RISC System/6000 Multimedia Environment: An AIX Ultimedia Services/6000 Overview

Document Number GG24-4254-00

May 1994

International Technical Support Organization
Austin Center

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page xi.

First Edition (May 1994)

This edition applies to Version 1.0 of Ultimedia Services/6000 for use with the AIX operating system.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 948S, Building 821 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Abstract

This publication presents a comprehensive discussion of multimedia under AIX on the IBM RISC System/6000. Multimedia support under AIX is provided by the Ultimedia Services/6000 licensed program product, and in order to provide a context in which the features and functions of this product can be understood, information on multimedia concepts in general are explained initially. Ultimedia Services/6000 utilizes an object oriented approach, and as such, an introduction to object oriented programming is also included. Ultimedia Services/6000 current features and functions as well as future directions are then covered.

This document is intended for AIX professionals, both as a general introduction to multimedia, and to provide detailed information on the current product available for AIX on the IBM RISC System/6000.

(94 pages)

Contents

Abstract . . . iii

Special Notices . . . xi

Preface . . . xiii
How This Document is Organized . . . xiii
Related Publications . . . xiv
International Technical Support Organization Publications . . . xiv
Acknowledgments . . . xiv

Chapter 1. Multimedia . . . 1
1.1.1 What is Multimedia? . . . 2
1.2 Multimedia Concepts . . . 5
1.2.1 What is a Media? . . . 5
1.2.2 Importance of the Media Concept . . . 6
1.2.3 Multimedia Standards . . . 7
1.3 Media - Multimedia's Building Blocks . . . 9
1.3.1 Text . . . 9
1.3.2 Audio . . . 9
1.3.3 Still Image . . . 9
1.3.4 Motion Video . . . 9
1.3.5 Interactive Links . . . 10
1.4 What Makes a Good Multimedia Environment? . . . 10

Chapter 2. From Structured Programming to Language Independent Object Model Programming . . . 13
2.1 Advantages of Object Oriented vs. Functional Programming . . . 13
2.1.1 Limitation of Functional Programming . . . 13
2.1.2 Object Oriented Programming: An Overview . . . 16
2.2 Advantages of Language Independent Object Model Programming . . . 22
2.2.1 Limitations of Object Oriented Programming . . . 22
2.2.2 The Solution: SOMobjects, a CORBA Standard Language Independent Object Model . . . 23
2.3 Conclusion: Making Reuse a Reality . . . 26

Chapter 3. Ultimedia Services/6000 . . . 29
3.1 Main Characteristics . . . 30
3.2 Functional Aspects: the Components . . . 33
3.2.1 Ultimedia Object's Library . . . 34
3.2.2 Tools Programs . . . 41
3.2.3 Demo Examples . . . 46
3.3 Programming Considerations . . . 48
3.3.1 Media Supported by AIX Ultimedia Services/6000 . . . 49
3.3.2 Limitations and Performance Considerations . . . 50
3.4 AIX Ultimedia Services/6000 at a Glance . . . 50

Chapter 4. Future Directions . . . 53
4.1 The Future for Ultimedia Services/6000 Objects Library . . . 53
4.2 The Future for Ultimedia Services/6000 Multimedia Tools . . . 54
4.3 Vendors Announcing Support for Ultimedia Services/6000 . . . 55

Appendix A. Multimedia Technologies 57
A.1 Audio 57
A.1.1 Analog Waveform Audio 58
A.1.2 Digital Waveform Audio 58
A.1.3 MIDI 63
A.2 Still Image 64
A.2.1 Raster Image 64
A.2.2 Vector Image 65
A.3 Motion Video 70
A.3.1 Analog 70
A.3.2 Digital 72

Appendix B. System Object Model (SOM) - A Brief Overview 77
B.1 Basic Concepts of the System Object Model (SOM) 77
B.2 Organization of the SOMobject Package 79
B.2.1 SOM Compiler 79
B.2.2 SOM Run-Time Library 80
B.2.3 Frameworks Provided in the SOMobjects Toolkit 80

Appendix C. Programming Examples 83
C.1 Example 1: File Type Detector Program 83
C.2 Example: Movie Play Program 84

Glossary 87

List of Abbreviations 91

Index 93

Figures

1. The Human Machine Interface . . . 2
2. Media . . . 3
3. Multimedia Market . . . 5
4. Multi-Sensory Communication . . . 7
5. Multimedia Software Technology Trend . . . 13
6. Software Costs . . . 14
7. Limitation of Traditional Programming Techniques . . . 15
8. Design process . . . 17
9. Encapsulation . . . 19
10. Inheritance . . . 20
11. Polymorphism . . . 21
12. SOM Compiler . . . 25
13. AIX Ultimedia Services/6000 Product . . . 29
14. AIX Ultimedia Services/6000: Technological Aspects . . . 31
15. AIX Ultimedia Services/6000 Components . . . 34
16. Object Library Components . . . 35
17. AIX Ultimedia Services/6000 Object Library . . . 36
18. Media Handler Objects . . . 36
19. Video Decoder and Encoder Objects . . . 38
20. File Access Objects . . . 39
21. Configuration Objects . . . 40
22. Audio Filter Objects . . . 40
23. Movie Editor . . . 43
24. Audio Waveform Editor . . . 45
25. Main Screen of Ultimedia Services/6000 Demos . . . 47
26. A typical Retail Shop Demo Screen . . . 48
27. AIX Ultimedia Services/6000 layer . . . 49
28. Audio Types . . . 57
29. Analog-To-Digital Conversion . . . 59
30. WAV Format Structure . . . 62
31. SND Format Structure . . . 63
32. Raster and Vector Representations . . . 65
33. GIF Format Structure . . . 67
34. TIFF Format Structure . . . 68
35. JPEG Format Structure . . . 69
36. Analog Motion Video Capture Diagram . . . 71
37. Example of Non-Interlaced and Interlaced Video . . . 72
38. AVS Format Structure . . . 74
39. AVI Format Structure . . . 75

Tables

1. Typical Multimedia Application Areas 3
2. Major Multimedia File Formats for Still Images and Motion 8
3. Comparison of Some Major Common Object Oriented Languages 21
4. Multimedia Requirements Provided by Different Development Approaches 26
5. Audio Encoding 37
6. Motion Video Codecs Characteristics 39
7. Media Supported by AIX Ultimedia Services/6000 49
8. Ultimedia Services Requirements 50
9. Vendors Announcing Support for Ultimedia Services/6000 55
10. Colors Available and Video Memory Required for Bits per Pixel 64
11. Main Characteristics of Vector and Raster Images 66

Special Notices

This publication is intended to introduce IBM marketing representatives, dealers, system engineers and AIX customers to the multimedia environment available on RS/6000 machines. The information in this publication is not intended as the specification for any programming interfaces that are provided by programming languages or application enablers covered in this document. See the PUBLICATIONS section of the IBM Programming Announcement for AIX Ultimedia Services/6000 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, Ct USA 06904.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AIX/6000
AIXwindows	IBM
Multimedia Presentation Manager/2	MVS/ESA
Operating System/2	OS/2
OS/400	POWERserver
POWERstation	PS/2
RISC System/6000	RS/6000
SOMobjects	Ultimedia
Ultimotion	

The following terms, which are denoted by a double asterisk (**) in this publication, are trademarks of other companies:

Apple, System 7	Apple Computer, Inc.
APPLIX, Applixware	Applix, Inc.
C++, C	AT&T, Inc.
Communique	Insoft, Inc.
DEC, Digital	Digital Equipment Corporation
DVI	Intel Corporation
Gain Momentum	Sybase, Inc.
Frame	Frame Technology
Hewlett-Packard, HP, HP/UX, DOMF	Hewlett-Packard Company
ICONAUTHOR	AimTech Corporation
Indeo	Intel Corporation
Intel	Intel Corporation
Interleaf	Interleaf, Inc.
IPX	Novell, Inc.
Lego	Lego, Inc.
Mac, Macintosh	Apple Corporation
MCI	Microsoft Corporation
Microsoft, Microsoft Windows	Microsoft Corporation
MMIO	Microsoft Corporation
MPower	Hewelet-Packard Company
Novell	Novell, Inc.
Objective-C	Stepstone, Inc.
OLE 2.0	Microsoft Corporation
OSF/Motif	Open Software Foundation, Inc.
QuickTime	Apple Corporation
Sybase	Sybase, Inc.
Smalltalk-80	Xerox, Inc.
Sun Microsystems, SunSoft, SunOS	Sun Microsystems, Inc.
Targa+	Truevision Corporation
UNIX	Unix System Laboratories, Inc.
Video for Windows	Microsoft Corporation
Window Multimedia Extension	Microsoft Corporation
XMedia	Digital Equipment Corporation
X-Windows	Massachusetts Institute of Technology

Preface

Multimedia is rapidly evolving into an integral part of modern computing. It is one of the fast growing areas of computer science, having moved from development labs into everyday day life in just a few years. This was possible, because it added a new dimension to *user-friendliness*, making it possible for people to communicate more naturally through computers. Multimedia opens up a whole range of possibilities in communications and is targeted at all audiences.

Multimedia has been developed mainly on the PC platform, and until now, the AIX world did not support multimedia. Today, IBM offers a complete multimedia solution for the AIX platform: the AIX Ultimedia Services/6000 product.

The goal of this book is to provide the reader with a basic understanding of what multimedia is and to give a comprehensive, introductory overview of the AIX Ultimedia Services/6000 environment provided on the AIX platform.

How This Document is Organized

The document is organized as follows:

- Chapter 1, " Multimedia"

  This chapter provides an overview on Multimedia.

- Chapter 2, "From Structured Programming to Language Independent Object Model Programming"

  This chapter provides a short overview of object oriented technology and System Object Model. It gives an insight into which software technology is best for multimedia applications development.

- Chapter 3, "Ultimedia Services/6000"

  This chapter describes AIX Ultimedia Services/6000 and introduces some basic concepts for programming with this product.

- Chapter 4, "Future Directions"

  An overview on the future of multimedia platforms in an AIX environment.

- Appendix A, "Multimedia Technologies"

  This appendix describes the most common computer representations of multimedia data and their related file formats.

- Appendix B, "System Object Model (SOM) - A Brief Overview"

  This appendix provides a brief description of System Object Model.

- Appendix C, "Programming Examples"

  This appendix shows some AIX Ultimedia Services/6000 programming examples.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- *Microsoft Window Multimedia Programmers Workbook*, ISBN 1-55615-390-2
- *Microsoft Window Multimedia Programmers Reference*, ISBN 1-55615-389-9
- *Object Oriented programming: An Evolutionary Approach*, ISBN 0-201-10393-1
- *Multimedia Making It Work*, ISBN 0-07-881869-9
- *S/390 Multimedia Application for Manufacturing*, G221-3535
- *The Common Object Request Broker: Architecture and Specification*, OMG Document 91.12.1, rev 1.1
- *SOMobjects Developer Toolkit*, G221-3651
- *AIX Ultimedia Services/6000 Guide and Reference*, SC23-2528-00

International Technical Support Organization Publications

- *IBM Personal System/2 Multimedia Fundamentals*, GG24-3653-01
- *Multimedia in a Network Environment*, GG24-3947-00
- *Object Technology in Application Development*, GG24-4290-00
- *Multimedia Presentation Techniques and Technology*, GG24-3975-00
- *IBM PS/2 Programming Multimedia under Windows 3.0: An Object-Oriented Approach*, GG24-3672-00

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

Bibliography of International Technical Support Organization Technical Bulletins, GG24-3070.

Acknowledgments

This publication is the result of a residency conducted at the International Technical Support Organization, Austin Center.

The advisor for this project was:

Al Pate, International Technical Support Organization, Austin Center

The author of this document was:

Andrea Fabrizi, IBM Italy

Thanks to the following people who produced the *AIX Multimedia Workshop*, which was the basis for this publication.

Nina Vogl (Workshop Advisor)

Andrea Fabrizi, IBM Italy

Jon Sober, IBM UK

Nick Higham, IBM UK

Paul Gunther, IBM Australia

Yoshio Takano, IBM Japan

Cyndy Sullivan, Technology Transfer Group, Austin

Chapter 1. Multimedia

In the past few years the incredible growth of hardware performance has allowed the *communication interface* between computer and users to become more natural for users. In the early 1980s, the only way to communicate with a computer was the so-called *command-line* interface. Typing a command was the usual way to indicate to a computer what it was to do. Image support was not standard on PCs. It was present only if an application required it. In the mid 1980s, the computer world began to change dramatically as graphical adapters became standard on PCs. This started the icon-based interface era. The computer's screen became a two dimensional metaphor for the physical desktop; icons (graphical representations of a desktop's objects), windows (areas of the screen), and menus (lists of possible actions) became the new ways to interact with computers.

Today, the human-computer interface is again undergoing tremendous change. Multimedia, the ability to use several media to concurrently and interactively communicate with machines has begun to change the human-computer interaction. Multimedia increases the effectiveness of communications by combining the interactivity of a computer with a natural user interface that includes many data types (see Figure 4 on page 7). Multimedia is rapidly evolving into an integral part of modern computing. It opens up a whole range of possibilities in communications and is targeted at all end-users.

Multimedia is the natural evolution of the human-computer interface: interaction with computers is closer to the human way of communication.

Multimedia handles several kinds of computer input data. It changes the command-line and icon-based user interfaces to a new *video user interface*, which is more interactive and more natural than the previous ones. Indeed multimedia adds a new dimension to *user-friendliness*, making it possible for people to communicate more naturally through machines.

Interactivity

Interactivity is the ability to *conversationally* exchange information. This means there is a *dialog* between human and computer, where the computer or the user accepts inputs or responds to inputs. This kind of interaction allows information to be accessed randomly.

Multimedia lets us communicate more naturally with computers by using more of our natural senses: sight, hearing, touch. Person-to-computer dialog is more spontaneous and more *fun*.

Organizations that today generate traditional information consisting of text, numbers and graphics, now believe their applications will be more valuable with the addition of multimedia content. Multimedia delivers information through the enterprise more effectively and efficiently, thus maximizing the value of the information transmitted.

Multimedia support is becoming standard in many lines of PCs and workstations today. This trend toward multimedia is a result of both the interactive capability of computers and the incremental and qualitative capability to communicate using sound, graphics, animation, video, as well as text and data.

1.1.1 What is Multimedia?

Multimedia is not a precise term and it is used to indicate many different things, such as a:

- Market
- New technology
- New communication media

Multimedia is all these things.

Multimedia as a New Communication Media: Multimedia is a new communication media which allows concepts to be illustrated with images or full-motion video rather than merely presented in text and numeric forms. Their significance can be explained with speech, and the desired emotions created with music and sound effects. Multimedia provides an opportunity to experience, or learn by doing, where understanding comes from an intuitive process, based primarily on the engaging of the senses (see Figure 1).

Multimedia is a completely different way of conceptualizing information and provides mechanisms to transfer this information.

One of its most important features is the easy user interaction, which increases the attention, understanding, and retention of information. People retain more information from an interactive multimedia presentation involving sight and sound than from the passive activity of reading the written word. Short-term retention increases significantly as more human senses are involved, especially if the involvement is interactive. This interactive involvement allows computer-based multimedia to became a powerful communication tool (see Figure 1 and Figure 4 on page 7).

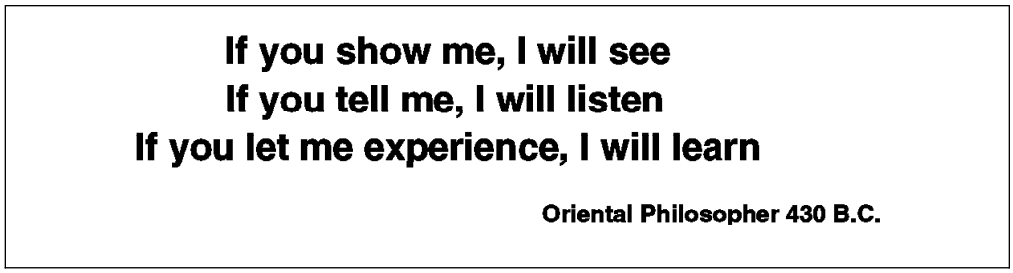

Figure 1. The Human Machine Interface. Providing an interactive experience people retain more information than from a passive activity.

Multimedia as a New Technology: Multimedia is a new technology. Computers, inherently interactive, devices have now acquired enhanced audio and video communications. In addition, traditional content (printed text, film, video, music) and telecommunications are quickly becoming fully digital. As a result, the term multimedia is sometimes used to define a class of computing, where multiple information types, such as text, graphics, image, sound, animation and video (see Figure 2 on page 3), are combined to improve the quality and efficiency of communication. Each of these data types represents technologies with disparate qualities, standards, and computing demands. They each have unique methods for user interaction and control. The goal of multimedia technology is to combine these data types in an effective manner where the value of multi-sensory communication can be realized.

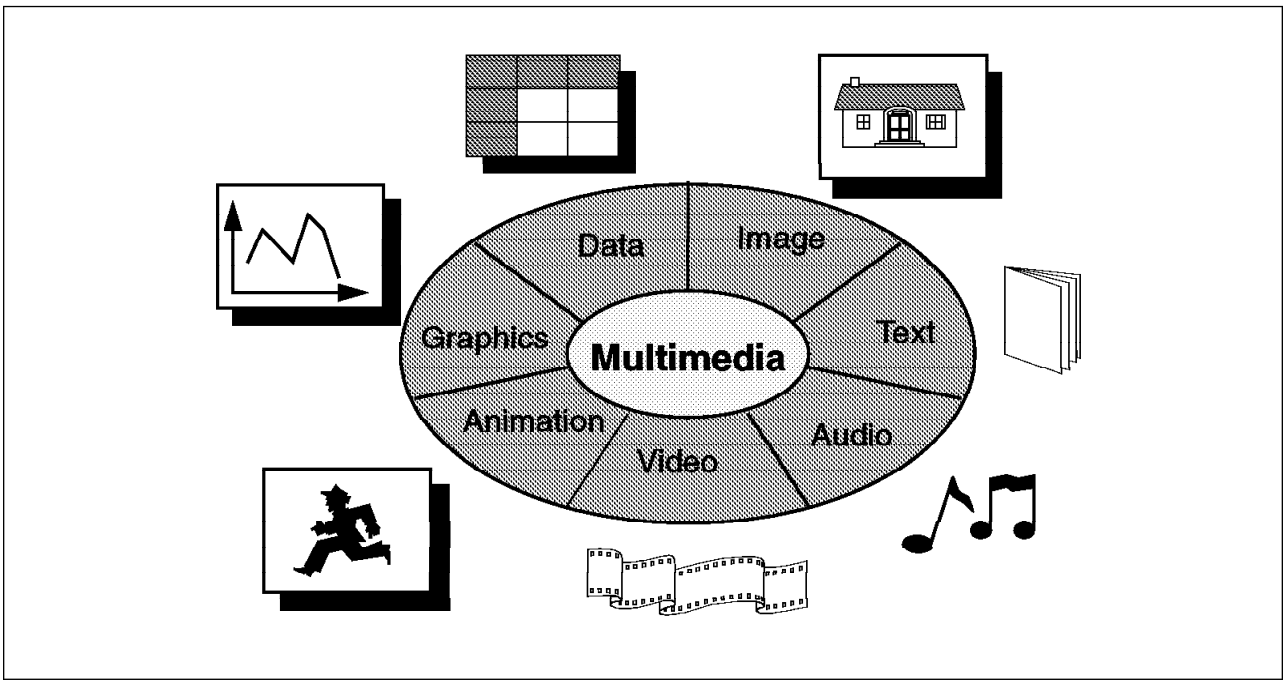

Figure 2. Media. The most common media available on computers.

Multimedia as a New Market: Multimedia is a set of enabling technologies that are evolving the two previously distinct devices, computer and television, into technological equivalents. This evolution has created a new market. Any computer or computer-based device now has the ability to access, interact, and play digital information and entertainment.

This may profoundly change the ways we communicate and the use of traditional media such as television, newspaper, fax, or phone. For example, a personal workstation with network and database access and enhanced with multimedia capability, becomes a versatile communication system, which allows one to communicate information to others in the format and in the location they prefer. In advertising, creative designers can use multimedia to enhance their ability to create, edit and develop digital Video Clips. In manufacturing, a plant floor worker can communicate from his or her station via videoconference with a development engineer to show a defective part. Medical professionals can collaborate by sharing diagnostic images using a multimedia distributed database. In education, the high quality of multimedia communication makes traditional lectures and laboratories more interesting and more fun.

Table 1 (Page 1 of 2). Typical Multimedia Application Areas

Market Area	**Description**
Video Intensive Creation	Multimedia capabilities allow enhanced automation of Video Clips editing and development.

Table 1 (Page 2 of 2). Typical Multimedia Application Areas

Market Area	**Description**
Video Information Services	Multimedia allows users to digitize and store video and audio information in databases. This information can be shared across networks and distributed to distributors, producers and so forth. Applications such as virtual museums are possible, where the artifacts and information displayed at the museum are made available to computer users in a multimedia presentation.
Kiosk Sales/Services	Multimedia kiosks make services more accessible (24 hours and 7 days per week) and more convenient (public places) to the public. Moreover, they can add video demonstrations, competitive comparisons and interactivity.
Multimedia Factory Floor	Automation of communication (for example videoconferencing or in-place training) inside a factory allows dramatically improved factory floor efficiency.
"Documedia" Multimedia Documentation	Integration of several media types (data, text, graphics, image, video, audio) into documents allows improved document quality and effectiveness.
Training and Education	Multimedia improves traditional lectures and education methods.
Corporate Communications	Corporate communications greatly benefit from multimedia. Electronic mail, information sharing and training applications are examples.

There are many strategic applications (see Figure 3 on page 5 and Table 1 on page 3) across all industries in which multimedia can improve competitiveness through increased efficiency, effectiveness and quality of communication.

In conclusion, multimedia can be considered as the integration of the human senses into a computer environment for the purpose of improving communication between the computer and its user and, even more important, between users.

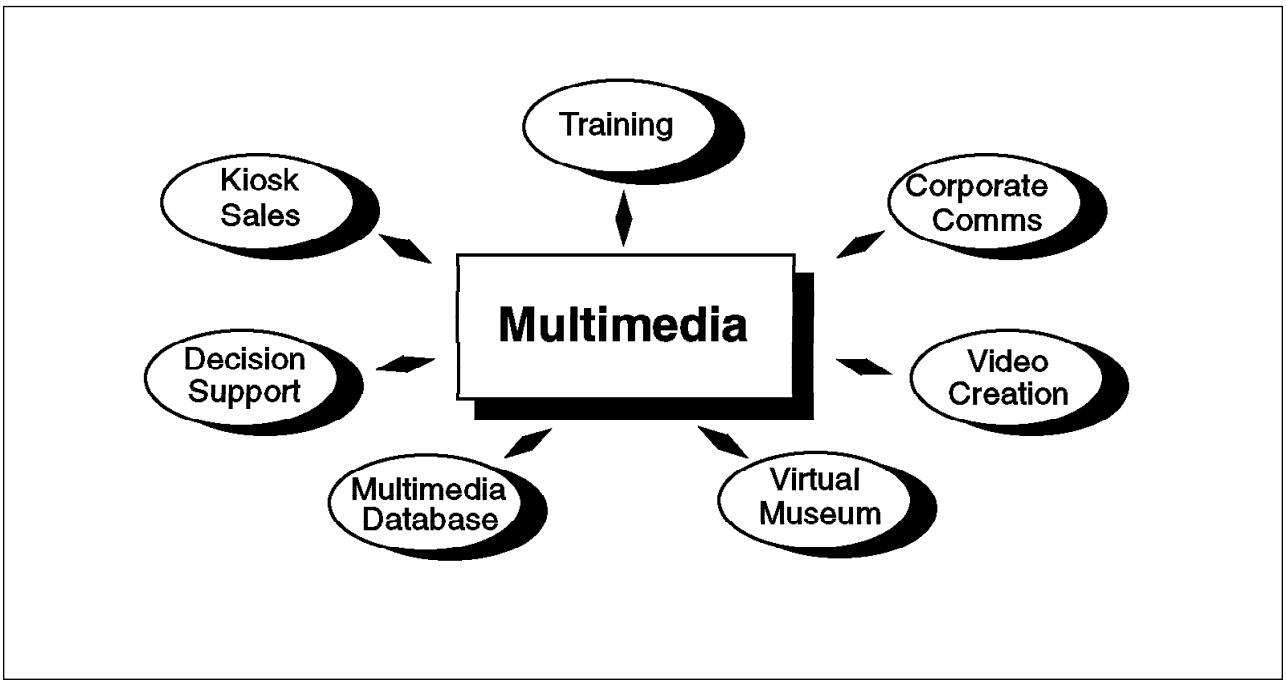

Figure 3. Multimedia Market. This picture shows some of the most important business areas where multimedia is used.

1.2 Multimedia Concepts

This section explains the main concepts of multimedia though a description of its main components and technologies. Multimedia is described in several steps, each introducing one main concept.

1.2.1 What is a Media?

In a computer science context, and throughout this book, media is used either as the singular or plural and means channel of communication: a way to transmit a particular information type, such as an image or text.

The human brain has evolved a set of powerful *natural processors* or senses to recognize these channels. Of the five senses:

- Sight
- Hearing
- Touch
- Smell
- Taste

the first three are used the most. More than 90% of information is transmitted using just sight, sound and touch.

During communication, each sense can simultaneously process different media.

Sight is the image processor which absorbs and processes visual information. It involves the following media:

- Still Image and Graphics, which allow the viewer to inspect details and understand their relationship to the complete image.
- Full Motion Image and Animation, which allow the perception of movement. Motion is perceived as a series of still images, each one slightly different from the previous, displayed in succession so fast that the illusion of movement is created.
- Text, which provides data-oriented information to enlighten the viewer.

Hearing is the audio processor, which receives and processes audio information over a large range of frequencies. It involves audio media such as voice and music.

Touch is the pressure processor, which receives and processes tactile information, such as the tactile feedback from our fingers. Touch is used as a communication media primarily by the visually and hearing impaired.

In the computer science world, the word *media* also means: *"a kind of information that can be transmitted from a source (computer or human being) to a destination (computer or human being)"*.

Any type of information that can be transmitted is a media.

1.2.2 Importance of the Media Concept

The concept of media is related to one of the most important problems of computer science: improving *communication* between users and computers.

Communication

The word communication indicates a process by which information is exchanged between two or more entities through a common system of symbols, signs and behavior. (Webster's Dictionary)

Studies on human exchange of information have shown that increasing the number of media used during communication greatly increases the information retained. In fact, the increase in information retained by the destination grows more exponential than linear as more than one media is used. Figure 4 on page 7 shows communication where information is exchanged using several media is more efficient than communication using only one media.

Multimedia, or increasing the number of media involved in a communication, expands the communication capability between humans and computers. It allows a user to interact with their computer at a more meaningful level through multi-sensory communication (see Figure 4 on page 7).

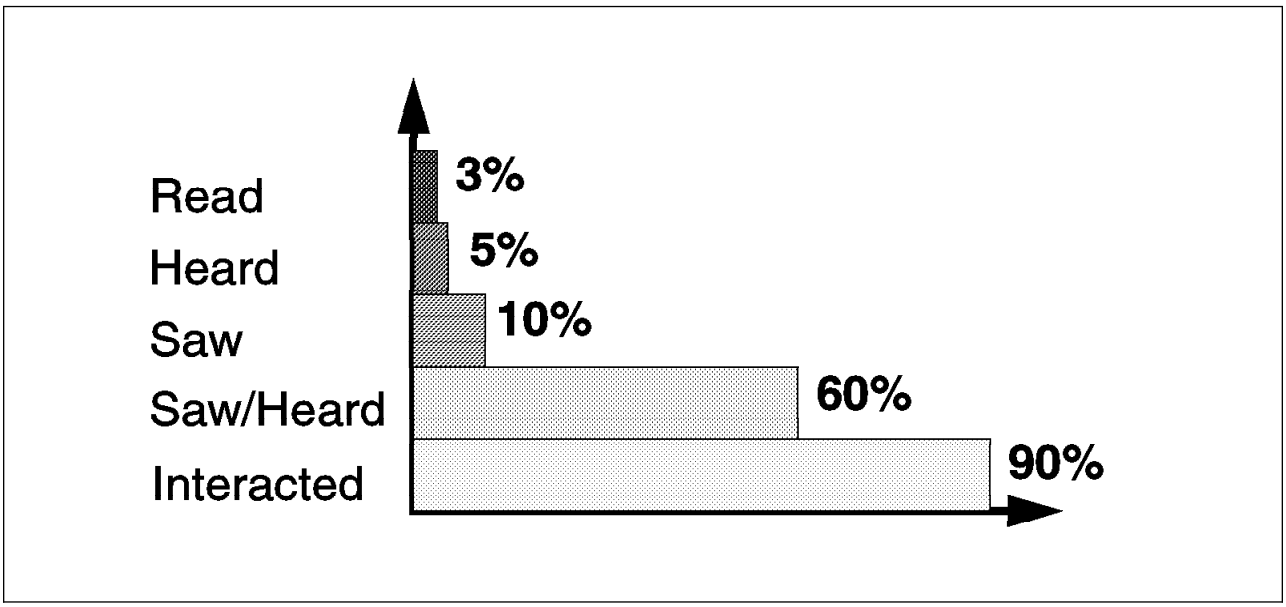

Figure 4. Multi-Sensory Communication. The figure shows an information retention increase with the number of media involved in communication and the resulting increased interaction level.

1.2.3 Multimedia Standards

The growth of multimedia can be accelerated greatly through unifying standards and integrated management of different media data types among heterogeneous platforms. Multimedia systems should be able to exchange data between different multivendor platforms. *Standards* are one of the critical success factors in the widespread acceptance and growth of multimedia applications and products.

Widely accepted multimedia data formats are a fundamental key for applications to work on different software platforms. Multimedia applications able to play only a few multimedia formats can *run* only on specific platforms, and will become obsolete as technology evolves. Only the unified standard file formats and document coding can protect customer investments and accelerate the growth of multimedia.

Multimedia standards are in early development and much work still must be carried out. This section will provide an overview of the major standards-related activities. For more information, read: *Multimedia in a Network Environment* and *IBM Personal System/2 Multimedia Fundamentals*.

ISO

The principal standards organization is the *International Standards Organization (ISO)*. This organization has several committees, subcommittees, and working groups that deal with multimedia standards.
In addition to the ISO, the *Consultative Committee on international Telegraph and Telephone (CCITT)* and other associations are also working to define new standards and recommendations.

Multimedia-related standards can be grouped into four major areas:

1. Data File Format Standards

This area includes all Data Format Standards able to describe a single media type when it is captured, recorded or played. Table 2 on page 8 shows the most important and most common file formats.

Video.

Table 2. Major Multimedia File Formats for Still Images and Motion

Area	Standard
Still Image	• JPEG # (Joint Photograph Experts Group) • JBIG # (Joint Bi-level Image experts Group) • GIF • TIFF • Targa+
Motion Video	• MPEG # (Motion Photograph Experts Group) • DVI (Digital Video Interactive) • QuickTime • Ultimotion • Video for Windows • Indeo • px64 #
Audio	• MIDI # (Musical Instruments Digital Interface) • Wave • SND
Storage	• CD-ROM (Compact Disk - Read Only Memory) • CD-ROM/XA (CD-ROM eXtended Architecture) • CD-I (Compact Disk Interactive) • CD-DA (Compact Disk Digital Audio)
Note: The # indicates file formats supported by ISO or by other organizations working on standards.	

2. Multimedia Language Technology

   A model is required to describe and represent the relations among multimedia objects. There are two main types of relationship between multimedia objects:

   - Spatial, the positional-relationship. For example, the spatial distance between two objects on the screen.
   - Temporal, the time-relationship between objects (for example, if two objects are displayed simultaneously or with an interval of 5 milliseconds.)

   The ISO jointly with the CCITT has formed a committee to address the development of a common base for multimedia and hypermedia applications. This committee is called *Multimedia and Hypermedia Experts Groups (MHEG)*. The goals of this standard are:

   - Support for the real-time exchange of multimedia information
   - Synchronization of multimedia object interaction
   - Provision of a final form representation of multimedia and hypermedia information

3. Network

   Today's local area network (LAN) and wide area network (WAN) are poorly suited for transporting multimedia data objects and data streams. New technologies are being explored to support real-time traffic with minimal delay over networks.

   There are three major standards efforts in this area:

   - Investigate 10Base-T (Ethernet) isochronous communication enhancement to support high data rates over unshielded twisted-pair copper cable.

- Investigate priority scheduling and bandwidth reservation for multimedia frame transmission over token-ring.
- Support emerging standards such as Asynchronous Transport Mode (ATM).

4. System Services

   Multimedia extensions to basic operating system function are required to to accommodate new data types and their unique features. Moreover, actual databases are needed to accommodate multimedia data.

1.3 Media - Multimedia's Building Blocks

This section addresses the information types handled by a computer. The principle goal of multimedia is the efficient management of these data types to optimize the value of multi-sensory communication. The technical details about how these media are represented and implemented on a computer are discussed in Appendix A, “Multimedia Technologies” on page 57.

1.3.1 Text

The most common way to transmit information is to write it! Text is a sequence of elements (characters, symbols, words, phrases, paragraph, sentences, tables and so forth) intended to convey a meaning. The interpretation is essentially based upon the reader's knowledge of some natural or artificial languages. This media is fundamental to computer systems and provides a basic building block for multimedia systems.

1.3.2 Audio

Sound and Audio are very useful in a multimedia system as a means to transmit parallel information: while watching a movie or reading text, our understanding of information presented can be reinforced using audio. By seeing and listening at the same time we receive similar information simultaneously. Moreover sound and music can be used to transmit particular sensations and feelings.

1.3.3 Still Image

Images are a fundamental part of multimedia systems. Still images are a wonderful way to illustrate information or concepts: *A picture is worth a 1000 words*. An image of a view, a histogram or a pie chart are more impressive than text or a table. Images allow much information to be transmitted in a short time (see Figure 4 on page 7).

1.3.4 Motion Video

Television has shown the huge potential of video as an information channel. Sometimes a concept cannot be explained easily using speech, text, still images or all these media together. In those cases, an animation or a video sequence may be a more effective way to illustrate a situation.

1.3.5 Interactive Links

As shown before, multimedia requires strong interactive techniques. A fundamental feature of a multimedia system is the ability to *follow* the information. For example, if we are reading text and we find an unknown concept, it would be very useful to provide a mechanism to ask for an explanation on that unknown concept. Moreover, if we were watching a video or listening to an audio file, it would be useful to have a button to control the playback of these media. Interactive links, together with the multimedia information they connect, are called *hypermedia*. The ability to press screen *hot areas* (for example, buttons, highlighted text, and so forth) to access more information or to cause a program response, are other fundamental aspects of a multimedia system.

1.4 What Makes a Good Multimedia Environment?

For a complete understanding of multimedia, there is one last important question: how must multimedia be integrated into the actual corporate and customer environments.

Organizations produce a lot of traditional information consisting of text, numbers and graphics. They store and retrieve this information through the use of databases, and share this information through the use of electronic mail, fax and telephone communications. These same organizations expect to work in the same way using multimedia information. They need a new application environment so multimedia data can be handled in the same way as the traditional information.

A primary goal of Multimedia System Services must be to provide an infrastructure for building a multimedia computing platform that supports *interactive* multimedia applications dealing with *synchronized, time-based* media in a *heterogeneous distributed* environment. (IMA Multimedia System Services 1.0)

This problem is not simple. Construction of large-scale, heterogeneous, multimedia, distributed systems introduces technical problems not found in conventional systems. For example, the additional, non-structured complexity of multimedia environments makes data encapsulation and abstraction mechanisms very important. There is a heavy emphasis on interfaces and objects. Many software experts agree that modelling a distributed system as a collection of interacting objects is appropriate for integrating distributed multimedia information and resources. Today, many people argue that object-oriented distributed computing is the natural step forward from the client-server systems of today.

Objects form a natural model for a multimedia system. They are a realistic representation of real world things both from the user's and the programmers point of view. For example, in the real world a user watches a movie by putting the video into a video tape player and pressing the play key. In the application world he does the same: he puts the movie's object into the videotape's object and presses the play button. From a programmer's point of view, it is the same: all resources (multimedia adapters, displays, and so forth) are treated as a commonly accessible collection of objects. The object's model hides the complexity of the resources and the device drivers from the programmer.

Object modelling is important but it is only a part of the solution. Indeed, object-oriented modelling allows flexibility, code reuse and expandability, but there is a hitch! The object oriented languages world is fragmented into many factions,

each writing code in their specialized programming languages. Each of these factions claim to follow the one right path to object-oriented programming and this has produced many incompatible object libraries. A major problem is the fact that most object oriented programming languages are not compatible with other object oriented languages. The solution to this problem is not to look for a universal object oriented language, but rather to find a way to enable objects written in different languages to work together within a single computer or within a network of computers.

Networking presents the other problem which must be solved. If multimedia is to reach its fullest potential, it must move beyond the limits of stand-alone technology. This requires new communications and transmission technologies to guarantee an exact delivery of real-time multimedia continuous data streams. But this is not enough; multimedia objects will have to be accessed remotely in a distributed heterogeneous environment. Applications must be able to access objects in other processes, even on different machines. The object's implementation as well as the object's location must be hidden from the clients. The client will access objects as if they are local.

In summary, the main requirements that a multimedia environment must satisfy are:

Open This is the ability to interoperate between a wide range of different systems while maintaining consistency across all. An important objective of an *open system* is to make a network of heterogeneous systems appear as a single system to users. This means all resources and services that users can access, local and remote, appear as local to such a system, despite differences in hardware, operating systems, networks, programming languages and databases.

Flexible This is the ability to create and modify software easily. A *flexible routine* is a routine that can be changed without requiring old programs which use it to be changed.

Expandable This is the ability to easily extend the functionality of existing applications. The necessity to extend the functionality of applications in one of the major problems in the computer industry.

Reusable This is the ability to use code written for an application in another application without modifications. The main goal of *reusable code* is to allow creation of applications by assembling independent objects. These objects are a kind of Lego** that programmers can arrange in any way they choose. Reusable code reduces development cost and time and allows creation of more uniform applications.

Heterogeneous This is the ability to communicate transparently between different hardware and software.

High Capability The code must be highly specialized to guarantee high performance.

Distributed This is the ability to work transparently across networks. Today's computers need to share CPU time as well as printers, external devices and information. The main goal of a *distributed system* is to share all computer resources (information, CPU and so forth) to all users on the the network.

Each of these points is essential in order for multimedia to reach its fullest potential.

In summary, we can say that the goals of a multimedia distributed environment are two: **open computing** and the **integration of multimedia extensions into strategic platforms**.

Open and Heterogeneous

Open System. A system whose characteristics comply with standards made available throughout the industry and therefore can be connected to other systems complying with the same standards.

Heterogeneous System. A system in which computers have dissimilar architecture, but still are able to communicate.

Multimedia Platforms Available

Multimedia support is becoming standard in many lines of PCs and workstations today. The main ones are:

- PCs:

  MacIntosh** system 7 with QuickTime

  Microsoft Windows** 3.1 Multi Media (Microsoft Windows 3.1 Multi Media is not really a multimedia platform, because it is not a real multitasking system, which is a fundamental requirement for data synchronization and presentation.)

  OS/2* 2.0 and more with MultiMedia Presentation Manager/2*

- UNIX** Workstations:

  Digital**: XMedia**

  HP**: MPower**

  SUN**: Gain Momentum** (Actually, Gain Momentum Authoring System and its related software is also available on the following platforms:

  – SUN
  – RS/6000*
  – HP

  IBM*: AIX* Ultimedia Services/6000*

Chapter 2. From Structured Programming to Language Independent Object Model Programming

Before discussing multimedia on the AIX platform, let's take a closer look at different software development technologies to understand which technology is best for developing multimedia applications.

This chapter examines why the development software technology has moved from a Traditional Functional Approach to Object Oriented (OO) Technologies and is now evolving towards Language Independent Object Model Approaches. It also discusses why these new programming technologies are so important for the development of multimedia applications.

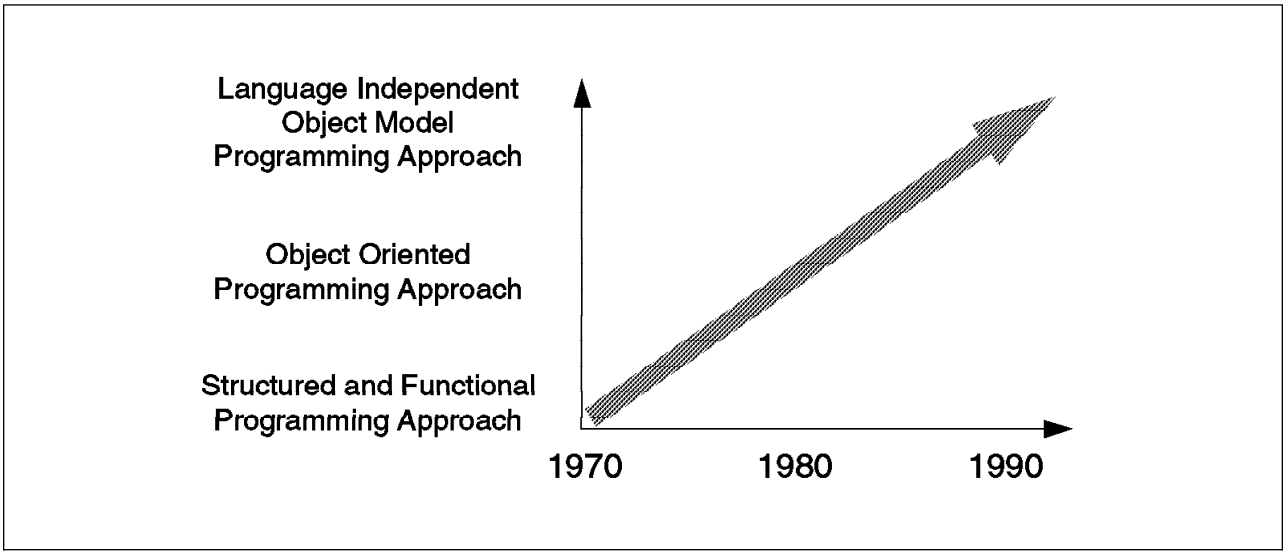

Figure 5. Multimedia Software Technology Trend. The software technology used to develop multimedia applications has moved from Functional Approaches (for example, MCI) to an Object Oriented Approach and now is moving from an OO Approach to a Language Independent Object Model Programming Approach (for example, the AIX Ultimedia Services/6000).

2.1 Advantages of Object Oriented vs. Functional Programming

The advantages of an object oriented programming approach versus a functional programming approach can best be understood by comparing the capabilities and limitations of both to application development technology requirements.

2.1.1 Limitation of Functional Programming

Today, the quantity and importance of information, especially multimedia data, are growing exponentially. The speed and performance of the computer hardware has easily kept pace. Application development technology required for creating software has evolved much more slowly. This has created a huge drag on business.

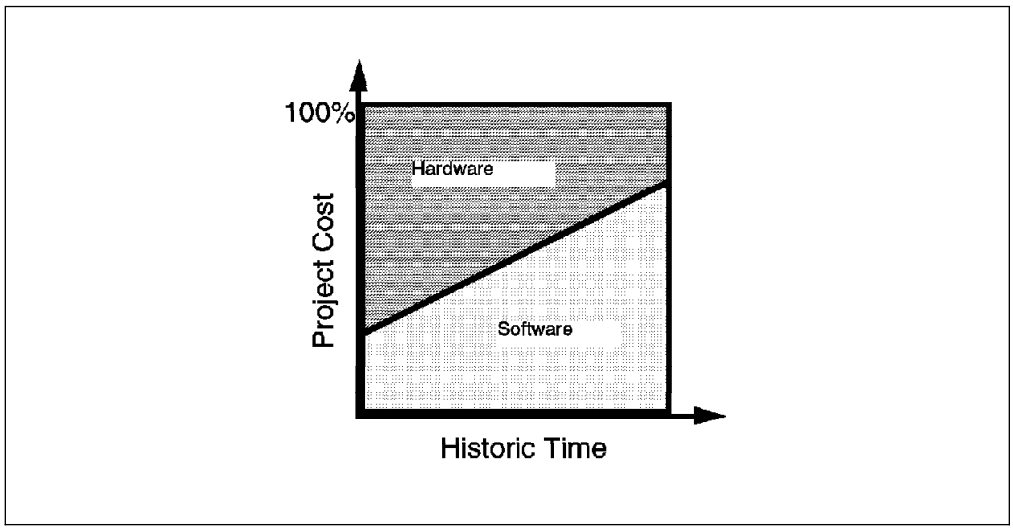

Figure 6. Software Costs. Software is playing a larger percentage in project costs.

There is no one specific answer to why software development has not kept pace. A look at what is required in an application program reveals some clues. Applications must accurately model the environment they describe. Application development requires creating an exact computer representation of the real world entities and transactions (business, department store, bank transactions, and so forth). Essentially, application programmers have to translate all real-world objects into computer objects. This requires programming languages that permit:

Information Hiding - Data must be accessible only through few and well-specified functions.

Data Abstraction - The ability to create new types of computer data. (For example, the complex number type, or the window type.)

Easy Software Enhancement - The software must be easy to enhance and modify.

Software Reuse - The ability to create new software simply by adding new features to the old ones is essential.

Easy project complexity management - The efforts to manage a project must not increase more than linearly with the dimension of the code.

Interoperability - The programs must be able to work and to exchange data with other programs.

A succession of approaches (for example, modular programming, structured programming, computer-aided software engineering (CASE), fourth-generation languages) have been proposed as the solutions to these requirements. Ultimately each has proved to have a serious down side: excessive programmer or computer resources needed, limitations on program size, difficulty (or impossibility) of modification (see Figure 7 on page 15), while time and costs to develop software continue to grow, as shown in Figure 6. The amount of code required increases as programs attempt to model more complex projects. Application cost can be expected to increase proportionally as complexity increases. The ability for humans to manage complexity becomes a factor as applications become more complex. Moreover, the maintenance costs of a project grow with the dimensions of the code.

Multimedia applications emphasize and amplify these problems in two ways.

- The high-level of interaction required by multimedia applications and the non-structured nature of user's actions dramatically increase the complexity of projects.
- The high abstraction level of the objects involved in typical multimedia applications (for example, the motion video editor object) makes computer representation and environment modeling more difficult and complex.

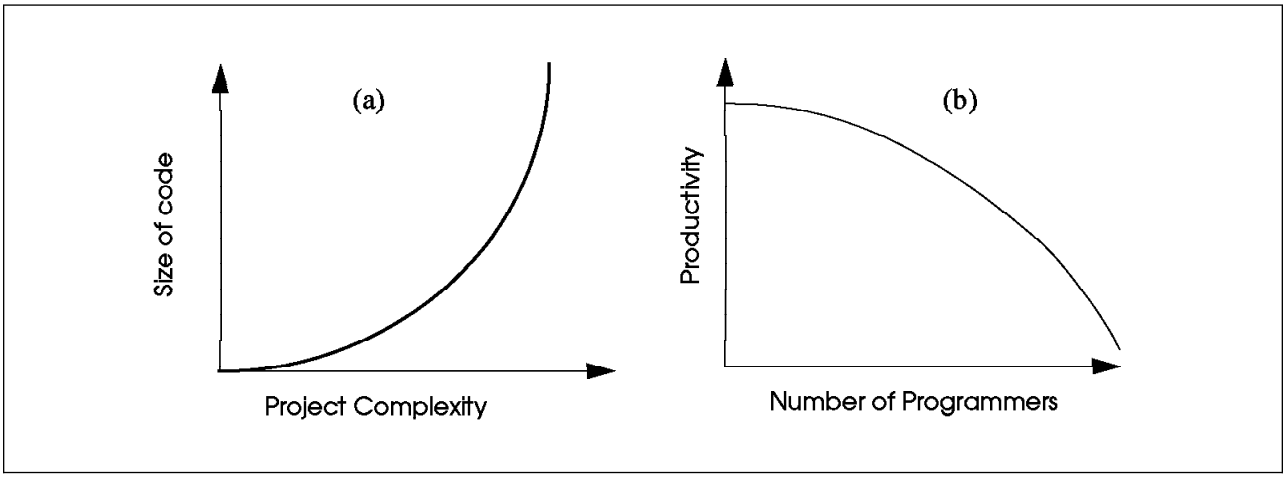

Figure 7. Limitation of Traditional Programming Techniques. The figure (a) shows what happens when programs attempt to manage more complex projects: the size of code increases exponentially with the project complexity. The figure (b) shows typical non-linear productivity response to increasing the number of programmers applied to a project. This decreased performance is due in part to the communication overhead incurred with more and more programmers.

Object Oriented (OO) Technology represents a fundamental change in the concept of software development. With the OO paradigm, the engineering and manufacturing techniques which enables hardware to grow in performance while declining in price are applied to programming. Object Oriented Technology provides software objects which are segments of code combining data and procedures. These objects will become standard, off-the-shelf parts. Programmers can then incorporate these objects in a system as easily as engineers incorporate an off-the-shelf semiconductor chip into a circuit board design. A hardware engineer would never consider designing a circuit board from scratch, rather he will select from a library of prebuilt components with well-defined interfaces and link these together. With OO Technology, creating an application becomes a bottom-up assembly of existing units.

The main goals of Object Oriented Technology are:

- Maintenance and development cost reduction
- Increased software quality
- Ease of software reuse

2.1.2 Object Oriented Programming: An Overview

Object Oriented Programming (OOP) is an important new programming technology that offers expanded opportunities for software reuse and extendibility. Object Oriented Programming shifts the emphasis of software development away from functional decomposition and toward the recognition of units, called *objects*, that encapsulate both code and data. As a result, programs become easier to maintain and enhance. Objects with the same behavior and common features, can be grouped into hierarchical categories, called *classes*. As a result, new objects can be built by reusing others. The traditional approach to application development has been called a *water fall process*. The steps to create an application are:

- Requirements
- Analyze
- Design
- Implement
- Test

All these steps are sequential, with limited feed back possibilities between steps. The creation process is not interactive and changes and/or enhancements are difficult to manage. The Object Oriented Approach allows a true interactive design process (called a *whirlpool* or *spiral process*). This approach allows program development within a section; the application is built by simply adding new features and objects. Object Oriented programs are typically more impervious to the *ripple effects* of subsequent design changes than their non-object-oriented counterparts (see Figure 8 on page 17). This, in turn, leads to improved programmer productivity.

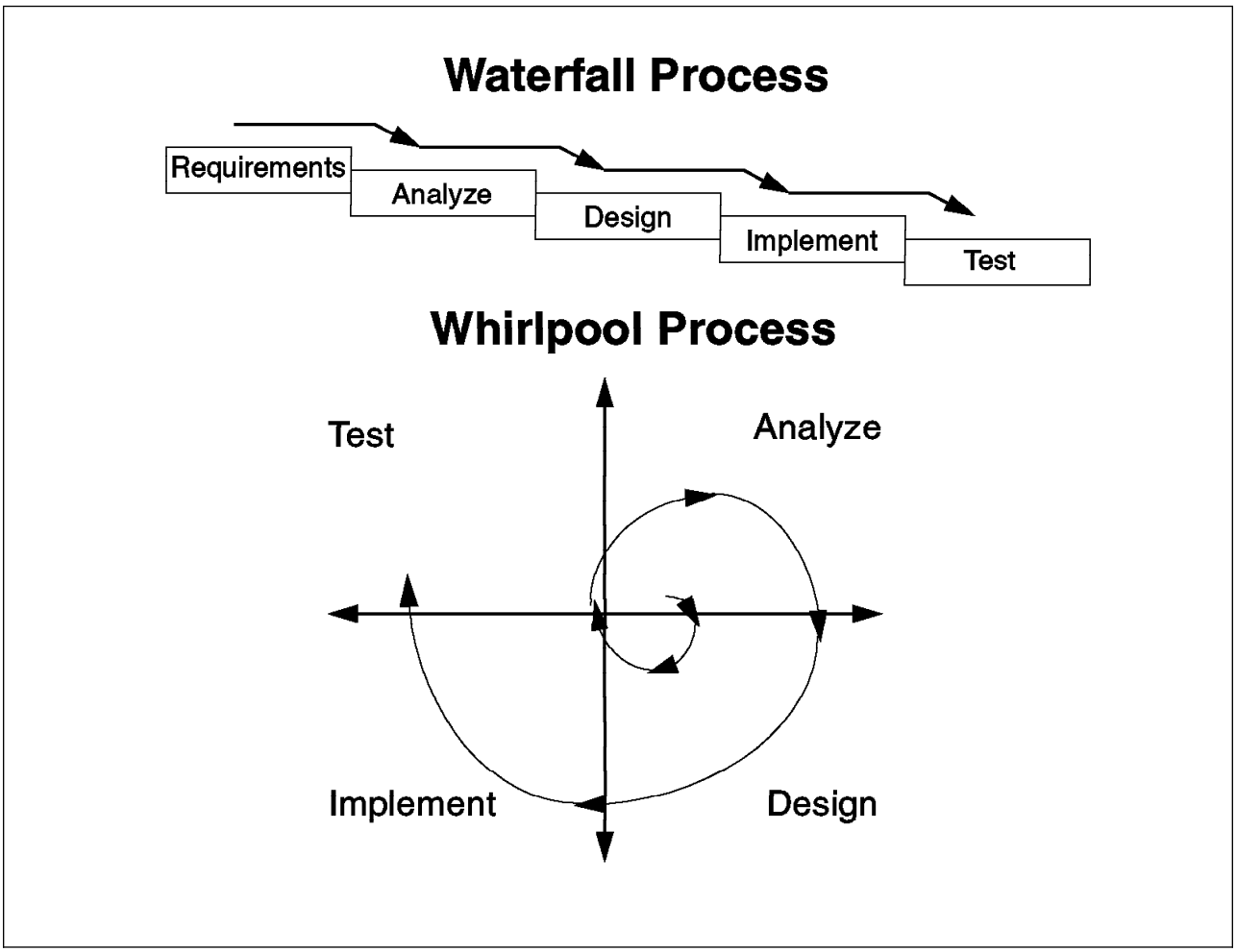

Figure 8. Design process. The figure shows the traditional "waterfall" approach to application development and the more interactive spiral process which is characteristic of Object Oriented Techniques.

2.1.2.1 Concepts and Terminology

The Basic Idea: Object Oriented Programming involves two basic concepts:

- Encapsulation
- Inheritance

Encapsulation means keeping data and functionality in one package (the object). Data is completely hidden to the rest of world and can be modified only by using the object's functionality (the methods). Encapsulation allows the creation of a *software wall* around data, as shown in Figure 9 on page 19. It is not known *how* an object works, but only *what* it does. This ensures data operations are only performed on appropriate legal data values. It is possible to modify data descriptions concurrently with last minute design changes without recoding major sections of the application. This is possible because the objects are *independent* software.

Inheritance is a code-sharing mechanism. It allows reuse of the behavior of a class in the definition of new classes. Subclasses of a class inherit the data structure and the functionality of their parent classes (called *superclasses*). Reuse is the key to

dramatically improved software productivity. Coding time is reduced and software quality improves as tried and trusted objects are reused.

A better description of Object Oriented Technology can be provided through its main components: Classes, Objects, Messages, Encapsulation, Inheritance, and Polymorphism.

Classes: Classes are a set of *things* that share common characteristics. For example, the biological sciences have used classification to categorize plants and animals. The plant and animal categories begin with very general descriptions. These general categories are then further described to create more specialized categories (for example, the mammals category). In this way, classes are often organized in a hierarchy of other classes. Classes are used to create objects.

Objects: Objects are produced from classes. They are the instances of classes. An object is a grouping of data and operations (called *methods*) on that data. The data determines the *state* of the object, while the methods determine the *behavior* of the object. An example of an object can be given using C programming language. In C, an object can be represented using the formal C type *struct*. In this case the object has both data and methods, but the data is not hidden.

Messages: A message is used to invoke an object's method. A general message contains:

- Object's name
- The requested method
- Any parameters required

Messages are the only way to invoke methods. For example, the *movie* object is a highly specialized software unit that can play, record and edit motion video files. When the *movie* object plays a movie, it simply sends the *OPEN_FILE <file_name>* message to the *file_handler* object (the *file_handler* objects is a highly specialized software unit that can open, close and save files) which opens the file and returns the file identifier to the *movie* object.

Encapsulation: Object Oriented Programming revolves around the concept of encapsulation. Encapsulation hides the implementation detail of the data from other objects. Access to internal data is allowed, but only by using the programming interface provided by the object. As a consequence, the internal structure of an object may change with no impact to other objects.

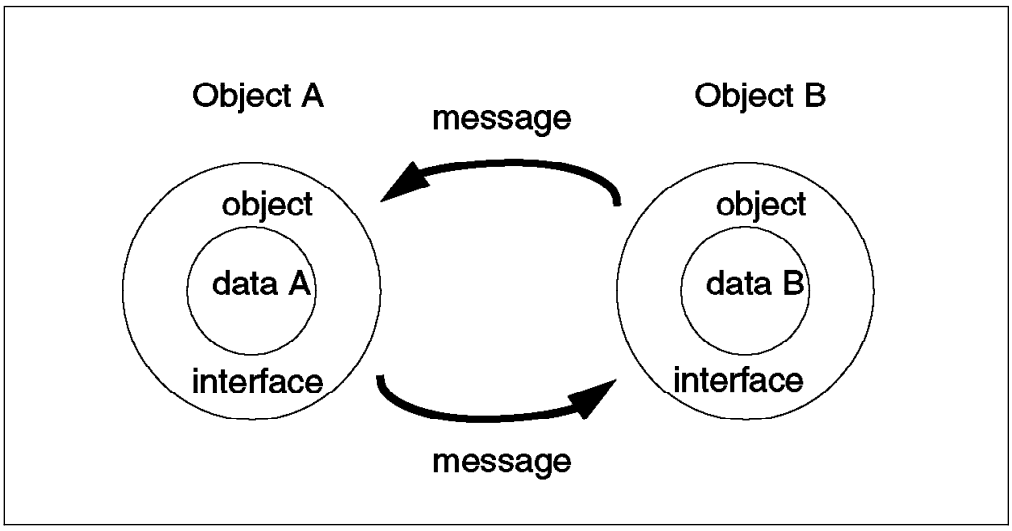

Figure 9. Encapsulation. Encapsulation means to bundle code and data

Inheritance: Inheritance is the ability to define new classes in terms of other classes. This means that a class acquires characteristics from one or more parent classes. The advantages of this mechanism are:

- Common code is implemented once
- Common data is described once

The biological sciences are again useful in explaining the concept of inheritance. Animal classification is organized based on common characteristics (mammals, as an example, are described by the way in which their young are fed). All animals in the same class have common characteristics. To say a human being, a dog or a whale are mammals means that these animals feed their young in a common way. The location of the class within the hierarchy determines how a class is specialized. A deep position in the class hierarchy means high specialization, while a high position means low specialization. Inheritance makes it possible to define new software in the same way a new concept is introduced to a newcomer: by comparing it with something that is already known. New classes can be created by specialization (the new class is similar to the old one, but it has new features) or by generalization (this class merges the features of old classes).

The two inheritance mechanisms are:

- **Linear Inheritance or Inheritance by Specialization.** In this case, the child classes are made by adding new methods to the parent class. Child classes are more *specialized* than the parent class.
- **Multiple Inheritance or Inheritance by Generalization.** In this case, the child class is produced by merging the parent class methods. The new class is a *generalization* of the parent classes (see Figure 10 on page 20).

Inheritance permits reusabilty of software.

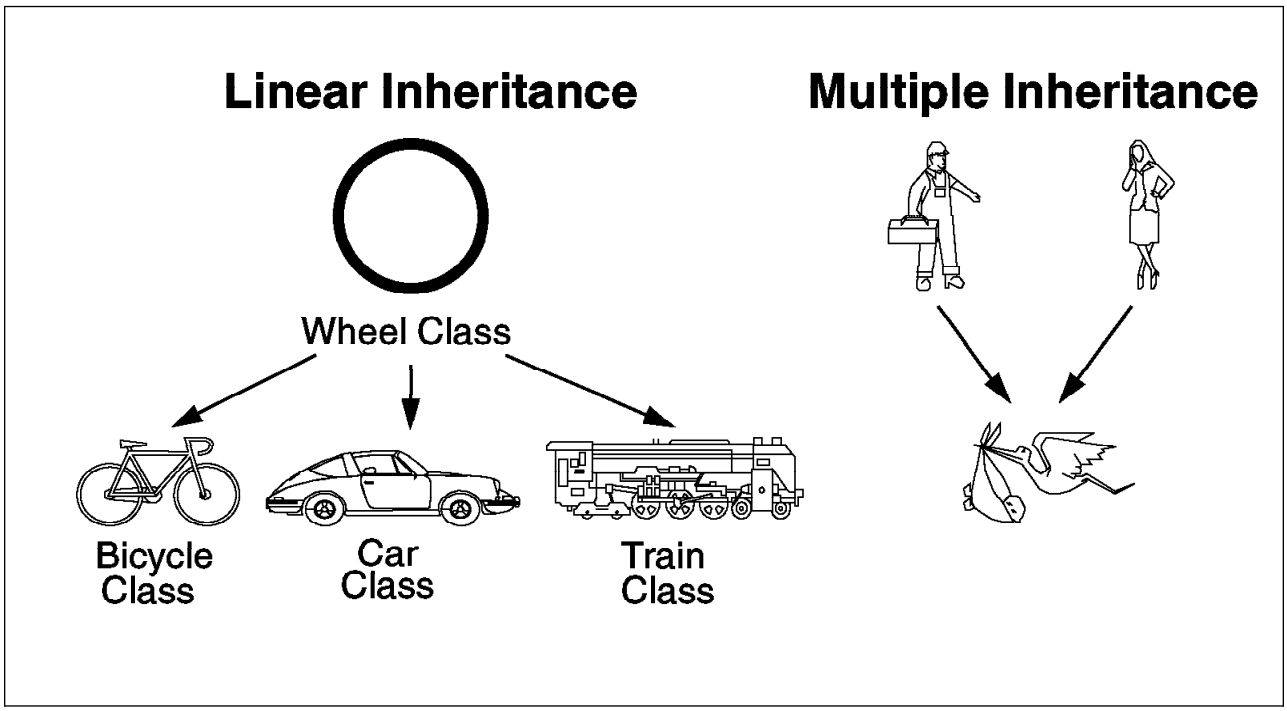

Figure 10. Inheritance.

Polymorphism: Polymorphism is the ability to *hide* different implementations behind a common interface: different objects can understand the same message, but they perform different actions. An example of polymorphism could be the arithmetic message of *addition* on several objects whose data is different (for example, integer, floating point, complex, character and so forth). The exact implementation of addition at the machine level is different on each data type, however the high level message *addition* is understood by each object (see Figure 11 on page 21).

These key concepts of Object Oriented Technology provide significant benefits for both end-users and developers. For example:

- Consistent user interface across many applications is possible by sharing common objects.
- Directly mapping real-world objects to programming objects results in simplified programming as extra layers of translation between programmers and users is eliminated.
- Identification of common parts between and within applications facilitates modelling and code reuse.

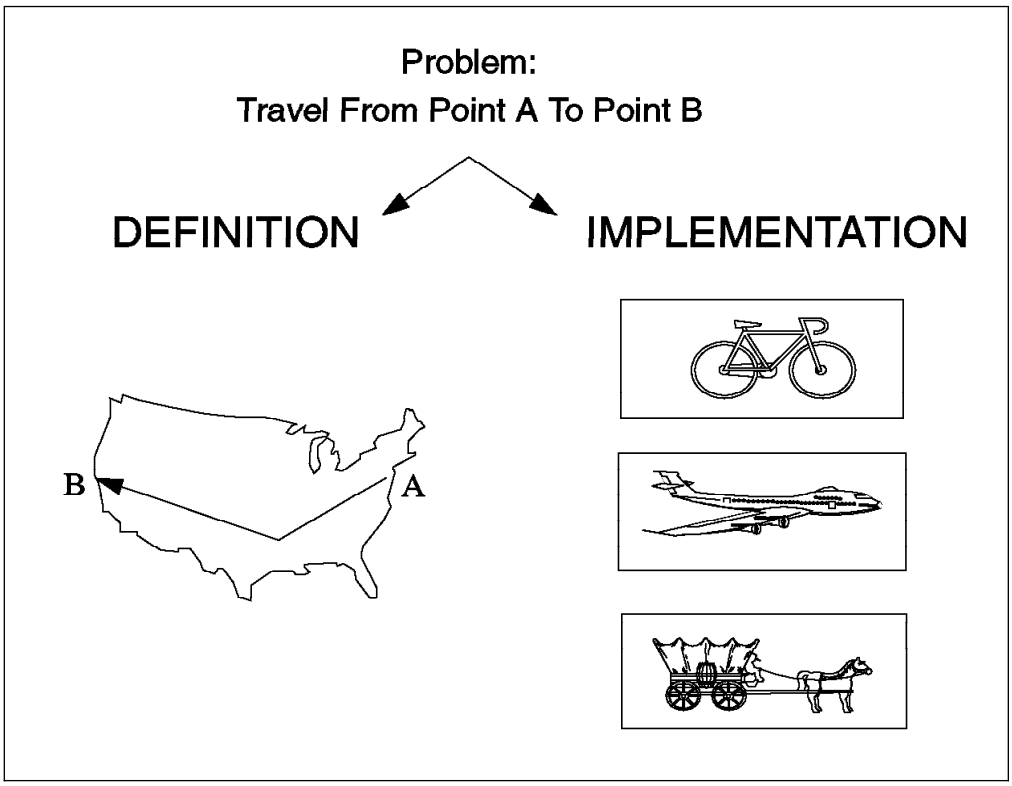

Figure 11. Polymorphism. Polymorphism is the ability to hide different implementations behind a common interface

2.1.2.2 Comparison of Some Object Oriented Languages

Table 3 (Page 1 of 2). Comparison of Some Major Common Object Oriented Languages

	C	C++	Objective C	Smaltalk-80	Ada
Class/Object	no	defined	defined	defined	no
Data Typing	yes can be overridden by "cast" operator	strong	yes	everything is an object	strong
Object Binding	compile time	either	either	Run-Time	compile time
Run-Time	none required	yes	n/a	An entire OO environment with tools is included. Automatic garbage collection is done for you	yes
Inheritance	no	yes	yes	yes	no
Multiple Inheritance	no	yes	yes	yes	no
Polymorphism (operator overloading)	no	yes	no	n/a	yes
Encapsulation	no native support	yes	yes	yes	yes

Table 3 (Page 2 of 2). Comparison of Some Major Common Object Oriented Languages					
	C	**C++**	**Objective C**	**Smaltalk-80**	**Ada**
Message Passing	no	yes	yes	yes	no
Comment	Not an OOPL. Only included for comparison	C++ programmers migrating from C do not necessarily see the OO benefits	Available on NeXTStep environment.	Large consumption of machine resource (memory and CPU)	A third generation languages with some OO features

2.2 Advantages of Language Independent Object Model Programming

Despite its promise, penetration of object-oriented technology in major commercial software products has progressed slowly. This is particularly true of products that offer only a binary programming interface to their internal object classes (that is, products that do not allow access to source code).

2.2.1 Limitations of Object Oriented Programming

The first obstacle that developers must confront is the choice of an object- oriented programming language.

So-called *pure* object-oriented languages (such as Smalltalk**) presume a complete run-time environment (sometimes known as a virtual machine), because their semantics represent a major departure from traditional, procedure-oriented system architectures. So long as the developer works within this environment, everything works smoothly and consistently. When the need arises to interact with foreign environments, however (for example, to make an external procedure call), the pure-object paradigm ends, and objects must be reduced to data structures for external manipulation. Unfortunately, data structures do not retain the advantages that objects offer with regard to encapsulation and code reuse.

Hybrid languages like C++**, on the other hand, require less run-time support, but sometimes result in tight bindings between programs that implement objects (called *class libraries*) and their clients (the programs that use them). That is, implementation detail is often unavoidably compiled into the client programs. Tight binding between class libraries and their clients means that client programs often must be recompiled whenever simple changes are made to the library. Furthermore, no binary standard exists for C++ objects, so the C++ class libraries produced by one C++ compiler cannot, in general, be used by C++ programs built with a different C++ compiler.

The second obstacle object-oriented software developers must confront is the fact that different object-oriented languages and toolkits embrace incompatible models of what objects are and how they work. Software developed using a particular language or toolkit is naturally limited in scope. Classes implemented in one language cannot be readily used from another. A C++ programmer, for example, cannot easily use classes developed in Smalltalk, or a Smalltalk programmer cannot make effective use of C++ classes. Object Oriented languages and toolkit boundaries become, in effect, barriers to interoperability.

Ironically, no such barrier exists for ordinary procedure libraries. Software developers routinely construct procedure libraries that can be shared across a

variety of languages, by adhering to standard linkage conventions. Object- Oriented class libraries are inherently different in that no binary standards or conventions exist to derive a new class from an existing one, or even to invoke a method in a standard way. Procedure libraries also enjoy the benefit of implementations which can be freely changed without requiring client programs to be recompiled, unlike the situation for C++ class libraries.

These are serious obstacles for developers who need to provide binary class libraries. In an era of open systems and heterogeneous networking, a single-language solution is frequently not broad enough. Certainly, mandating a specific compiler from a specific vendor in order to use a class library might be grounds not to include the class library with an operating system or other general-purpose product.

The solution to these problems is a “Standardized Object System Model,” which is used to describe the object's interfaces. This description model makes object languages neutral. It preserves key object-oriented characteristics without requiring that the user of a class and the implementer of that class use the same language.

The Object Management Group (OMG) has defined a Standardized Object System Model called *Common Object Request Broker Architecture (CORBA)*.

IBM's implementation of this standard is *The System Object Model objects (SOMobjects*)*.

Note

SOMobjects is not the only Standardized Object System Model, but it is the only one which is fully CORBA compliant. Other Standardized Object System Models are:

- Microsoft's OLE**
- Taligent's Object Oriented Operating System
- HP's DOMF**

2.2.2 The Solution: SOMobjects, a CORBA Standard Language Independent Object Model

The System Object Model (SOM) is a new object-oriented programming technology for building, packaging, and manipulating binary class libraries, which indeed fulfills the promise of object oriented technology. The advantages of SOM are:

- Class implementors describe the interface for a class of objects (names of the methods supported, return types, parameter types, and others) in a standard language called the Interface Definition Language, or IDL.
- Methods may be implemented in a preferred programming language (which may be either an object-oriented programming language or a procedural language such as C*).

This means that programmers can begin using SOM quickly, and also extends the advantages of OOP to programmers who do not use object-oriented programming languages.

A principal benefit of using SOM is that SOM accommodates changes in implementation details and even in certain facets of a class's interface, without breaking the binary interface to a class library and without requiring recompilation of client programs. As a rule of thumb, if changes to a SOM class do not require source-code changes in client programs, then those client programs will not need to be recompiled. This is not true of many object-oriented languages, and it is one of the chief benefits of using SOM. For instance, SOM classes can undergo the following structural changes yet retain full backward, binary compatibility:

- Adding new methods
- Changing the size of an object by adding or deleting instance variables
- Inserting new parent (base) classes above a class in the inheritance hierarchy
- Relocating methods upward in the class hierarchy

In short, implementers can make the typical kinds of changes to an implementation and its interfaces that evolving software systems experience over time.

SOM is language neutral unlike the object models found in formal object-oriented programming languages. It preserves the key OOP characteristics of encapsulation, inheritance, and polymorphism, without requiring that the of a SOM user class and the implementer of a SOM class use the same programming language.

SOM is said to be language-neutral for four reasons:

1. All SOM interactions consist of standard procedure calls. On systems that have a standard linkage convention for system calls, SOM interactions conform to those conventions. Most programming languages that can make external procedure calls can use SOM.
2. The form of the SOM Application Programming Interface, or API (the way that programmers invoke methods, create objects, and so on) can vary widely from language to language as a benefit of the SOM bindings. Bindings are a set of macros and procedure calls that make implementing and using SOM classes more convenient by tailoring the interface to a particular programming language.
3. SOM supports several mechanisms for method resolution that can be readily mapped into the semantics of a wide range of object-oriented programming languages. Thus, SOM class libraries can be shared across object-oriented languages that have differing object models. A SOM object can potentially be accessed with three different forms of method resolution:
   - *Offset resolution:* roughly equivalent to the C++ *virtual function* concept. Offset resolution implies a static scheme for typing objects, with polymorphism based strictly on class derivation. It offers the best performance characteristics for SOM method resolution. Methods accessible through offset resolution are called static methods, because they are considered a fixed aspect of an object's interface.
   - *Name-lookup resolution:* similar to that employed by Objective-C** and Smalltalk. Name resolution supports untyped (sometimes called *dynamically* typed) access to objects, with polymorphism based on the actual protocols that objects honor. Name resolution offers the opportunity to write code to manipulate objects with little or no awareness of the type or shape of the object when the code is compiled.

- *Dispatch-function resolution:* a unique feature of SOM that permits method resolution based on arbitrary rules known only in the domain of the receiving object. Languages that require special entry or exit sequences or local objects that represent distributed object domains are good candidates for using dispatch-function resolution. This technique offers the highest degree of encapsulation for the implementation of an object, with some cost in performance.

4. SOM conforms fully with the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standards. In particular,
   - Interfaces to SOM classes are described in CORBA's Interface Definition Language, IDL, and the entire SOMobjects Toolkit supports all CORBA-defined data types.
   - The SOM bindings for the C language are compatible with the C bindings prescribed by CORBA.
   - All information about the interface to a SOM class is available at run time through a CORBA-defined Interface Repository.

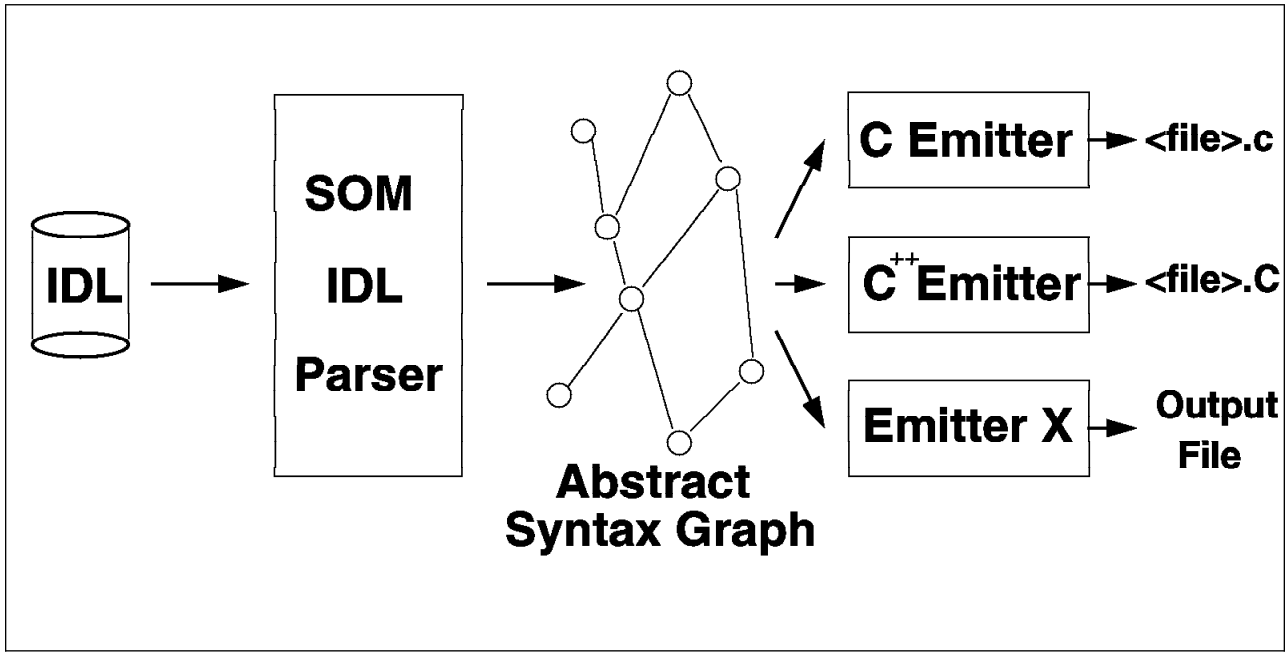

Figure 12. SOM Compiler. How SOM translates IDL file to required output.

SOM is not intended to replace existing object-oriented languages. Rather, it is intended to complement them so that application programs written in different programming languages can share common SOM class libraries. For example, SOM can be used with C++ to:

- Provide upwardly compatible class libraries. When a new version of a SOM class is released, client code needn't be recompiled, so long as no changes to the client's source code are required.
- Allow other language users (and other C++ compiler users) to use SOM classes implemented in C++.
- Allow C++ programs to use SOM classes implemented using other languages.

- Allow other language users to implement SOM classes derived from SOM classes implemented in C++.
- Allow C++ programmers to implement SOM classes derived from SOM classes implemented using other languages.
- Allow encapsulation (implementation hiding) so SOM class libraries can be shared without exposing private instance variables and methods.
- Allow dynamic (run-time) method resolution in addition to static (compile-time) method resolution (on SOM objects).
- Allow information about classes to be obtained and updated at run time. (C++ classes are compile-time structures that have no properties at run time.)

The user of a SOM class and the implementer of a SOM class need not use the same programming language, and neither is required to use an object-oriented language. The independence of client language and implementation language also extends to subclassing: a SOM class can be derived from other SOM classes, and the subclass may or may not be implemented in the same language as the parent class(es). Moreover, SOM's run-time environment allows applications to access information about classes dynamically (at run time).

2.3 Conclusion: Making Reuse a Reality

Creating interchangeable, reusable software components is very much a reality today: developers can choose from a variety of object oriented programming languages, design and analysis aids, and visual programming tools. They can also find OO enablers and frameworks built into the latest releases of some operating systems (for example IBM Operating System/2* Versions 2.0 and 2.1). What were not available, until now, are some key pieces needed to complete a system that works as smoothly and openly as the hardware development process.

With SOM objects tools, the programmers can supply these missing pieces by breaking objects free from ties to a specific programming language (see Table 4). On a single machine, the System Object Model provides an object-structured protocol that allows applications to access and use objects and object definitions, regardless of what programming language created them, with no need to recompile the application. SOM's language-neutral character allows robust software objects to be easily used and reused wherever they're needed. It also enables a greater degree of openness than ever before in the development and use of object oriented programming (OOP) facilities across multiple operating platforms.

Table 4 (Page 1 of 2). Multimedia Requirements Provided by Different Development Approaches

	Structured Approach	**Object Oriented**	**Independent Object Model**
Complexity Management	no	yes	yes
Open	partial	no	yes
Reuse	partial	yes	Yes
Data Abstraction	limited	yes	yes

Table 4 (Page 2 of 2). Multimedia Requirements Provided by Different Development Approaches

	Structured Approach	Object Oriented	Independent Object Model
Flexibility	no	yes	yes
Portability	yes	no	yes
Heterogeneity	yes	no	yes
Distribution	no	no	partial
Expandability	limited	yes	yes

SOMobjects incorporates Distributed SOM (DSOM) technology that provides a base for OOP development and use over entire networks. With the IBM SOMobjects Developer Toolkit, programmers can start taking advantage of SOM and DSOM immediately. SOMobjects complies with industrywide standards of the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA). It provides an extensive set of facilities for putting the power and speed of OO technology to work.

Chapter 3. Ultimedia Services/6000

The continued growth of multimedia within the business environment is promoting demand of application program support with improved capabilities to meet the expectations of the current customer market. The AIX Ultimedia Services/6000 is IBM's response to this customer's demand.

AIX Ultimedia Services/6000 are installable extensions to the base AIX Operating System that support the use of audio and video data on RISC System/6000*.

Ultimedia Services/6000 is an integral part of the IBM desktop computing solution for technical and commercial AIX/6000* environments. It provides ease of use in accessing various multimedia types and developing multimedia applications, thereby enhancing customer productivity and communications.

It provides the ability to enhance interfaces and applications with the use of audio and video facilities. It enables customers to use standard multimedia types to provide a more effective communication media. Customers may quickly enhance current application programs or develop new applications in a true multimedia environment.

It provides enhanced facilities for working with various media types on RISC System/6000s and offers a robust Application Programming Interface (API), new Graphical User Interface (GUI) and other tools to support audio and video. Sample code, video clips and audio clips are provided for user convenience and to illustrate usage of multimedia in a business environment.

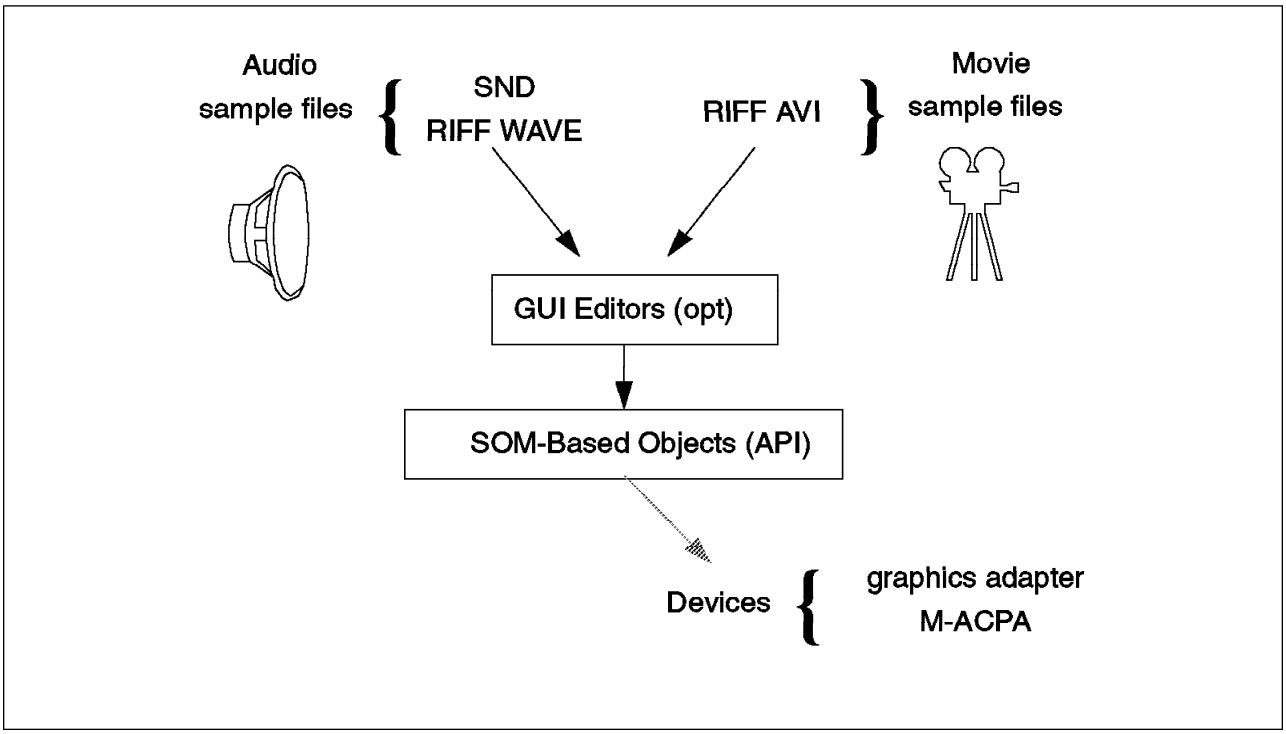

Figure 13. AIX Ultimedia Services/6000 Product

An Application Programming Interface and a Graphic User Interface enable basic video and audio services on the RISC System/6000 platform. Also included are

System Object Module (SOM) objects that demonstrate media presentation and translation capability. These encourage multimedia application programmers to promote the development, enablement, or migration of multimedia applications to the AIX platform and facilitate media compatibility across applications.

AIX Ultimedia Services/6000 has been designed to support industry standard audio and video formats from PC and workstation environments. They promote the sharing of audio and video information while protecting investment in existing hardware, software and multimedia content. It uses the existing IBM M-Audio Capture and Playback Adapter (M-ACPA) for CD quality sound support. Movies are played using advanced decompression and playback routines that rely on the power of the RISC System/6000 instead of requiring specialized hardware. This scalability of service allows multimedia performance to follow the trend of processor speeds. The goal is to provide basic services so the user has a choice and is not forced to use a PC to meet basic multimedia needs.

Note

UNIX workstations have a significant advantage as video support is moving to software playback instead of using specialized hardware. The available processing power can support and manage more complex video editing and playback scenarios such as multiple audio/video streams or higher resolution images. Generally, UNIX network services provide greater capability for networked multimedia solutions. The PC areas currently have more options and applications including access to a wide range of multimedia titles not yet available on the workstation.

3.1 Main Characteristics

Ultimedia Services/6000 provides an object-oriented, easy-to-use set of libraries which greatly eases creation of multimedia applications. This Object-Oriented interface to multimedia devices and libraries greatly reduces programming time. A facility for editing audio and video clips is provided. Sample applications are available for demonstrations and technique analysis. This allows the developer to study and emulate existing modules when creating new applications.

AIX Ultimedia Services/6000, one of the most advanced multimedia computer environments, was created through the fusion of four different technologies:

- **Multimedia -** the ability to manage different types of data.
- **Object-Oriented -** the ability to write reusable and easily extendable software.
- **Media Control Interface** (MCI)** and **Multi Media Input Output** (MMIO) -** the PC standard to control device drivers and file Input/Output, respectively.

  Media Control Interface (MCI)

  MCI is a high-level standard command control interface to multimedia devices and resource files. It provides applications with device-independent capabilities for controlling audio and visual peripherals. It has been developed jointly by IBM and Microsoft**.

- **SOM** and **DSOM,** which makes all heterogeneous, CORBA compliant, object-oriented software available.

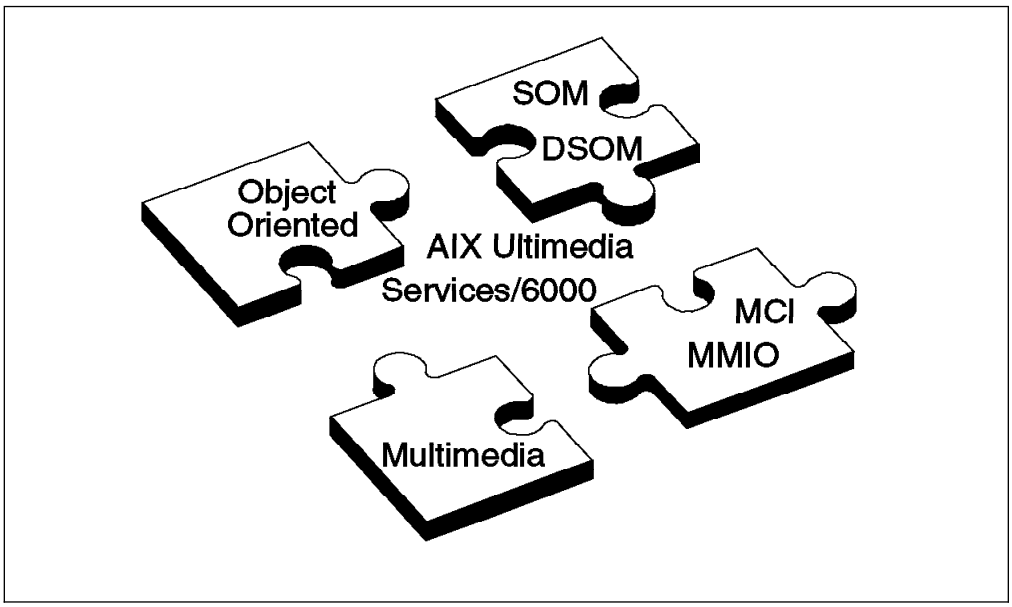

Figure 14. AIX Ultimedia Services/6000: Technological Aspects

The capabilities of all these technologies are required to guarantee a good multimedia environment as described in section 1.4, “What Makes a Good Multimedia Environment?” on page 10.

3.1.1.1 Ultimedia Services/6000 is Open and Heterogeneous

AIX Ultimedia Services/6000 supports industry standard data types and interchange formats. An an increasing number of customers require adherence to industry standard object-oriented program models to establish and maintain products.

The application programming environment, provided by this product, allows applications to use current and upcoming hardware offerings without recompiling the application. This programming environment contains an expandable set of objects created with IBM's System Object Model. This guarantees complete interoperability and portability with all CORBA standard objects. Applications written in C and C++ are able to use these objects and Ultimedia Services/6000's objects are able to use other CORBA compliant library objects.

File interoperability across platforms is achieved by supporting popular file formats and media compression algorithms used in PC and workstation arenas.

3.1.1.2 Ultimedia Services/6000 is Flexible

Objects are the natural way to implement flexibility in a software system because they communicate with each other using only messages addressed to well-defined interfaces. Developers are only required to specify the functionality of their programs: that is the messages exchanged by the objects. Ultimedia Services/6000 object library allows code to be written using the *plug and play* approach. Multimedia objects, provided by the Ultimedia Services/6000, can be used to easily add audio and video capability within the framework of existing applications.

The object-oriented library encourages use of object-oriented software engineering programming such as data abstraction, encapsulation and sub-classing, but it does

not require them. Developers can also use Ultimedia Services/6000 as they would use a normal library, to create traditional structured applications.

3.1.1.3 Ultimedia Services/6000 is Extendable

The Ultimedia Services/6000 object-oriented approach allows code to be easily extended. The inheritance mechanism allows Ultimedia Services/6000 objects to be used to produce new objects. Using this library, a developer has to write only the code needed to differentiate new classes from the old ones. With Ultimedia Services/6000 all CORBA compliant class libraries can be used, modified and extended.

Ultimedia Services/6000 was written using SOM technology; therefore, it is possible to use the OO inheritance mechanism to extend classes of objects provided by the product. It is also possible to use and extend any other CORBA compliant binary class libraries.

3.1.1.4 Ultimedia Services/6000 is Reusable

Reuse is the key to dramatically improved software productivity. Coding time is reduced and software quality is improved through the reuse of tried and trusted objects. A significant benefit of the object-oriented approach used for the AIX Ultimedia Services classes is the increased potential for reusable code. The independent nature of these objects enables them to be coupled together in various ways. Applications that manipulate multimedia data can be easily created and assembled. This reduces the time and effort to produce new applications or to enhance old ones.

Common functionality in different applications can be realized by using common shared objects. This leads to a more consistent user interface across the system as a whole.

Identification of common parts between and within applications greatly facilitates modelling and code reuse.

3.1.1.5 Ultimedia Services/6000 Allows High Specialization

Objects can be specialized more easily than functions, because backward compatibility is much easier with objects than with functions. For example, programs written to use an object could continue to use the original features without change. Any new programs could take advantage of new features. Obtaining the same result with a function is more complex.

The object-oriented approach used to create the Ultimedia Services/6000 library, allows software to be created using a *plug and play* approach; the same approach used in the assembly of PC hardware. This allows production of highly specialized software components. In fact, the application production chain can be broken into two parts: programmers who develop new applications (by assembly of objects) and programmers who create and improve objects. Developers can become highly specialized on well defined areas of software engineering as their hardware colleagues are. The Ultimedia Services/6000 objects are reusable, highly-specialized, continuously improved units of code. Moreover, the use of SOM allows them to be CORBA compliant.

3.1.1.6 Ultimedia Services/6000 Is Not Yet Distributed

This release of AIX Ultimedia Services/6000 (1.1) is SOM compliant, but it is not DSOM compliant. It does not allow transparent access to objects across networks. In this release, object distribution is provided only through the X-Window** System. Support is provided to distribute still images and motion video (without audio), but not to distribute audio.

Distribution

Two steps are required to create an integrated, distributed multimedia information system:

- Interconnectivity
- Interoperability

Interconnectivity means that two or more applications running on the same machine or on different machines can exchange messages (this means they can communicate), but they cannot cooperate. For example, in the client-server model, clients send requests for services and the server sends back the requested services. There is communication, because client and server exchange information, but there is not cooperation, because the server works without using any client resources.

Interoperability means that two or more applications can *interact* to execute tasks jointly. Clients can request services involving resources that reside anywhere across networks. In this way, objects can request services from other objects and work together, independent of object and resource location.

The next release of this product is expected to be DSOM compliant, so Ultimedia Services/6000 objects would be shareable across networks. This will allow true interoperability between networked computers.

3.2 Functional Aspects: the Components

Ultimedia Services/6000 is organized in three different parts:

1. **The Ultimedia Services/6000 binary class library**, developed using SOM technology, allows the creation of object oriented, event-driven multimedia applications.
2. **The Ultimedia Services/6000 audio player/editor/recorder application suite and a motion video player/editor**, an extensive library of audio samples, audio/video samples and application demonstrations are provided to stimulate customer acceptance and promote multimedia development on the RISC System/6000.
3. **Sample desktop applications**, which demonstrate the usage of multimedia functions in an interactive environment. Samples are interactive from the mouse/keyboard interface and employ animations and sound in synchronous format. Four samples are included which demonstrate the usage of multimedia in banking, assembly, real estate and public information.

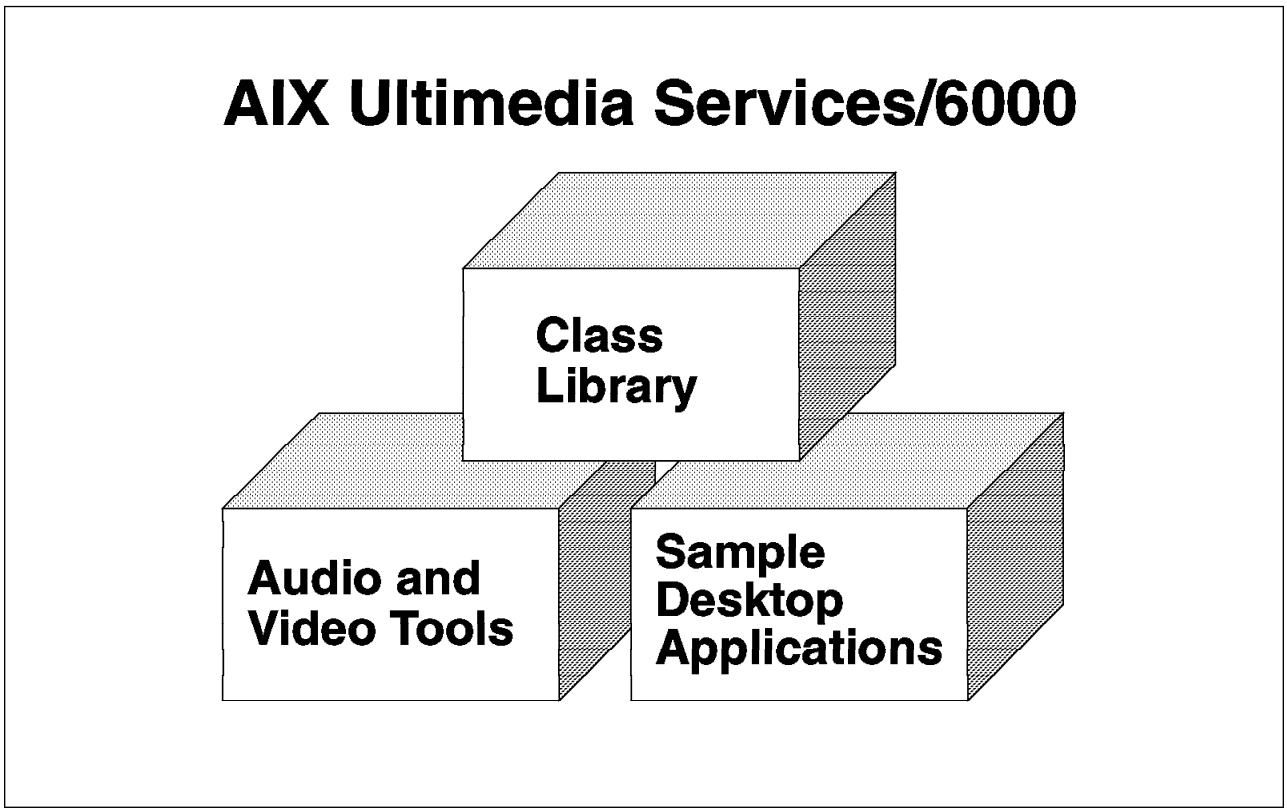

Figure 15. AIX Ultimedia Services/6000 Components

Ultimedia Services/6000 also includes some sample programs which show how some objects are made and how they work. Audio and video samples are also included.

3.2.1 Ultimedia Object's Library

AIX Ultimedia Services/6000 features programming support for various services including:

- Software video compression and decompression (CODEC) algorithms
- Audio and Video editors/player
- Support of numerous popular multimedia file formats
- Audio recording

These services are provided through the use of the following classes of objects (see Figure 16 on page 35):

- Media Handler Objects
- Video Decoder and Encoder (CODEC) Objects
- File Access Objects
- Configuration Objects
- Audio Filter Objects

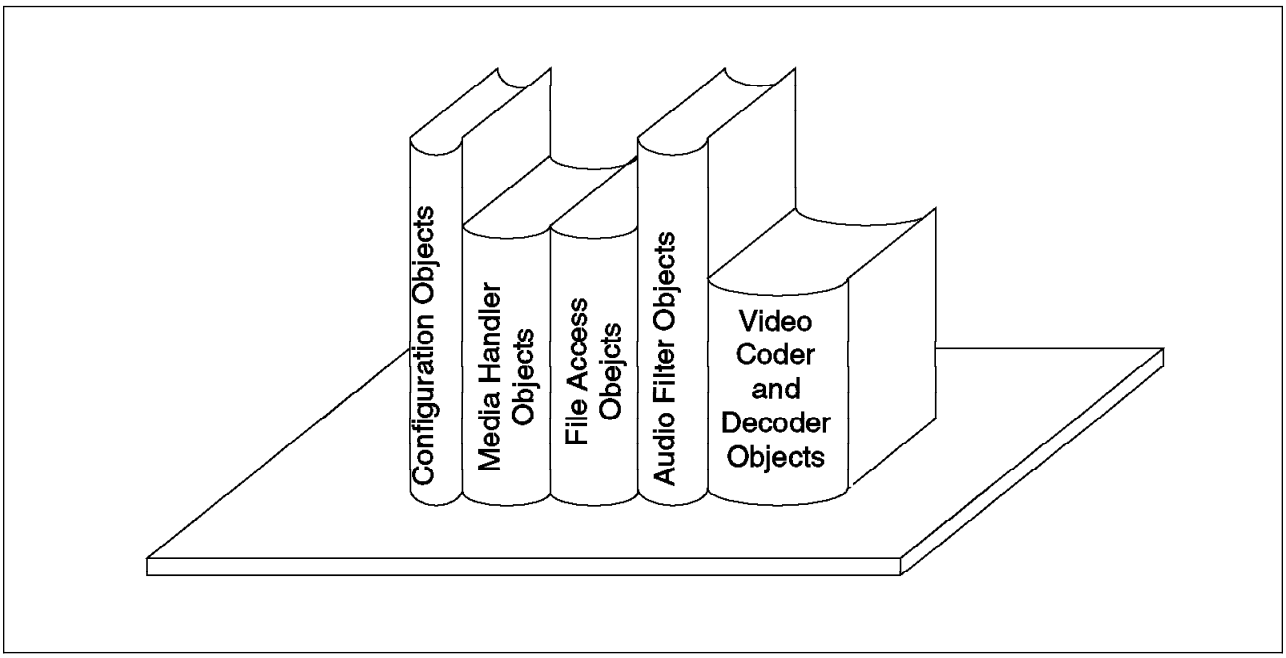

Figure 16. Object Library Components

These objects can be divided in two groups as shown in Figure 17 on page 36. Some objects are used to handle media, and others are used to translate media to and from its basic presentation form.

- The first group of objects called *media handler objects* are used to play the media within an application. These objects, offering presentation capability, share concepts with the Media Control Interface (MCI) found on Microsoft Window Multimedia Extension** and on Multimedia Presentation Manager/2 (MMPM). They play audio, video or synchronized audio and video using methods much like the MCI messages.
- The second group of objects is actually composed of two kinds of objects:
  - Media Conversion Objects (video codec and filter objects)
  - File Access Objects (configuration and file access objects)

  The media conversion objects include only video decoders and encoders and audio filters. There are no objects to perform audio encoding and decoding or for video format translation. Audio encoding and decoding is performed directly by hardware (this is a accomplished by loading microcode on an audio card). Instead of a movie filter, some sample programs are provided, but no real object class. This group of objects offering media translation capability share concepts with the Multimedia Input Output (MMIO) library also found on PC products. They offer support to more sophisticated applications by providing them with shared system services for reading and interpreting standard formats and for compression and decompression of most of these same file formats.

  Figure 17 on page 36 shows how Ultimedia Services/6000 Object Library is logically organized.

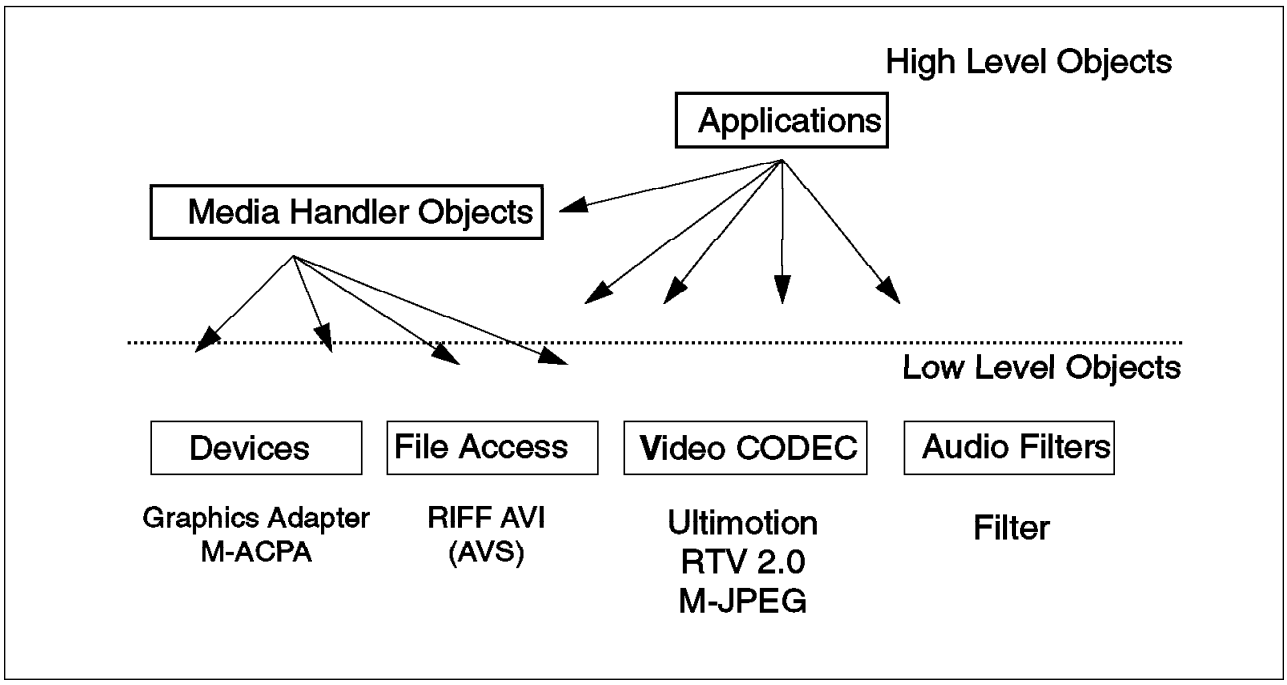

Figure 17. AIX Ultimedia Services/6000 Object Library. The picture shows the objects included in Ultimedia Services/6000. They can be organized into two levels: high level objects (presentation level) and low level objects (device and file management)

3.2.1.1 Media Handler Objects

Ultimedia Services/6000 media handler objects provide a means of controlling audio and video data much like audio and video home entertainment systems: a movie can be loaded into a movie player, played, and then paused when an interesting sequence occurs. The complexity of moving the information through the system is hidden by the media handler object, which uses other processes to control the data flow and handle device-specific dependencies.

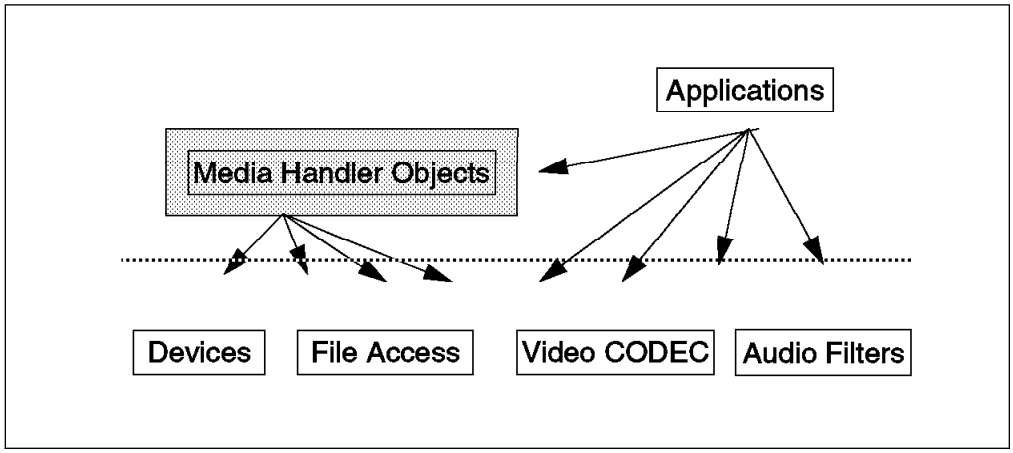

Figure 18. Media Handler Objects

The media handler objects are an *audio player/recorder* and a *movie player*.

- The audio player/recorder object can be used to play and record audio in the PCM, ADPCM, A-LAW and μ-LAW audio standard. For these types of audio standards, the following variety of file formats can be used:

- WAVE (.wav)
- RIFF (.au)
- SND (.snd)
- RAW (.raw)

(The first three file formats contain attributes within the file, which describe how the audio sample was encoded.) The encoding characteristics of WAVE and SND formats are shown in the following table.

Table 5. Audio Encoding

File Format	Data Format	Sample Rates	Bits per Sample
WAVE	PCM	8000, 11025, 22050, 44100	8 or 16
	A-LAW	8000, 11025, 22050, 44100(mono only)	8
	μ-LAW	8000, 11025, 22050, 44100(mono only)	8
SND	PCM	8000, 11025, 22050, 44100	8 or 16
	A-LAW	8000, 11025, 22050, 44100(mono only)	8
	μ-LAW	8000, 11025, 22050, 44100(mono only)	8

The .raw format does not contain this type of information. For more details on multimedia file formats see Appendix A, “Multimedia Technologies” on page 57).

- The movie player object can be used to play synchronized audio and video from a file. The movie player works with the AIXwindows* system, using an AIXwindows window to display the video frames. The playback of video frames does not need special hardware adapters. An audio adapter is required only if the video also contains sound. It is capable of playing movies recorded in the following formats:
  - Motion Joint Photograph Experts Group (M-JPEG)
  - Ultimotion Matinee

  These are two kind of *Resource Interchange File Format (RIFF)* file formats. This type of format allows one video track and several audio tracks to be interleaved into the file. These files are indicated by a .AVI extension. For more information on these file formats see Appendix A, “Multimedia Technologies” on page 57.

Synchronization

During normal playback, synchronization between real-time media such as audio and video, occurs as the information streams physically proceed through the player. Time is measured in samples of audio or frames of video, or in milliseconds.

Because these measurements of progress are tightly related, any sample of audio or video can be thought of as having a time stamp associated that represents the length of time to be played. If the playback of the media is altered, such as speeding up the play, the time stamp association would not be consistent with real time, but would still be useful as a reference specifying a consistent position in the media.

3.2.1.2 Video Decoder and Encoder Objects

Ultimedia Services/6000 provides video compression and decompression through the encoder class and decoder class respectively. To use one of these classes, an application first initializes the object and then calls the object's methods to perform the needed actions.

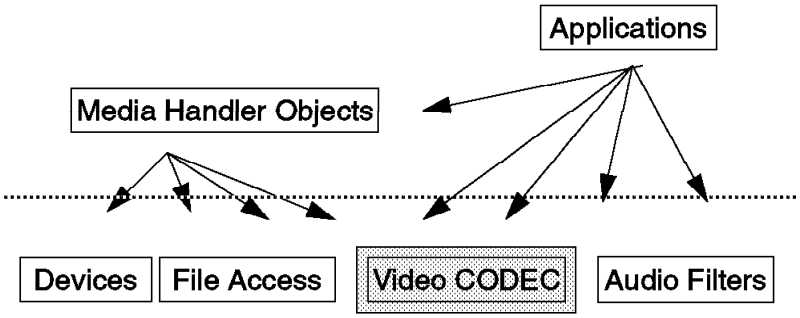

Figure 19. Video Decoder and Encoder Objects

Ultimedia Services/6000 supports three video CODECS:

- Motion Joint Photographs Experts Group (M-JPEG)
- Ultimotion Matinee
- RTV 2.0

Ultimedia Services/6000 provides both encoder and decoder for the first two; while only the decoder is provided for the RTV 2.0 format.

These codecs are intended for continuous-tone input; for example digitized photographs and motion video. For more information on these formats see Appendix A, “Multimedia Technologies” on page 57.

Table 6. Motion Video Codecs Characteristics	
Algorithm	**Advantages**
M-JPEG	• Arbitrary frame size • Higher visual quality • Non-proprietary algorithm • Compression in software
Ultimotion Matinee	• Presentation-quality video • Faster decoding in software • Fully compatible with PS/2 motion video • Compression in software
RTV 2.0	• Presentation-quality video • Fully compatible with PS/2 motion video

3.2.1.3 File Access Objects

File Access Objects allows access to fields and chunks of information in files kept in standard formats.

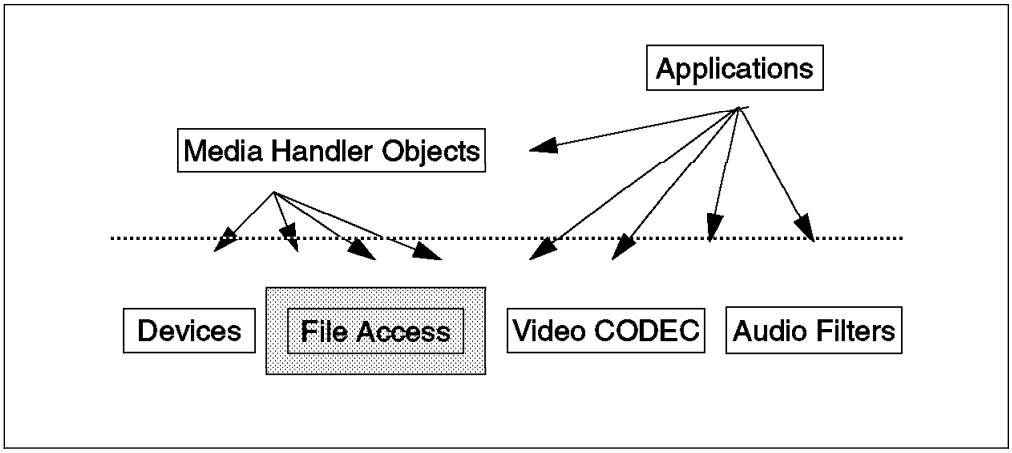

Figure 20. File Access Objects

These objects are useful for applications which need to read, change, or write file formats. Ultimedia Services/6000 provides the following objects:

- **RIFF File Access (UMSRiffReadWrite).** This object provides methods to access, read, write and parse files written in the RIFF format (for more details on RIFF formats, see Appendix A, “Multimedia Technologies” on page 57).
- **AVI File Access (UMSAVIReadWrite).** This object provides methods to locate, read, write the data and the header in the AVI format (for more details on AVI formats, see Appendix A, “Multimedia Technologies” on page 57).
- **AVS File Access (UMSAVSReadWrite).** This object provides methods to locate, read, write the data and the header in the AVS format (for more details on AVS formats, see Appendix A, “Multimedia Technologies” on page 57).
- **Filetype Detector (UMSFiletypeDetector).** This object is used to detect the type of a given media file.

3.2.1.4 Configuration Objects

Ultimedia Services/6000 configuration objects are used to allow the filetype detector object to recognize the set of possible file types and to allow applications to request an audio device.

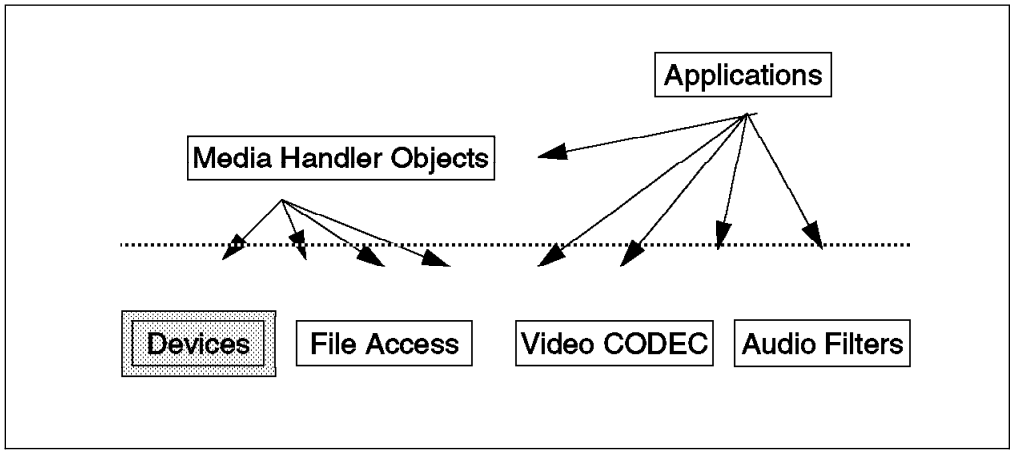

Figure 21. Configuration Objects

There are two configuration objects:

- **UMSConfig Object** - used generally by the Filetype Detector Object to detect the input file types.
- **UMSAddConfig Object** - used to add new device-driver aliases to an application. It can also be used to add new file types or to add alternate devices to use when one is busy.

3.2.1.5 Audio Filter Objects

Ultimedia Services/6000 audio filter objects allow the changing of both the audio format of a file (for example from PCM to μ-LAW) and the bit sampling.

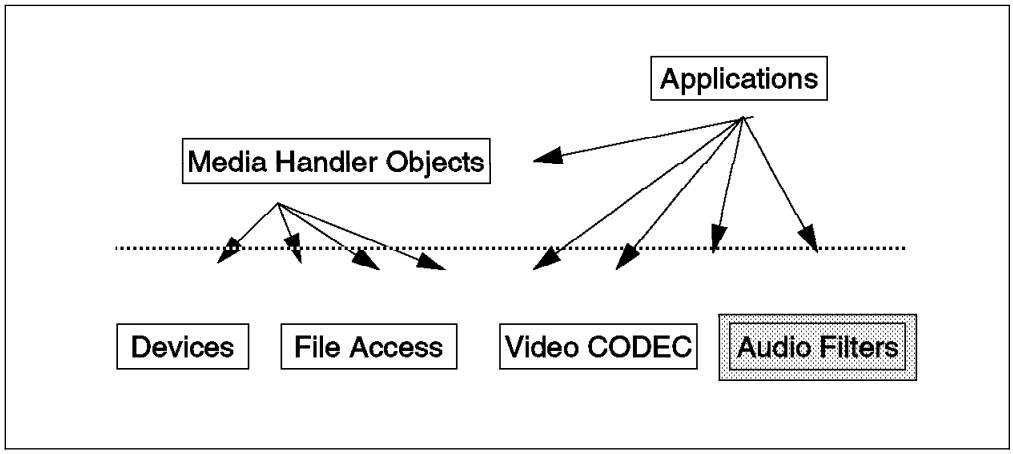

Figure 22. Audio Filter Objects

The Audio Filter Objects provided are:

- **UMSFilter Object -** A parent class defining the methods for the whole family of filters. It exists only for inheritance.

- **UMSADPCMtoPCM16 Object -** Converts file format from ADPCM to PCM at 16-bit.
- **UMSPCM16toADPCM Object -** Converts file format from PCM at 16-bit to ADPCM.
- **UMSALAWtoPCM16 Object -** Converts file format from A-LAW to PCM at 16-bit.
- **UMSPCM16toALAW Object -** Converts file format from PCM at 16-bit to A-LAW.
- **UMSMULAWtoPCM16 Object.** Converts files format from μ-LAW to PCM at 16-bit.
- **UMSPCM16toMULAW Object -** Converts file format from PCM at 16-bit to μ-LAW.
- **UMSPCM8toPCM16 Object -** Converts file format from PCM at 8-bit to PCM at 16-bit.
- **UMSPCM16toPCM8 Object -** Converts file format from PCM at 16-bit to PCM at 8-bit.
- **UMSByteOrder Object -** Changes the byte order of 16-bit PCM.
- **UMSChainFilter Object -** Allows creation of audio filter object *pipes* where the output of the first object is passed as input to the second, and so forth.
- **UMSSamplingRate Object -** Changes sampling rate of PCM at 16-bit.

These objects provide the capability to:

- Convert various audio formats to 16-bit pulse code modulation (PCM)
- Convert 16-bit PCM into other audio formats
- Modify the sampling rate of 16-bit PCM
- Convert the byte order of 16-bit PCM
- Chain several of these objects together

3.2.2 Tools Programs

AIX Ultimedia Services also provides two fully Common User Access (CUA) compliant and icon-based multimedia tools:

- An audio player/editor/recorder, which can play audio from a variety of sources, file formats, and compression types.
- An audio/video *movie* player/editor, which is able to present movie files compressed in Ultimotion and Motion JPEG formats. All 8 and 24-bit IBM color graphics adapters are supported.

These two players provide an easy way for the novice user to explore multimedia data types. Easy-to-use tools provide simple cut/copy/paste commands for video and audio editing. End-users can easily edit and merge current audio and movie files to update old material or create new deliverables.

Simple audio and audio/video samples are provided for demonstration purposes.

These tools are an example of the use of the Ultimedia services/6000 object library. These tools can run from any AIX window or desktop and use the AIX Ultimedia Services/6000 objects to perform multimedia play or record operations.

3.2.2.1 Movie Player/Editor Program

The Ultimedia Services/6000 Movie Editor plays and provides editing capability for audio/video interleaved (AVI) files with Ultimotion Matinee and M-JPEG video compression.

The Movie Editor shown in (Figure 23 on page 43), shows the main window that is displayed each time the Movie Editor is started. From this panel the user can perform every movie operation allowed in a natural way.

The Movie Editor is organized as follows:

- In the center of the panel, there is the display window (320x240 pixel) where the movie is displayed.
- Buttons on the bottom of the panel allow the movie to be played, stopped, paused, scanned and played step-by-step (both in forward and in backward mode).
- Four pulldown menu buttons on the top bar allow the user to perform editing functions (typically merging parts of the edited file with other movie files).
- A volume scroll bar and the mute button on the left permit the audio volume to be controlled.

A record button is also provided, but movie recording is not supported in this release of the product.

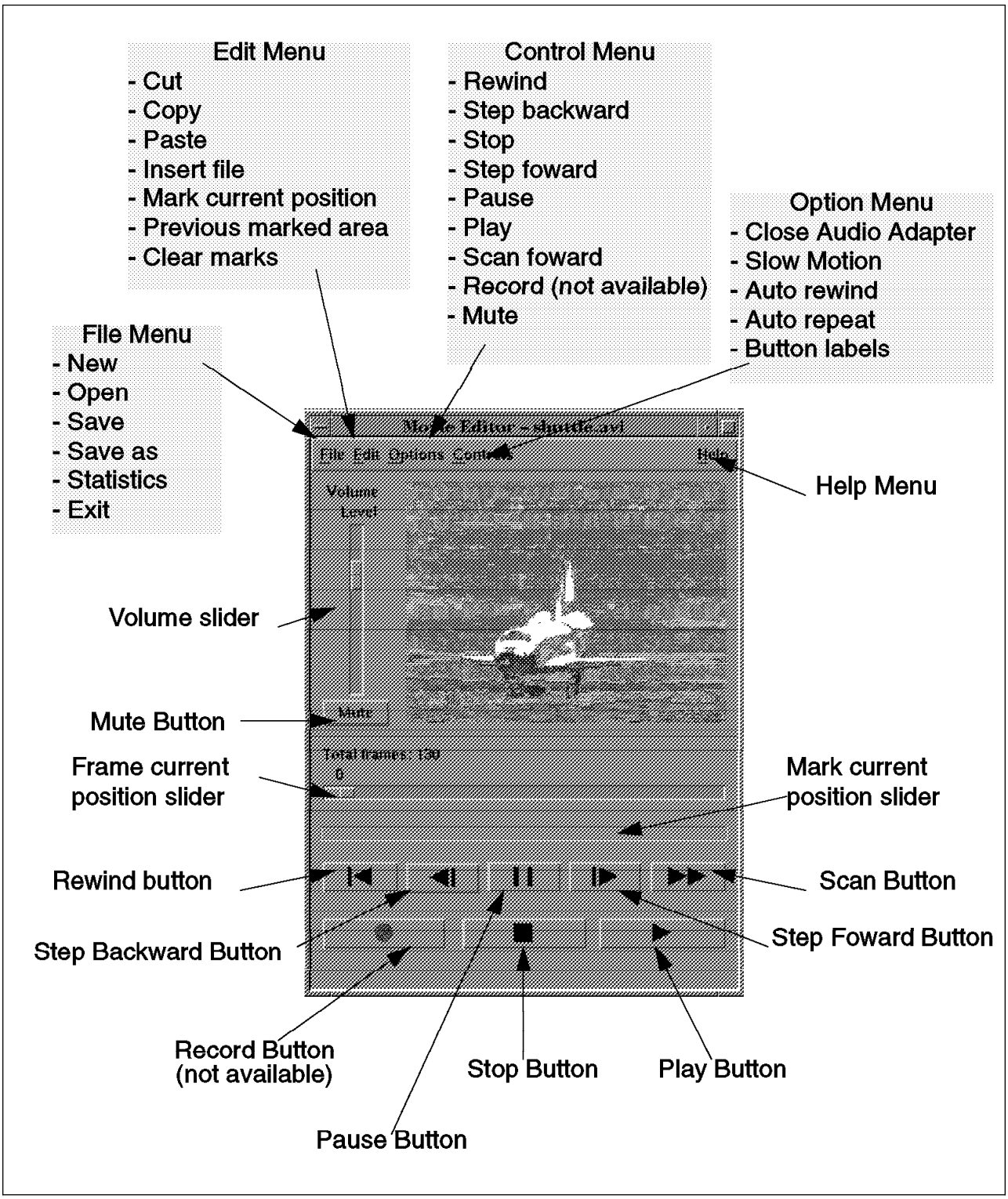

Figure 23. Movie Editor

3.2.2.2 Audio Player/Editor/Recorder Program

The Ultimedia Services/6000 Audio Waveform Editor plays, records, and provides editing capability for audio file formats.

The Audio Editor displayed in (Figure 24 on page 45), shows the main panel that is displayed each time the Audio Editor is started. From this panel every audio operation allowed can be performed in a natural way.

The Audio Editor panel is organized in the following way:

- In the center of the panel, there are two areas where the audio file waveform is displayed. It displays the full file waveform and current waveform of the played segment. The segment of the waveform can be enlarged or reduced using the vertical detail buttons.
- On the bottom, there are six buttons: play, stop, pause, begin, end and record, respectively, which allow the usual audio operation.
- The four pulldown menu buttons on the top allow the editing operations (cut, past and merge).
- A volume scroll bar and the mute button on the left permit audio volume to be controlled.

This tool provides simple, but complete audio support to create, edit or play voice, music and sound.

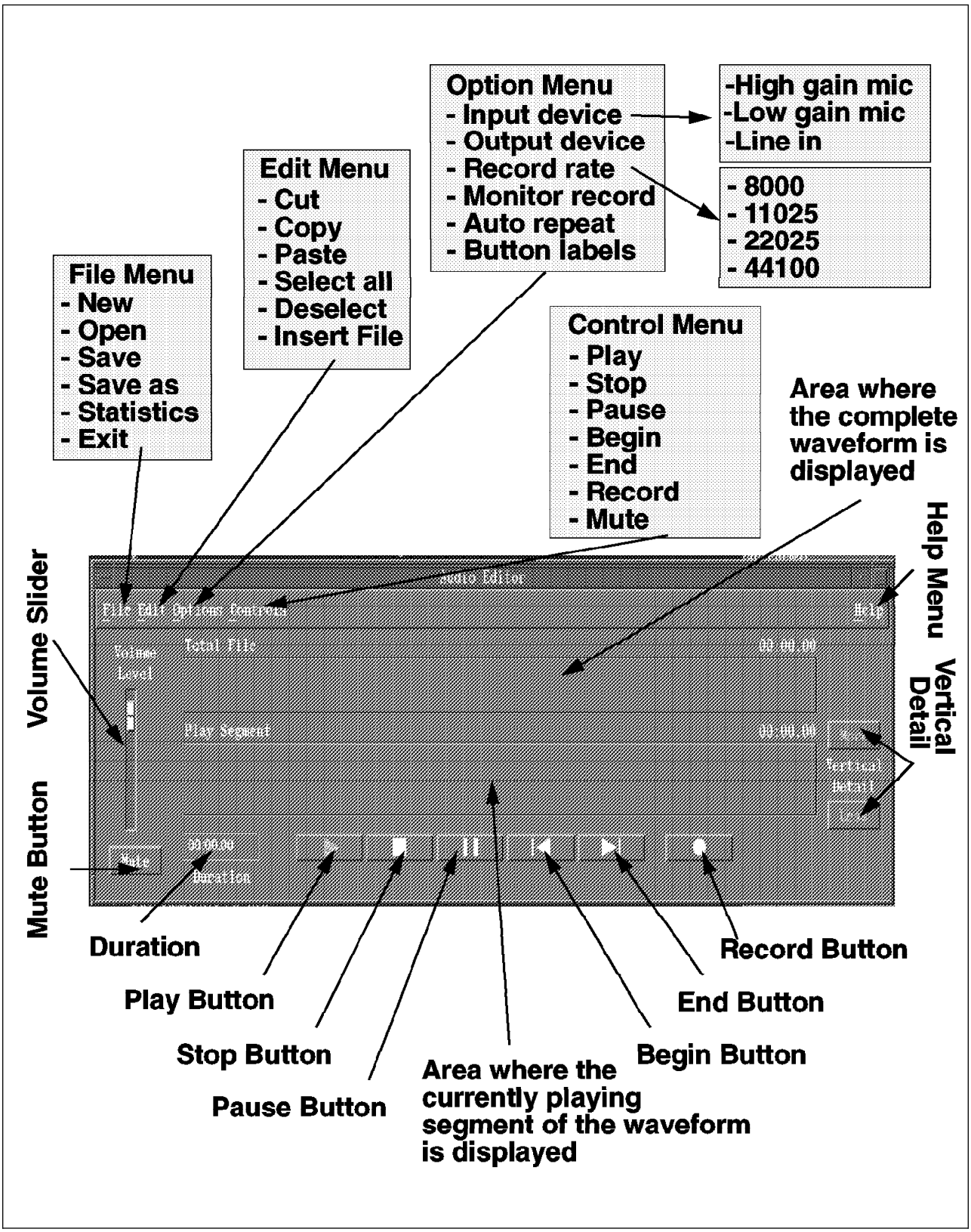

Figure 24. Audio Waveform Editor

3.2.3 Demo Examples

Four demo examples are provided with AIX Ultimedia Services/6000 showing how business applications can be improved dramatically using a multimedia environment.

The four business areas are:

- Banking Industry
- Realty Market
- Retail Business
- Employee Training

This suite of four applications was conceived and developed by Sybase** Inc., using Gain Momentum software and AIX Ultimedia Services/6000.

Note

Gain Momentum software provides an object oriented application development environment for delivering open client-server business solutions.

This is a typical example of software enhancing using Ultimedia Services/6000 Each demo is organized in three parts:

1. The first part shows common problem areas in this type of business.
2. The second part shows the proposed solutions for the problems:

   The bank's customers are provided with a list of interactive services available (nearest ATM services, and so forth).

   Realty market customers and employees are provided with a virtual visit to a house. The user can see every part of the house, general area, and neighborhood by simply pressing buttons on the multimedia workstation.

   The retail sport shopper is provided with an interactive catalog and database of sport products.

   Factory workers are provided with an interactive assembly course. Employees can use this interactive course to improve or to check on their skills.
3. The third part shows the hardware solution proposed.

Figure 25 on page 47 and Figure 26 on page 48 show the first screen of the Ultimedia Services/6000 demos and a typical screen from the retail demo, respectively.

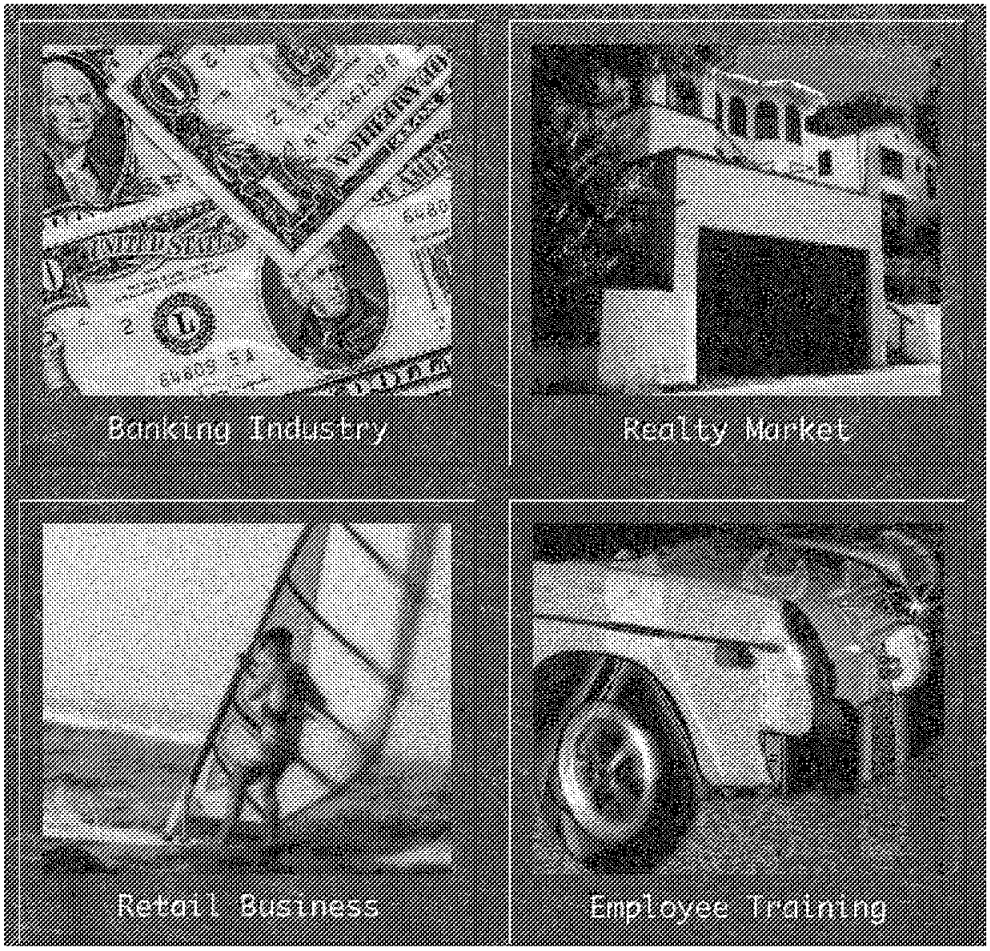

Figure 25. Main Screen of Ultimedia Services/6000 Demos. From this screen it is possible to access the four demos available. Each demo is represented by a button, which can be pressed to start the selected demo

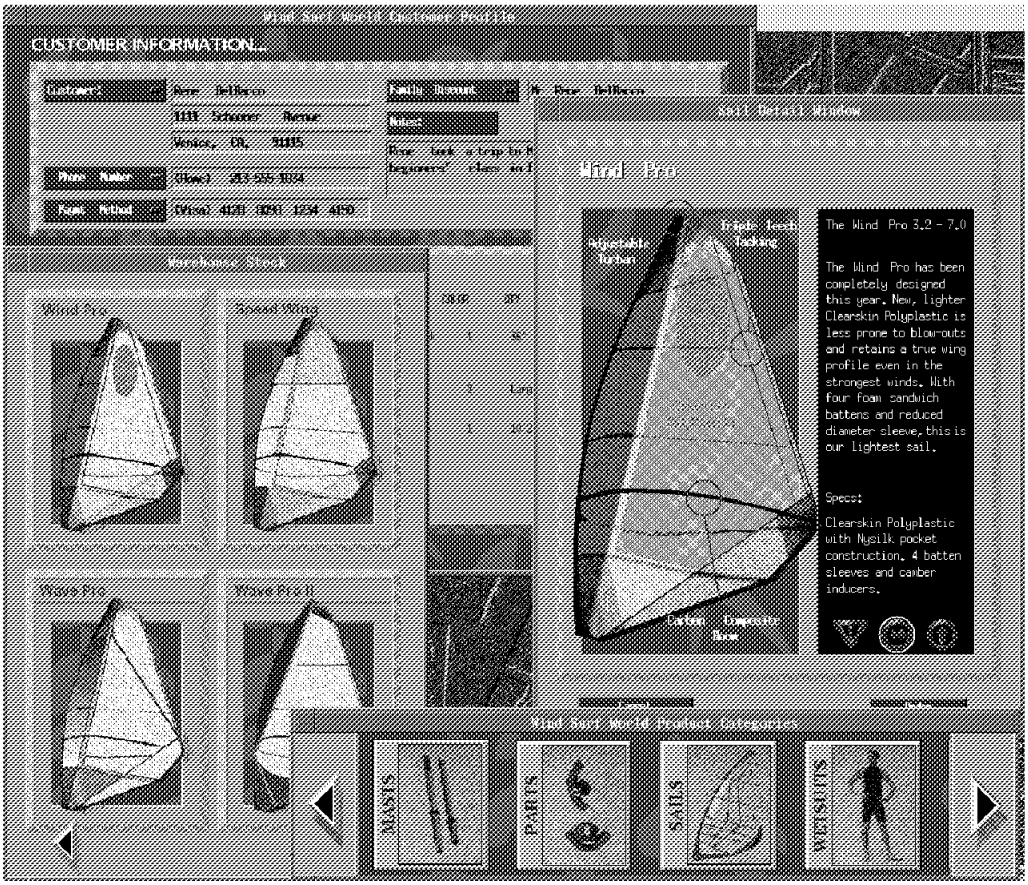

Figure 26. A typical Retail Shop Demo Screen. There are several panels describing: the customer information (top panel on the left), a selection panel used to help the customer to select the product (long panel on bottom), a list of sails available (this panel, which appears on the left, is activated by pressing the "sails" button on the previous panel), and a detailed panel describing the selected sail (panel on the right).

3.3 Programming Considerations

It is necessary to understand both the internal objects provided for processing multimedia information and the external mechanisms for obtaining and representing the information itself in order to use all of the services available with AIX Ultimedia Services/6000.

Access to information means providing methods for storing audio and visual data in a form that can be processed by the system; for example the M-ACPA card on the RISC System/6000. Having the hardware devices to allow connection of the external data feeds is not sufficient however. It also necessary to have software to drive the hardware adapters and interpret the information. Ultimedia Services/6000 objects make the task of handling devices drivers and interpreting audio and video data easy.

The multimedia representation scheme, that is, how multimedia data is handled by computers, is explained in Appendix A, "Multimedia Technologies" on page 57. This shows, in detail, which technologies (both hardware and software) are involved in multimedia and how multimedia information is converted into computer data.

This section explains how the Ultimedia Services/6000 objects can be used to interact with multimedia data and create multimedia applications. Figure 27 on page 49 shows that Ultimedia Services/6000 objects are logically positioned as a high-level programming interface between the application and AIXwindows-OSF/Motif**. As the graphic depicts, the Ultimedia Services/6000 objects form an extension to the AIXwindows environment that is able to handle audio and video data.

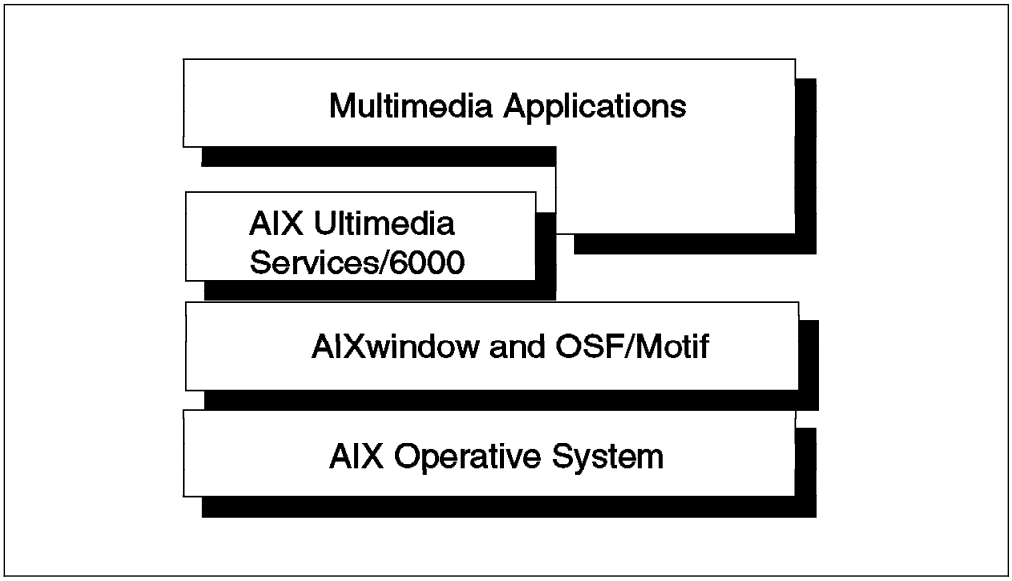

Figure 27. AIX Ultimedia Services/6000 layer. The Ultimedia Services/6000 objects library, fully integrated with AIXwindows and OSF/Motif, provides a set of objects to access and use multimedia files and devices.

There are various types of multimedia information that developers may want to access. Some of these (audio and video) as we saw in previous sections can be managed by Ultimedia Services/6000, but multimedia data also includes things such as text, graphics and still images, which Ultimedia Services/6000 objects are not currently able to manage. In order to incorporate this kind of information into a multimedia application, it is necessary to use an alternative mechanism such as the AIXwindows widget set, coupled with Ultimedia Services/6000 objects. With these two mechanisms, an application is able to use all the major multimedia information streams now available on computers.

3.3.1 Media Supported by AIX Ultimedia Services/6000

AIX Ultimedia Services/6000 does not provide objects to manage all major multimedia media. Robust support is provided for motion video and audio, but, no support is provided for text, graphics or still images, However, programmers can use the support provided by AIXwindows and OSF/Motif for these media as indicated in the following table.

Table 7 (Page 1 of 2). Media Supported by AIX Ultimedia Services/6000

	Ultimedia Services	**AIXwindows and OSF/Motif**
Motion Video	yes	no
Audio	yes	no

Table 7 (Page 2 of 2). Media Supported by AIX Ultimedia Services/6000

	Ultimedia Services	AIXwindows and OSF/Motif
Text	no	yes
Still Image	no	partial
Animation	partial	no
Graphics	no	yes

Notes

AIXwindow and OSF/Motif provide support only for pixmaps, but many free UNIX tools are available to support or to convert other still image file formats.

The only way to provide animation is to record it as motion video, for example using the M-JPEG format.

3.3.2 Limitations and Performance Considerations

Performance will be affected by the type/performance of individual storage devices, as well as the customer-specified parameters for file and data storage management and archival/retrieval functions. Performance will also be affected by factors such as processor speed, memory configuration, available fixed disk storage and the application environment. For more information on performance considerations see: *AIX Ultimedia Services/6000 Version 1.1.0*.

3.4 AIX Ultimedia Services/6000 at a Glance

Table 8 (Page 1 of 2). Ultimedia Services Requirements

Hardware Required	200, 300 or 500 Series POWERstations and POWERservers configured with a minimum of: • One supported display, display adapter keyboard, and mouse • M-Audio Capture and Playback Adapter (for audio).
Memory	• Minimum: 16MB • Recommended: 32 MB
Fixed Disk Storage	• Minimum 10MB for Ultimedia Services/6000 Code • Minimum 50MB for Ultimedia Services/6000 demo and sample clips • Additional fixed disk required for user-created audio and video files
Operating System	AIX Version 3.2.4 or later
Other Software	• AIXwindow/6000 Version 1.2.4 or later • OSF/Motif 1.2

Table 8 (Page 2 of 2). Ultimedia Services Requirements	
Hardware Supported	Ultimedia Services/6000 is designed to support: • IBM 8 and 24-bit graphics adapters • CD-ROM • M-ACPA card

Chapter 4. Future Directions

The future user's requirements can be divided into two main categories:

- Developer's needs
- End-User's needs

For the first category, the problem is to have a multimedia programming interface which is flexible, expandable and distributed (see 1.4, “What Makes a Good Multimedia Environment?” on page 10). Todays standard interface technology is not extendable. The interface developer must design all of the flexibility that users might eventually need in advance. This could possibly be realized for traditional applications, but in multimedia applications the user's requirements cannot be fully projected. Creating a new multimedia programming interface means to provide some functionality, from which developers can easily obtain new functionality. Moreover, these new interfaces must be accessible by a variety of different programming languages. Finally, these interfaces must provide transparent distribution of data, resources and CPU between networked computers.

The end-user's requirements are multimedia and distribution. They need to manage multimedia data like they handle traditional data. Single-user multimedia applications are not the wave of the future. These applications manage the new types of data, but the information can not be distributed or shared throughout the corporation. Customers need distributed multimedia business applications which are able to share new types of data (audio, video, animation and so forth), able to work on different platforms, and which allow videoconferencing and fully interactive real-time collaboration between networked end-users.

AIX Ultimedia Services/6000 responds to these requirements by offering multimedia programming interfaces along with true objects which can be subclassed and are accessible from a variety of languages. It provides a set of multimedia tools which can be used to create new multimedia information or to reorganize existing information. Ultimedia Services/6000 satisfies a large part of, but not all, customer and developer requirements for a good multimedia programming environment. The addition of the support provided by AIXwindows and OSF/Motif fulfills most of the current requirements.

4.1 The Future for Ultimedia Services/6000 Objects Library

The AIX Ultimedia Services/6000 Object Library will improve over time by increasing the multimedia capabilities and increasing distribution and heterogeneous features.

Ultimedia Services/6000 has significant advantages from being developed using SOM technology. Its SOM features allows Ultimedia Services/6000 to become a heterogeneous and distributed *bus*, able to link a variety of software developed on different operating systems. Ultimedia Services/6000 will benefit from SOM improvements. Every improvement to the SOM software (for example, support for a new operating system or for the binding of a new language) will extend Ultimedia Services/6000 capabilities.

SOM is fully CORBA compliant and it is quickly becoming a cross-platform for CORBA implementations. SOM provides support for NetBIOS, IPX** and TCP/IP, and it also has added C++ language bindings, cross-process and cross-system support. SOM currently provides support only for OS/2 and AIX, but it will provide support for Windows, Apple System 7**, HP/UX**, MVS/ESA*, and OS/400*, and plans are being discussed for Novell** and SunSoft**. IBM wants to port SOM into OpenDoc. This would offer an alternative to OLE 2.0 that is open, standards-based, multi-platform, and technically superior. Ultimedia Services/6000 will be able to work on several different operating systems, to link code written in a growing variety of programming languages and to transparently distribute objects between networked computers. This means improved flexibility, distribution and heterogeneous features.

Ultimedia Services/6000 is organized into five object classes (see 3.2.1, “Ultimedia Object's Library” on page 34), which support the following multimedia operations:

- Software video compression and decompression
- Audio and Video playback
- Audio recording
- Translation between Audio file formats
- Adding virtual device drivers or new file types

There is no support for video recording, and this is the main deficiency of Ultimedia Services/6000 Version 1.1.

The next release of Ultimedia Services/6000 will address this deficiency. It will provide support for video capture and for video file format translation by adding new classes and new features to the current classes. New *state-of-the-art* features (scaling, quality improvements, and others) will also be added to current CODEC algorithms to bring them more in line with PC CODEC capability.

4.2 The Future for Ultimedia Services/6000 Multimedia Tools

The next release of the Ultimedia Services/6000 tools will include full integration into the new AIX Common Desktop Environment (CDE) for AIX/6000.

The intent is to fully integrate multimedia data into the new AIX desktop. Both multimedia and traditional data will be managed in the same way. All standard operations available on icons representing traditional data (text, executable programs and so forth) will also be supported for new multimedia data (audio and video). For example, a double-click on a particular icon on a current traditional desktop indicates that the user wants to perform the default action on that icon. On an icon representing a text file, a double click indicates the user wants to edit that file. The same action on an icon representing a executable program indicates that the user wants to execute the program. Integrating multimedia data into CDE means that these types of behavior will also be possible for multimedia icons. A double-click on an audio icon or on a movie icon will run the audio or the movie player respectively. From the end-user point of view, all data icons will have similar predictable behavior.

This is a big step toward the homogenization of data handling. Customers want to use their data (both traditional and multimedia) in a simple and predictable way. A multimedia desktop is a step toward uniformity of computer information.

Other enhancements will provide a video capture feature which will be included in the video/audio tool. It will then be possible to record motion video directly from the RISC System/6000.

There are also plans to extend the Ultimedia Services/6000 tools to provide a framework for videoconferencing and for workgroup collaboration (for example, a clipboard shared in real time by users, or the ability to work on the same document).

4.3 Vendors Announcing Support for Ultimedia Services/6000

Several vendors in the UNIX area have announced intentions or plans to develop their applications with AIX Ultimedia Services/6000. The following table summarizes the action taken by some vendors:

Table 9. Vendors Announcing Support for Ultimedia Services/6000

Aim Tech	Signed a development agreement to provide a RISC System/6000 version of ICONAUTHOR, a multimedia authoring package.
Applix Inc.	Has announced their intent to use AIX Ultimedia Services/6000 to support audio and video in its ApplixWare desktop application.
Frame Technology	Inserted calls to Ultimedia Services/6000 within its Frame environment.
Insoft Inc.	Has signed a development agreement with IBM to provide a RISC System/6000 version of Communique, a teleconferencing package that supports collaboration in the UNIX environment.
Interleaf Inc.	Used AIX Ultimedia Servics/6000 in it's application.
Sybase Inc.	Gain Momentum has a multimedia demonstration developed with AIX Ultimedia Services/6000.

Appendix A. Multimedia Technologies

This appendix shows how different multimedia information types are handled by a computer and how media is converted into computer data. This is not a trivial problem, because all of this information is represented by different technologies with disparate qualities, standards, computer devices, and unique methods for user interaction. One of the main goals of multimedia is the management of these data types in an efficient way to optimize the value of multi-sensory communication.

A.1 Audio

The human ear recognizes audio signals as continuous variation of air pressure on the ear's membrane. This variation can be represented in a two dimensional space, where the x-axis represents time and the y-axis represents the instantaneous pressure value. The curve in this space is called a *wave* or *audio wave* or *waveform*, and the instantaneous value of this curve is called the *Amplitude of the wave*. For this reason the word *audio* is used in relation to any acoustic, mechanical, or electrical frequencies corresponding to normal audible sound waves, which are of frequencies ranging from 15 to 20,000 Hz (Webster's Dictionary).

Audio can be in one of the following forms:

- Analog Waveform
- Digital Waveform
- MIDI

The hierarchical diagram of this representation is given in Figure 28.

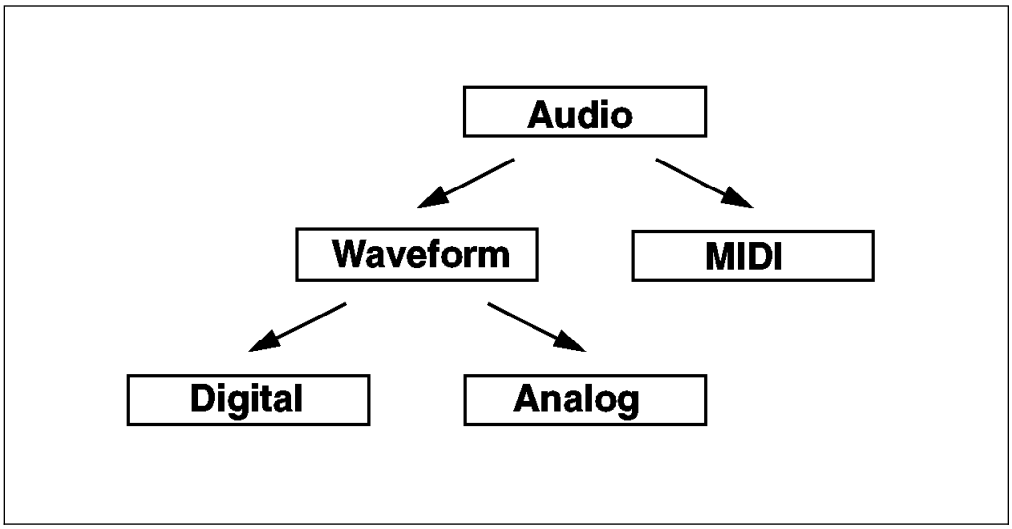

Figure 28. Audio Types

A.1.1 Analog Waveform Audio

Sound can be seen as a continuous variation of a *quantity* with time and its analog waveform is the continuous magnitude of that signal. This means that the magnitude of the signal itself represents the loudness, and the magnitude variation the frequency. This is the way analog devices such as a cassette recorder record sound: as an electric signal, where electric voltage replaces the sound loudness and voltage variation is the frequency.

Analog audio is subject to noise, error, distortion and the limited life of the storage devices. However, it is widely used and it is available from many devices such as compact disc (CD) players, digital audio tape (DAT) players, cassette players, or open reel players. The signal generated by these devices can be routed directly to an audio amplifier and speakers, without any conversion or processing.

Computers are not able to manage analog signals (they manage only digital data): therefore, these signal must be converted into computer-form data.

Computers can store and generate sound in two different formats:

- Waveform Audio File
- MIDI Audio File

A.1.2 Digital Waveform Audio

A digital representation of a sound can be obtained from an analog waveform through a process called *Analog-To-Digital Conversion (ADC)*. This process is organized into two main sub-processes:

- Sampling, where the analog signal's amplitude is periodically measured
- Quantization, where the analog signal's amplitude measurements are converted into digital data (see Figure 29 on page 59)

This process is also called *Pulse Code Modulation (PCM)*.

At the end of these sub-processes, the analog waveform is transformed into numerical data which represents the value of the waveform's amplitude at specific instants. In this way, the digital data values represent the volume of the sound wave (for example, a large numerical value indicates a loud sound, while a big change between two adjacent numerical values indicates a high frequency).

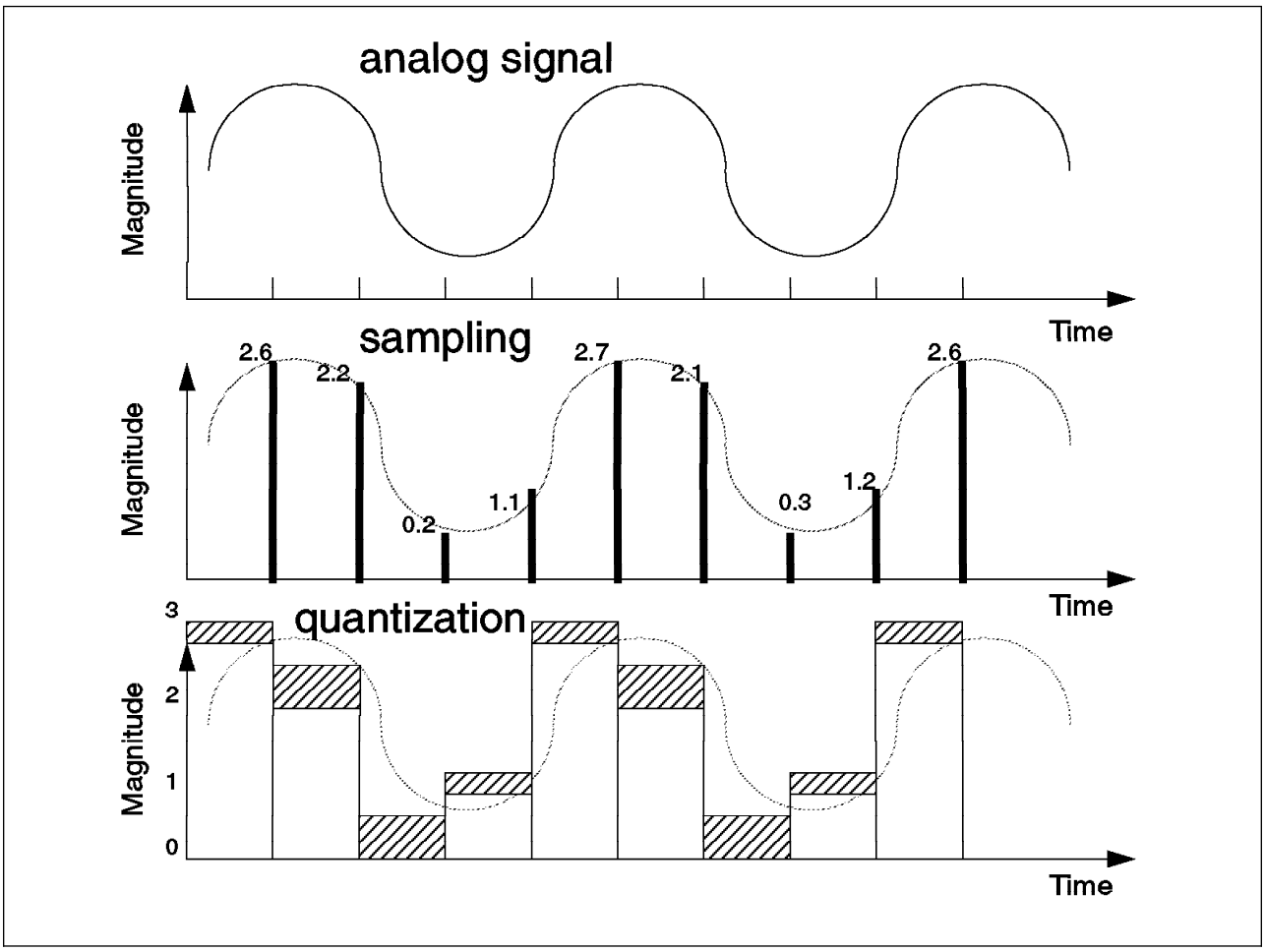

Figure 29. Analog-To-Digital Conversion. The picture shows the three phases of analog-to-digital conversion. The top picture shows the original analog signal. The middle picture shows the sampling operation: periodically the value of the analog signal is recorded. The lower picture shows the quantization phase: each value is rounded to the nearest integer. The shadow areas are so-called "quantization errors."

The accuracy of the conversion depends on two main factors:

Sampling Rate, which indicates how often the analog audio signal is sampled. It determines the upper frequency response.

Sample Size, which indicates how many bits are used in the quantization process.

The minimum value for the sampling rate is given by the Nyquist theorem which says that if an analog signal has an upper frequency equal to W, then the minimum sampling rate must be equal to 2W. Sampling rates smaller than this value produce noise called *Aliasing Distortion*.

More bits per sample means a more accurate signal representation, less information loss, or lower quantization error. High frequency content of an analog signal requires a high sampling rate.

Digital data has the advantage of being more easily reproduced and duplicated without any loss or signal distortions. Digital storage devices have longer life times than analog ones; however, digital audio requires heavy processing and large storage overheads.

A.1.2.1 Audio Compression Techniques and Formats

PCM has large storage requirements. For example, one minute of high-fidelity music requires about 10MB (16-bits per sample x 2 channels of stereo music x 44,100 samples per second).

To reduce the storage requirement, many compression techniques have been developed.

A compression technique is a digital process that allows data to be stored or transmitted using less than the normal number of bits, eliminating or reducing gaps, redundancies and unnecessary data.

Compression techniques are divided into two classes:

- Lossless - where information is compressed without loss of information.
- With loss - where some information is lost during the compression process. This technique is used when detail is less important than minimizing the volume of data requiring storage.

Compression techniques allow increased throughput and reduced storage requirements, but they require intensive use of CPU or dedicated hardware. Data compression requires processing both for the compression during input (when the signal is recorded) and then for the decompression during output (when the signal is restored for reproduction). These techniques imply intensive use of the computer's CPU or they require a special hardware adapter card.

The most common methods are: truncation, interpolation and prediction (for example, Adaptive Differential Pulse Code Modulation or ADPCM) and logarithmic compression of a signal's dynamic range (A-LAW and μ-LAW). Each of these techniques produces a different type of audio format.

The Resource Interchange File Format (RIFF)

The *Resource Interchange File Format* (RIFF) is a standard file format used for storing multimedia information. The format allows video, audio, text and other multimedia elements to be stored in a common format and accessed simultaneously. The RIFF format constitutes a base format or syntax upon which more specific file formats can be defined (for example the .WAV and .AVI formats, discussed later, are based on the RIFF formats). For this reason, RIFF is not really a file format, since it does not represent a specific kind of information, but is rather a standard container for multiple information formats.

A RIFF file consists essentially of chunks, each of which may consist of sub chunks. Navigation through the file is accomplished by moving from chunk to chunk, or by descending (or ascending) to (from) a sub-chunk.

Every chunk consists of a header containing a four byte identifier for the chunk, followed by the four bytes containing the size of the data portion of the chunk.

Due to the flexibility of this approach, applications need not recognize all chunks in a file; they can simply ignore the chunks they cannot understand. The RIFF format places few constraints on information contained with it, merely providing a flexible framework for the storage of large elements of information.

The most common audio formats are:

- .WAV
- .SND

The WAV format: WAV audio files are stored in the RIFF format. The RIFF chunk has its *type field* set to the value WAVE. This chunk may have up to three sub-chunks as follows:

- Fmt: contains information on the audio attributes
- List: contains information regarding the audio data in the file
- Data: contains the data stream

Figure 30 shows the WAV format structure.

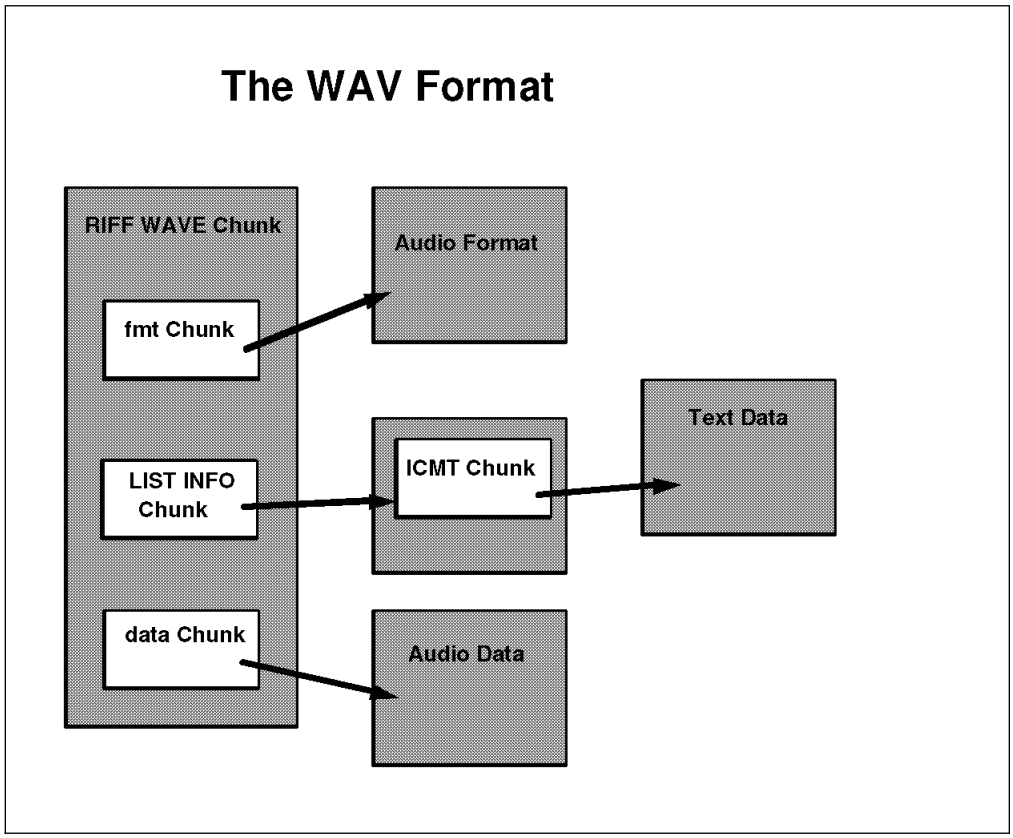

Figure 30. WAV Format Structure

The SND format: The SND format, commonly referred to as the NeXt/SUN SND format, consists of a simple header structure followed by the audio data. The header contains all information necessary to play the data contained in the audio section of the file.

Figure 31 shows the SND format structure.

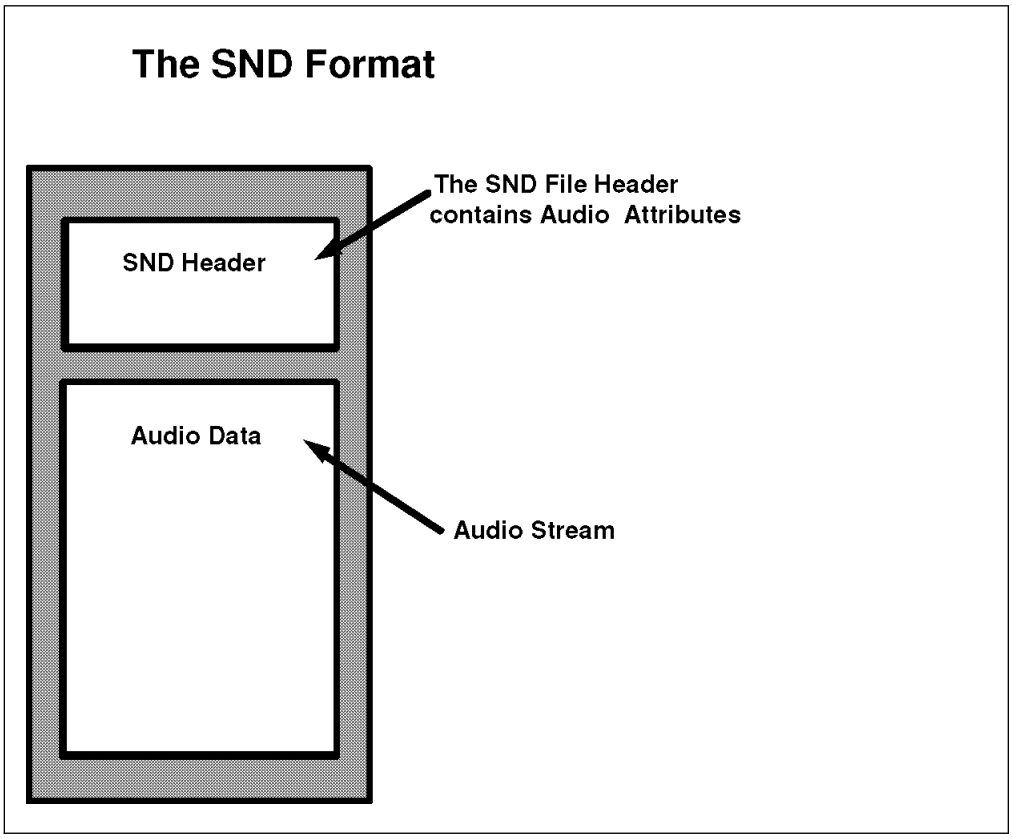

Figure 31. SND Format Structure

Other audio formats: Other audio formats are:

- .AU file format. This format is quite similar to the .SND format.
- .RAW file format. This format is raw audio data that may or not may contain audio attributes describing the format of the information.

A.1.3 MIDI

MIDI (Musical Interface Digital Interface) is the data protocol of electronic music and is a digital encoding of musical performance information (for example, notes, volume, and so forth).

MIDI does not directly relate to the original sound wave, as does the analog or digital sound representation. It is rather a description of musical events, for example which piano key is pressed and when, or for how many milliseconds the key remains pressed, and so forth. So MIDI defines a specific note; it does not represent the actual sound of that note. MIDI represents music using far less storage and allows greater editing flexibility, but does require a music synthesizer to generate music and is very limited in speech reproduction.

MIDI augments the use of waveform audio in multimedia. Waveform audio is normally used to reproduce speech and many non-musical sounds, but requires a lot of memory to faithfully reproduce music.

A.2 Still Image

Everything in the world emits electrical energy called radiation. The parts of this electrical spectrum with frequencies corresponding to the frequencies detectable by the human eye are called visible light. An image is the mapping of all *visible* light emitted by the object onto the eye's retina. If we scan the eye's retina, it appears as a two dimensional array of points. The color of a point can be represented by two different values:

Luminance, which is the magnitude of the light signal emitted by the picture element.

Chrominance, which is the difference between a color and a reference white of the same luminous intensity.

If frequency values for any image points do not change with time, the image is called a *still image*, otherwise *it is called a moving image* or *motion video*.

This array can be reproduced, for example, on a TV display using an analog signal. The analog signal contains information to move a beam of electrons across the TV screen. The electron beam causes phosphorus dots on the screen to glow in response to the video signal's amplitude. A computer screen responds to the content of digital display memory within the computer to control the electron beam.

The analog video signals related to an image must be converted into computer-form data, in a similar fashion to the audio signal previously described, in order to be used by computers. Computers can store and draw images in two different ways:

- Raster Image
- Vector Image

A.2.1 Raster Image

This is the simplest way to obtain a digital representation of an image. The image is described as a matrix describing the individual dots, that are the smallest elements of resolution on a computer screen, other display, or printer device. Each of these elements are referred to as a *pixel* or *pel*. Every pixel can be represented in computer memory using 1, 2, 4, 8 or more bits. The bits used for each pixel indicate the color quality of the image. For example, suppose we have only one bit per pixel available to store color information. There are not many ways to interpret these pixel values. The only information available is whether this pixel is turned on or off. The result is a monochrome picture. But let's assume we have 4 bits per pixel available. In this case, for each pixel 2^4 combinations are possible. So, we can have 16 different shades of grey or 16 different colors.

Table 10 (Page 1 of 2). Colors Available and Video Memory Required for Bits per Pixel

Bits Per Pixel	Number of Colors	MB (MByte) (*)
1	2	0.164

Table 10 (Page 2 of 2). Colors Available and Video Memory Required for Bits per Pixel		
Bits Per Pixel	**Number of Colors**	**MB (MByte) (*)**
4	16	0.655
8	256	1.311
24	16M	3.932
Note: (*) These values are calculated for a 1280x1024 pixel screen.		

Raster representation is used for photo-realistic images and in presentation graphics where artistic consideration and the need to deal with photographic information is important. However, it requires a large amount of storage space.

A.2.2 Vector Image

An image can also be drawn, that is, it can be generated by drawing instructions. For example, a line can be represented by its initial and final points. The CPU is then responsible for converting the description of the graphic into pixel data that can be displayed. This process of converting a geometric image to pixel data is known as *rasterization*, which gets its name from the fact that a raster display is used to display the information.

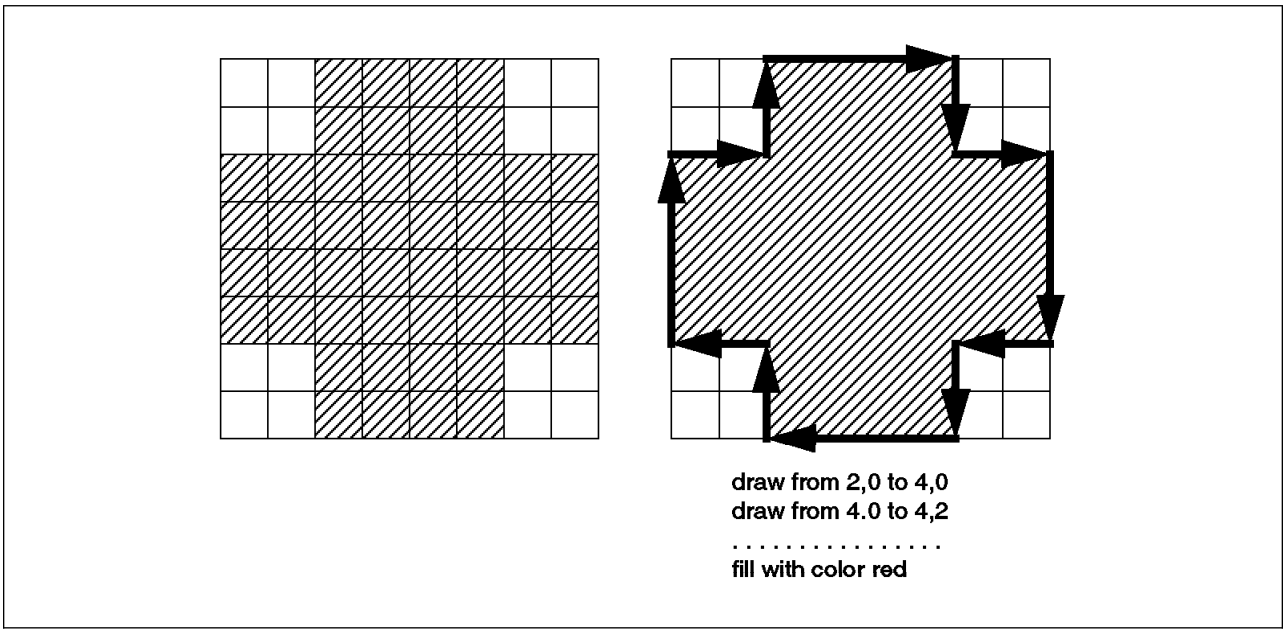

Figure 32. Raster and Vector Representations. The left picture shows a raster representation of a cross; while the right one shows the vector representation.

Vector images are almost always used in CAD/CAM applications, where precise scaling and element relationships are important. The main advantages of these images is that they do not require large amounts of storage and precise scaling and transformations are possible. However, they do require a lot of CPU processing when being rasterized for display.

Table 11. Main Characteristics of Vector and Raster Images	
Vector	**Raster**
• High-Precision output to both raster hardware (display and printer) and vector (plotter). • Precision scaling and transforming of objects. • Easy definition of what an element/object is for (cut and paste outline, rotating and so forth).	• Rapid display. • Input from raster and analog devices (scanner, video cameras). • “Paint” functions including complex shading.

A.2.2.1 Image Compression Techniques and Formats

The previous section shows how an image can be represented using digital data. This representation is not unique as there are many different ways which can be used to memorize pixel information. For example, every value associated with a pixel can be an index in a color list (called *a palette*) or they can be the three RGB color components. Generally, this and other information is contained in the *file header* of the image file. The *features* of this header differentiate images type files, which are called *image formats*

The file header is not the only characteristic which differentiates image files. Images require a lot of storage space to be recorded and a lot of computer and network bandwidth to be transmitted. This has resulted in many different image compression techniques. Each has the goal of reproducing good images with less storage requirements than the original number of bits. Each tries to reach this aim through reduction or elimination of image redundancies. Most digital raster images contain a high degree of redundancy, which means the same data is transmitted more than once. The most common redundancies in still raster images are:

- Redundancy between *single pixels*. For example, when a single color spans more than a single pixel location in any area of the image.
- Redundancy between *lines*. For example, when a scene contains predominantly vertically oriented object there is a possibility that two or more adjacent lines will contain identical data.

These types of redundancies are called *spatial redundancies*.

The image compression techniques range from simple methods of truncation, interpolation and prediction (similar to audio ones) to more advanced code transformation and statistical coding.

The most common still image formats are:

- .GIF
- .TIF
- JPEG

The GIF File Format: The GIF image format is used to store a single image. A GIF file is organized into a series of headers and colormap information, followed by the image data itself.

- **GIF file header** - Contains the GIF identifier, used for both version and file type checking, as well as image width and height information, and whether a colormap is included.
- **Colormap information** - Immediately follows the header and contains a series of RGB values which constitute the colormap information (the *palette*).
- **Image Information** - Immediately after the colormap, it contains futher information describing the image.

Figure 33 shows the GIF format structure.

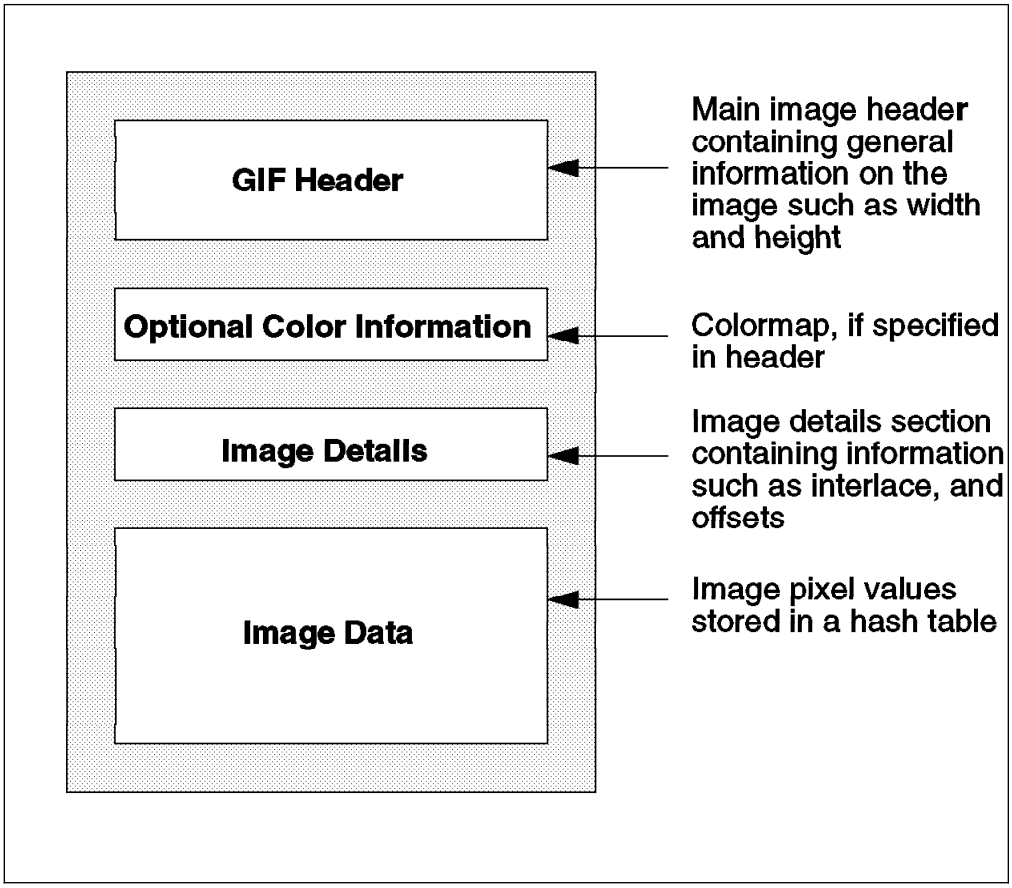

Figure 33. GIF Format Structure

The Tagged Image File Format (TIFF): There is not just one TIFF format! There is a whole family of different TIFF file formats, each one slightly different from the other. The main difference between these formats are the different *TIFF tags* used.

The TIFF format, used to store still RGB images, has a representation mechanism widely different from the GIF one. The image can be stored either as RGB values, as indexes into a colormap, or a number of other pixel representation schemes. Moreover, all the information describing the image is stored as tagged values. These tags specify a value and a type describing the information that follows.

Figure 34 shows the TIFF format structure.

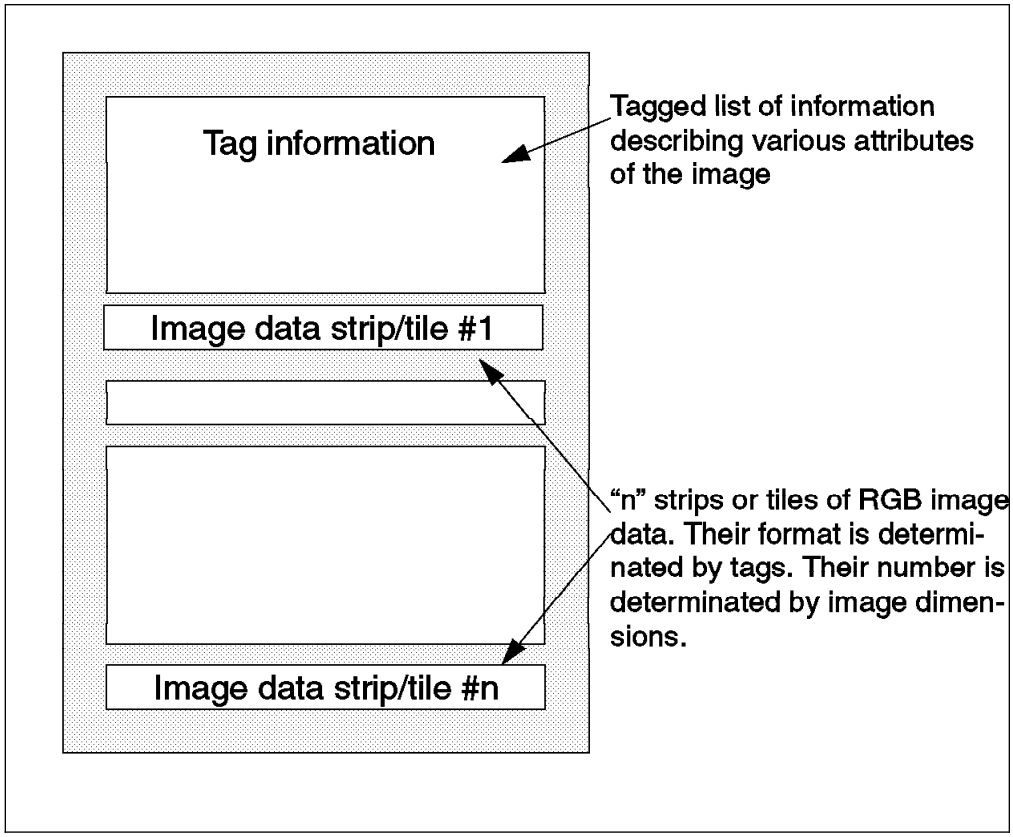

Figure 34. TIFF Format Structure

The Joint Photograph Experts Group (JPEG) File Format: The Joint Photograph Experts Group file format is a standard file format and at the same time a compression format. It allows image data to be stored without significant image degradation. The JPEG format allows image information to be stored in a scalable way. This means that when the image is shown it is possible to define the quality of the image itself.

JPEG images are compressed using the following three steps:

- **Encoding**. The source image data is fed one 8 by 8 pixel block at a time into an encoder which transforms them using a *Discrete Cosine Transformation (DCT)*. This step does not alter the data, but converts it to a form more easily processed by the successive stages.
- **Quantization**. Each 8 by 8 block is rounded to the nearest integer by the quantization process.
- **Entropy coding**. Each DCT coefficient is encoded using the Huffman Statistical Method.

Figure 35 shows the JPEG format structure.

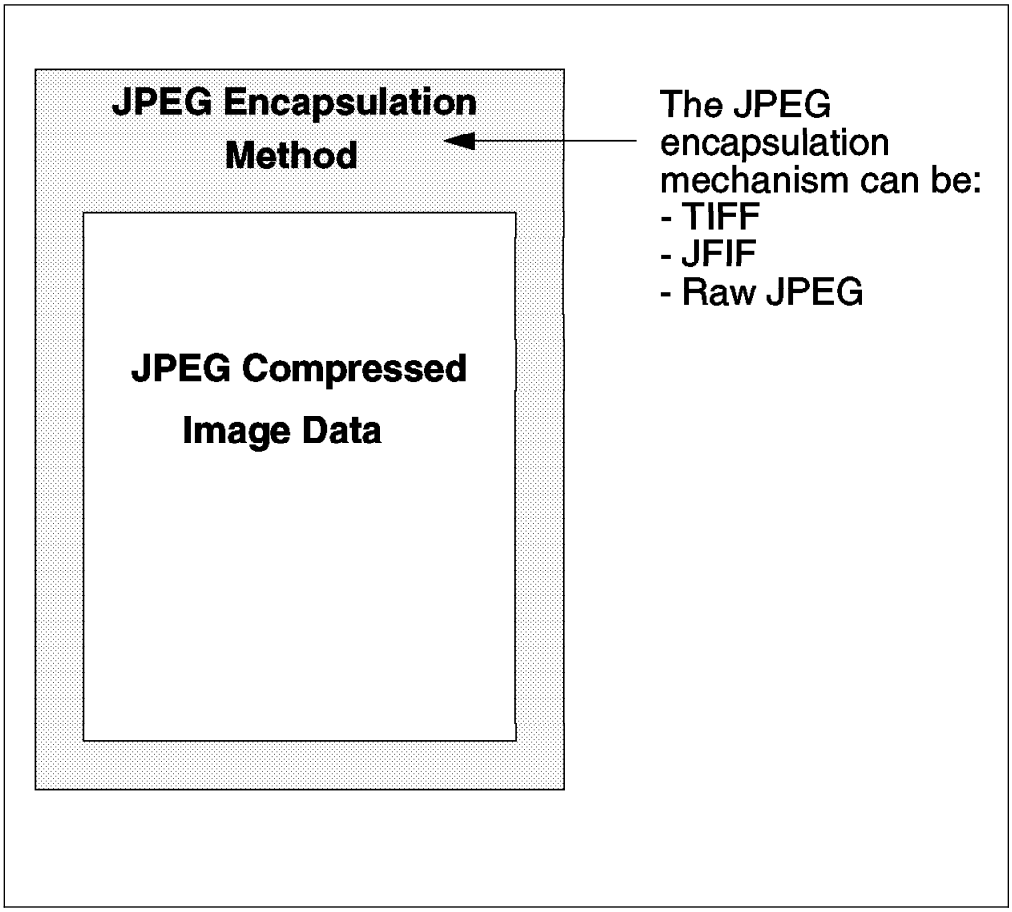

Figure 35. JPEG Format Structure

A.3 Motion Video

Motion Video is a collection of still images, slightly different from each other, displayed in rapid sequence to provide the illusion of continuous motion on the screen. To feel the picture is moving the images have to be presented at at least 15 frames per second for video without audio and 24 frames per second for movies with sound. Motion video can be in two forms: Analog and Digital.

A.3.1 Analog

A variety of video cameras can be used for video acquisition or *capture*. Acquisition (or capture) is the ability to transform a light signal into an electrical signal, which can be stored in magnetic or optical format.

Any analog device captures an image by scanning it from top to bottom and left to right, and splitting the light at each scanned point into it's three color components (red, green and blue). After this, each color component is converted into an analog electric signal.

After the image is seen by the camera sensors, it must be coded into electrical signals. There are several ways to code an analog video signal. The most common are:

- **Composite:** The lowest quality of acceptable video coding is called composite video. The color image is produced by taking only one light signal and multiplexing it to produce the three color signals: red, green and blue. With this system, all picture information is carried by a single signal, with a separate audio signal. The resolution provided by this type of coding is normally in the region of 200 lines per frame.
- **Y/C or S-Video:** The next highest quality signal coding is Y/C or S-Video. In this case, the information regarding the black and white and color levels is still present in separate components. The black and white component is called Y (Luminance or Brightness) and the color component is called C (Chroma or Color Intensity). It provides a resolution in the region of 400 lines per frame.
- **RGB:** The highest quality analog signal is called RGB. In this coding, the image information is split into three separate color channels (red, green and blue). Each of these channels has a detector, which converts the light to signals. These three signals must be synchronized to obtain the image. There are two synchronization techniques:
  - RGB w/Sync: the synchronization signal is separated from the color signal.
  - RGB: the synchronization signal is carried by the green signal.

  Because the original RGB signals are not mixed, the picture quality is higher than Composite and Y/C coding. This kind of coding provides a quality signal better than 400 lines per frame.

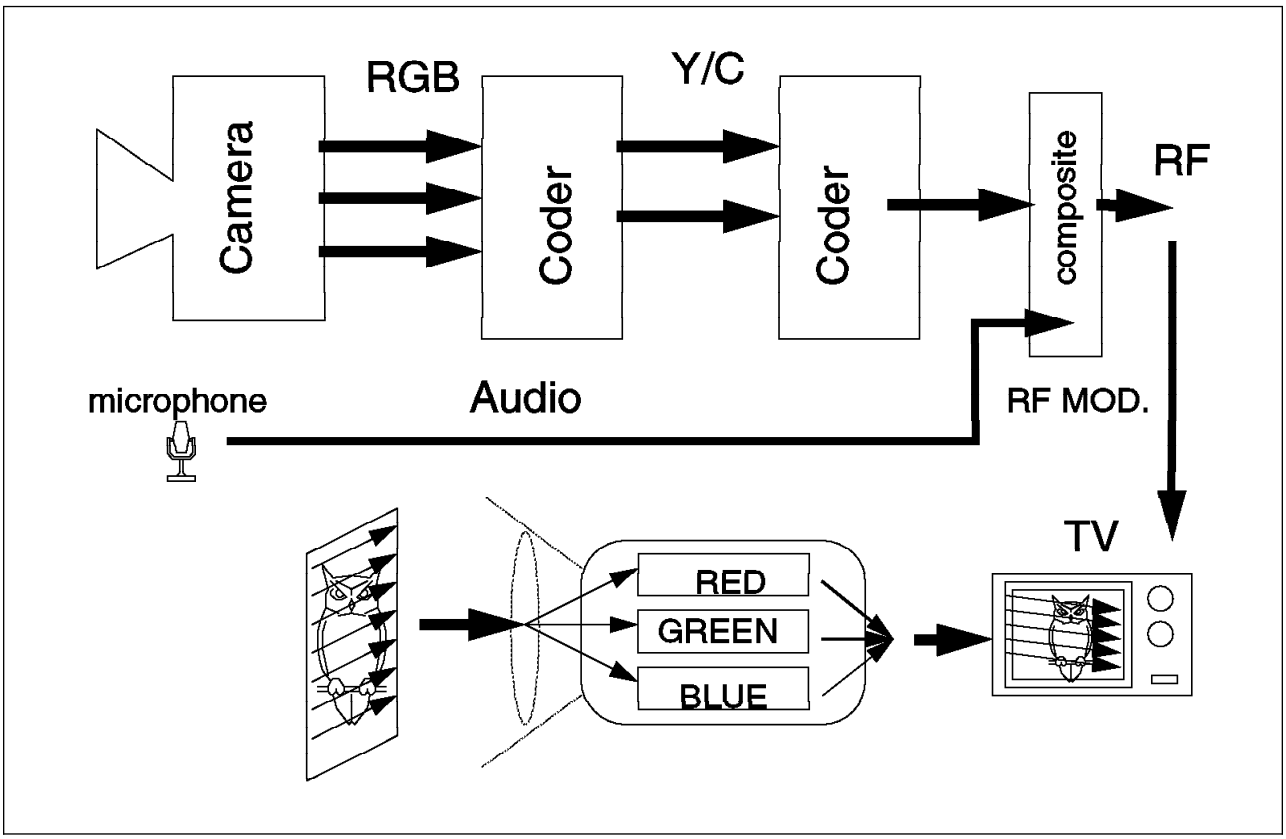

Figure 36. Analog Motion Video Capture Diagram

After capture, the video must be stored or transmitted. There are several standard formats for storing video information. The most common (BETA, VHS, S-VHS, Hi8, VIDEO8) use some form of cassette, which is not compatible between systems. The differences between these formats are not in the signal, but in the way they are separated or in the cassette's size or speed. The most common way to transmit video is *Television Broadcasting*.

TV Distribution Standards in use worldwide

The different TV distribution standards in use worldwide are:

- PAL (Phase Alternation Line format) is a European standard and it uses a TV scan rate of 25 frames (50 half-frames) per second and a line count per frame of 625. It is used mainly in Europe, Australia and South America.
- SECAM (Sèquentiel Couleurs Ávec Mèmoire) is a French standard and is very similar to PAL, but with different internal video and audio frequencies. It is used mainly in France and Eastern Europe.
- NTSC (National Television Standard Committee) is the USA standard and it uses a TV scan rate of 30 frames (60 half-frames) per second and a line count per frame of 575. It is used mainly in USA, Japan and Canada.

Because the images are displayed from top to bottom and from left to right, when the monitor displays the upper part of a new picture, the down part still contains the previous picture. To avoid this effect (called flickering) in the video transmission a display method called *interlacing* is used. Each picture or frame is divided into two complete sets of lines (odd and even). The monitor displays first the odd lines and

then the even ones (see Figure 37 on page 72). Obviously the transmission data rate is doubled. A NTSC video scans the screen 60 times per second constituting 1/2 of a complete video frame, while a PAL video scans 50 times per second.

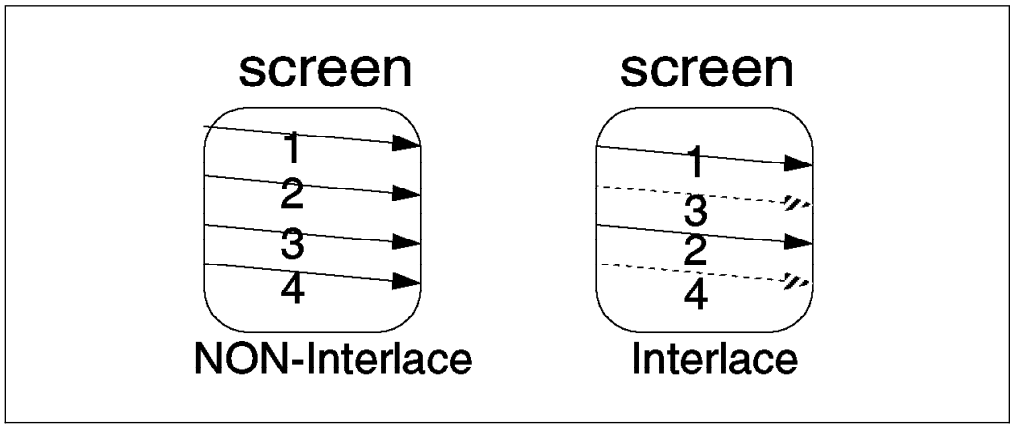

Figure 37. Example of Non-Interlaced and Interlaced Video. In a non-interlaced display (left picture), the video is refreshed by drawing all lines sequentially. In an interlaced display (right picture), the odd lines are displayed first (solid arrows) and then the even lines (dashed lines).

A.3.2 Digital

A digital representation of a motion video can be produced by digitizing successive frames from a video camera, by animation software that generates movement from still images, or by allowing the user to generate an animation sequence from scratch. At the end of these processes, a video is transformed into digital data.

A great deal of storage is required for storing motion video. For example, to store a picture of 640x480 pixel and 24 bits colors, 922KB is required. That means that one second of motion video (at 30 frames per second) requires about 27.6MB and one minute requires about 1.66GB! These values are huge not only from a storage point of view, but also from computer throughput.

A.3.2.1 Motion Video Compression Techniques and Formats

Motion video is a collection of still images displayed rapidly; therefore, motion video file compression has all of the same requirements as image compression. In addition, there are new problems due to the time-based nature of a movie (for example, fast display, short response time, and so forth).

The need for fast access to data requires higher file compression to reduce the storage requirements and to increase the access speed, response time and display time. Video compression is a crucial element in a fully capable multimedia system.

Video files also have redundancies. In addition to the spatial redundancies found in still images, video files have another type of redundancy:

- Redundancy between *frames*. For example, when a scene is stationary or slightly moving, adjacent frames in time are similar or even identical.

These types of redundancies are called *temporal redundancies*.

Compression techniques for motion video act in two ways: compression on spatial redundancy (called *intraframe compression*) and on temporal ones (called *interframe compression*). Interframe compression notes the difference between consecutive frames in the sequence and discards redundant information.

Motion video compression also covers another kind of compression: An audio track is usually associated with the movie. In this case the audio track must be compressed and it must be synchronized with the movie sequence.

A video compressor (sometimes called a CODEC) generally introduces some distortion into the reconstructed output.This occurs because every video CODEC tries to obtain the best tradeoff between good visual quality of the decompressed output (more computation) and a frame rate that is high enough to make the motion appear smooth (less computation).

IBM deals primarily with the following Motion Video Formats:

- AVS
- AVI
- MJPEG
- MPEG

The AVS Format: The AVS file format is mainly used in association with the ActionMedia II card.

The AVS multimedia file format allows multiple information streams to be contained within a single file structure. Figure 38 shows the AVS format structure.

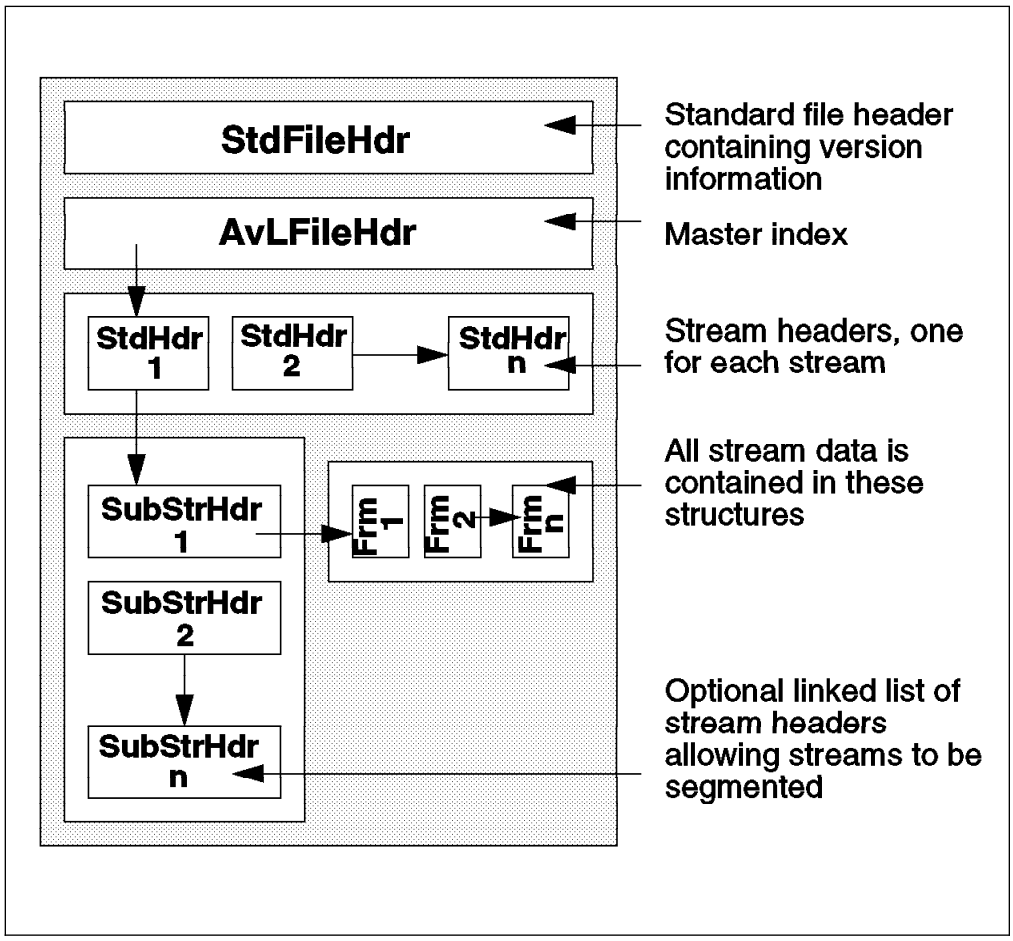

Figure 38. AVS Format Structure

The AVI Format: The AVI format is the one currently used by the multimedia movie player.

The AVI multimedia file format is based on the Resource Interchange File Format (RIFF). Figure 39 shows the AVI format structure.

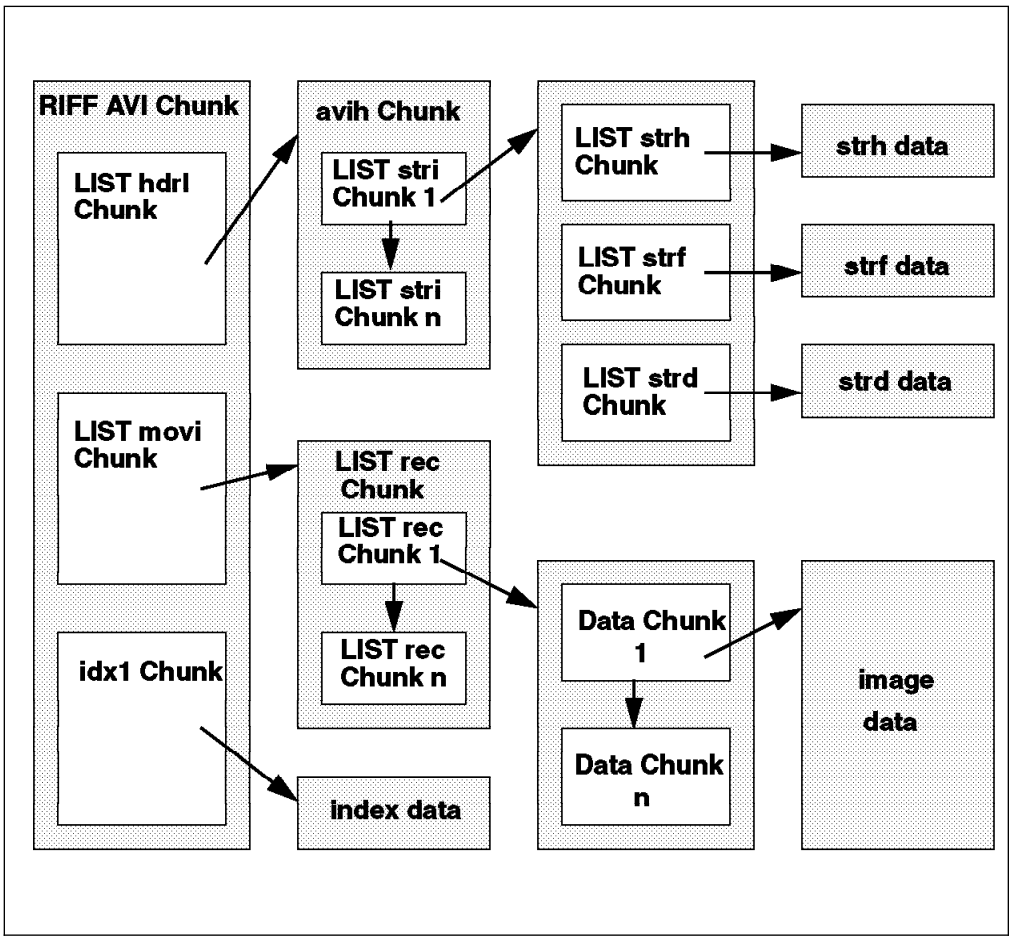

Figure 39. AVI Format Structure

The MJPEG and MPEG formats: Motion JPEG (MJPEG) utilizes the JPEG compression techniques to encode individual frames which are then stored in an AVI file format. A Motion JPEG file has a lower effective bit rate than other mechanisms.

MPEG is a standardization effort by the Motion Picture Expert Groups to establish an international digital video compression standard for moving pictures. MPEG can achieve three times more compression than MJPEG.

Appendix B. System Object Model (SOM) - A Brief Overview

This chapter describes the most important features of the IBM System Object Model (SOM).

B.1 Basic Concepts of the System Object Model (SOM)

The System Object Model (SOM), provided by the SOMobjects Developer Toolkit, is a set of libraries, utilities, and conventions used to create binary class libraries that can be used by application programs written in various object- oriented programming languages, such as C++ and Smalltalk, or in traditional procedural languages, such as C and COBOL. The following paragraphs introduce some of the basic terminology used when creating classes in SOM:

- In SOM, an object is a run-time entity with a specific set of methods and instance variables. The methods are used by a client programmer to make the object exhibit behavior (that is, to do something), and the instance variables are used by the object to store its state.

  An object's implementation is determined by the procedures that execute its methods, and by the type and layout of its instance variables. The procedures and instance variables that implement an object are usually encapsulated (hidden from the caller), so a program can use the object's methods without knowing anything about how those methods are implemented. Instead, a user is given access to the object's methods through its interface (a description of the methods in terms of the data elements required as input and the type of value each method returns).

  An interface through which an object may be manipulated is represented by an object type. That is, by declaring a type for an object variable, a programmer specifies the interface that is intended to be used to access that object. SOM IDL (the SOM Interface Definition Language) is used to define object interfaces. The interface names used in these IDL definitions are also the type names used by programmers when typing SOM object variables.

- In SOM, as in most approaches to object-oriented programming, a class defines the implementation of objects. That is, the implementation of any SOM object (as well as its interface) is defined by some specific SOM class. A class definition begins with an IDL specification of the interface to its objects, and the name of this interface is used as the class name as well. Each object of a given class may also be called an instance of the class, or an instantiation of the class.

- SOM classes can take advantage of multiple inheritance, which means that a new class is jointly derived from two or more parent classes. In this case, the derived class inherits methods from all of its parents (and all of its ancestors), giving it greatly expanded capabilities. In the event that different parents have methods of the same name that execute differently, SOM provides ways for avoiding conflicts.

  In the SOM run time, classes are themselves objects. That is, classes have their own methods and interfaces, and are themselves defined by other classes. For this reason, a class is often called a class object. Likewise, the terms class methods and class variables are used to distinguish between the methods/ variables of a class object vs. those of its instances. (Note that the

type of an object is not the same as the type of its class, which as a *class object* has its own type.)

A class that defines the implementation of class objects is called a *metaclass*. Just as an instance of a class is an object, so an instance of a metaclass is a class object. Moreover, just as an ordinary class defines methods that its objects respond to, so a metaclass defines methods that a class object responds to. For example, such methods might involve operations that execute when a class (that is, a class object) is creating an instance of itself (an object). Just as classes are derived from parent classes, so metaclasses can be derived from parent metaclasses, in order to define new functionality for class objects.

- The SOM system contains three primitive classes that are the basis for all subsequent classes:

  – SOMObject: the root ancestor class for all SOM classes

  – SOMClass: the root ancestor class for all SOM metaclasses

  – SOMClassMgr: the class of the SOMClassMgrObject, an object created automatically during SOM initialization, to maintain a registry of existing classes and to assist in dynamic class loading/unloading

  SOMClass is defined as a subclass (or child) of SOMObject and inherits all generic object methods; this is why instances of a metaclass are class objects (rather than simply classes) in the SOM run time.

SOM classes are designed to be language neutral. That is, SOM classes can be implemented in one programming language and used in programs of other languages. To achieve language neutrality, the interface for a class of objects must be defined separately from its implementation. That is, defining interface and implementation requires two completely separate steps (plus an intervening compile), as follows:

- An interface is the information that a program must know in order to use an object of a particular class. This interface is described in an interface definition (which is also the class definition), using a formal language whose syntax is independent of the programming language used to implement the class's methods. For SOM classes, this is the SOM Interface Definition Language (SOM IDL). The interface is defined in a file known as the IDL source file (or, using its extension, this is often called the .idl file).
- An interface definition is specified within the interface declaration (or interface statement) of the .idl file, which includes:

  1. The interface name (or class name) and the name of the class's parent(s)
  2. The names of the class's attributes and the signatures of its new methods

  Each method signature includes the method name, and the type and order of its arguments, as well as the type of its return value (if any). Attributes are instance variables for which *set* and *get* methods will automatically be defined, for use by the application program. (By contrast, instance variables that are not attributes are hidden from the user.)

- Once the IDL source file is complete, the SOM Compiler is used to analyze the .idl file and create the implementation template file, within which the class implementation will be defined. Before issuing the SOM Compiler command, `SC`, the class implementor can set an environment variable that determines which emitters (output-generating programs) the SOM Compiler will call and, consequently, which programming language and operating system the resulting binding files will relate to. In addition to the implementation template file itself, the binding files include two language-specific header files that will be included in the implementation template file and in application program files. The header files define many useful SOM macros, functions, and procedures that can be invoked from the files that include the header files.

  The implementation of a class is done by the class implementor in the implementation template file (often called just the implementation file or the template file). As produced by the SOM Compiler, the template file contains stub procedures for each method of the class. These are incomplete method procedures that the class implementor uses as a basis for implementing the class by writing the corresponding code in the programming language of choice.

In summary, the process of implementing a SOM class includes using the SOM IDL syntax to create an IDL source file that specifies the interface to a class of objects, that is, the methods and attributes that a program can use to manipulate an object of that class. The SOM Compiler is then run to produce an implementation template file and two binding (header) files that are specific to the designated programming language and operating system. Finally, the class implementor writes language-specific code in the template file to implement the method procedures.

At this point, the next step is to write the application (or client) program that uses the objects and methods of the newly implemented class. (Observe, here, that a programmer could write an application program using a class implemented entirely by someone else.) If not done previously, the SOM compiler is run to generate usage bindings for the new class, as appropriate for the language used by the client program (which may be different from the language in which the class was implemented). After the client program is finished, the programmer compiles and links it using a language-specific compiler, and then executes the program. (Notice again, the client program can invoke methods on objects of the SOM class without knowing how those methods are implemented.)

B.2 Organization of the SOMobject Package

The SOMobjects Developer Toolkit contains the following packages:

B.2.1 SOM Compiler

The SOMobjects Toolkit contains a tool, called the SOM Compiler, that helps implementors build classes in which interface and implementation are decoupled. The SOM Compiler reads the IDL definition of a class interface and generates:

- Implementation skeleton for the class
- Bindings for implementors
- Bindings for client programs

Bindings are language-specific macros and procedures that make implementing and using SOM classes more convenient. These bindings offer a convenient interface to SOM that is tailored to a particular programming language. For instance, C programmers can invoke methods in the same way they make ordinary procedure calls. The C++ bindings wrap SOM objects as C++ objects, so that C++ programmers can invoke methods on SOM objects in the same way they invoke methods on C++ objects. In addition, SOM objects receive full C++ typechecking, just as C++ objects do. Currently, the SOM Compiler can generate both C and C++ language bindings for a class. The C and C++ bindings will work with a variety of commercial products available from IBM and others. Vendors of other programming languages may also offer SOM bindings. Check with your language vendor about possible SOM support.

B.2.2 SOM Run-Time Library

In addition to the SOM Compiler, SOM includes a run-time library. This library provides, among other things, a set of classes, methods, and procedures used to create objects and invoke methods on them. The library allows any programming language to use SOM classes (classes developed using SOM) if that language can:

- Call external procedures
- Store a pointer to a procedure and subsequently invoke that procedure
- Map IDL types onto the programming language's native types

Thus, the user of a SOM class and the implementor of a SOM class need not use the same programming language, and neither is required to use an object-oriented language. The independence of client language and implementation language also extends to subclassing: a SOM class can be derived from other SOM classes, and the subclass may or may not be implemented in the same language as the parent class(es). Moreover, SOM's run-time environment allows applications to access information about classes dynamically (at run time).

B.2.3 Frameworks Provided in the SOMobjects Toolkit

In addition to SOM itself (the SOM Compiler and the SOM run-time library), the SOMobjects Developer Toolkit also provides a set of frameworks (class libraries) that can be used in developing object-oriented applications. These include Distributed SOM, the Interface Repository Framework, the Persistence Framework, the Replication Framework, and the Emitter Framework, described below.

B.2.3.1 Distributed SOM

Distributed SOM (or DSOM) allows application programs to access SOM objects across address spaces. That is, application programs can access objects in other processes, even on different machines. DSOM provides this transparent access to remote objects through its Object Request Broker (ORB): the location and implementation of the object are hidden from the client, and the client accesses the object as it if were local. The current release of DSOM supports distribution of objects among processes within a workstation, and across a local area network consisting of OS/2 systems, AIX systems or a mix of both. Future releases may support larger enterprise-wide networks.

B.2.3.2 Interface Repository Framework

The Interface Repository is a database, optionally created and maintained by the SOM Compiler, that holds all the information contained in the IDL description of a class of objects. The Interface Repository Framework consists of the 11 classes defined in the CORBA standard for accessing the Interface Repository. Thus, the Interface Repository Framework provides run-time access to all information contained in the IDL description of a class of objects. Type information is available as TypeCodes: a CORBA-defined way of encoding the complete description of any data type that can be constructed in IDL.

B.2.3.3 Persistence Framework

The Persistence Framework is a collection of SOM classes that provide methods for saving objects (either in a file or in a more specialized repository) and later restoring them. This means that the state of an object can be preserved beyond the termination of the process that creates it. This facility is useful for constructing object-oriented databases, spreadsheets, and so forth. The Persistence Framework includes the following features:

- Objects can be stored singly or in groups.
- Objects can be stored in default formats or in specially designed formats.
- Objects of arbitrary complexity can be saved and restored.

B.2.3.4 Replication Framework

The Replication Framework is a collection of SOM classes that allows a replica (copy) of an object to exist in multiple address spaces, while maintaining a single-copy image. In other words, an object can be replicated in several different processes, while logically it behaves as a single copy. Updates to any copy are propagated immediately to all other copies. The Replication Framework handles locking, synchronization, and update propagation, and guarantees mutual consistency among the replicas. The Replication Framework includes these important features:

- Good response times for both readers and writers
- Fault-tolerance against node failures and message loss
- Simple coding rules (that can be automated) for building replicated objects
- Graceful degradation under wide area networks
- Minimal overhead when replication is not activated

B.2.3.5 Emitter Framework

The Emitter Framework is a collection of SOM classes that allows programmers to write their own emitters. Emitter is a general term used to describe a back-end output component of the SOM Compiler. Each emitter takes as input information about an interface, generated by the SOM Compiler as it processes an IDL specification, and produces output organized in a different format. SOM provides a set of emitters that generate the binding files for C and C++ programming languages (header files and implementation templates). In addition, users may wish to write their own special-purpose emitters. For example, an implementor could write an emitter to produce documentation files or binding files for programming languages other than C and C++. The Emitter Framework is separately documented in the SOMobjects Developer Toolkit: Emitter Framework Guide and Reference.

B.2.3.6 Collection Classes Framework

The Collection Classes Framework provides a collection of classes which contains the most common elements used in user interfaces, such as lists, sets, queues, dictionaries and so forth. The programmers can inherit from and use these classes in their applications with no need to recode or retest the functions.

B.2.3.7 Workstation and Workgroup Runtimes

The Workstation Runtimes enables the execution of SOM-based applications in a single machine environment; while the Workgroup Runtimes enables the execution of SOM-based applications across a multiple-node workgroup LAN distributed environment. Both these runtimes work on AIX and OS/2 platforms.

Appendix C. Programming Examples

This appendix shows some programming examples for storing multimedia information.

C.1 Example 1: File Type Detector Program

This example shows a demo program which is able to detect file types.

```
/*************************************************************************
 *
 * COMPONENT_NAME: detector_examples
 *
 * FUNCTIONS: main
 *
 * ORIGINS: 27
 *
 *  (C) COPYRIGHT International Business Machines Corp. 1993
 *  All Rights Reserved
 * Licensed Materials - Property of IBM
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 *************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "UMSFiletypeDetector.h"
#include "SOMDetector.h"

/*************************************************************************/
/* This program prints out the type of the file listed in command line */
/* Usage:       detector_example <file1> ffl<file2> <file3> ....“        */
/*************************************************************************/


main(int argc, char **argv)
{
  UMSFiletypeDetector   *ds;
  int                    i;
  Environment            ev;

  SOM_InitEnvironment(&ev);     /* Initialization of SOM environment */

  ds=UMSFiletypeDetectorNew();  /* Creation of new file detector object */

  /* Go through each argument: parsing of input line */
  for (i=0; i<argc; i++)
  {
     char *tmp=NULL;

     printf("<%s>: \n", arg[i]);  /* print of filename */
     tmp=UMSFiletypeDetector_determine_file_type(ds, &ev, argv[i]);

     if (tmp)
       printf("<%s>: \n", tmp);   /* print file type returned by detector */
     else
       printf("unknown file type\n);
  }
```

```
  _somFree(ds);               /* Destruction of detector object */
  SOM_UninitEnvironment(&ev); /* Deinitialitazion of SOM envirnment */

  exit(0);
}
```

This example shows how easy it is to work with the AIX Ultimedia Services/6000 Object Library. It is interesting to observe that it was *not* necessary to use object oriented programming concepts, instead using the traditional procedural approach. The AIX Ultimedia Services/6000 library encourages the use of object oriented methodologies, but does not require them.

C.2 Example: Movie Play Program

This example shows a simple program for playing a JPEG file.

```
/*****************************************************************************
 *
 * COMPONENT_NAME: examples
 *
 * FUNCTIONS: main
 *
 * ORIGINS: 27
 *
 *  (C) COPYRIGHT International Business Machines Corp. 1993
 *  All Rights Reserved
 * Licensed Materials - Property of IBM
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 *****************************************************************************/

#include <stdio.h>
#include <sys/stat.h>
#include "UMSJPEGDecoder.h"
#include "UMSAudioPlayer.h"

main(int argc, char *argv[])
{
  char  *infile;
  char          *denfile;              /* input  filename      */
  char          *dendev;               /* input  filename      */
  char          *outfile;              /* output filename      */
  FILE          *fip;                  /* file pointer to input  file     */
  FILE          *fop;                  /* file pointer to output file     */
  unsigned char *imagedata;            /* pointer to compressed image     */
  unsigned char *decbuffer;            /* pointer to decompressed image   */
  struct stat   statb;                 /* input file status    */
  long          width;                 /* frame width          */
  long          height;                /* frame height         */
  long          size;                  /* frame size           */
  long          format;                /* image format         */
  long          Nf;                    /* component number     */
  char          ch;
  int           i,rc,volume;
  UMSAudioPlayer_EventData *denevent;

  Environment   *ev;                   /* SOM environment variable */
  UMSAudioPlayer  audioobj;            /* pointer to audioplayer object */
```

```
/*****************************************************************************
* Parse the command line
*****************************************************************************/

  if (argc < 2)
  {
     fprintf(stderr, "Usage: <inputfile> <m,l,h>volume \n");
     exit(0);
  }

  if (!strcmp(argv[2],"m"))
    volume = 75;
  else if (!strcmp(argv[2],"h"))
  {
    volume = 100;
  }
  else
    volume = 50;

  fprintf(stderr,"volume is %d\n",volume);
  infile = argv[1];

/*****************************************************************************
* Create the audio object
*****************************************************************************/

  ev=somGetGlobalEnvironment();
  audioobj=UMSAudioPlayerNew();

  if (audioobj == NULL)
  {
    fprintf(stderr, "can't create audioplayer object\n");
    goto ErrorExit;
  }

/*****************************************************************************
* Open the input file
*****************************************************************************/

  rc = UMSAudioPlayer_load(audioobj,ev,infile,UMSAudioPlayer_LocalFile,NULL);
  if (rc != UMSAudioPlayer_Success)
  {
      fprintf(stderr, "error is %d\n", rc);
      fprintf(stderr, "cannot open %s\n", infile);
      goto ErrorExit;
  }
  fprintf(stderr, "opened the file OK\n");

/*****************************************************************************
*  Open device
*****************************************************************************/

  rc = UMSAudioPlayer_open(audioobj,ev,"Audio.audio_editting.audio_editor");
  if (rc != UMSAudioPlayer_Success)
  {
      fprintf(stderr, "cannot open device /dev/acpa0 rc=%d\n",rc);
      goto ErrorExit;
  }

  if (UMSAudioPlayer_Success != UMSAudioPlayer_set_volume(audioobj,ev,volume,0))
      fprintf(stderr, "cannot set volume \n");

  rc = UMSAudioPlayer_play(audioobj,ev);
```

```
  if (rc != UMSAudioPlayer_Success)
  {
      fprintf(stderr, "cannot play %s\n", infile);
      goto ErrorExit;
  }

  while (1)        /* Main Loop */
  {
    if (denevent->type == 1)
    {
      UMSAudioPlayer_close(audioobj,ev);
      exit(0);
    }
  }

/*************************************************************************
* Error Routine                                                          *
*************************************************************************/

ErrorExit:
  if (audioobj != NULL)
      _somFree(audioobj);
      return(0);

}
```

This example shows a simple program for playing movie files. Note the size of the code: 84 lines! The AIX Ultimedia Services/6000 object oriented approach provides an easy and quick way to integrate multimedia into applications or to create new ones. Developers do not need to create any AIX windows or the equivalent because the Ultimedia Audio-Player object organizes and presents its data automatically. Programmers can concentrate only on the *core* of their applications, that is how the application uses objects, and not on the objects themselves.

Glossary

ACPA. The Audio Capture and Playback Adapter is a card that allows the capture and playback of audio. Incoming audio, via jacks, is converted in digital information and stored on disk. Digital audio, stored on disk, is converted by the card to analog audio, and output to a line out connection or headphone socket.

Adapter. A printed circuit card that modifies the system unit to allow it to operate in a particular way. See also *card*.

ADPCM. In audio, adaptive differential pulse code modulation (ADPCM) is a compression technique that computes the difference between samples and quantizies the difference in a way that changes according to the input signal.

AIX. IBM's UNIX implementation is called Advanced Interactive eXecutive which abbreviates to AIX.

A-law. A method of audio encoding that maps 13-bit input samples to 8-bit numbers. The mapping is not linear and increases the signal quality at low level at the expense of distortion in the high-level signal. See also *μ-law.*

Aliasing. The phenomenon of generating false (alias) frequencies, along with the correct ones. This can occur as a result of sampling a signal at discrete points. Contrast with *Antialiasing*.

Analog Video. Video in which all information representing images is in a continuous-scale electrical signal for both amplitude and time.

Animation. A series of images, each one slightly different from the previous, displayed in succession so fast that they appear to be moving.

Antialiasing. The process of eliminating the aliasing phenomenon. Contrast with *Aliasing*.

API. An Application Programming Interface is: (1) the Interface through which an application program interacts with an access method. (2) A functional interface supplied by the operating system or by a separate licensed program that allow an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

Asynchronous. (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. Contrast with *synchronous*.

Audio. Records information that can be heard when played back.

Audio device alias. Used by audio player/recorder and by movie player to access the device driver.

Audio filter object. In Ultimedia Services/6000, it is an object used to convert various audio formats to other audio formats.

AVI. The audio-video interleaved (AVI) file format is a RIFF file specification that permits audio and video data to be interleaved in a file.

AVS. The audio-video sub-system (AVS) is a file format used by Intel's** Digital Video Interactive** (DVI) system and IBM's ActionMedia* and ActionMedia II* adapters.

Binding. Language-specific macros and procedures that make implementing and using SOM classes more convenient.

Capture. An operation where analog data such as video or audio signals are converted from analog to digital representation and stored in a computer.

Card. (1) An electronic circuit board that is plugged into a slot in a system unit. See also *Adapter.*

CD-ROM. The Compact Disk-Read Only Memory (CD-ROM) is an adaptation of the compact disk digital audio device (CD-DA or CD) for use with general digital data.

Chrominance. The difference between a color component and a reference white of the same luminous intensity.

Chunk. A basic building block of a RIFF file.

Class. A group of objects that share a common definition and that therefore share common properties, operation and behavior.

Codec. Abbreviation for (en)coder/decoder. A compressor-decompressor pair.

CORBA. The common object request broker architecture (CORBA) is a standard that provides the mechanism by which object make requests and receive responses. It provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

Compression. A process of transforming a block of data into a smaller block of data from which the former can be reconstructed. Contrast with *decompression*.

Compressor. An algorithm or software or hardware entity that does compression. Contrast with *decompressor*.

Decode. (1) To convert data by reversing the effect of some previous encoding. (2) To interpret code. See also *encode*.

Decoder. In multimedia, a synonym for *decompressor*. Contrast with *encoder*.

Decompression. A process of expanding a compressed data block into its original uncompressed form. Contrast with *compression*.

Decompressor. An algorithm or software or hardware entity that does decompression. Contrast with *compressor*.

Device driver. A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisk player, or an added card.

Digital. A method of signal representation with a set of discrete numeric values.

Digital Video. Video in which all the information representing images is in computer form.

Digitize. To convert analog signals to digital format.

Encapsulation. A method for representing abstract data, where data values are hidden, but operations are permitted on these values by functions called "methods."

Encode. to convert data by the use of a code in such manner that reconversion to the original data is possible. See also *compression*. Contrast with *decode*.

Encoder. In multimedia, a synonym for *compressor*. Contrast with *decoder*.

File type. The kind of files from different operating systems or file formats that are supported.

Frame. An image that corresponds to a scene at a particular instant.

GUI. The graphical user interface (GUI) is a visual metaphor that uses icons representing desktop objects that a user can manipulate with a positioning device.

Hypertext. This is the ability to access information by interaction with explicit links to cross-reference related objects in a natural manner.

Hypermedia. A method of presenting information in discrete units, or nodes, that are connected by links. The information may be presented using a variety of media such as text, graphics, audio, video, animation, image or executable documentation.

Hz. A Hertz (Hz) is a unit of frequency equal to one cycle per second.

Icon. A pictorial representation of a desktop object such as a text file or an executable program. It can be manipulated using a pointing device.

Information. In information processing, knowledge concerning such things as facts, concepts, objects, events, ideas, and processes that within a certain context has a particular meaning.

Inheritance. The passing of class resources or attributes from a parent to a child class.

JPEG. (1) The joint photograph experts group is a group that are working to establish standards for compression of digitalized continuous-tone still images. (2) The standard under development of this group.

Lossness. In compression, refers to a codec that introduces no distortion into a signal.

Lossy. In compression, refers to a codec that introduces distortion into a signal.

Luminance. The amount of light, measured in lumens, that is emitted by a source.

Media. Material or technical means of artistic communication using form such as film, art, voice, computer programming, and so forth.

Media handler object. Object useful to software that need to play (or record) audio to (from) an audio device, or play a movie into an X window.

Method. A combination of procedure and name, such that many different procedures can be associated with the same name (polymorphism).

MHEG. (1) The Multimedia and Hypermedia Information Coding Experts Group is a group that is working to establish standards for multimedia and hypermedia information coding. (2) The standard under development by this group.

MIDI. The Musical Instrument Data Interface is a serial standard data bus for interfacing digital musical instruments.

Mu-law. See *μ-law*.

Multimedia. The integrated treatment of more than two kind of different data. The term is also used as an adjective; for example multimedia network, multimedia application. In this case it means the possibility of handling different kinds of data.

Multitasking. A mode of operation that provides for concurrent performance, or interleaved execution of two or more tasks.

MJPEG. (1) The Motion Joint Photograph Experts Group is a group that are working to establish standards for JPEG compression of digitalized continuous-tone sequences of still images. (2) The standard under development by this group. (3) A compression algorithm used to record sequences of still images played as a movie. See also *JPEG*

NetBIOS. Networking Basic Input/Output System. An operating system interface for application programs used on IBM personal computers that are attached to the IBM Token-Ring* network.

Noise. Random undesirable picture or sound interference.

Object. A software abstraction consisting of data and routines that operate on the data.

OMG. The object management group (OMG) is an industry consortium founded to advance the use of object technology in distributed, heterogeneous environments. It includes IBM, SUN, Hewlett-Packard, and more than 200 other software vendors.

OOP. Object-oriented programming (OOP) is a method of structuring programs as hierarchically organized classes describing the data and the operations of objects that may interact with other objects.

Open. This defines the ability to interoperate across a wide range of different systems, maintaining consistency across networks.

PCM. In audio, refers to simple quantization of samples in an audio signal with no further processing. See also *ADPCM* and *quantization*.

Pixel. An abbreviation for picture element.

Pixmap. An abbreviation for pixel map, which is a 3-dimensional array of bits or a 2-dimensional array of numbers representing an image.

Quantization. The subdivision of the range of values of a variable into a finite number of non-overlapping, but not necessarily equal, subranges or intervals, each of which is represented by assigned values within the subrange.

Raster. A predetermined pattern of lines that provides uniform coverage of a display space.

Real-time. The processing of information that returns a result so rapidly that the interaction appears to be instantaneous.

Resolution. A measure of the sharpness of an image, expressed as the number of lines and columns on the display screen or the number of pixels per unit of area.

RGB. (1) The primary colors of light (red, green and blue), which are mixed to produce a video image. (2) In video, RGB refers to a system in which the three primary colors are kept isolated and delivered from the source to the display device over separate wires.

RIFF. The resource interchange file format (RIFF) is a standard file format used for storing multimedia files.

RISC System/6000. IBM's UNIX workstation implementing a Reduced Instruction Set Cycle computer system (RISC) is called RISC System/6000. The UNIX version on this computer is called AIX.

RTV. In DVI technology, this is a lower-quality video compression process. It allows real-time video compression.

S-Video. Super video is a video signal with separated chrominance (C) and luminance (Y) components.

Sample rate. The rate at which the samples occur.

Sampling. The conversion of an analog signal, varying continuously in time, into a sequence of numbers (or samples) that represent the signal values at discrete moments in time.

Signal. Information transmitted as an electric impulse.

SOM. The system object model (SOM) is IBM's implementation of CORBA. An object-oriented programming technology for building, packaging, and manipulating binary class libraries. See also *CORBA*.

Sound. The pressure variations in the air that are detected by our ears to create the sense of hearing.

Still Image. This is the digital representation of an image.

Synchronous. (1) Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (2) Occurring with a regular or predictable time relationship. Contrast with *asynchronous*.

Text. Text is a sequence of elements (characters, symbols, words, sentences, phrases, tables, paragraphs and so forth) intended to convey a meaning, whose

understanding is essentially based upon the reader's knowledge of some natural or artificial language.

Track. in multimedia file format, a stream of one type of information; for example, video frames or mono or stereo audio.

Ultimedia. IBM's family of multimedia products and services which deliver the ultimate in multimedia solution and support. Ultimedia goes beyond text and graphics to include high-quality images, full motion video, animation, high-fidelity music, and touch-based interaction.

Video codec object. Conversion object that implements video compression and decompression.

X-window. (1) An X-window or window is a rectangular zone of a display screen in an X-windowing environment where a user may activate a computer process and perhaps produce graphics while simultaneously working with other similar processes. (2) X-windows is a device-independent graphics standard for controlling windows (X-windows) within a distributed environment. It provides programmers with a software toolkit to implement window systems and display 2-dimensional graphics.

μ**-law**. A method of audio encoding that maps 14-bit input samples to 8-bit numbers. The mapping is not linear and increases the signal quality at low level at the expense of distortion in the high-level signal. See also *a-law.*

List of Abbreviations

ADPCM	Adaptive Differential Pulse Code Modulation
AIX	Advanced Interactive eXecutive
API	Application Programming Interface
AVI	Audio-Video Interleaved
AVS	Audio-Video Sub-system
CD	Compact Disc
CD-ROM	Compact Disc-Read Only Memory
CODEC	Coder-Decoder or Compression-Decompression
CORBA	Common Object Request Broker Architecture
DAT	Digital Audio Tape
DVI	Digital Video Interactive
DLL	Dynamic Link Libraries
DSOM	Distributed System Object Model
GUI	Graphical User Interface
IBM	International Business Machines Corporation
IDL	Interface Descriptor Language
IMA	International Multimedia Association
ISO	International Standards Organization
ITSO	International Technical Support Organization
JPEG	Joint Photograph Experts Group
MCI	Multimedia Control Interface
MHEG	Multimedia and Hypermedia information coding Expert Group
MMIO	MultiMedia Input Output
MMPM/2	MultiMedia Presentation Manager/2
MPEG	Motion Photograph Experts Group
NTSC	National Television System Committee
OMG	Object Management Group
OOP	Object Oriented Programming
OS/2	IBM Operating System/2
PAL	Phase Alternation Line format
PCM	Pulse Code Modulation
PLV	Production Level Video
RIFF	Resource Interchange File Format
RISC	Reduced Instruction Set Cycle Computer
RS/6000	RISC System/6000
RTV	Real Time Video
SECAM	Sèquentiel Couleurs Àvec Mèmoire
SOM	System Object Model
TIFF	Tagged Image File Format
YUV	Intensity (Y from Yellow) with color vectors (U and V)

Index

A

A-LAW 36, 60, 87
abbreviations 91
ACPA 30, 48, 87
acquisition 70
acronyms 91
adapter 87
ADC 58
ADPCM 36, 60, 87
Aim Tech 55
AIX 54, 87
aliasing 87
animation 2, 87
antialiasing 87
API 24, 29, 87
apple system 7 54
Applix 55
ApplixWare 55
as media 9, 54, 87
asynchronous 87
audio 54
  analog 58
  as media 2, 49
  definition 57
  digital 58
  introduction 57
  MIDI 8, 63
  types 57
audio device alias 87
audio filter objects 34, 40, 87
AVI 37, 38, 39, 73, 75, 87
AVS 8, 39, 73, 74, 87

B

binding 87

C

C 22, 23, 77, 81
C&plus+ 77
C++ 22, 24, 25, 31, 54, 80, 81
capture 87
card 87
CCITT 8
CD 30, 58
CD-DA 8
CD-I 8
CD-ROM 8, 87
CD-ROM/XA 8
CDE 54
chrominance 64, 87
chunk 87
class 16, 19, 24
  definition 18
  hierarchy 19
  libraries 22
  library 33
  methods 18
  objects 18
COBOL 77
codec 73, 87
communication 2, 6, 57
Communique 55
composite 70
compression 38
compression techniques 60
compressor 88
configuration objects 34, 40
CORBA 23, 25, 27, 32, 54, 81, 87
CUA 41

D

DAT 58
data abstraction 14, 26
decoder 88
decompression 38
decompressor 88
definition 87
device drivers 88
devices driver 48
distribution 11, 27, 33
DSOM 27, 30, 80, 82

E

encapsulation 18, 88
encoder 88
encoding 69
entropy coding 69

F

file access objects 34, 35, 39
file format
  AVI 73, 75
  AVS 8, 38, 73, 74, 87
  GIF 8, 66, 67
  JBIG 8
  JPEG 8, 66, 69, 88
  MIDI 8, 63, 88
  MJPEG 37, 38, 73, 75, 89
  MPEG 8, 73, 75

file format *(continued)*
mu-law 88
PCM 89
RIFF 37, 60, 75, 89
RTV 89
SND 8, 37, 61, 63
TIFF 8, 66, 68
ultimotion 8, 37
WAV 8, 37, 61, 62
filetype detector 39
flickering 71
Frame 55
Frame Technology 55
functional programming
design process 16
limits 13—15

G

Gain Momentum 12, 55
GIF 8, 66, 67
glossary 87
graphics 2, 49
GUI 29, 88

H

heterogeneous 27, 31
heterogeneous system 11, 12
hot areas 10
HP/UX 54
hypermedia 10, 88
hypertext 88

I

icon 88
ICONAUTHOR 55
IDL 23, 25, 77
Indeo 8
information hiding 14
inheritance 88
definition 19
introduction 17
linear 19
multiple 19
Insoft 55
interactive links 10
interactivity 1
interconnectivity 33
interlacing 71
Interleaf 55
interoperability 14, 33
IPX 54
ISO 8

J

JBIG 8
JPEG 8, 66, 69, 88

L

luminance 64, 88

M

media
audio 9
communication 6
definition 5, 6, 88
interactive links 10
introduction 5
motion video 9
sight 5
sound 6
still image 9
text 9
touch 6
media control interface 30, 35
media conversion objects 35
media handler objects 34, 35, 36, 88
message 18
methods 17, 18, 24, 88
MHEG 8, 88
MIDI 8, 63, 88
MJPEG 37, 38, 41, 73, 75, 89
MMPM 35
motion video
analog 70, 87
as media 2, 6
definition 64
description 70—72
digital 72, 88
file format 8, 72, 75
MPEG 8, 73, 75
MPower 12
mu-law 88
multi media input output 30, 35
multimedia
communication 1, 6
concepts 5—12
distribution 10—12
goals 2, 9, 10
history 1
interactivity 1
introduction 2, 89
languages 8
market 3
media 2, 9
platforms 12
problems 10—12, 15, 26, 53
standards 7

multimedia *(continued)*
technology 2, 57
trend 13
multitasking 12, 89
MVS 54

N

NetBIOS 54, 89
noise 89
NTSC 71
Nyquist theorem 59

O

object 16, 18, 89
object oriented
advantages 10, 20
concepts 17—22, 89
design process 16
goals 15
introduction 16—17
languages 21
limits 22—23, 26
technology 15
terminology 17—22
objective-C 24
OLE 2.0 54
OMG 23, 25, 89
open 11, 26, 31, 89
open system 11, 12
ORB 80
OS/2 12, 54
OS/400 54

P

PAL 71
PCM 36, 58, 60, 89
pel 64
pixel 64, 65, 72, 89
pixmap 89
polymorphism 20
px64 8

Q

quantization 58, 59, 69, 89
QuickTime 8, 12

R

raster 64, 65, 89
rasterization 65
real-time 89
redundancy
spatial 66, 72
redundancy *(continued)*
temporal 72
resolution 89
reuse 11, 14, 20, 26, 32
RGB 66, 70, 89
RIFF 37, 39, 60, 89
RTV 38, 73, 74

S

S-video 70, 89
sample 89
sampling 58, 59
sampling ratio 59
SECAM 71
sight 5
smalltalk 22, 24, 77
smell 5
SND 8, 37, 61, 63
SOM
advantages 23, 26—27
Classes 24, 77
Collection Class Framework 82
Compiler 79
components 24—26, 79—82
Description 77—79
Distributed SOM 80
Emitter Framework 81
Frameworks 80—82
IDL 78—79
Interface Repository Framework 81
introduction 23—24
object oriented 24
Objects 77
Persistence Framework 81
Replication Framework 81
Run-time library 80
Workgroup Runtime 82
Workstation Runtime 82
sound 5
standards 7
still image 2, 6, 8, 9, 49, 50, 64, 70, 72, 89
Sybase 55
synchronization 8, 37
system 7 12

T

taste 5
TCP/IP 54
television 3, 9, 64, 71
text 2, 6, 9, 49, 54, 89
μ-law 36, 60, 90
TIFF 8, 66, 68
touch 5

track 90
TV 3, 9, 64, 71

U

Ultimedia Services/6000
  AIXwindow 49
  Audio filter Objects 40
  audio player/recorder 34, 36, 41, 44
  characteristics 30—33
  class library 54
  classes library 33, 34—41, 53
  components 33
  Configuration objects 40
  demo programs 33, 46—48
  distribution 33
  expendable 32
  file access objects 39
  flexible 31
  heterogeneous 31
  introduction 29—30
  limits 50
  media 49
  media handler objects 36
  open 31
  reusable 32
  summary 50
  tools 33, 41—45, 54, 55
  video codecs objects 38
  video player 34, 36, 37, 41, 42
ultimotion 8, 37, 38, 39, 73, 75
UMSAddConfig Object 40
UMSADPCMtoPCM16 Object 41
UMSALAWtoPCM16 Object 41
UMSAVIReadWrite Objects 39
UMSAVSReadWrite Objects 39
UMSByteOrder Object 41
UMSChainFilter Object 41
UMSConfig Object 40
UMSFiletypeDetector Objects 39
UMSFilter Object 40
UMSMULAWtoPCM16 Object 41
UMSPCM16toADPCM Object 41
UMSPCM16toALAW Object 41
UMSPCM16toMULAW Object 41
UMSPCM16toPCM8 Object 41
UMSPCM8toPCM16 Object 41
UMSRiffReadWrite Objects 39

V

vector 64, 65
video
  as media 49
  compression 34, 54, 73
  decompression 34, 54
video codecs objects 34, 38, 90
virtual function 24

W

WAV 8, 37, 61, 62
Windows 12, 54

X

XMedia 12

Y

Y/C 70

ITSO Technical Bulletin Evaluation
RED000

RISC System/6000 Multimedia Environment:
An AIX Ultimedia Services/6000 Overview

Publication No. GG24-4254-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:
   Do you provide billable services for 20% or more of your time? Yes____ No____
   Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____
   If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

Name _____ Address _____

Company or Organization _____

Phone No. _____

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department JN9B, Building 045
Internal Zip 2834
11400 BURNET ROAD
AUSTIN TX
USA 78758-3493

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.