **Remove Topic Definitions** removes topics, located in a topic file, from the database. It can also be used to remove all topics from a database.

Remove Unreferenced Topic Def is used to remove topics from the database that are not referenced by any objects stored in the database.

Finally, **Add Exception Words** and **Remove Exception Words**, the twelfth and thirteenth administrative options, are used to add and remove exception words from the database. For both options, the exception words are read from a file supplied by the user that contains a list of words. The file is a standard ASCII text file created with a word processor.

Accessing and Manipulating Objects

One way to access an object is to search for it. The second way is to directly access it by its object handle (object ID). Once an object has been accessed it can be manipulated. The options for manipulating an object are the same whether the object was found through a search or by directly accessing it.

Figure 53 shows the main MAX menu, in public mode, with the **Search For Objects** option highlighted. If the search option is selected, by clicking on a mouse button or hitting the ENTER key, a search options menu is displayed (Figure 54). The search options menu allows search criteria to be specified and a search to be performed. If no search criteria are specified, and **Perform The Search** is selected, all objects in the database will be found.

The scope of a search can be limited by specifying the topics that should be included in the search, by specifying the types of objects to be included in the search, by specifying the statuses of objects to be included in the search, and by specifying search words that are to be associated with objects included in the search. A user may include none, some, or all of these criteria in a search specification.

Figure 53. The main MAX menu is shown with the search option highlighted. Note the description of the option at the bottom of the screen.

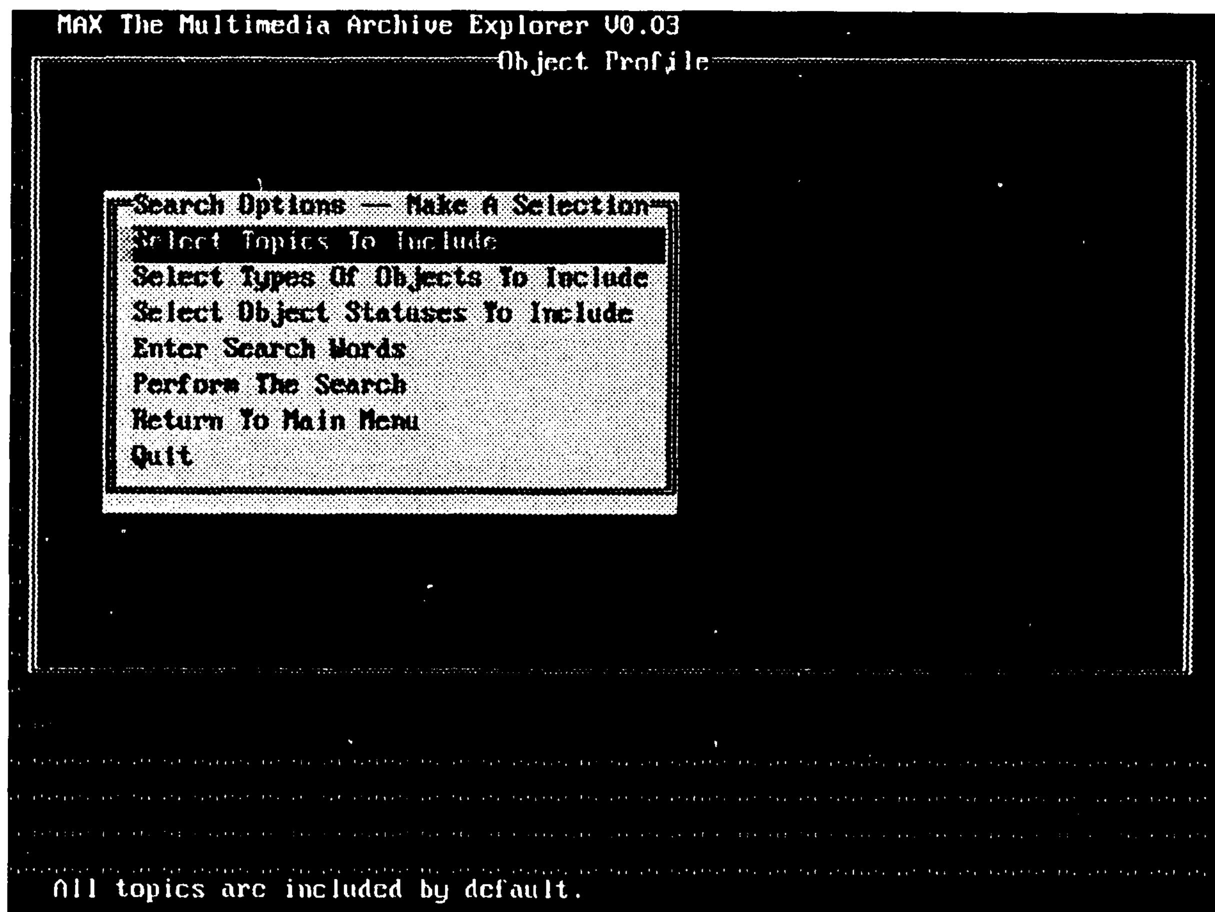


Figure 54. The MAX search options menu is shown. The options are used to set search criteria and to activate the search.

If the user wants to limit the search to a particular set of topics, the **Select Topics To Include** option is used. When **Select Topics To Include** is chosen, a scrollable listbox is displayed that contains a list of topics supported by the database (Figure 55). The user can select one or more topics from the listbox by highlighting the desired entries and clicking the mouse button or pressing the ENTER key. Check marks are placed beside the selected topics.

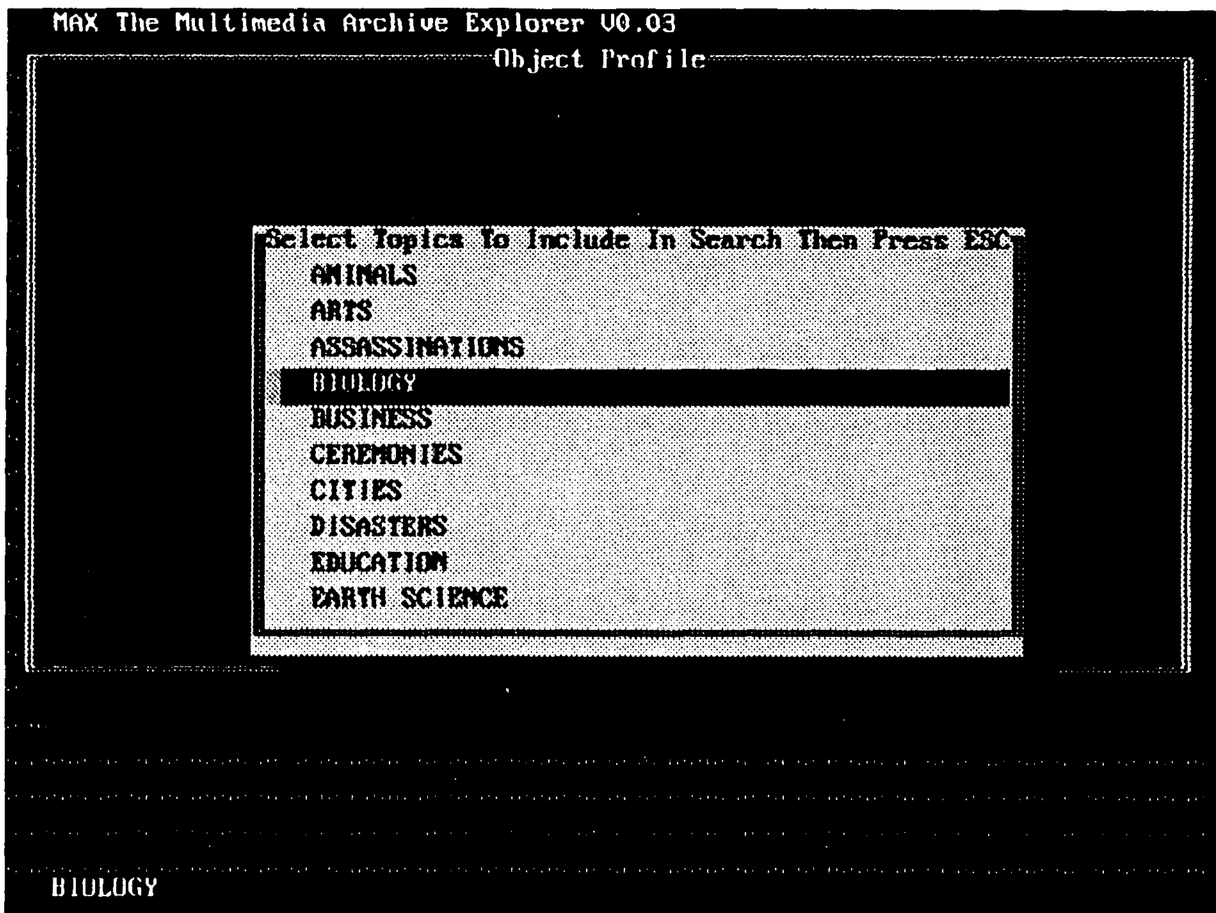


Figure 55. The topic list menu is displayed after choosing to add topics to the search criteria.

If the user wants to limit the types of objects located in the search, the **Select Types Of Objects To Include** option is used. When **Select Types Of Objects To Include** is chosen, a scrollable listbox is displayed which contains a list of object types supported by

the database (Figure 56). The user can select one or more object types from the listbox by highlighting the desired entries and clicking the mouse button or pressing the ENTER key. Check marks are placed beside the selected topics.

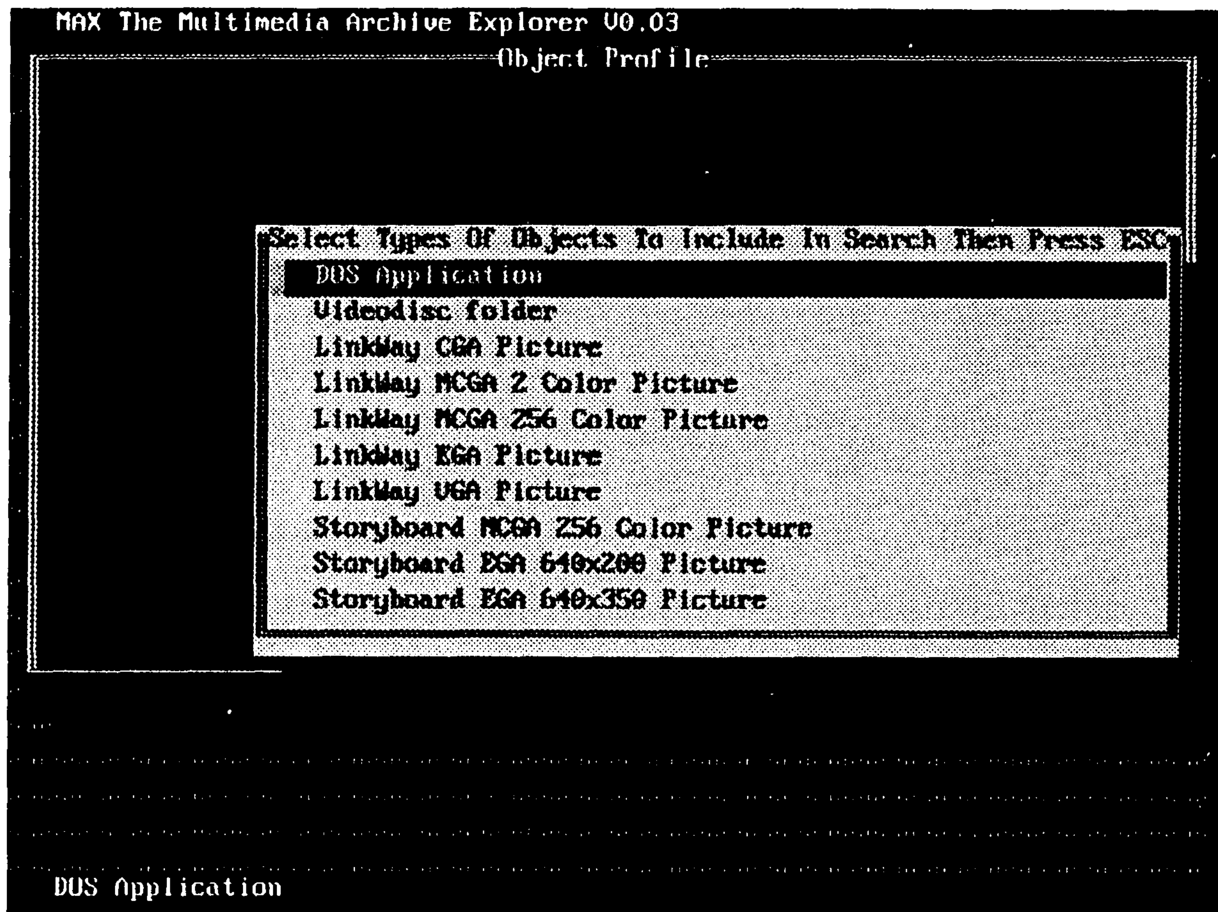


Figure 56. The class list menu is displayed after choosing to add object types to the search criteria. Multiple classes may be selected.

If the user wants to limit the statuses of objects located in the search, the **Select Object Statuses To Include** option is used. When **Select Object Statuses To Include** is chosen, a listbox is displayed which contains a list of possible object statuses. The list displayed in the listbox depends on whether the user is in public mode or administrative mode. Figure 57 shows the list of statuses displayed in public mode and Figure 58 shows the list of statuses displayed in administrative mode. The user can select one or more

statuses from the listbox by highlighting the desired entries and clicking the mouse button or pressing the ENTER key. Check marks are placed beside the selected statuses.

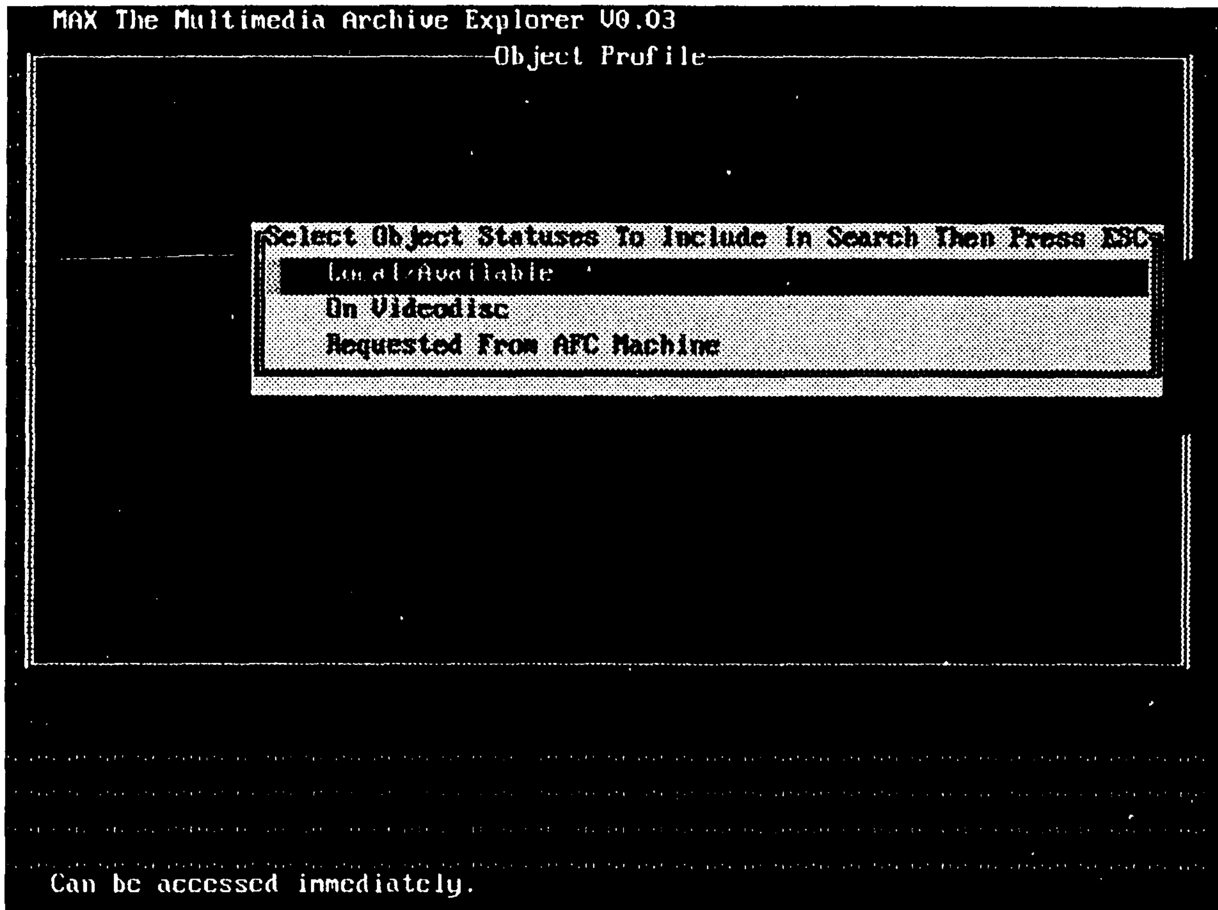


Figure 57. The status list menu is displayed after choosing to add object statuses to the search criteria. Multiple statuses may be selected. The status list shown is for the public mode.

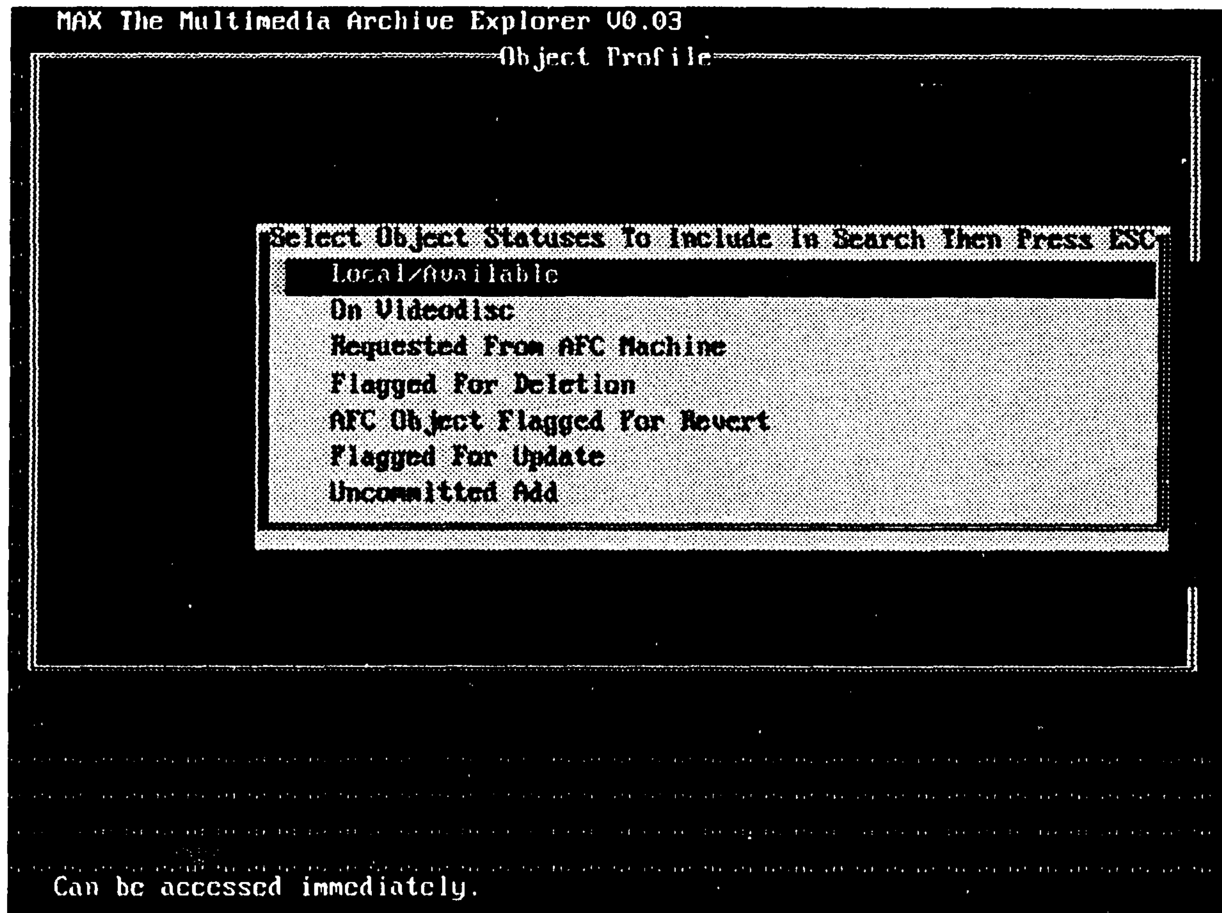


Figure 58. The status list for an administrative user is shown. The status list is longer for an administrative user than for a public user.

Search words are entered by selecting **Enter Search Words**. A prompt is displayed that allows the user to enter one or more words separated by spaces (Figure 59). An object must be associated with all entered search words to be found.

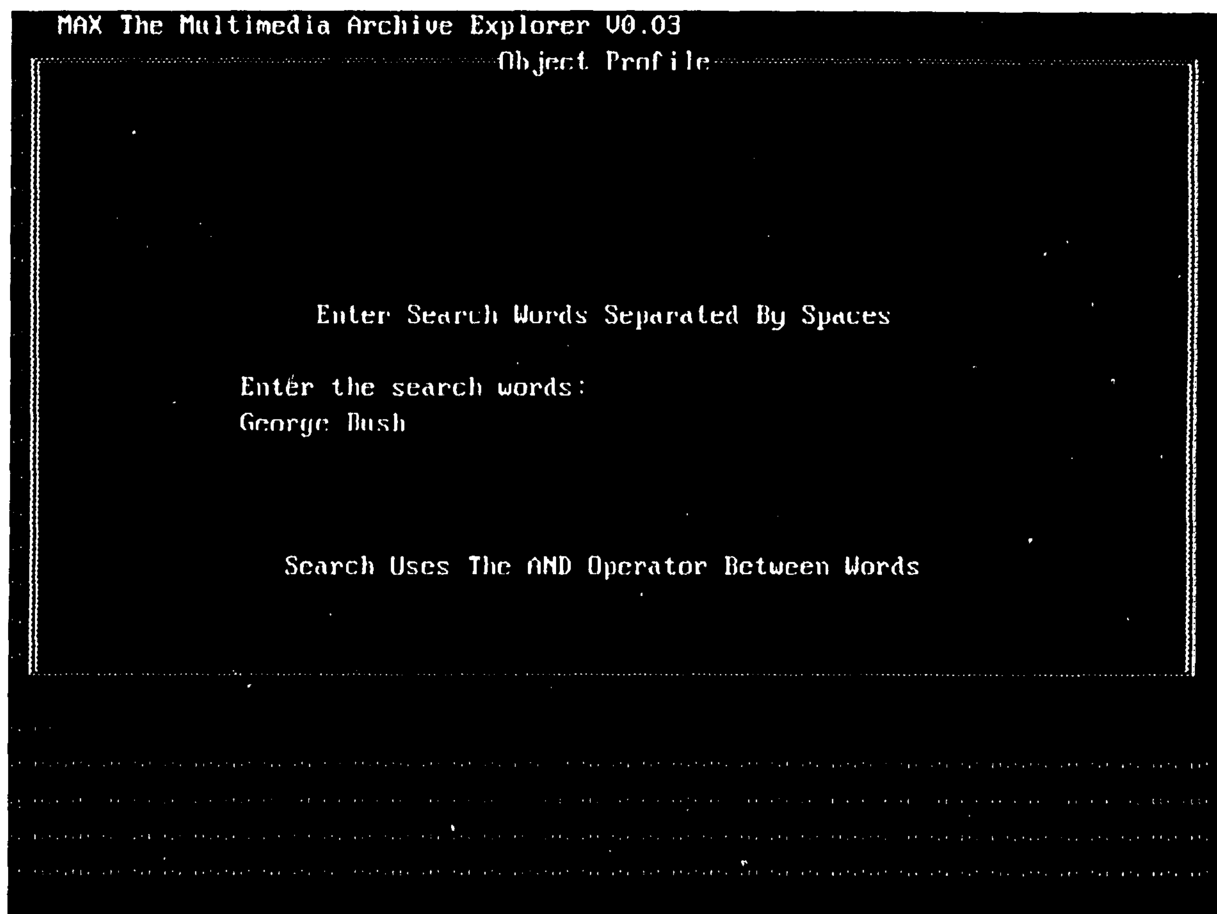


Figure 59. A search word prompt is displayed after choosing to add search words to the search criteria. Multiple search words may be entered.

Once search criteria have been specified, the search is performed by selecting **Perform The Search** (Figure 60). The first object found is displayed in the Object Profile window for the user to see while the computer continues to search for other objects that match the search criteria.

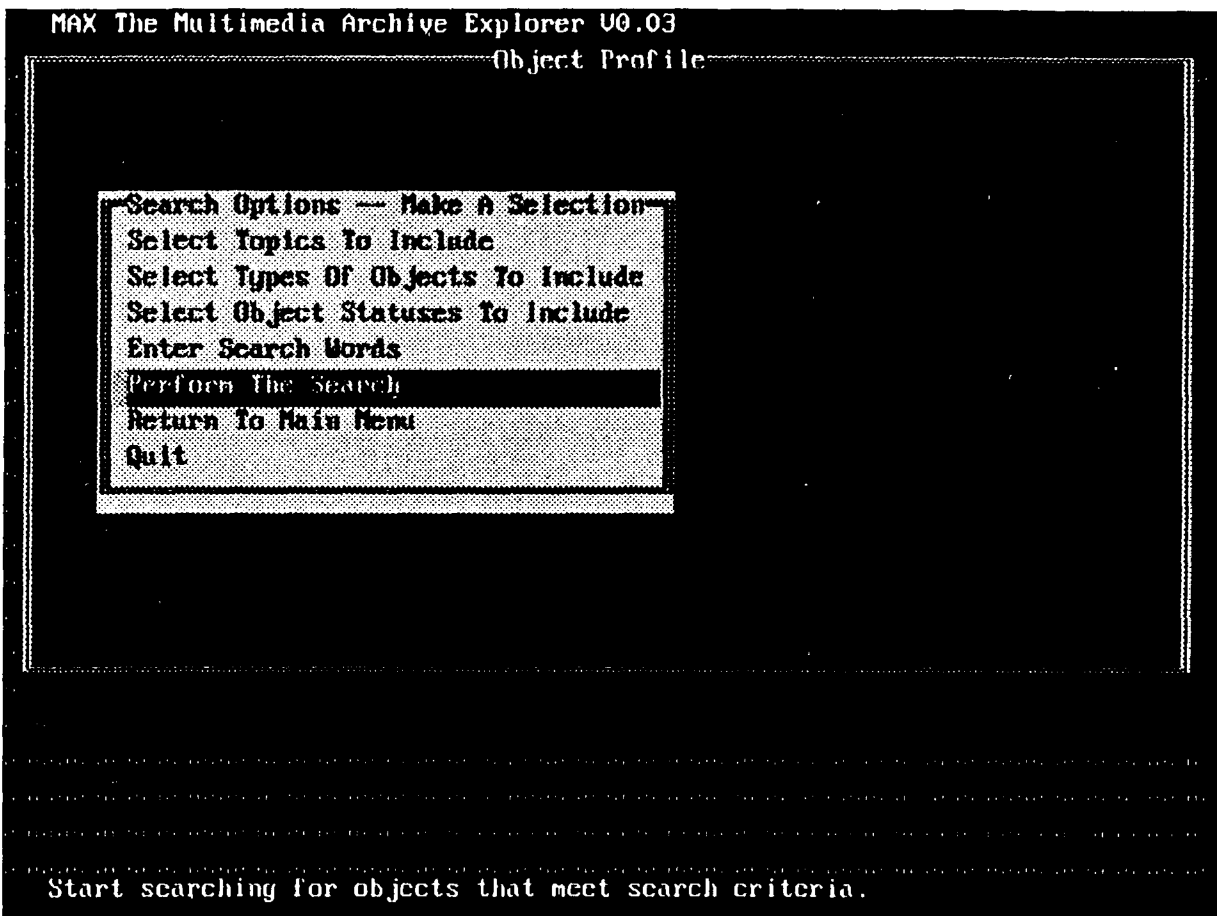


Figure 60. After search criteria have been established, the search is performed by selecting "Perform The Search" from the search options menu.

Figure 61 shows the profile for an object located in a search. Note the status line at the bottom of the screen in Figure 61 which shows that 142 objects have been found, but the computer is still searching. The user can use the **Next** button to move to the next object, in the list of objects which have been found, while the search is still in progress. The search continues during slack time between user input events. Figure 62 shows the profile for the twenty fifth object found in the search. Note the status line at the bottom of the screen in Figure 62 which shows that the search is complete and 179 objects have been found.

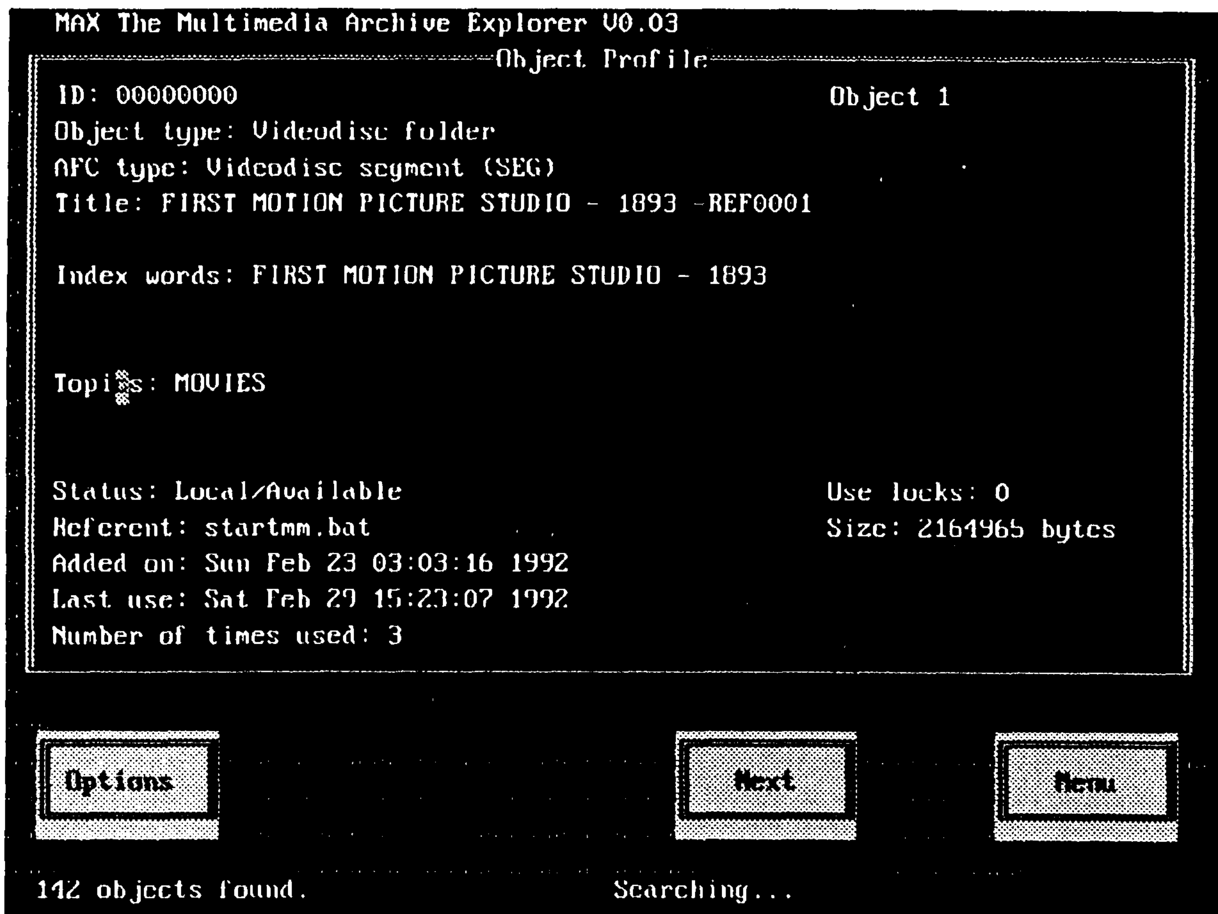


Figure 61. The profile for the first "found" object is displayed. According to the status line at the bottom of the screen, 142 objects have been found and the search is continuing.

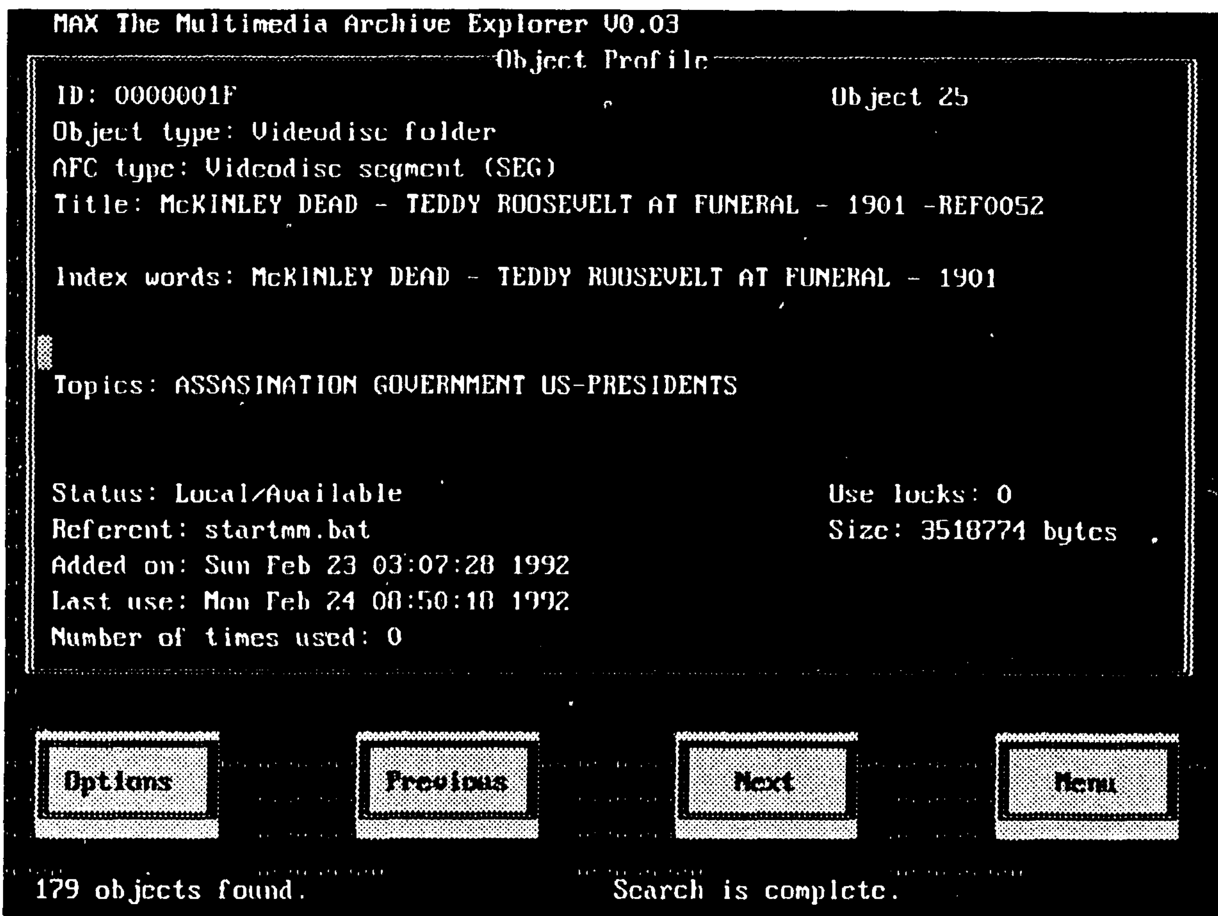


Figure 62. The profile for the twenty-fifth "found" object is displayed. According to the status line at the bottom of the screen, 179 objects have been found and the search is complete.

The information contained on the Object Profile screen is catalog and status information for the object. The information is updated once every ten seconds to reflect changes which may have occurred to the database. For example, if another user is using an object shown in the Object Profile, the number of use locks will increase.

Four buttons are provided along the bottom of the screen shown in Figure 62: Options, Previous, Next, and Menu. The Previous and Next buttons are used to move backward and forward through the list of objects which have been found. At the end of the list Next is removed from the screen, and at the beginning of the list Previous is

removed from the screen. The Menu button takes the user back to the search options menu to specify another search, to return to the main menu, or to quit.

The Options buttons is used to access operations which can be performed on the object displayed in the Object Profile window. The list of options supplied in the options menu is determined by the type of object, the status of the object, and the access mode. A user is only presented with options that apply to the object displayed in the Object Profile window for the current access mode. The options menu for the local videodisc segment on McKinley and Roosevelt is shown in Figure 63 for the public mode and in Figure 64 for the administrative mode.

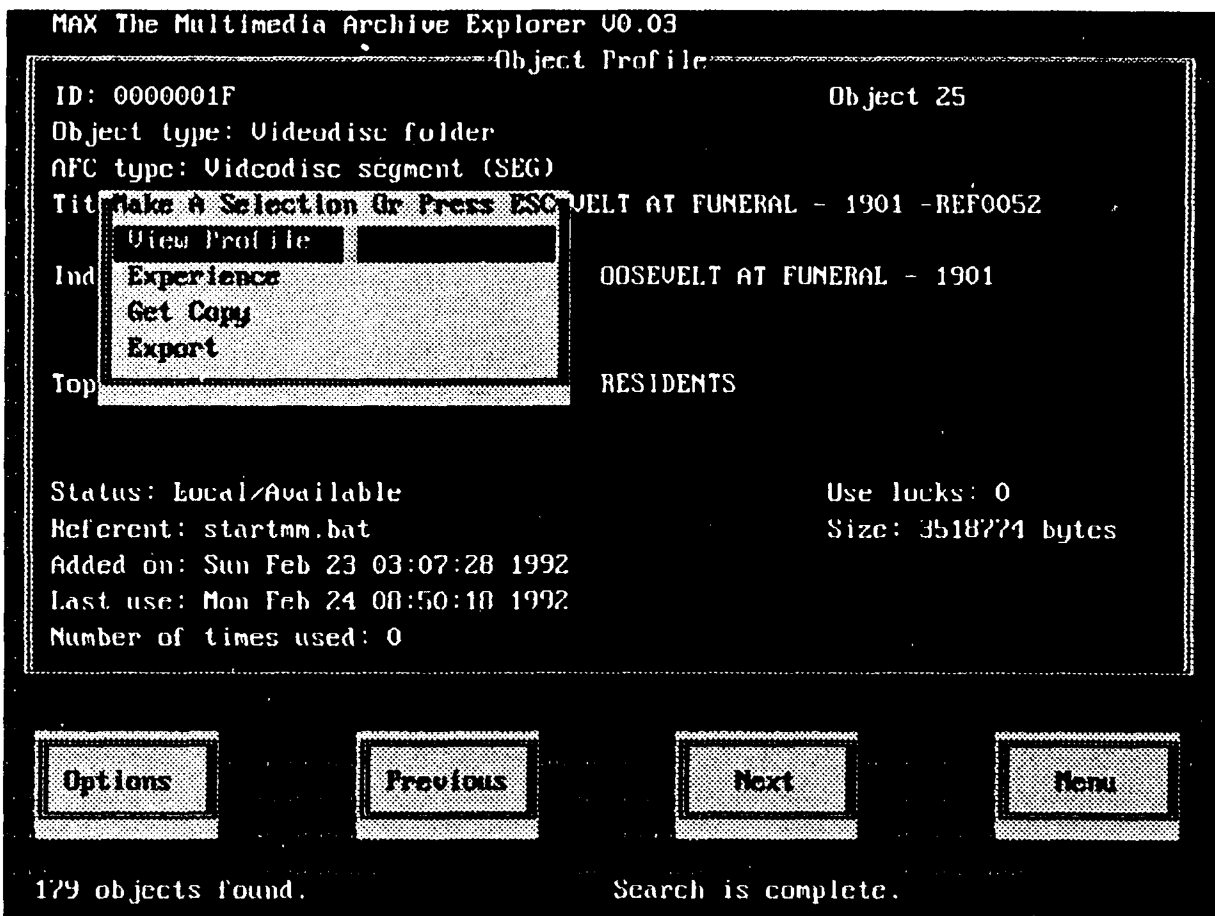


Figure 63. The options menu for a public user is displayed for an object.

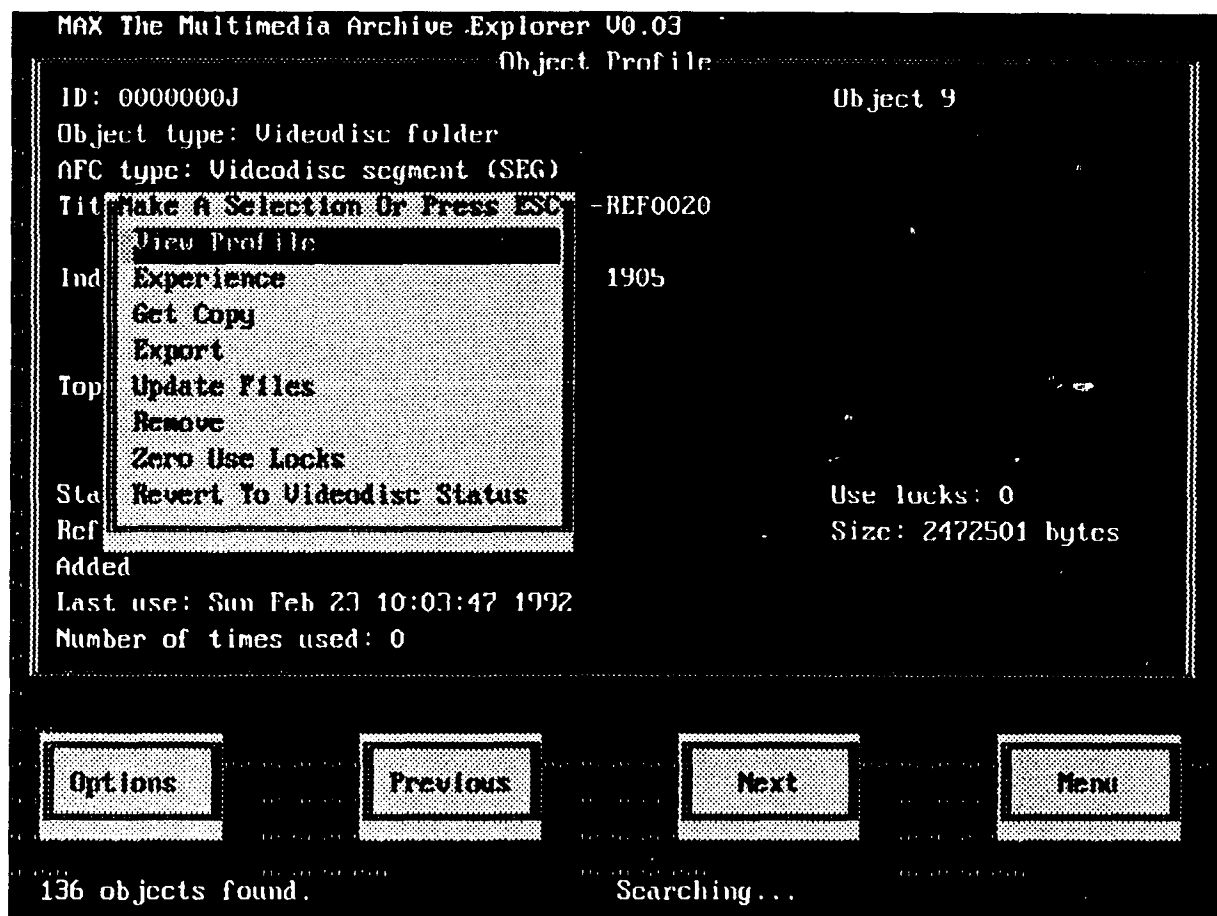


Figure 64. The options menu for an administrative user is displayed for an object.

In the public access mode, the user can view the profile, experience the object, get a copy of the object, or export the object. In the administrative access mode, the user has all the options available in the public access mode, plus the user can update the object, remove the object, set the use locks for the object to zero, and revert the object to videodisc status.

If the user chooses to get a copy of, or export, the object, the menu shown in Figure 65 is displayed. The menu allows the user to specify the location for the copy of the object. In an ICLAS environment, the user would probably want the copy to be placed in the home directory, which is the user's current directory. The **Put object in current directory** option provides the easiest method to get a copy of the object and place it in

the home directory. Optionally, the user may want to put a copy of the object on a floppy diskette. The **Specify a directory for the object** option allows the user to specify a floppy drive as the destination location for the copy of the object.

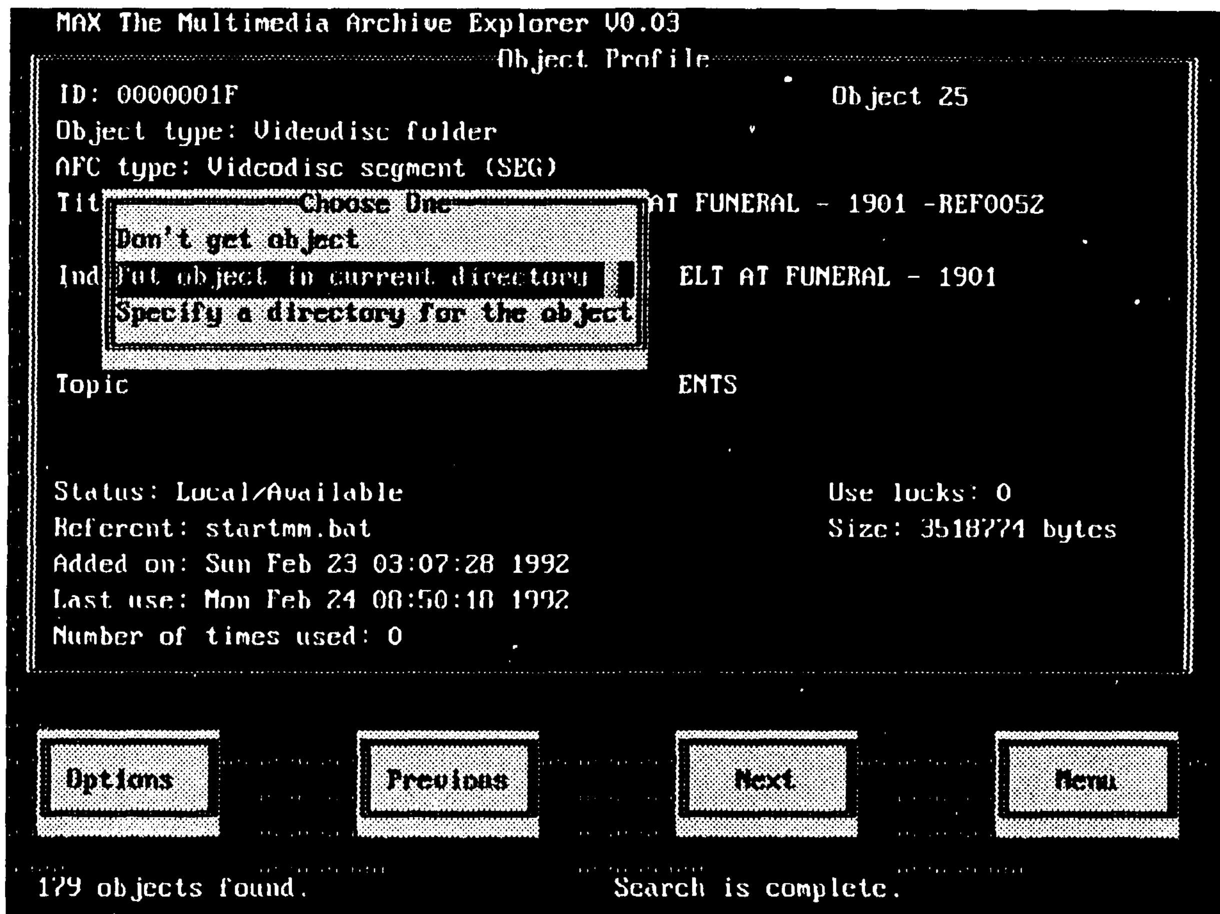


Figure 65. If a copy of an object is requested, the user is prompted for the location to put the copy of the object.

If the user wants to experience the object, then **Experience** is selected from the options menu. To experience an object, most of MAX is removed from memory to make enough room available to run the appropriate experience method. After the user is finished experiencing the object, MAX is put back in memory the way it was before it was removed. The state of the screen and the search remain intact after an object is experienced.

Figure 66 shows the options menu for a videodisc segment that has a status which indicates the segment is on videodisc and must be requested (digitized by the Automatic Folder Creator). **Experience, Get Copy,** and **Export** are not options because the object must be digitized and placed in the database before these operations can be used. Instead, the option **Request Creation** is provided.

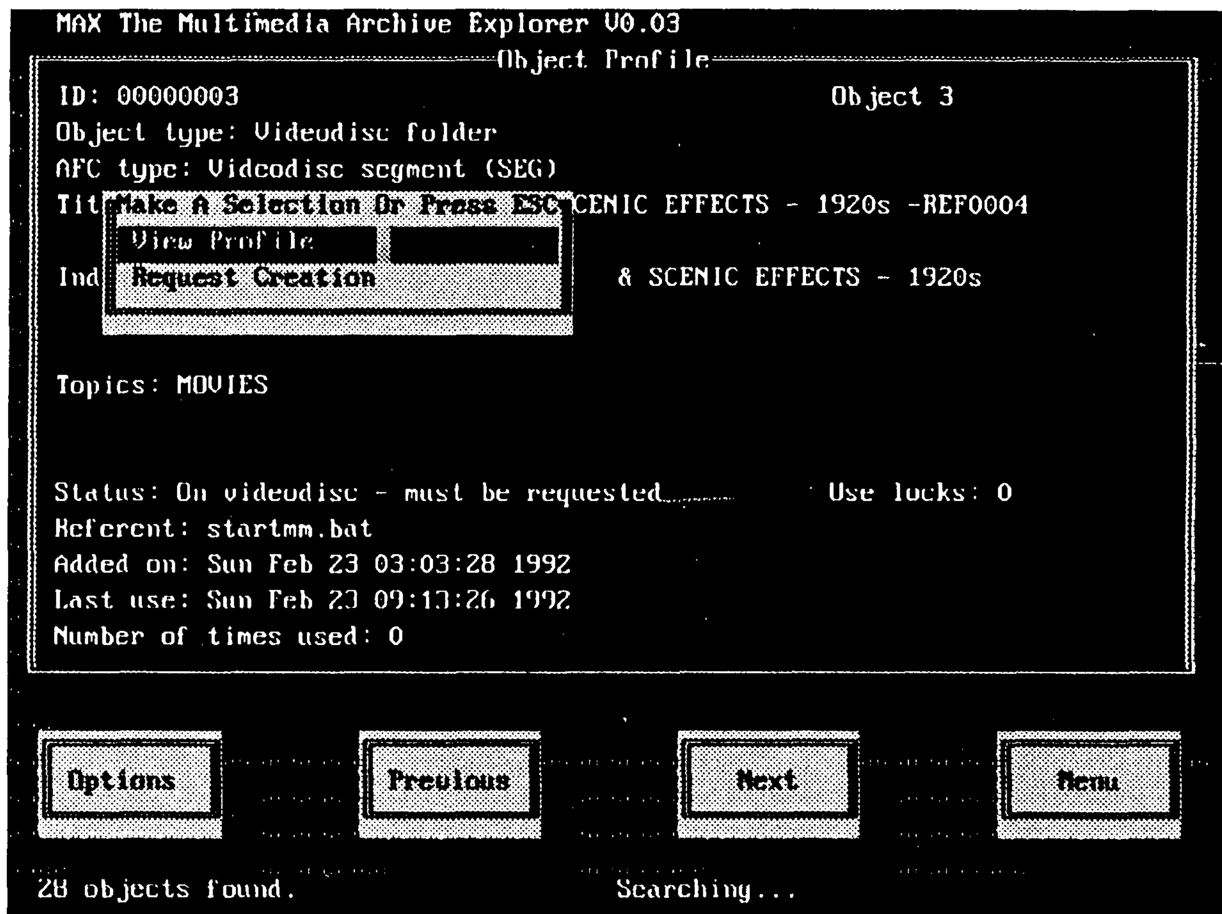


Figure 66. The options menu is displayed for a videodisc object. Since the object is on videodisc, a request must be placed to digitize the object.

When **Request Creation** is selected, another menu of create types is displayed (Figure 67). The user can select the way the Automatic Folder Creator (AFC) digitizes the videodisc segment. The options are **Create LinkWay Slideshow** and **Create LinkWay Movie**; either option can be selected by the user. The option chosen

determines what the user will see when the AFC is finished digitizing the object. After **Request Creation** is chosen, the status of the object is updated to "requested" status. When the AFC has completed the digitization and placed the object in the database, the status of the object is updated to "local/available" and the options to experience, get a copy, and export the object are placed on the options menu.

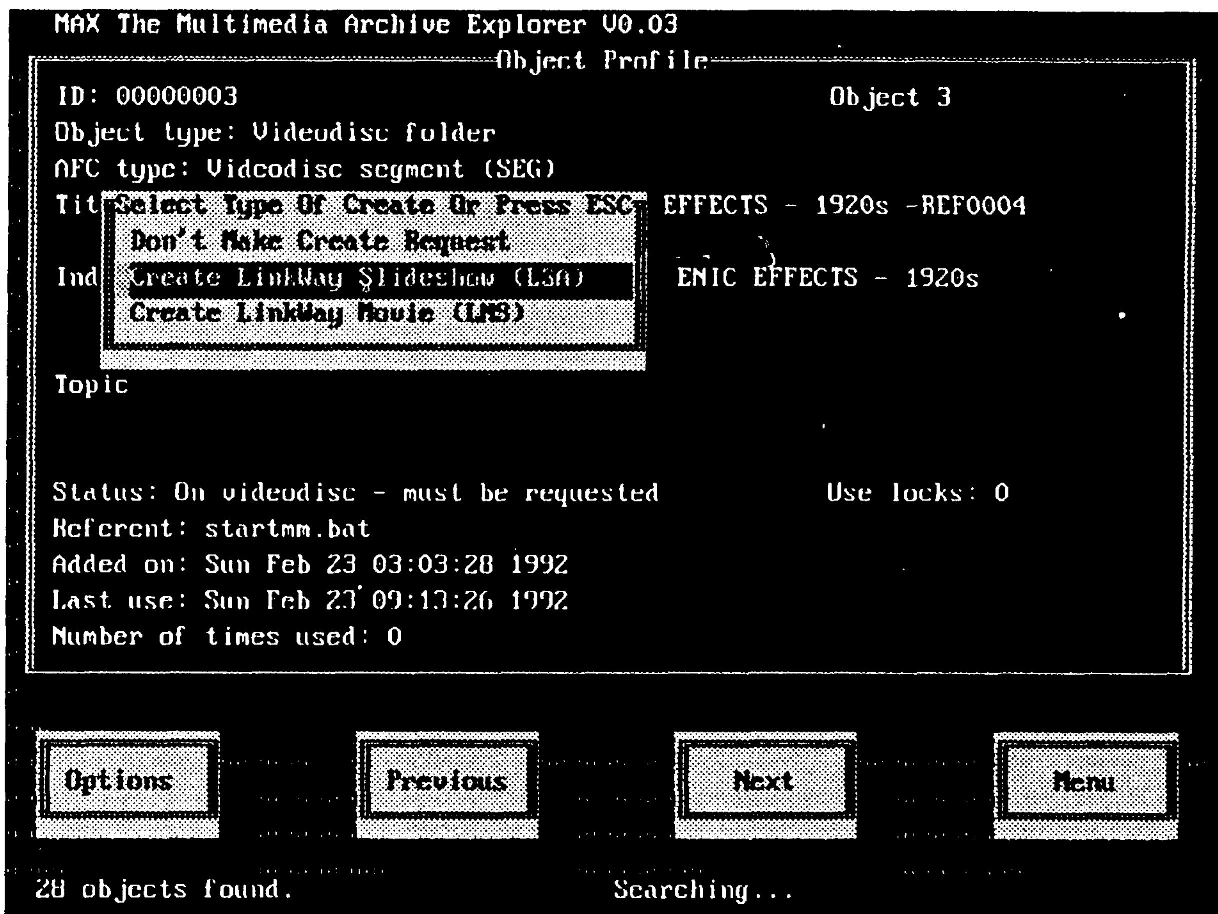


Figure 67. After a videodisc object is selected to be created, the user chooses a create type from a list of choices.

Summary

The Multimedia Archive Explorer (MAX) is a database application written using the multimedia database system application programming interface. MAX can be used with IBM LinkWay as a development tool or without LinkWay as a research, exploration, and presentation tool.

MAX recognizes two types of users: public and administrative. Public users can search for objects, directly access objects, experience objects, copy objects, and export objects. Administrative users have the same options as public users plus a set of options for adding objects, deleting objects, and maintaining the database. Access to administrative mode can be controlled using a password.

The options listed on the main MAX menu provide access to operations on the database. After objects have been located through a search or accessed directly, options are provided on other menus to manipulate the objects. Menus only display options that can be used in the context in which they are displayed. Menus also reflect the access mode, public or administrative, which determines the operations a user can perform.

CHAPTER XI

SUMMARY AND FUTURE RESEARCH

In this research project a multimedia database management system (MMDBMS) was developed for educators and students in grades K through 12 to store, organize, and retrieve multimedia objects. The first section of this chapter provides a summary of information about the MMDBMS and the second section discusses future research and development needed to expand the capabilities of the MMDBMS.

Summary

Early computers were developed to perform calculations . . . to process numerical information. Today computers process more complex forms of information including images, animations, digital video, sounds, voice, music, text, and their combinations. The term *multimedia* has gained popular usage in describing computers and software that can handle these more complex forms of information.

Multimedia has begun to find its way into educational environments because of its dynamic nature, richness, and interactivity. Although no empirical evidence is available to prove that multimedia improves instruction, anecdotal evidence suggests that multimedia serves a useful purpose in education.

Missing from the multimedia computing environment in education and elsewhere is the ability to store, organize, and retrieve multimedia objects. The multimedia database management system (MMDBMS) created in this research project provides the missing functionality.

An MMDBMS can be used by educators and students in four ways: as an exploration and research tool, as a presentation tool, as a development tool, and as a resource for educational software. No single database application serves all purposes for all populations. The MMDBMS uses a layered approach in its design to accommodate the development of a variety of user interfaces and applications.

The MMDBMS operates on a single local area network (LAN) containing a Novell Netware fileserver and DOS client workstations. The database is located on the fileserver and is manipulated by processes running on the client workstations. The object oriented programming (OOP) paradigm was used in the conceptualization and development of the MMDBMS.

The Data Model

A data model defines the objects that can be stored in the database and the operations that can be performed on the objects. Four types of objects are defined in the data model: the database object, the exception word object, the topic object, and the password object.

A database object is a multimedia object plus pieces of information that are associated with the multimedia object when it is added to the database. A multimedia object is a collection of files and the associated pieces of information include the object handle, the referent, status information, and catalog information. An exception word object is a word with little or no meaning that is removed from word index lists and titles when indexes are built to speed searches and reduce storage. A topic object is a pre-defined domain of information within which an object can be classified. And, a password object is a character string used to implement database security.

An object handle is a number associated with a multimedia object that provides a reference to an object; object handles are similar to card catalog numbers for books in a

library. A referent is a filename or a set of start instructions that the method for experiencing a multimedia object requires to execute. Status information is used to track the state of a database object and includes five pieces of information: a status flag, when the object was added to the database, when the object was last used, the number of times the object was used, and the number of use locks placed on the object. The status flag and the number of use locks value are used to manage concurrency of database use on a network. The status flag is also used to track the state of an object. Catalog information provides a description of an object and includes four pieces of information: the class of the object, a title for the object, a set of topics within which the object is classified, and a set of index words associated with the object. A user searches the catalog to locate objects in the database.

The data model includes a classification of multimedia objects supported by the MMDBMS and is presented as a class hierarchy. The root of the class hierarchy is the class **Multimedia Objects** and the leaves of the class hierarchy are the types of multimedia objects supported by the MMDBMS. The class hierarchy is constructed using the knowledge that different formats for multimedia objects exist and that multimedia objects have hardware dependencies when they are experienced.

The database system operations defined in the data model are used to create a database, establish a connection with the database, manipulate objects in the database, and maintain the database. Most of the operations for manipulating database objects can be used without regard for the specific class membership of an object on which operations are being performed.

The Automatic Folder Creator

Analog information, such as undigitized audio and video, cannot be distributed to workstations on a LAN, which prohibits the distribution of audio and video played from

a videodisc player. Videodiscs have traditionally been included in the definition of multimedia, but since the information played from a videodisc player is analog, the ability to store videodisc material in a multimedia database, for access by many users, is impossible unless a method is implemented to digitize the information.

The Automatic Folder Creator (AFC) is integrated within the MMDBMS to provide transparent access to videodisc objects. The AFC digitizes videodisc audio and video based on requests deposited in a request queue by the MMDBMS. After the videodisc object is digitized it is placed in database storage for access by the requesting user, and others, using a method called *REALIZE*. REALIZE contains a least recently used algorithm that replaces digital versions of videodisc objects that have not recently been used with newly digitized videodisc objects. The database serves as a cache for videodisc objects which allows access to large numbers of multimedia objects even when database storage is limited.

The Multimedia Database System Architecture

The multimedia database system (MMDBMS) uses a three layer approach in its design. The layers are the access method, the communication interface, and the user interface.

The access method contains operating system and network services for storing, retrieving, and experiencing multimedia objects located in directories on the fileserver and a relational record manager for storing and retrieving information associated with the multimedia objects

The communication interface contains C language functions built using the services provided in the access method. The collection of C functions forms a multimedia database management system (MMDBMS) application programming interface (API) that

is used by software developers to create database applications. The C functions provide access to the MMDBMS operations described in the data model.

Many different user interfaces can be designed to serve a variety of purposes and populations. A database application is created when a user interface is created that uses the MMDBMS API functions. The Multimedia Archive Explorer (MAX) is a prototype database application described in this document.

The Multimedia Archive Explorer

The Multimedia Archive Explorer (MAX) is a database application that is used with IBM LinkWay as a development tool or without LinkWay as an exploration, research, and presentation tool. MAX recognizes two types of users: public and administrative. Public users can search for objects, directly access objects, experience objects, copy objects, and export objects. Administrative users have the same options as public users plus a set of options for adding objects, deleting objects, and maintaining a database. Access to the administrative mode can be controlled using a password.

Future Research and Development

Multimedia database systems (MMDBMS) research is in its infancy. An MMDBMS for teachers and students in grades K through 12 is a new phenomenon. At this early stage in MMDBMS research there are many questions; every aspect of a multimedia database system needs to be reviewed and questioned. Many who, what, when, where, and why questions relating to the database system's use in educational environments exist. Five areas are of particular importance in the evolution of an MMDBMS for use in educational environments:

1. What types of applications and user interfaces are needed to meet the needs of the educational community?

2. What information should be contained in a catalog of multimedia objects and how should that information be used to access multimedia objects in the database?
3. Are there alternative architectures for the MMDBMS that provide advantage over the existing architecture?
4. What is the teacher's and student's experience with the multimedia database system?
5. How can access to multimedia databases be expanded?

Applications and User Interfaces

The multimedia database system application programming interface provides an opportunity to explore a variety of user interfaces which have been designed for specific populations and tasks. There are many populations with differing needs in K-12 education: elementary students, middle school students, high school students, special needs students, science teachers, math teachers, history teachers, English teachers, administrators, high achievers, low achievers, visual learners, and others. Research needs to be performed to determine what user interfaces and applications will serve the needs of these populations.

It was previously stated that a database system could be used four ways in education: as a research and exploration tool, as a development tool, as a presentation tool, and as a resource for educational software. Applications need to be developed for each category of use, and research is needed to determine the appearance of these applications and how they will function. There may also be other ways that a database system could be used. This possibility needs to be explored.

Cataloging and Indexing

Research is needed to determine requirements for information that should be associated with multimedia objects stored in a database. Are topics, index words, title, and type of object enough? What information is important to users of the database system? The US MARC format is used by libraries for cataloging books and other materials housed in a library. Does the US MARC format meet the needs of a database system? If not, could it be modified to support objects stored in a multimedia database?

Information associated with objects is used to locate objects stored in a database. Research is needed to determine how this information can be used to efficiently locate objects. Efficiency is a measure of speed, precision, and accuracy. Accuracy is determined by whether or not the search found objects the that the user desired. Precision is determined by whether or not the search yielded only the objects the user desired.

Alternative Architectures

Research is needed to determine if there are alternative architectures that are more suitable for the storage and retrieval of multimedia objects. The object oriented programming (OOP) paradigm was used, but not fully exploited, in the design of the MMDBMS. The OOP paradigm may provide a way of restructuring the MMDBMS. It is also possible that a new MMDBMS architecture requires a change in the definition of multimedia objects leading to research regarding the nature and structure of multimedia objects.

Examining the User's Experience

Research is needed to examine a whole range of who, what, when, where, and why questions relating to the use of a multimedia database system. These questions need to

be answered to make sure the database system is serving the needs of its users. Some possible questions follow:

1. Who is using the database system and why are they using it?
2. Who is not using the database system and why aren't they using it?
3. Where is the database system being used?
4. When is the database system being used?
5. What do users want to do that they can't with the current system?
6. What effects does the presence of a multimedia database system have on instruction?
7. What problems do users experience with using the database system?

Expanding Access

The MMDBMS resulting from this research project was designed for use on a single local area network. Research is needed to determine how to expand the scope of a database system beyond the walls of a single school. Distributed and interconnected database systems are needed to provide students and teachers with access to databases in other institutions (schools, libraries, research facilities, businesses) which may be in other cities, states, or countries. With an interconnected global system of databases, a world of information could be placed at the finger tips of students and teachers. The database system could serve as a means of communication between diverse groups from around the world. Imagine the possibilities if a student could access the holdings of the Library of Congress, the Smithsonian, National Geographic, NASA, and the Jet Propulsion Laboratory.

Remote access from homes and libraries are also possible avenues for expanding access to a database. Students and teachers could access the database at any time if home

access were provided. Access from libraries would make the database system available to a population larger than just those who are attending or work at a school.

APPENDIX A

AFC REQUEST FILE FORMATS

Each Automatic Folder Create (AFC) create type has a different request file format. A request file has a filename suffix of VRQ and a filename prefix which is the same as the object handle. The version of the request file formats that follow is 01.

Request file format for LSA and LMS create types:

# of bytes	Type	Description
2 + <crlf>	(fixed length char)	Request file version (01)
3 + <crlf>	(fixed length char)	Create type (LSA or LMS)
1-10 + <crlf>	(variable len char)	User ID of requestor
8 + <crlf>	(fixed length char)	Object handle
1-80 + <crlf>	(variable len char)	Text description/title
1- 8 + <crlf>	(variable len char)	Disk ID
1 + <crlf>	(fixed length char)	Disk side (1 or 2)
1 + <crlf>	(fixed length char)	Color info (1 or 2)
1 + <crlf>	(fixed length char)	Audio info (0, 1, 2, or 3)
5 + <crlf>	(fixed length char)	Start frame/time
5 + <crlf>	(fixed length char)	Stop frame/time
1 + <crlf>	(fixed length char)	'+' (Terminating character)

For the LSA request type, the maximum difference allowed between start and stop frames/times is 300 seconds.

For the LMS request type, the maximum difference allowed between start and stop frames/times (movie length) depends on the amount of memory installed in the AFC machine. If the AFC machine contains 8Mb of memory (7471104 bytes EMS free), then the maximum movie length is 82.5 seconds. Doubling the memory should double the maximum movie length. Regardless of the memory installed in the AFC machine, the maximum movie length can never exceed 493 seconds.

Example LSA request:

01
LSA
Keith
ABCDEFGH
This is the text description of the segment.
CELVE039
2
1
3
00100
00200
+

The file format for the LSB create type:

# of bytes	Type	Description
2 + <crlf>	(fixed length char)	Request file version (01)
3 + <crlf>	(fixed length char)	Create type (LSB)
1-10 + <crlf>	(variable len char)	User ID of requestor
8 + <crlf>	(fixed length char)	Object handle
1-80 + <crlf>	(variable len char)	Text description/title
1- 8 + <crlf>	(variable len char)	Disk ID
1 + <crlf>	(fixed length char)	Disk side (1 or 2)
1 + <crlf>	(fixed length char)	Color info (1 or 2)
1 + <crlf>	(fixed length char)	Audio info (0, 1, 2, or 3)
5 + <crlf>	(fixed length char)	Start frame/time
5 + <crlf>	(fixed length char)	Stop frame/time
5 + <crlf>	(fixed length char)	Frame/Time 1
5 + <crlf>	(fixed length char)	Frame/Time 2
5 + <crlf>	(fixed length char)	Frame/Time 3
...		
5 + <crlf>	(fixed length char)	Frame/Time N
1 + <crlf>	(fixed length char)	'+' (Terminating character)

The maximum number of frames/times in the frame list is 30. The maximum audio file length (given by stop frame/time minus start frame/time) is 1000 seconds.

Example LSB request:

01
LSB
Suzanne
12345678
This is the text description for the LSB request.
CELVE030
1
3
00100
00200
00100
00107
00111
00123
00130
00141
+

The file format for the LPM create type:

# of bytes	Type	Description
2 + <crlf>	(fixed length char)	Request file version (01)
3 + <crlf>	(fixed length char)	Create type (LPM)
1-10 + <crlf>	(variable len char)	User ID of requestor
8 + <crlf>	(fixed length char)	Object handle
1-80 + <crlf>	(variable len char)	Text description/title
1- 8 + <crlf>	(variable len char)	Disk ID
1 + <crlf>	(fixed length char)	Disk side (1 or 2)
1 + <crlf>	(fixed length char)	Color info (1 or 2)
5 + <crlf>	(fixed length char)	Frame/time
1 + <crlf>	(fixed length char)	'+' (Terminating character)

Example LPM request:

01
LPM
Bradley
123ABC00
This is the text description of the picture.
CELVE011
1
1
01034
+

Request file format for the LAD request type:

# of bytes	Type	Description
2 + <crlf>	(fixed length char)	Request file version (01)
3 + <crlf>	(fixed length char)	Create type (LAD)
1-10 + <crlf>	(variable len char)	User ID of requestor
8 + <crlf>	(fixed length char)	Object handle
1-80 + <crlf>	(variable len char)	Text description/title
1- 8 + <crlf>	(variable len char)	Disk ID
1 + <crlf>	(fixed length char)	Disk side (1 or 2)
1 + <crlf>	(fixed length char)	Audio info (0, 1, 2, or 3)
5 + <crlf>	(fixed length char)	Start frame/time
5 + <crlf>	(fixed length char)	Stop frame/time
1 + <crlf>	(fixed length char)	'+' (Terminating character)

The maximum difference between start and stop frames/times (audio recording length) is 1000 seconds.

Example LAD request:

01
LAD
Dallas
EYEYOHHH
This is the text description of the audio file.
CELVE002
1
2
00200
00410
+

The following notes apply to the request file formats:

-User ID is the network user ID of the user making the request.
In an ICLAS installation this would be the contents of
the environment variable ID.

- the disk side must be a '1' or a '2'

- the audio info must be a '0', '1', '2', or a '3'
  0 = no audio
  1 = right channel
  2 = left channel
  3 = stereo (both channels)

- the color info must be a '1' or '2'
  1 = black and white
  2 = color

APPENDIX B

DESCRIPTIONS OF MULTIMEDIA OBJECT TYPES SUPPORTED BY THE MULTIMEDIA DATABASE MANAGEMENT SYSTEM

This appendix describes the multimedia object types supported by the multimedia database management system (MMDBMS): LWCGA, LWMCGA2, LWMCGA256, LWEGA, LWVGA16, MACPAINT, SBMCGA256, SB16EGA350, SB16EGA200, SBVGA16, ASCIITEXT, LWOB, LWPG, LWBIT, LWFNT, LWFOLDER, DS201CVSD, ACPASOUND, PVI, DVI, RTV, SEG, SFL, FRM, DOSAPP, GENERIC.

Images

LinkWay Images

- LWCGA
  - 320 x 200 pixels
  - 4 colors
  - Requires video adapter that supports CGA mode to be experienced
  - Filename suffix: PCC
  - Referent: filename (ex: BIRD.PCC)
- LWMCGA2
  - 640 x 480 pixels
  - 2 colors
  - Requires video adapter that supports MCGA mode to be experienced
  - Filename suffix: PCH
  - Referent: filename (ex: BIRD.PCH)

LinkWay Images

- LWMCGA256
  - 320 x 200 pixels
  - 256 colors out of 256,000
  - Requires video adapter that supports MCGA mode to be experienced
  - Image filename suffix: PCM
  - Palette filename suffix: P13
  - Referent: image filename (ex: BIRD.PCM)

- LWEGA
  - 640 x 350 pixels
  - 16 colors out of 63 colors
  - Requires video adapter that supports EGA mode to be experienced
  - Image filename suffix: PCE
  - Palette filename suffix: P10
  - Referent: image filename (ex: BIRD.PCE)
- LWVGA16
  - 640 x 480 pixels
  - 16 out of 256,000 colors
  - Requires video adapter that supports VGA mode to be experienced
  - Image filename suffix: PCV
  - Palette filename suffix: P12
  - Referent: image filename (ex: BIRD.PCV)

MacPaint Images

- MACPAINT
  - 576 x 720 pixels
  - 2 colors
  - Requires video adapter that supports MCGA mode to be experienced
  - Filename extension: PNT
  - Referent: filename (ex: BIRD.PNT)

Storyboard Images

- SBMCGA256
  - 320 x 200 pixels
  - 256 colors out of 256,000
  - Requires video adapter that supports MCGA mode to be experienced
  - Filename extension: PIC
  - Referent: filename (ex: BIRD.PIC)
- SB16EGA350
  - 640 x 350 pixels
  - 16 colors out of 63
  - Requires video adapter that supports EGA mode to be experienced
  - Filename extension: PIC
  - Referent: filename (ex: BIRD.PIC)

- SB16EGA200
    - 640 x 200 pixels
    - 16 colors
    - Requires video adapter that supports EGA mode to be experienced
    - Filename extension: PIC
    - Referent: filename (ex: BIRD.PIC)

Storyboard Images

- SBVGA16
    - 640 x 480 pixels
    - 16 colors out of 256,000
    - Requires video adapter that supports VGA mode to be experienced
    - Filename extension: PIC
    - Referent: filename (ex: BIRD.PIC)

Text

- ASCIITEXT
    - ASCII text file
    - Requires video adapter that supports character mode to be experienced
    - Text filename extension: any
    - Referent: text filename (ex: LETTER.TXT)

LinkWay Pieces

- LWOB
    - LinkWay object cut from LinkWay page
    - Cannot be experienced
    - Filename extension: OB
    - Referent: filename (ex: BUTTON.OB)
- LWPG
    - LinkWay page cut from LinkWay folder
    - Cannot be experienced
    - Filename extension: PG
    - Referent : filename (ex: SAMPLE.PG)

- LWBIT
    - LinkWay bitmap cut from image
    - Requires video adapter that supports bitmap graphics to be experienced
    - Filename extension: BIT
    - Referent: filename (ex: BIRD.BIT)
- LWFNT
    - LinkWay font
    - Requires video adapter that supports bitmap graphics to be experienced
    - Filename extension: LFN
    - Referent: filename (ex: FONT1.LFN)
- LWFOLDER
    - LinkWay folder
    - Cannot be experienced
    - Filename extension: LWC, LWH, LWM, LWE, or LWV
    - Referent: filename (ex: BIRD.LWM)

Sounds

- DS201CVSD
    - Digispeech CVSD sound with header format compatible with LinkWay Digispeech audio record utility
    - Requires Digispeech adapter to be experienced
    - Filename extension: any
    - Referent: filename (ex: BIRD.DIG)
- ACPASOUND
    - Audio Capture Playback Adapter (ACPA) sound compatible with files generated by RECORD program, in LinkWay package
    - Requires ACPA to be experienced
    - Filename extensions: _AU (file 1), _AD (file 2)
    - Referent: filename prefix (ex: BIRD)

Digital Video

- PVI
    - Movie generated by IBM Movie Maker (Photomotion)
    - Requires video adapter that supports MCGA mode to be experienced
    - Filename extension: V
    - Referent: filename (ex: BIRD.V)

- DVI
    - Intel's Digital Video Interactive (DVI) using Intel proprietary compression
    - Requires DVI playback adapter to be experienced
    - Filename extension: any
    - Referent: filename (ex: BIRD.VID)
- RTV
    - Intel's Real Time Video (RTV) using hardware compression
    - Requires DVI playback adapter to be experienced
    - Filename extension: any
    - Referent: filename (ex: BIRD.RTV)

Videodisc Segments

- SEG
    - Segment of videodisc specified by start and end frames/times
    - Can be realized as an LSA, LMS, or LAD Automatic Folder Creator create ype
    - When realized, is a DOSAPP
    - Referent: application start instructions
- SFL
    - Segment of videodisc with a frame list
    - Can be realized as an LSA, LSB, LMS, or LAD Automatic Folder Creator create type
    - When realized, is a DOSAPP
    - Referent: application start instructions
- FRM
    - Frame on a videodisc specified by a frame or time
    - Can be realized as an LPM Automatic Folde Creator create type
    - When realized, is an LWMCGA256 image
    - Referent: image filename

Applications

- DOSAPP
  - IBM or Microsoft DOS application
  - Application determines if hardware environment is appropriate to run the application
  - A collection of files including a program type recognized by the command interpreter (EXE, COM, or BAT)
  - Referent: start instructions

Generic

- GENERIC
  - An unclassified collection of files
  - The object cannot be experienced
  - Referent: not applicable

APPENDIX C

VIDEODISC INFORMATION FILE (VIF) FORMAT

A Videodisc Information File (VIF) contains object information for objects of type Videodisc Segments. Each object is a record in the VIF file. Each Videodisc Segments object type (SFL, SEG, and FRM) has a different record format. An unlimited number of objects may be placed in one file, but it is suggested that a single VIF file contain the objects located on one side of one videodisc, since the objects are added to the database as a logical group.

VIF record type: SFL

Record Entry	Description
<type><crlf>	The three letters SFL followed by a carriage return line feed.
<disk name><crlf>	The name of a videodisc (8 chararacters) followed by a carriage return line feed.
<disk side><crlf>	A single character (1 or 2) followed by a carriage return line feed.
<audio info><crlf>	A single character (0, 1, 2, or 3) followed by a carriage return line feed.
<color info><crlf>	A single character (1 or 2) followed by a carriage return line feed.
<start><crlf>	A five digit number (represented using characters) followed by a carriage return line feed.
<end><crlf>	A five digit (represented using characters) number followed by a carriage return line feed.
<frame list><crlf>	A string of five digit numbers (represented using characters) separated by spaces and followed by a carriage return line feed (30 five digit numbers maximum).
<title><crlf>	A variable length string of alphanumeric characters followed by a carriage return line feed (80 characters maximum).

VIF record type: SFL (Continued)

<index words><crlf>	A variable length string of words separated by spaces and followed by a carriage return line feed (160 characters maximum).
<topic pointers><crlf>	A variable length string of topic pointers separated by spaces and followed by a carriage return line feed (160 characters maximum).
Char 219 decimal	This character delimits the record.

VIF record type: SEG

Record Entry	Description
<type><crlf>	The three letters SEG followed by a carriage return line feed.
<disk name><crlf>	The name of a videodisc (8 chars) followed by a carriage return line feed.
<disk side><crlf>	A single character (1 or 2) followed by a carriage return line feed.
<audio info><crlf>	A single character (0, 1, 2, or 3) followed by a carriage return line feed.
<color info><crlf>	A single character (1 or 2) followed by a carriage return line feed.
<start><crlf>	A five digit number (represented using characters) followed by a carriage return line feed.
<end><crlf>	A five digit number (represented using characters) followed by a carriage return line feed.
<title><crlf>	A variable length string of alphanumeric characters followed by a carriage return line feed (80 characters maximum).

VIF record type: SEG (Continued)

<index words><crlf>	A variable length string of words separated by spaces and followed by a carriage return line feed (160 characters maximum).
<topic pointers><crlf>	A variable length string of topic pointers separated by spaces and followed by a carriage return line feed (160 characters maximum).
Char 219 decimal	This character delimits the record.

VIF record type: FRM

Record Entry	Description
<type><crlf>	The three letters FRM followed by a carriage return line feed.
<disk name><crlf>	The name of a videodisc (8 chars) followed by a carriage return line feed.
<disk side><crlf>	A single character (1 or 2) followed by a carriage return line feed.
<audio info><crlf>	A single character (0, 1, 2, or 3) followed by a carriage return line feed.
<color info><crlf>	A single character (1 or 2) followed by a carriage return line feed.
<frame><crlf>	A five digit number (represented using characters) followed by a carriage return line feed.
<title><crlf>	A variable length string of alphanumeric characters followed by a carriage return line feed (80 characters maximum).
<index words><crlf>	A variable length string of words separated by spaces and followed by a carriage return line feed (160 characters maximum).

VIF record type: FRM (Continued)

<topic pointers><crlf>	A variable length string of topic pointers separated by spaces and followed by a carriage return line feed (160 characters maximum).
Char 219 decimal	This character delimits the record.

The first record in a VIF file is preceded by the version number of the VIF record format followed by a carriage return line feed. The format presented in this section is 01. The last record in a VIF file must be followed by a Char 219 decimal and a carriage return line feed (this is in addition to the Char 219 decimal that is part of the VIF record.) Two consecutive Char 219 characters indicate the end of a VIF file.

Description of VIF file record information:

<disk name>
- the 8 character videodisc identifier
- example: CELVE012

<disk side>
- a single character (1 or 2) which specifies the side of the videodisc (side A or side 1 = 1, side B or side 2 = 2)
- example: 1

<audio info>
- a single character (0, 1, 2, 3) which specifies the availability of audio
    - 0 = no audio
    - 1 = audio on right channel
    - 2 = audio on left channel
    - 3 = audio on both right and left channels (stereo)
- example: 3

Description of VIF file record information (Continued):

<color info>

- a single character (1, 2) which specifies whether the images (video) is in black and white or in color
  - 1 = black and white
  - 2 = color

<start>

- a five digit number (represented using characters) indicating the start of a videodisc segment. This is the frame or time as displayed by the videodisc player.
- exmaple: 00120

<end>

- a five digit number (represented using characters) indicating the end of a videodisc segment. This is the frame or time as displayed by the videodisc player.
- example: 00130

<frame>

- a five digit number (represented using characters) indicating a single frame or time. This is the frame or time as displayed by the videodisc player.
- example: 00125

<frame list>

- a list of frames/times separated by spaces. The frames/times are as displayed by the videodisc player.
- exmaple: 00100 00110 00120 00125 00131

<title>

- title is a variable length (up to 80 characters) string which is a title or description of the segment or frame.
- example: Albert Einstein lectures on the general theory of relativity.

<index words>

- a list of words each of which is a description of the segment or frame . The words will be used when a word search is performed.

<topic pointers>

- a list of topic pointers (references to topics) under which the segment or frame can be classified.

APPENDIX D

RELATIONAL TABLE AND INDEX SPECIFICATION

This appendix specifies the tables and indices managed by the relational record manager.

Table Name: NEXTID
Field 1: ID (ASCII string, 8 characters)
Index: None

The NEXTID table holds the next available object handle. When a new handle is needed, the handle is read from the ID field and an incremented handle is written back to the ID field. The handle stored in the ID field is 8 characters long and represents a base 36 number.

Table Name: EXCEPTNS
Field 1: EXWORD (ASCII string, 20 characters)
Primary Index: Field 1

The EXCEPTNS table holds exception words. Exception words are words that are excluded from the WORDINDX table (table of index words associated with each object) and from user supplied strings of search words.

Table Name: TOPICS
Field 1: TOPICPTR (ASCII string, 20 characters)
Field 2: TOPICDEF (ASCII string, 40 characters)
Primary Index: Field 1

The TOPICS table stores the topic list, and relates topic pointers to their descriptions. The TOPICPTR field holds the pointer to a topic and the TOPICDEF field holds the related definition of the topic.

Table Name: TOPCLIST
Field 1: ID (ASCII string, 8 characters)
Field 2: TOPICPTRS (ASCII string, 160 characters)
Primary Index: Field 1

The TOPCLIST table holds a list of topic pointers that are associated with an object. The ID field holds the object's handle and the TOPICPTRS field holds the associated list of topic pointers.

Table Name: WORDLIST
Field 1: ID (ASCII string, 8 characters)
Field 2: WORDS (ASCII string, 160 characters)
Primary index: Field 1

The WORDLIST table holds a list of index words associated with an object. Index words are stored in capitalized form. The ID field holds the object 's handle and the WORDS field holds the associated list of index words.

Table Name: TITLES
Field 1: ID (ASCII string, 8 characters)
Field 2: TITLE (ASCII string, 80 characters)
Primary Index: Field 1

The TITLES table holds the title associated with an object. The ID field holds the object 's handle and the TITLE field holds the associated title.

Table Name: TOPCINDX
Field 1: ID (ASCII string, 8 characters)
Field 2: TOPICPTR (ASCII string, 20 characters)
Primary Index: Field 1 and Field 2 (provides a unique index)
Secondary Index: Field 2

The TOPCINDX table holds the associations between objects and topics. Each object has one entry in the table for each topic associated with the object. An object with N topic references will contain N entries in the TOPCINDX table. The ID field holds the object's handle and the TOPICPTR field holds the associated TOPICPTR.

Table Name: WORDINDX
Field 1: ID (ASCII string, 8 characters)
Field 2: WORD (ASCII string, 20 characters)
Primary Index: Field 1 and Field 2 (provides a unique index)
Secondary Index: Field 2

The WORDINDX table holds associations between objects and index words. Each object has one entry in the table for each index word associated with the object. An object associated with four index words will contain four entries in the TOPCINDX table. The ID field holds the object's handle and the WORD field holds the associated index word.

Table Name: OBJINFO
Field 1: ID (ASCII string, 8 characters)
Field 2: CLASS (integer)
Field 3: STATUS (1 character)
Field 4: LASTUSE (long integer)
Field 5: NUMUSES (long integer)
Field 6: WHENADD (long integer)
Field 7: USELOCKS (integer)
Primary Index: Field 1
Secondary Indexes: Field 2, Field 3

The OBJINFO table stores class and status information for an object. The ID field holds the object's handle. The CLASS field holds an enumeration of the class of the object. The STATUS field holds the value of the status flag associated with the object. The LASTUSE field holds the date and time the object was last used. The NUMUSES field holds the number of times the object was accessed by a user. The WHENADD field holds the date and time the object was added to the database. And, the USELOCKS field holds the number of users currently using the object.

Table Name: REFERENT
Field 1: ID (ASCII string, 8 characters)
Field 2: REFERENT (ASCII string, 40 characters)
Primary Index: Field 1

The referent table holds the referent associated with an object. The ID field holds the object's handle and the REFERENT field holds the associated referent.

Table Name: VDINFO
Field 1: ID (ASCII string, 8 characters)
Field 2: TYPE (ASCII string, 3 characters)
Field 3: DISKNAME (ASCII string, 8 characters)
Field 4: DISKSIDE (1 character)
Field 5: AUDINFO (1 character)
Field 6: COLORINF (1 character)
Field 7: STARTPOS (ASCII string, 5 characters)
Field 8: ENDPOS (ASCII string, 5 characters)
Primary Index: Field 1

The VDINFO table holds object information about an object of type Videodisc Segments. The ID field holds the object's handle. The TYPE field holds the type of videodisc object (SEG, SFL, or FRM). The DISKNAME field holds the name of the videodisc on which the object can be found. The DISKSIDE field holds the side of the videodisc containing the object. The AUDIOINFO field holds information about which audio channels contain audio (0 = no audio, 1 = right channel, 2 = left channel, and 3 = stereo). The COLORINF field holds information that indicates whether the video segment is predominately in color or black and white (1 = black and white and 2 = color). The STARTPOS field holds the frame or time on the videodisc which is the start of the segment. And, the ENDPOS field holds the frame or time on the videodisc which is the end of the segment.

Table Name: FRAMELST
Field 1: ID (ASCII string, 8 characters)
Field 2: LIST (ASCII string, 180 characters)
Primary Index: Field 1

The FRAMELST table holds the list of frames associated with videodisc objects that have a type of SFL (segment with frame list). The ID field holds the object's handle and the LIST field holds the list of frames associated with the object. The frame list can contain 30 frames which are separated by spaces.

APPENDIX E

MULTIMEDIA DATABASE MANAGEMENT SYSTEM APPLICATION PROGRAMMING INTERFACE

The multimedia database management system (MMDBMS) application programming interface (API) functions were written using Microsoft C 6.0 with the large memory model library. Four C libraries (in addition to the C 6.0 library) are needed to support the API. These libraries are:

- Novell Netware C Interface for DOS (Version 1.2)
- Hold Everything from South Mountain Software (Version 1.05)
- C Utility Library from South Mountain Software (Version 5.0)
- Paradox Engine from Borland (Version 2.0)

In the first section, macro and type definitions are listed. In the second section, the ObjInfo structure is described. In the third section, the API function prototypes are listed. And, in the fourth section, expanded descriptions of the API functions are provided.

Macro and Type Definitions

The following macro and type definitions are referenced in the MMDBMS API:

```
/* Constraints */
#define LOCKTIMEOUT          30
#define OBJHANDLELEN          8
#define MAXWORDLISTLEN      160
#define MAXTOPCLISTLEN      160
#define MAXWORDINDEXES       40
#define MAXWORDINDEXLEN      20
#define MAXTOPCINDEXLEN      20
#define MAXTOPCDEFLEN        40
#define MAXTITLELEN          80
#define STATUSLEN             1
#define MAXREFERENTLEN       40
#define AFCTYPELEN            3
#define MAXDISKNAMELEN        8
#define DISKSIDELEN           1
#define AUDINFOLEN            1
#define COLORINFLEN           1
#define VDPOSITIONLEN         5
#define MAXFRAMELISTLEN     180
#define MAXPATHLEN           80
#define MAXTABLENAMELEN       8
#define MAXFIELDNAMELEN      25
#define MAXUSERIDLEN         10
#define MAXFILENAMELEN       12
#define MINDOSDISKFREE   (unsigned long)20000
#define MAXCLASSDESCLEN      40

/* Type defines */
typedef char  PATH[MAXPATHLEN + 1];
typedef char  OBJECTHANDLE[OBJHANDLELEN + 1];
typedef int   OBJECTCLASS;
typedef char  OBJECTSTATUS[STATUSLEN + 1];
typedef long  OBJECTLASTUSE;  /* time_t */
typedef long  OBJECTNUMUSES;
typedef long  OBJECTWHENADD;  /* time_t */
typedef int   OBJECTUSELOCKS;
typedef char  OBJECTTITLE[MAXTITLELEN + 1];
typedef char  OBJECTREFERENT[MAXREFERENTLEN + 1];
typedef char  OBJECTWORDLIST[MAXWORDLISTLEN + 1];
typedef char  OBJECTTOPCLIST[MAXTOPCLISTLEN + 1];
```

```
typedef unsigned long OBJECTSIZE;
typedef char  OBJAFCTYPE[AFCTYPELEN + 1];
typedef char  OBJAFCDISKNAME[MAXDISKNAMELEN + 1];
typedef char  OBJAFCDISKSIDE[DISKSIDELEN + 1];
typedef char  OBJAFCAUDINFO[AUDINFOLEN + 1];
typedef char  OBJAFCCOLORINF[COLORINFLEN + 1];
typedef char  OBJAFCSTARTPOS[VDPOSITIONLEN + 1];
typedef char  OBJAFCENDPOS[VDPOSITIONLEN + 1];
typedef char  OBJSFLFRAMELST[MAXFRAMELISTLEN + 1];
typedef int   INFOTYPE;
typedef char  TABLENAME[MAXTABLENAMELEN + 1];
typedef int   AFCREQUESTTYPE;
typedef char  USERID[MAXUSERIDLEN + 1];
typedef char  TOPICPTR[MAXTOPCINDEXLEN + 1];
typedef char  TOPICDEF[MAXTOPCDEFLEN + 1];
typedef char  SEARCHWORD[MAXWORDINDEXLEN + 1];
typedef int   CHECKOFFLIST;
typedef char  CLASSDESC[MAXCLASSDESCLEN + 1];
```

ObjInfo Structure Description

The ObjectInfo structure is used to hold object information for an object. The declaration of the ObjectInfo structure follows, along with a description of each of its elements.

```
struct ObjectInfo
 {
 /* Version info */
 char                    Version[7];
 /* Base Multimedia Class Info */
 OBJECTHANDLE            objHandle;
 OBJECTCLASS             objClass;
 OBJECTSTATUS            objStatus;
 OBJECTLASTUSE           objLastUse;
 OBJECTNUMUSES           objNumUses;
 OBJECTWHENADD           objWhenAdd;
 OBJECTUSELOCKS          objUseLocks;
 OBJECTTITLE             objTitle;
 OBJECTREFERENT          objReferent;
 OBJECTWORDLIST          objWordList;
 OBJECTTOPCLIST          objTopcList;
 OBJECTSIZE              objSize;
 /* AFC Class Additional Info */
 OBJAFCTYPE              objAFCType;
 OBJAFCDISKNAME          objAFCDiskName;
 OBJAFCDISKSIDE          objAFCDiskSide;
 OBJAFCAUDINFO           objAFCAudInfo;
 OBJAFCCOLORINF          objAFCColorInf;
 OBJAFCSTARTPOS          objAFCStartPos;
 OBJAFCENDPOS            objAFCEndPos;
 /* AFC Class, SFL Type, Additional Info */
 OBJSFLFRAMELST          objSFLFrameLst;
 /* Location of object files for add operation */
 PATH                    objAddSourcePath;
 };
```

ObjInfo structure element descriptions:

Version - when an object is exported, a version number is placed in Version. Then, when the object is imported this version number is verified to make sure the structure is compatible. Version is a 6 character string (NULL terminated).

objHandle - is the handle for the multimedia object in the MMDB. objHandle is an 8 character string (NULL terminated). The handle is a base 36 number that has a range of 00000000 to ZZZZZZZZ.

objClass - is the class of the multimedia object. objClass is an integer.

objStatus - is the status of the multimedia object. objStatus is a 1 character string (NULL terminated). The following is a list of possible statuses and their meanings:

V =	virtual (video disk based material)
L =	locally available/real
R =	the object has been requested from an automatic folder creation (AFC) machine
+ =	the object is being added to the MMDB, but has not yet been committed
D =	the object is flagged for deletion. An object is given this status just prior to the time it is deleted.
M =	the AFC object is in the process of being reverted from an L status to a V status
U =	the object's files are being updated

objLastUse - the last time the object was last experienced or copied. objLastUse is a long integer and is the number of seconds that has elapsed since 00:00:00 hour, January 1, 1970 (GMT).

objNumUses - the number of times the object has been experienced or copied. objNumUses is a long integer.

objWhenAdd - when the object was added to the MMDB. objWhenAdd is a long integer and is the number of seconds that has elapsed since 00:00:00 hour, January 1, 1970 (GMT).

objUseLocks - the number of use locks on the object. objUseLocks is an integer. A use lock is placed on an object while it is being experienced or copied.

objTitle - is the title for the object. objTitle is an 80 character string (NULL terminated).

objReferent - is the name of the object, or the instruction for executing the object. For example, if the object is a LinkWay MCGA 256 Color Picture of a dog, the referent would be DOG.PCM (the name of the picture.) If the object is a DOS Application that runs when STARTMM.BAT is executed, the object's referent would be STARTMM.BAT. objReferent is a 40 character string (NULL terminated). The following table describes the referent for each object class.

objWordList - is a string that holds a list of index words that are separated by spaces. objWordList is a 160 character string (NULL terminated). The index words are used in performing searches.

objTopcList - is a string that holds a list of topic pointers that are separated by spaces. objTopcList is a 160 character string (NULL terminated). The topic pointers are used in performing searches. A topic pointer is a 20 character (NULL terminated) maximum string that refers to a topic definition in the MMDB.

objSize - is the size (in bytes) of the object. OBJECTSIZE is an unsigned long type.

objAFCType - is the type of AFC object. There are three types of AFC objects: SFL, SEG, FRM. SFL is a a segment on a videodisc where an explicit list of frames for digitization has been specified. SEG is a segment on a videodisc. FRM is a single frame on a videodisc. objAFCType is a 3 character string (NULL terminated.) This information is supplied only for AFC objects.

objAFCDiskName - is a name of a videodisc. A videodisc name is an 8 character ID given to a videodisc which is in a videodisc player attached to an AFC machine. objAFCDiskName is an 8 character string (NULL terminated). This information is supplied only for AFC objects.

objAFCDiskSide - is the side of a videodisc (1 or 2). objAFCDiskSideis a character string (NULL terminated). This information is supplied only for AFC objects.

objAFCAudInfo - is audio information for an AFC object. objAFCAudInfo is a 1 character string (NULL terminated). The possible values and their descriptions are:

0 = no audio
1 = audio on right channel
2 = audio on left channel
3 = audio on both left and right channels (stereo)

This information is supplied only for AFC objects.

objAFCColorInf - is color information for an AFC object. objAFCColorInf is a 1 character string (NULL terminated). The possible values and their descriptions are:

1 = black & white
2 = color

This information is supplied only for AFC objects.

objAFCStartPos - is the start time/frame for a segment on a videodisc (SEG and SFL types) or the location of a time/frame (FRM type). objAFCStartPos is a 5 character string (NULL terminated). This information is supplied only for AFC objects.

objAFCEndPos - is the end time/frame for a segment on a videodisc (SEG and SFL types) or the location of a time/frame (FRM type). objAFCEndPos is a 5 character string (NULL terminated). This information is supplied only for AFC objects.

objSFLFrameLst - is a list of times/frames (maximum 30) separated by spaces in a segment on a videodisc. objFrameLst is a 180 character string (NULL terminated). A frame list is supplied for AFC objects of SFL type.

objAddSourcePath - when an object is added to the MMDB, objAddSourcePath is set to the path pointing to the directory holding the object's files. objAddSourcePath is an 80 character string (NULL terminated).

API Function Declarations

Create an instance of an MMDB

```
int CreateMMDB(char *path, char *MethodsSourcePath,
                    struct MMDBConfig *MMDBcfg);
```

Open an MMDB

```
int MMDBOpen(char *path, int restorescreen);
```

Close an MMDB

```
int MMDBClose(void);
```

Set a password in the MMDB
The password is placed in the file ACCESS.CTL.

```
int SetPassword(char *password);
```

Get the password from the MMDB
The password is read from ACCESS.CTL.

```
int GetPassword(char *password, int passwordBufSize);
```

Add an exception word to the EXCEPTNS table

```
int AddExceptionWord(char *ExceptionWord);
```

Remove an exception word from the EXCEPTNS table

```
int RemoveExceptionWord(char *ExceptionWord);
```

Remove all exception words from the EXCEPTNS table

```
int RemoveAllExceptionWords(void);
```

Add or remove exception words listed in a file

```
int AddOrRemvExWords(char *FilePathName, int Action, int DoBeep);
```

Add a topic pointer definition to the MMDB

```
int AddTopicPtrDefinition(TOPICPTR TopicPtr, TOPICDEF TopicDef);
```

Remove a topic pointer definition from the MMDB

int RemoveTopicPtrDef(TOPICPTR TopicPtr);

Remove all topic pointer definitions from the MMDB

int RemoveAllTopicPtrDefs(void);

Add or delete topic definitions listed in a file

int AddOrRemvTopicDefs(char *FilePathName, int Action, int DoBeep);

Remove unreferenced topic pointer definitions

int RemoveUnreferencedTopPtrDefs(int DoBeep);

Make notification that an add has started

int NotifyStartOfAdd(void);

Add an object to the MMDB

int AddObject(struct ObjectInfo *objInfo);

Import an object into the MMDB
The import function adds an object to the MMDB just like AddObject() except that the information describing the object is stored with the object. The information describing the object is in a file called OBJECTIN.FO.

int ImportObject(char *SourcePath, struct ObjectInfo *objInfo);

Commit the add

int CommitAdd(void);

Roll back the add

int RollBackAdd(void);

Zero the use locks on an object

int ZeroUseLock(OBJECTHANDLE objHandle);

Update Object Files

```
int UpdateObjectFiles(OBJECTHANDLE objHandle, char *SourcePath, int Action);
```

Remove an object

```
int RemoveObject(OBJECTHANDLE objHandle);
```

Revert an AFC object from L to V

```
int RevertAFCObjToVirtual(OBJECTHANDLE objHandle);
```

Revert the least recently used (LRU) AFC object from L to V

```
int RevertLRUAFC(OBJECTHANDLE objHandle);
```

Remove all objects that have an add (+) status
This function is used to remove objects which were added and were not committed or back out successfully.

```
int RemoveObjsWithAddStatus(int DoBeep);
```

Get information about an object

```
int GetObjInfo(struct ObjectInfo *objInfo, INFOTYPE WhatInfo);
```

Get a copy of an object

```
int GetCopyOfObject(OBJECTHANDLE objHandle, char *DestPath);
```

Export an object from the MMDB
This function gets a copy of the object and places it in DestPath just like GetCopyOfObject(). But, in addition, it places a file called OBJECTIN.FO in DestPath that describes the object. An exported object can be imported into an MMDB using ImportObject().

```
int ExportObject(char *DestPath, OBJECTHANDLE objHandle);
```

Get a list of object classes and their descriptions

```
int GetListOfClasses(OBJECTCLASS **Class, int **ExpInd,
                     CLASSDESC **ClassDesc, int *numClasses);
```

Destroy the list of classes obtained with GetListOfClasses

This function deallocates the memory obtained for a list. No reference should be made to a list after this function is called.

```
void DestroyListOfClasses(void);
```

Get a list of topic definitions

```
int GetTopicDefinitionList(TOPICPTR **TopicPtr, TOPICDEF **TopicDef,
                           int *numTopics);
```

Destroy a topic definition list

This function deallocates memory obtained for a topic definition list. No reference should be made to a list after it is destroyed.

```
void DestroyTopicDefinitionList(void);
```

Create an AFC machine request

```
int CreateAFCRequest(OBJECTHANDLE objHandle, AFCREQUESTTYPE reqType);
```

Realize an AFC object (used by AFC machine)

```
int RealizeAFCObject(OBJECTHANDLE objHandle, char *SourcePath);
```

Add AFC objects from a Video Information File (VIF)

```
int AddVIF(char *VIFPath, int DoBeep, int *RecNum);
```

Change Search Defaults

```
int ChangeSearchDefaults(int maxTopics, int maxSearchWords,
                         int maxClasses, int maxStatuses);
```

Open a search

```
int OpenSearch(void);
```

Close a search

```
int CloseSearch(void);
```

Add topic to a search

```
int AddTopicToSearch(TOPICPTR TopicPtr);
```

<u>Add a search word to the search</u>

```
int AddSearchWordToSearch(SEARCHWORD SearchWord);
```

<u>Add a class to the search</u>

```
int AddClassToSearch(OBJECTCLASS objClass);
```

<u>Add a status to the search</u>

```
int AddStatusToSearch(OBJECTSTATUS objStatus);
```

<u>Perform the search and retrieve a list of found object handles</u>

```
int ObjectSearch(OBJECTHANDLE *objHandles, int *numHandles,
                 int Interruption, int *MouseX, int *MouseY,
                 int *SearchStatus);
```

<u>Experience an object</u>

```
int ExperienceObject(OBJECTHANDLE objHandle);
```

<u>Get an error message associated with an error number</u>

```
char *MDErrMsg(int err);
```

Create an instance of an MMDB

```
int CreateMMDB(char *path, char *MethodsSourcePath,
                    struct MMDBConfig *MMDBcfg);

int ret;
char path[MAXPATHLEN + -20 + 1], MethodsSourcePath[MAXPATHLEN + 1];
struct MMDBConfig MMDBcfg;
ret = CreateMMDB(path, MethodsSourcePath, &MMDBcfg);
```

path - [IN] path to directory which will hold the MMDB. This directory must exist before the function is called. The path should be no longer than MAXPATHLEN - 20 plus a NULL.

MethodSourcePath - [IN] the path to the directory holding the MMDB methods. The path should be no longer than MAXPATHLEN plus a NULL.

MMDBcfg - [IN] a structure holding the configuration for the MMDB to be created. The structure is given below.

```
struct MMDBConfig
    {
    unsigned long MinDiskFree;
    };
```

MinDiskFree - is the value of free disk space that the MMDB will attempt to not go below when adding a multimedia object.

Returns: MDSUCCESS - the MMDB was created successfully.
MDERR_MMDBISOPENERR - an MMDB is open. It must be closed before an MMDB can be created.
MDERR_STRINGTOOLONG - a specified path is too long.
MDERR_ERROROPENMMDBCFG4WRITE - unale to open the MMDB.CFG file for write operation.
MDERR_INVALIDPATH - the path specified for the location of the MMDB cannot be located.
MDERR_NOATTACHTOSERVER - a volume was specified, but the workstation is not attached to a server.
MDERR_VOLDOESNOTEXIST - the volume specified in the path does not exist.

MDERR_UNABLETOSETDRV - unable to map a network drive.
MDERR_METHODSOURCENOTLOC - the source directory for the methods or the directory for one of the methods cannot be located.
MDERR_ERRSETTINGATTRIB - error setting attributes on files in \MMDB\TABLES.
MDERR_NOTABLEFILESFOUND - no files were found in \MMDB\TABLES.
MDERR_PATHCREATEERR - error creating a directory with \MMDB.
Other errors provide indication of reason for failure.

Open an MMDB

int MMDBOpen(char *path, int restorescreen);

int ret, restorescreen;
char path[MAXPATHLEN - 20 + 1];
ret = MMDBOpen(path, restorescreen);

path - [IN] path to directory holding the MMDB. The path should be no longer than MAXPATHLEN - 20 plus a NULL.

restorescreen - [IN] indicates whether the screen should be restored following a call to ExperienceObject(). A screen restoration can only happen in a text mode. Options are RESTORESCREEN and DONTRESTORESCREEN.

Returns: MDSUCCESS - the MMDB was opened successfully.
MDERR_MMDBALREADYOPEN - the MMDB is already open.
MDERR_INVALIDPATH - the MMDB cannot be located.
MDERR_NOATTACHTOSERVER - a volume was specified, but the orkstation is not attached to a server.
MDERR_VOLDOESNOTEXIST - the volume specified in the path does not exist.
MDERR_UNABLETOMAPDRV - unable to map a network drive.
MDERR_MMDBDOESNTEXIST - the MMDB doesn't exist.
MDERR_TABLEPATHMISS - the TABLES directory within the MMDB cannot be located.
MDERR_OBJPATHMISS - the OBJECTS directory within the MMDB cannot be located.
MDERR_REQUPATHMISS - the REQUESTS directory within the MMDB cannot be located.
MDERR_METHODPATHMISS - the METHODS directory within the MMDB cannot be located.
MDERR_CANTGETUSERID - unable to get user ID.
MDERR_ERROPENMMDBCFG4READ - error opening the MMDB.CFG file for read.
MDERR_STRINGTOOLONG - the path is too long.
MDERR_LOCKTIMEOUT - a lock timeout occured attempting to open the tables.
Other errors indicate reason for failure to open the MMDB.

Close an MMDB

```
int MMDBClose(void);

int ret;
ret = MMDBClose();
```

Returns: MDSUCCESS - MMDB was closed successfully.
MDERR_MMDBNOTOPEN -the MMDB is not open, so a close cannot be performed.
Other errors indicate reason for failure to close the MMDB properly.

Set a password in the MMDB

The password is placed in the file ACCESS.CTL.

```
int SetPassword(char *password);

int ret;
char password[41];
ret = SetPassword(password);
```

password - [IN] is the password to be set in the MMDB.

Returns: MDSUCCESS - the password has been set successfully.
MDERR_ERROPENACCCTL4WRITE - error opening the file ACCESS.CTL for write operation.
MDERR_ERRWRITINGACCCTL - an error occurred writing to the ACCESS.CTL file.

Get the password from the MMDB

The password is read from ACCESS.CTL.

int GetPassword(char *password, int passwordBufSize);

int ret;
char password[41];
int passwordBufSize;
passwordBufSize = sizeof(password);
ret = GetPassword(password, passwordBufSize);

password - [OUT] this variable holds the returned password.
passwordBufSize - [IN] the number of bytes allocated for the password variable.

Returns: MDSUCCESS - the password has been retrieved successfully.
MDERR_ERROPENACCCTL4READ - an error occurred opening the ACCESS.CTL file for read operation.
MDERR_BUFTOOSMALL - the password buffer is too small to hold the password.

Add an exception word to the EXCEPTNS table

```
int AddExceptionWord(char *ExceptionWord);

int ret;
char ExceptionWord[MAXWORDINDEXLEN + 1];
ret = AddExceptionWord(ExceptionWord);
```

ExceptionWord - [IN] is the exception word to be added to the EXCEPTNS table. The exception word should be no longer than MAXWORDINDEXLEN plus a NULL.

Returns: MDSUCCESS - the exception word was added sucessfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_DUPLICATIONOFENTRY - the exception word is a duplicate of an exception word already in the EXCEPTNS table.
Other errors provide indication as to why an exception word could not be added sucessfully.

Remove an exception word from the EXCEPTNS table

```
int RemoveExceptionWord(char *ExceptionWord);

int ret;
char ExceptionWord[MAXWORDINDEXLEN + 1];
ret = ExceptionWord(ExceptionWord);
```

ExceptionWord - [IN] is the exception word to be removed from the EXCEPTNS table. The exception word should be no longer than MAXWORDINDEXLEN plus a NULL.

Returns: MDSUCCESS - the exception word was removed successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_EXWORDISNOTINTABLE - the exception word was not found in the EXCEPTNS table.
Other errors provide indication as to why an exception word could not be successfully removed.

Remove all exception words from the EXCEPTNS table

```
int RemoveAllExceptionWords(void);

int ret;
ret = RemoveAllExceptionWords();
```

Returns: MDSUCCESS - all exception words were removed from the EXCEPTNS table.
MDERR_MMDBNOTOPEN - open the MMDB first.
Other errors rpovide indication as to why an exception word could not be successfully removed.

Add or remove exception words listed in a file

int AddOrRemvExWords(char *FilePathName, int Action, int DoBeep);

int ret, Action, DoBeep;
char FilePathName[MAXPATHLEN + 1];
ret = AddOrRemvExWords(FilePathName, Action, DoBeep);

FilePathName - [IN] the path and filename for the file holding the list of exception words to be added to or removed from the EXCEPTNS table. The FilePathName should be no longer than MAXPATHLEN plus a NULL.

Action - [IN] the action to be taken. The options are: ADDBUTCLEARFIRST, ADD, and REMOVE. ADDBUTCLEARFIRST adds exception words, but clears all exception words from the EXCEPTNS table first. ADD adds exception words to the EXCEPTNS table. REMOVE removes exception words from the EXCEPTNS table.

DoBeep - [IN] this parameter determines whether or not a beep sounds each time an exception word is processed. The options are YES and NO.

Returns: MDSUCCESS - the exception words were added or removed successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_OPENEREXWORDSFILE - an error occurred attempting to open the file specified in SourcePath for read operation.
MDERR_ERRREADINGEXWORDSFILE - an error occurred reading exception words from the file.
MDERR_INVALIDACTION - an invalid action was specified.
Other errors provide indication as to why the addition or removal of exception words was not successful.

Add a topic pointer definition to the MMDB

int AddTopicPtrDefinition(TOPICPTR TopicPtr, TOPICDEF TopicDef);

int ret;
TOPICPTR TopicPtr;
TOPICDEF TopicDef;
ret = AddTopicPtrDef(TopicPtr, TopicDef);

TopicPtr - [IN] a character string which references a topic. The string should not be longer than MAXTOPICPTRLEN plus a NULL.

TopicDef - [IN] a string holding the definition of a topic. The string should be no longer than MAXTOPICDEFLEN plus a NULL.

Returns: MDSUCCESS - the topic reference was added to the MMDB.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_DUPLICATIONOFENTRY - the TopicPtr value matches a topic pointer already stored in the database.
Other errors indicate why a topic reference could not be added to the MMDB.

Remove a topic pointer definition from the MMDB

int RemoveTopicPtrDef(TOPICPTR TopicPtr);

```
int ret;
TOPICPTR TopicPtr;
ret = RemoveTopicPtrDef(TopicPtr);
```

TopicPtr - [IN] a character string which points to a topic pointer definition which is to be removed from the MMDB. The string should be no longer than MAXTOPCINDEXLEN plus a NULL.

Returns: MDSUCCESS - the topic reference was removed from the MMDB successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_TOPCPTRNOTINTABLE - the topic pointer definition could not be located.
Other errors indicate why a topic reference could not be removed from the MMDB.

Remove all topic pointer definitions from the MMDB

```
int RemoveAllTopicPtrDefs(void);

int ret;
ret = RemoveAllTopicPtrDefs();
```

Returns: MDSUCCESS - all topic references have been removed successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
Other errors indicate why the topic references could not be removed from the MMDB.

Add or delete topic definitions listed in a file

```
int AddOrRemvTopicDefs(char *FilePathName, int Action, int DoBeep);

int ret, Action, DoBeep;
char FilePathName[MAXPATHLEN + 1];
ret = AddOrRemvTopicDefs(FilePathName, Action, DoBeep);
```

FilePathName - [IN] path and filename of the file holding the topic references to be added to or removed from the MMDB. FilePathName should be no longer than MAXPATHLEN plus a NULL.

Action - [IN] the action to be taken. Choices are: ADDBUTCLEARFIRST, ADD, REMOVE. ADDBUTCLEARFIRST adds the topic pointer definitions from the file, but clears all previously stored topic pointer definitions first. ADD adds topic pointer definitions from the file and REMOVE removes the topic pointer definitions from the file.

DoBeep - [IN] the value provided for DoBeep determines whether or not a beep sounds for each topic pointer definition processed. Options are YES and NO.

Returns: MDSUCCESS - the topic pointer definitions were successfully added or removed.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_OPENERRTOPPTRDEFFILE - an error occurred opening the topic pointer definition file for read.
MDERR_ERRREADTOPPTRDEFFILE - an error occurred reading the topic pointer definition file.
MDERR_ERRINTOPCDEFFILE - there is an error in the topic pointer definition file.
MDERR_INVALIDACTION - an invalid action was specified.
Other errors provide indication as to why adding or deleting topic pointer definitions did not complete successfully.

File format for topic definition file:

Each topic entry is composed of two parts: a topic pointer and a topic definition. The topic pointer has a maximum length of 20 characters and the topic definition has a maximum length of 40 characters. The topic pointer and the topic definition are separated by a █ (Char 219 decimal). A topic entry is concluded with a carriage return-linefeed. The end of a topic definition file is signaled with a █ and a carriage return line feed as the first characters of a record.

Entry format:
<topic pointer>█<topic definition><crlf>
Example:
FORGOVT█Foreign Affairs<crlf>

Sample topic definition file:
FORGOVT█Foreign Affairs<crlf>
MATH█Mathematics<crlf>
PHYSICS█Physics<crlf>
█<crlf>

Remove unreferenced topic pointer definitions

int RemoveUnreferencedTopPtrDefs(int DoBeep);

int ret, DoBeep;
ret = RemoveUnreferencedTopPtrDefs(DoBeep);

DoBeep - [IN] indicates whether a beep should sound as objects are processed. Options are YES and NO.

Returns: MDSUCCESS - unreferenced topic pointer definitions were removed successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHOPEN - a search is open. Close the search before using this function.
Other errors provide indication as to why removal of the unreferenced topic pointer definitions failed.

<u>Make notification that an add has started</u>

```
int NotifyStartOfAdd(void);

int ret;
ret = NotifyStartOfAdd();
```

Returns: MDSUCCESS - A start of add notification has been successfully made.
MDERR_MMDBNOTOPEN - The MMDB is not open; open it first.
MDERR_ADDINPROGRESS -An add is already in progress.
Other errors indicate reason for failure to make notification of the start an add.

Add an object to the MMDB

```
int AddObject(struct ObjectInfo *objInfo);

int ret;
struct ObjectInfo objInfo;
ret = AddObject(&objInfo);
```

objInfo - [IN/OUT] is a structure which holds information about the object to be added. What information must be placed in objInfo depends on the class of the object being added. The structure and the dependencies are provided below:

Dependencies:

If the object has a class of AFC, the following information must be placed in the objInfo structure:

objClass, objTitle, objWordList, objTopcList, objAFCType, objAFCDiskName, objAFCDiskSide, objAFCAudInfo, objColorInf, objAFCStartPos, objAFCEndPos

In addition, if the AFC object has a type (objAFCType) of SFL objSFLFrameLst must be supplied.

If the object is a member of any other class, the following information must be placed in the objInfo structure:

objClass, objTitle, objReferent, objWordList, objTopcList, objAddSourcePath

Returns: MDSUCCESS - the object was added to the MMDB successfully.

MDERR_MMDBNOTOPEN - the MMDB must be opened first.

MDERR_ADDNOTSTARTED - use NotifyStartOfAdd() first.

MDERR_STRUCTCORRUPT - the objInfo structure has been corrupted. This happens when a string has overwritten a variable or if a required piece of information was not placed in the structure.

MDERR_INVALIDCLASS - an unknown object class was specified.

MDERR_OPENOBLIST - an error occurred opening the file holding the list of objects which have been added to the MMDB for write operation.

MDERR_ERRWRITEOBLIST - an error occurred writing to the file holding the list of objects which have been added to the MMDB.

MDERR_SOURCEPATHNOTFOUND - the source path for the object being added cannot be located.

MDERR_INSUFFICIENTDISKSPACE - there is not enough free disk space to store the object in the MMDB.

MDERR_OBJFILECOPYERR - an error was encountered copying the object's file(s).

MDERR_OBJDIRCREATEERR - error occurred creating a directory for the object.
Other errors provide indication as to why the add operation failed.

Import an object into the MMDB

The import function adds an object to the MMDB just like AddObject() except that the information describing the object is stored with the object. The information describing the object is in a file called OBJECTIN.FO.

```
int ImportObject(char *SourcePath, struct ObjectInfo *objInfo);

int ret;
char SourcePath[MAXPATHLEN];
struct ObjectInfo objInfo;
ret = ImportObject(SourcePath, &objInfo);
```

SourcePath - [IN] the path to the directory holding the object to be imported.
objInfo - [OUT] returned information about the imported object.

Returns: MDSUCCESS - the object was successfully imported.
MMDBNOTOPEN - open the MMDB first.
MDERR_STRINGTOOLONG - SourcePath is longer than MAXPATHLEN plus a NULL.
MDERR_ERROPENOBJINF4READ - error opening OBJECTIN.FO for read operation.
MDERR_ERRREADINGOBJINFO - error reading the file OBJECTIN.FO.
MDERR_OBJINFOVERLEVWRONG - the version level of the OBJECTIN.FO file is wronng.
MDERR_SOURCEPATHNOTFOUND - the source path cannot be located.
MDERR_ADDNOTSTARTED - the add dis not start properly.
MDERR_STRUCTCORRUPT - the objInfo structure has been corrupted. This happens when a string has overwritten a variable or if a required piece of information was not placed in the structure.
MDERR_INVALIDCLASS - an unknown object class was specified.
MDERR_OPENOBLIST - an error occurred opening the file holding the list of objects which have been added to the MMDB for write operation.
MDERR_ERRWRITEOBLIST - an error occurred writing to the file holding the list of objects which have been added to the MMDB.
MDERR_INSUFFICIENTDISKSPACE - there is not enough free disk space to store the object in the MMDB.
MDERR_OBJFILECOPYERR - an error was encountered copying the object's file(s).
MDERR_OBJDIRCREATEERR - error occurred creating a directory for the object.
Other errors provide indication as to why the add operation failed.

Commit the add

```
int CommitAdd(void);

int ret;
ret = CommitAdd();
```

Returns: MDSUCCESS - the commit operation has succeeded.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_ADDNOTSTARTED - use NotifyStartOfAdd() first.
MDERR_ERROPENOBLIST4RD - an error occurred opening the file holding the list of objects, which have been added, for read operation.
MDERR_COMMITNEADD - the number of committed objects does not equal the number of added objects.
Other errors provide indication as to why the commit of added objects has failed.

Roll back the add

```
int RollBackAdd(void);

int ret;
ret = RollBackAdd();
```

Returns: MDSUCCESS - the roll back operation has succeeded.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_ADDNOTSTARTED - use NotifyStartOfAdd() first.
MDERR_ERROPENOBLIST4RD - an error occurred opening the file holding the list of objects which have been added for read operation.
MDERR_ROLLBKNEADD - the number of objects rolled back does not equal the number of added objects.
Other errors provide indication as to why the roll back of added objects has failed.

Zero the use locks on an object

```
int ZeroUseLock(OBJECTHANDLE objHandle);

int ret;
OBJECTHANDLE objHandle;
ret = ZeroUseLock(objHandle);
```

objHandle - [IN] handle to the object for which the use lock value is to be zeroed.

Returns: MDSUCCESS - the use lock value has been zeroed.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_OBJREFNOTFOUND - the object could not be located.
MDERR_LOCKTIMEOUT - a lock timeout occurred getting a lock on the record.
Other errors indicate reason for not being able to zero the use locks on an object.

Update Object Files

```
int UpdateObjectFiles(OBJECTHANDLE objHandle, char *SourcePath, int Action);

int ret, Action;
OBJECTHANDLE objHandle;
char SourcePath[MAXPATHLEN + 1];
ret = UpdateObjectFiles(objHandle, SourcePath, Action);
```

objHandle - [IN] handle to the object that is to have its files updated.
SourcePath - [IN] path to the directory holding the files to be copied into the object's directory. The path should be no longer than MAXPATHLEN plus a NULL.
Action - [IN] determines how the update occurs. The options are ADD and ADDBUTCLEARFIRST. ADD copies the files from SourcePath to the object directory overwriting any files with the same name. ADDBUTCLEARFIRST erases all the files in the object directory before copying the files from SourcePath to the object directory.

Returns: MDSUCCESS - the object's files were updated successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_OBJREFNOTFOUND - the object could not be located.
MDERR_SOURCEPATHNOTFOUND - the source path cannot be located.
MDERR_OBJFILECOPYERR - an error occurred copying the files to the onject's directory.
MDERR_ERRSETTINGSTTRIB - an error occurred setting the object files' attributes.
MDERR_STRINGTOOLONG - the path is longer than MAXPATHLEN plus a NULL or the object handle is too long.
MDERR_WRONGSTATUS4REQ - the object has a status which is incompatible with fulfilling the request. The status must be an L.
MDERR_INVALIDACTION - an invalid action was specified.
MDERR_USELOCKONOBJECT - a use lock is on the object. The object cannot be updated until the number of use locks is zero.
Other errors provide indication as to why the file update failed.

Remove an object

```
int RemoveObject(OBJECTHANDLE objHandle);

int ret;
OBJECTHANDLE objHandle;
ret = RemoveObject(objHandle);
```

objHandle - [IN] handle to the object that is to be removed.

Returns: MDSUCCESS - the object has been removed successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_OBJREFNOTFOUND - the object could not be located.
MDERR_USELOCKONOBJECT - there is a use lock on the object. It cannot be removed until the number of use locks is zero.
MDERR_ERRREMOVINGOBJFILES - an error occurred removing the object files from the object directory.
MDERR_STRINGTOOLONG - the object handle is too long.
Other errors provide indication as to why the removal of the object failed.

Revert an AFC object from L to V

```
int RevertAFCObjToVirtual(OBJECTHANDLE objHandle);

int ret;
OBJECTHANDLE objHandle;
ret = RevertAFCObjToVirtual(objHandle);
```

objHandle - [IN] handle to the object that is to be reverted from L to V.

Returns: MDSUCCESS - the object has be reverted successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERROBJREFNOTFOUND - the object could not be located.
MDERR_USELOCKONOBJECT - there is a use lock on the object. It cannot be reverted until the number of use locks is zero.
MDERR_ALREADYVIRTUAL - the AFC object already is virtual (has a status of V).
MDERR_WRONGCLASS4REQ - the object is NOT an AFC object.
MDERR_ERRREMOVINGOBJFILES - an error occurred removing the object files from the object directory.
MDERR_STRINGTOOLONG - the object handle is too long.
Other errors provide indication as to why the revert operation did not complete successfully.

Revert the least recently used (LRU) AFC object from L to V

```
int RevertLRUAFC(OBJECTHANDLE objHandle);

int ret;
OBJECTHANDLE objHandle;
ret = RevertLRUAFC(objHandle);
```

objHandle - [IN] the handle of the deleted object is returned in objHandle.

Returns: MDSUCCESS - an LRU AFC object with a status of L was found and reverted to V.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_NOLRUFOUND - no LRU AFC object with a status of L was found.
MDERR_STRINGTOOLONG - the object handle is too long.
Other errors provide indication as to why the revert LRU did not complete successfully.

Remove all objects that have an add (+) status

This function is used to remove objects which were added and were not committed or back out successfully.

```
int RemoveObjsWithAddStatus(int DoBeep);
```

```
int ret, DoBeep;
ret = RemoveObjsWithAddStatus(DoBeep);
```

DoBeep - [IN] indicates whether a beep should sound as objects are processed. Options are YES and NO.

If, for some reason (I don't know how), there is a use lock on an object with an add status, this object will be skipped.

Returns: MDSUCCESS - objects with a status of + were successfully removed (even if number removed is zero).
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHOPEN - a search is open. The search must be closed before this function can be used.
Other errors provide indication as to why the removal of objects with a status of + did not complete successfully.

Get information about an object

int GetObjInfo(struct ObjectInfo *objInfo, INFOTYPE WhatInfo);

int ret;
INFOTYPE WhatInfo;
struct ObjectInfo objInfo;
ret = GetObjInfo(&objInfo, WhatInfo);

objInfo - [IN/OUT] the structure in which the information will be returned. The information which will be placed in objInfo depends on the setting of WhatInfo. The object handle for the object for which information is to be obtained should be placed in the objHandle variable in the objInfo structure. For more information on the ObjectInfo structure see the section titled ObjectInfo Structure Description.

WhatInfo - determine the information that will be returned by GetObjectInfo(). The choices are: OBJINFOINFO, CLASSINFO, STATUSINFO, REFERENTINFO, TITLEINFO, WORDLISTINFO, TOPCLISTINFO, LISTSINFO, SIZEINFO, AFCINFO, and ALLINFO. The structure variables that will be filled for each WhatInfo choice is provided below:

OBJINFOINFO - objClass, objStatus, objLastUse, objNumUses, objWhenAdd, objUseLocks
CLASSINFO - objClass
STATUSINFO - objStatus
REFERENTINFO - objReferent
TITLEINFO - objTitle
WORDLISTINFO - objWordList
TOPCLISTINFO - objTopcList
LISTSINFO - objWordList, objTopcList
SIZEINFO - objSize
AFCINFO - objClass, objStatus, objLastUse, objNumUses, objWhenAdd, objUseLocks, objTitle, objAFCType, objAFCDiskSide, objAFCAudInfo, objAFCStartPos, objAFCEndPos, objSFLFrameLst (if type is SFL)
ALLINFO - all applicable structure variables are filled.

Returns: MDSUCCESS - the requested information was obtained successfully.
MDERR_MMDBNOTOPEN - open the MMDB open first.
MDERR_STRUCTCORRUPT - the objInfo structure is corrupt. This occurs if the object handle placed in the structure overwrote the variable boundary or was not null terminated.
MDERR_INVALIDINFOTYPE - WhatInfo contains an unknown value.

MDERR_WRONGCLASS4REQ - an info type was specified that is not valid for the class of the object referenced by the object handle. (Eg. if AFCINFO is specified for an object that is of not AFC type.)
MDERR_WRONGSTATUS4REQ - the object has the wrong status to fulfill the request.
MDERR_OBJREFNOTFOUND - the object could not be located.
Other errors indicate why getting the information was not successfull.

Get the size of an object in bytes

```
int GetObjectSize(OBJECTHANDLE objHandle, unsigned long *Size);

int ret;
OBJECTHANDLE objHandle;
unsigned long Size;
ret = GetObjectSize(objHandle, &Size);
```

objHandle - [IN] handle to the object for which size information is to be obtained.

Size - [OUT] the size of the object is placed in Size upon successfull execution of the function.

Returns: MDSUCCESS - the size information was obtained successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_OBJREFNOTFOUND - the object could not be located.
MDERR_WRONGSTATUS4REQ - the object does not have a status of L.
MDERR_STRINGTOOLONG - the object handle is too long. Other errors indicate why the size of the object could not be determined.

Get a copy of an object

```
int GetCopyOfObject(OBJECTHANDLE objHandle, char *DestPath);

int ret;
OBJECTHANDLE objHandle;
char DestPath[MAXPATHLEN + 1];
ret = GetCopyOfObject(objHandle, DestPath);
```

objHandle - [IN] handle to the object for which a copy is to be obtained.
DestPath - [IN] the path to the directory which is to hold the copy of the object. The path should be no longer than MAXPATHLEN plus a NULL. The directory pointer to by DestPath must exist before this function is called.

Returns: MDSUCCESS - a copy of the object was obtained successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_WRONGSTATUS4REQ - the object does not have a status of L.
MDERR_OBJREFNOTFOUND - the object could not be located.
MDERR_ERRCOPYINGOBJ - an error occurred copying the object.
MDERR_STRINGTOOLONG - the path is longer than MAXPATHLEN plus a NULL or the object handle is too long.
Other errors indicate why the copy did not complete successfully.

Export an object from the MMDB

This function gets a copy of the object and places it in DestPath just like GetCopyOfObject(). But, in addition, it places a file called OBJECTIN.FO in DestPath that describes the object. An exported object can be imported into an MMDB using ImportObject().

```
int ExportObject(char *DestPath, OBJECTHANDLE objHandle);
```

```
int ret;
char DestPath[MAXPATHLEN + 1];
OBJECTHANDLE objHandle;
ret = ExportObject(DestPath, objHandle);
```

DestPath - [IN] is the location to place the exported object. DestPath should be no longer than MAXPATHLEN plus a NULL.
objHandle - [IN] is the handle for the object to be exported. objHandle should be no longer than OBJHANDLELEN plus a NULL.

Returns: MDSUCCESS - the object was successfully exported to the directory pointed to by DestPath.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_WRONGSTATUS4REQ - the object has the wrong status to be exported.
MDERR_ERROPENOBJINF4WRITE - error opening the OBJECTIN.FO file for write operation.
MDERR_ERRWRITINGTOOBJINF - error writing to the OBJECTIN.FO file.
MDERR_OBJREFNOTFOUND - the object could not be located.
MDERR_ERRCOPYINGOBJ - an error occurred copying the object.
MDERR_STRINGTOOLONG - DestPath is longer than MAXPATHLEN plus a NULL or the object handle is too long.
Other errors indicate why the export function did not complete successfully.

Get a list of object classes and their descriptions

```
int GetListOfClasses(OBJECTCLASS **Class, int **ExpInd,
            CLASSDESC **ClassDesc, int *numClasses);

int ret, NumClasses;
OBJECTCLASS *Class;
int *ExpInd;
CLASSDESC *ClassDesc;
ret = GetListOfClasses(&Class, &ExpInd, &ClassDesc, &numClasses);
```

Class - [OUT] holds the list of object classes.
ExpInd - [OUT] holds a list of experience indicators. If the indicator is set to 1 the class of object can be experienced, if the indicator is set to 0 the class of object cannot be experienced.
ClassDesc - [OUT] holds the list of class descriptions.
numClasses - [OUT] the number of classes which have been returned.

Returns: MDSUCCESS - the list of classes was obtained successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_MEMALLOCERROR - error allocating memory for the list.
Other errors provide indication as to why a list could not successfully be obtained.

Destroy the list of classes obtained with GetListOfClasses

This function deallocates the memory obtained for a list. No reference should be made to a list after this function is called.

void DestroyListOfClasses(void);

Returns: none.

Note: no harm will be done if this function is called if no list exists. :)

Get a list of topic definitions

int GetTopicDefinitionList(TOPICPTR **TopicPtr, TOPICDEF **TopicDef,
int *numTopics);

int ret, numTopics;
TOPICPTR *TopicPtr;
TOPICDEF *TopicDef;
ret = GetTopicRefList(&TopicPtr, &TopicDef, &numTopics);

TopicPtr - [OUT] holds the returned topic pointer strings
TopicDef - [OUT] holds the returned topic definition strings
numTopics - [OUT] is the number of returned topic definitions

Returns: MDSUCCESS - the topic reference list was retrieved successflly.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_MEMALLOCERROR - error allocating memory for the list.
Other errors provide indication as to why a list could not be obtained successfully.

Destroy a topic definition list

This function deallocates memory obtained for a topic definition list. No reference should be made to a list after it is destroyed.

```
void DestroyTopicDefinitionList(void);
```

Returns: none.

Note: no harm will be done if this function is called if no list exists. :)

Create an AFC machine request

int CreateAFCRequest(OBJECTHANDLE objHandle, AFCREQUESTTYPE reqType);

```
int ret;
OBJECTHANDLE objHandle;
AFCREQUESTTYPE reqType;
ret = CreateAFCRequest(objHandle, reqType);
```

objHandle - [IN] handle to the object for which an AFC machine request is to be created.

reqType - [IN] the type of AFC request that is to be made. Choices are: LSA, LMS, LSB, LAD, and LPM. There is a dependency between allowed request types and the object's AFC type. The SFL AFC type will allow LSA, LSB, LMS, and LAD requests. The SEG AFC type will allow LSA, LMS, and LAD requests. The SFL AFC type will allow an LPM request.

Returns: MDSUCCESS - the request was created.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_AFCOBJEXISTS - the object already exists.
MDERR_AFCOBJALREADYREQ - the object has already been requested.
MDERR_WRONGSTATUS4REQ - object has wrong status to be requested.
MDERR_INVALIDREQTYPE - an invalid request type was specified.
MDERR_INVALIDAFCTYPE - an invalid AFC type was encountered.
MDERR_REQFILEOPENERR - an error occurred opening a request file for write operation.
MDERR_REQWRITEERR - an error occurred writing to the request file.
MDERR_STRINGTOOLONG - object handle is too long.
Other errors provide indication of why the creation of a request met with an error.

Realize an AFC object (used by AFC machine)

int RealizeAFCObject(OBJECTHANDLE objHandle, char *SourcePath);

int ret;
OBJECTHANDLE objHandle;
char SourcePath[MAXPATHLEN + 1];
ret = RealizeAFCObject(objHandle, SourcePath);

objHandle - [IN] is the handle to the object which is to be realized (changed from V to L).
SourcePath - [IN] the path to the object's files which are to be placed in the MMDB.

Returns: MDSUCCESS - the object was "realized" successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_WRONGCLASS4REQ - the object has the wrong class to perform the action.
MDERR_WRONGSTATUS4REQ - the object has the wrong status to perform the action.
MDERR_SOURCEPATHNOTFOUND - the directory pointer to by source path cannot be located.
MDERR_INSUFFICIENTDISKSPACE - there is not enough free disk space to perform the action.
MDERR_OBJDIRCREATEERR - an error occurred creating a directory to hold the object's files.
MDERR_NOSOURCEFILESFOUND - no files could be found in the directory pointed to by SourcePath.
MDERR_FILECOPYERR - an error occurred copying the object's files to the object directory.
MDERR_ERRSETTINGSTTRIB - an error occurred setting the object files' attributes.
MDERR_STRINGTOOLONG - the path is longer than MAXPATHLEN plus a NULL or the object handle is too long.
Other errors provide indication as to why the operation did not succeed.

Add AFC objects from a Video Information File (VIF)

```
int AddVIF(char *VIFPath, int DoBeep, int *RecNum);

int ret, DoBeep, RecNum;
char VIFPath[MAXPATHLEN + 1];
ret = AddVIF(VIFPath, DoBeep, &RecNum);
```

VIFPath - [IN] is the path to the directory holding the VIF along with the name of the VIF. (Eg. A:\VIFS\CELVE001.VIA)

DoBeep - [IN] determines if a beep sound each time an AFC object is added to the MMDB.

RecNum - [OUT] indicates the number of entries in the VIF processed before the function returned. If the function returned with MDSUCCESS, then it is the number of AFC objects added to the MMDB. If the function returns with an error, it is the record in the VIF that was being processed when the error occurred.

Returns: MDSUCCESS - the AFC objects in the VIF file were added successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_STRINGTOOLONG - the VIFPath is too long. Maximum length is MAXPATHLEN.
MDERR_CANNOTOPENVIF - the VIF file could not be opened for read operation.
MDERR_READINGVIF - an error occurred reading the VIF file.
MDERR_ROLLBACKADDFAILED - after an error the rollback of the AFC objects already added to the MMDB failed.
Other errors provide indication as to why AddVIF() failed.

Change Search Defaults

```
int ChangeSearchDefaults(int maxTopics, int maxSearchWords,
                         int maxClasses, int maxStatuses);

int ret, maxTopics, maxSearchWords, maxClasses, maxStatuses;
ret = ChangeSearchDefaults(maxTopics, maxSearchWords,
                maxClasses, maxStatuses);
```

maxTopics - [IN] the maximum number of topics that may be specified in a search.
maxSearchWords - [IN] the maximum number of search words that may be specified in a search.
maxClasses - [IN] the maximum number of classes that may be specified in a search.
maxStatuses - [IN] the maximum number of statuses that may be specified in a search.

Returns: MDSUCCESS - the defaults were changed successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHOPEN - a search is open. The defaults cannot be changed while a search is in progress.

Open a search

```
int OpenSearch(void);

int ret;
ret = OpenSearch();
```

Returns: MDSUCCESS - the search was opened successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHALREADYOPEN - a search is already open.
MDERR_MEMALLOCERROR - an error occurred allocating memory for the search.

Close a search

```
int CloseSearch(void);

int ret;
ret = CloseSearch();
```

Returns: MDSUCCESS - the search was closed successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHNOTOPEN - a search is not open to close.

<u>Add topic to a search</u>

```
int AddTopicToSearch(TOPICPTR TopicPtr);

int ret;
TOPICPTR TopicPtr;
ret = AddTopicToSearch(TopicPtr);
```

TopicPtr - [IN] a topic pointer to be added to the search.

Returns: MDSUCCESS - the addition of the topic pointer was successful.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHNOTOPEN - open a search first.
MDERR_MAXTOPICSREACHED - the maximum number of topics has been reached. The maximum can be changed in ChangeSearchDefaults().
MDERR_DUPLICATIONOFENTRY - the topic pointer is a duplicate of one already in the search.
MDERR_SEARCHINPROGRESS - a search is in progress. A new topic may not be added to the search once it is in progress.

Add a search word to the search

```
int AddSearchWordToSearch(SEARCHWORD SearchWord);

int ret;
SEARCHWORD SearchWord;
ret = AddSearchWordToSearch(SearchWord);
```

SearchWord - [IN] the search word to be added to the search.

Returns: MDSUCCESS - the addition of the search word was successful.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHNOTOPEN - open a search first.
MDERR_MAXSEARCHWORDSREACHED - the maximum number of search words has been reached. The maximum can be changed in ChangeSearchDefaults().
MDERR_DUPLICATIONOFENTRY - the search word is a duplicate of one already in the search.
MDERR_SEARCHINPROGRESS - a search is in progress. A new search word may not be added to the search once it is in progress.

Add a class to the search

```
int AddClassToSearch(OBJECTCLASS objClass);

int ret;
OBJECTCLASS objClass;
ret = AddClassToSearch(objClass);
```

objClass - [IN] the class to add to the search.

Returns: MDSUCCESS - the addition of an object class was successful.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHNOTOPEN - open a search first.
MDERR_MAXCLASSESREACHED - the maximum number of classes has been reached. The maximum can be changed in ChangeSearchDefaults().
MDERR_DUPLICATIONOFENTRY - the object class is a duplicate of one already in the search.
MDERR_SEARCHINPROGRESS - a search is in progress. A new class may not be added to the search once it is in progress.

Add a status to the search

int AddStatusToSearch(OBJECTSTATUS objStatus);

int ret;
OBJECTSTATUS objStatus;
ret = AddStatusToSearch(objStatus);

objStatus - [IN] the status to be added to the search.

Returns: MDSUCCESS - the addition of an object status was successful.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHNOTOPEN - open a search first.
MDERR_MAXSTATUSESREACHED - the maximum number of statuses has been reached. The maximum can be changed in ChangeSearchDefaults().
MDERR_DUPLICATIONOFENTRY - the object status is a duplicate of one already in the search.
MDERR_SEARCHINPROGRESS - a search is in progress. A new status may not be added to the search once it is in progress.

Perform the search and retrieve a list of found object handles

```
int ObjectSearch(OBJECTHANDLE *objHandles, int *numHandles,
                 int Interruption, int *MouseX, int *MouseY,
                 int *SearchStatus);

int ret, numHandles, Interruption, MouseX, MouseY, SearchStatus;
OBJECTHANDLE objHandles[10];
numHandles = 10;
ret = ObjectSearch(objHandles, &numHandles, Interruption,
                   &MouseX, &MouseY, &SearchStatus);
```

objHandles - [OUT] is a pointer to an array of object handles. The object handles found during the search are placed in this array.

numHandles - [IN/OUT] on the call of the function the number of handles that the objHandles array can hold (or some smaller number) is to be placed in numHandles. On a return from the function, numHandles holds the number of object handles that have been found that match the search criteria.

Interruption - [IN] this variable determines whether a mouse or keyboard action will interrupt the search. There are four choices for Interruption: NOINTERRUPT, MOUSE, KEY and MOUSEKEY. NOINTERRUPT specifies that no interruption should be allowed. MOUSE specifies that a mouse button click can interrupt the search. KEY specifies that a keypress can interrupt the search. MOUSEKEY specifies that a mouse click and/or a keypress can interrupt the search.

MouseX, MouseY - [OUT] MouseX and MouseY hold the X and Y coordinates of the mouse if a MOUSE or MOUSEKEY interruption occurred.

SearchStatus - [OUT] indicates the status of a search upon return of the function. The possible values are FINISHED and NOTFINISHED.

Returns: MDSUCCESS - the search operation was successful.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_SEARCHNOTOPEN - open a search first.
MDERR_INVALIDSOURCE - an invalid source was encountered (this should never happen).
MDERR_INTERUPTERR - an error was encountered while checking for an interruption.
MDERR_KEYINTERRUPT - a keypress interruption occurred.
MDERR_MOUSEINTERRUPT - a mouse button press interruption occurred.

MDERR_MOUSEKEYINTERRUPT - a mouse button and a keypress interruption occurred.
Other errors provide indication as to why a search has met with failure.

Since ObjectSearch can be interrupted, it can be used to search for objects in the database while the user is looking over information returned about previously located objects. This allows the user interface to be written such that a search can be proceeding even though the user is performing other tasks. In general, immediately after ObjectSearch returns it should be called again. There is no need to wait for the user to indicate they want to see more.

Experience an object

```
int ExperienceObject(OBJECTHANDLE objHandle);

int ret;
OBJECTHANDLE objHandle;
ret = ExperienceObject(objHandle);
```

objHandle - [IN] handle to the object to be experienced.

Returns: MDSUCCESS - the object was experienced successfully.
MDERR_MMDBNOTOPEN - open the MMDB first.
MDERR_STRINGTOOLONG - the object handle is too long.
MDERR_WRONGSTATUS4EXP - the object has the wrong status to be experienced.
MDERR_WRONGCLASS4EXP - the object has a class which does not allow the object to be experienced.
Other errors provide indication as to why experiencing the object met with a failure.

Get an error message associated with an error number

```
char *MDErrMsg(int err);

int err;
printf("%s\n" MDErrMsg(err));
```

err - is an error from the MMDB API code.

Returns: the function returns a pointer to a string containing the error message.

LIST OF REFERENCES

Ambron, S. (1988). Introduction. In S. Ambron & K. Hooper (Eds.), Interactive Multimedia: Visions of Multimedia for Developers, Educators, & Information Providers (pp. 2-11). Redmond, WA: Microsoft Press.

Barkakati, N. (1991). Object-oriented programming in C++. Carmel, Indiana: SAMS.

Barker, P. G., & Yeates, H. (1981). Problems associated with multimedia data bases. British Journal of Educational Technology, 12, 158-175.

Beiser, K. (1990). Database software for the 1990s. Database, 13(3), 15-20.

Blumberg, R. E., & Walters, D. H. (1989). A PC-based multimedia document manager. IEEE Journal on Selected Areas in Communications, 7, 283-289.

Bordogna, G., Carrara, P., Gagliardi, I., Merelli, D., Naldi, F., & Padula, M. (1990). A system architecture for multimedia information retrieval. Journal of Information Science: Principles & Practice, 16, 229-238.

Cattell, R. G. G. (1991). Object data management: object-oriented and extended relational database systems. New York: Addison Wesley Publishing Company.

Christodoulakis, S., Vanderbroek, J., Li, J., Li, T., Wan, S., Wang, Y., Papa, M., & Bertino, E. (1984). Development of a multimedia information system for an office environment. Proceedings of the Tenth International Conference on Very Large Databases (pp. 261-271). Saratoga, CA: VLDB Endowment.

Engelbart, D. C., & Hooper, K. (1988). The augmentation framework. In S. Ambron & K. Hooper (Eds.), Interactive Multimedia: Visions of Multimedia for Developers, Educators, & Information Providers (pp. 16-31). Redmond, WA: Microsoft Press.

Faggin, F. (1992). The birth of the microprocessor. BYTE, 17(3), 145-150.

Gano, S. (1988). Multimedia technology is for casual everyday use. In S. Ambron & K. Hooper (Eds.), Interactive Multimedia: Visions of Multimedia for Developers, Educators, & Information Providers (pp. 254-265). Redmond, WA: Microsoft Press.

Gibbs, S., Tsichritzis, D., Fitas, A., Konstantas, D., & Yeorgaroudakis, Y. (1987). Muse: A multimedia filing system. IEEE Software, 4(2), 4-15.

Greenfeld, E. (1991, March/April). Playing the century's greatest (historical) "hits". Instruction Delivery Systems: The Magazine of Interactive Multimedia Computing, pp. 24-25.

Groff, J. R., & Weinberg, P. N. (1990). Using SQL. New York: Osborne McGraw-Hill.

Guide to multimedia: How it's changing the way we teach and learn. (1991). Electronic Learning, 11(1), 22-26.

Gupta, R., & Horowitz, E. (Eds.). (1991). Object-oriented databases with applications to CASE, networks, and VLSI CAD. Englewood Cliffs, NJ: Prentice Hall.

Harvey, B. (1991). Symbolic programming vs. the A. P. curriculum. The Computing Teacher, 18(5), 27-29, 56.

Holzner, S., & The Peter Norton Computing Group. (1991). C++ programming: The accessible guide to professional programming. New York: Brady Publishing.

Hughes, J. G. (1991). Object-oriented databases. New York: Prentice Hall.

Illich, I. (1973). Tools for conviviality. New York: Harper and Row.

International Business Machines. (1991). IBM classroom LAN administration system: installation instructions. International Business Machines.

Irven, J. H., Nilson, M. E., Judd, T. H., Patterson, J. F., & Shibata, Y. (1988). Multimedia information services: A laboratory study. IEEE Communications Magazine, 26(6), 27-44.

Kim, W. (1991). Introduction to object-oriented databases. Cambridge, MA: The MIT Press.

Kim, W. (1991). Object-oriented database systems: strengths and weaknesses. Journal of Object-Oriented Programming, 4(4), 21-29.

Kim, W., & Lochovsky, F. H. (Eds.). (1989). Object-oriented concepts, databases, and applications. New York: ACM Press.

Masunaga, Y. (1987). Multimedia databases: A formal framework. IEEE Computer Society Office Automation Symposium (pp. 36-45). Washington, D.C.: Computer Society Press of the IEEE.

Mehrotra, R., & Grosky, W. I. (1985). Reminds: A relational model-based integrated image and text database management system. Proceedings of the 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management (pp. 348-354). Washington, D. C.: IEEE Computer Society Press.

Merrill, M.D., & Tennyson, R. D. (1977). Teaching concepts: an instructional design guide. Englewood Cliffs, NJ: Educational Technology Publications.

Multimedia: What the excitement's all about. (1989). Electronic Learning, 8(8), 26-31.

National Geographic Society. (1990). Mammals: A multimedia encyclopedia [CD ROM]. Washington, D. C.: National Geographic Society.

Navathe, S. B., & Elmasri, R. (1989). Fundamentals of database systems. New York: Benjamin/Cummings Publishing Company, Inc.

Pollak, R. A. (Ed.). (1990). The videodisc compendium: 1990-1991 edition. St. Paul, MN: Emerging Technology Consultants Inc.

Sculley, J. (1988). Foreword. In S. Ambron & K. Hooper (Eds.), Interactive Multimedia: Visions of Multimedia for Developers, Educators, & Information Providers (pp. vii-ix). Redmond, WA: Microsoft Press.

Tello, E. R. (1989). Object-oriented programming for artificial intelligence. New York: Addison-Wesley Publishing Company, Inc.

Yoon, B. D., Suzuki, F., Ishikawa, H, & Makinouchi, A. (1987). Experimental multimedia DBMS using an object oriented approach. Proceedings of the Eleventh Annual International Computer Software and Applications Conference (pp. 632-640). Washington, D.C.: Computer Society Press of the IEEE.