THESIS / THÈSE

MASTER IN COMPUTER SCIENCE

Comparison of two Graphical User Interfaces : XView and Motif

Gillard, Serge

Award date:
1991

Awarding institution:
University of Namur

Link to publication

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique

Rue Grandgagnage, 21, B-5000 NAMUR (Belgium)

Comparison of two Graphical User Interfaces :
XView and Motif

Serge GILLARD

ABSTRACT

This thesis first of all presents the general principles of windowing systems used on networked graphical terminals. We detail afterwards two toolkits (XView and Motif), or interface creation languages. Then we present two versions of an application each of which uses one of these toolkits and we define a generic interface creation language from the two specific ones. A comparison is finally established between XView and Motif and is partially based on these applications.

RESUME

Ce mémoire présente tout d'abord les principes généraux des systèmes de fenêtrage utilisés sur des terminaux graphiques connectés en réseau. Nous détaillons ensuite deux boîtes à outils graphiques (XView et Motif) ou langages de création d'interfaces. Nous présentons alors deux versions d'une application utilisant chacune un de ces "toolkits" et nous définissons un langage générique de création d'interfaces sur base des deux langages spécifiques présentés. Une comparaison est finalement établie entre XView et Motif en se basant notamment sur ces applications.

Acknowledgements

We would like to thank all the people who helped us in the writing of this thesis.

First of all, we wish to thank Bernard Geubelle from BIM, who gave us the opportunity for performing that traineeship. We would like to thank him for his helpful guidance and for his precious time he devoted to our work.

It would be unfair to forget a few people from BIM who occasionnally helped us for specific difficulties : Danny Backx, Thierry Delhaye, Philippe Dromelet, Olivier Dubois, Bernard Heuse and Peter Strickx.

We would like especially to thank our promoter, Baudouin Le Charlier, who read our drafts and made many interesting comments, as well as his collaborator, Yves Deville.

Table of Contents

List of figures

To my parents,
for their generous and constant support
throughout all these years spent in college

Introduction

This thesis is the realization of a traineeship at the BIM (Belgian Institut of Management, in Everberg, near Brussels). The idea was to establish a synthesized presentation of many concepts and tools related to windowing systems in order to study and compare two emerging interface creation languages : XView and Motif.

XView and Motif are competing at different levels on the field of graphical user interfaces : the programming approach and the look and feel. Both toolkits have their supporters and their detractors and until now it does not appear that XView or Motif is completely supplanting the other. We shall especially consider the programming approach. We intend to show that they both have their own specific characteristics, advantages, and disadvantages.

Another goal of this work has a more theoretical aspect. The problem is to determine a way of expressing generic algorithms of applications in order to either produce XView or Motif applications. Nevertheless, the reader should not expect to be able to write XView or Motif applications at the end of this thesis.

The first chapter presents principles and concepts of windowing systems, among which are toolkits. The second and the third chapters detail two particular of these toolkits, XView and Motif. Both chapters are divided in two parts : the first one deals with the graphical object. Many figures illustrate the most interesting of them, in order to present at the same time respectively the Open Look and the Motif look & feel. The second part presents the functions that can be used to manipulate these objects. The fourth chapter presents an application, its algorithms for the XView and the Motif versions, and from them, defines a generic language used to describe generic algorithms that can be instanciated to either an XView or a Motif one. The fifth chapter presents a comparison of the two toolkits from a particular point of view : the programming approach.

Chapter 1
Principles of windowing systems

The aim of the first chapter is to define the general frame for the next ones to be placed in. However, this chapter also presents the necessary background for understanding further non-specific readings on the subject.

As windowing systems are only a small part of computer science, this chapter will first delimit the field of study concerned with this work and present the underlying requirements.

This field is quite new and still evolving, in comparison with other ones. Therefore, we shall respectively present basic concepts needed by the reader to understand the core of this work and the terminology mostly used in technical literature.

An historical part follows and leads to a presentation of some current products related to the basic concepts defined before.

1.1. Field of study and hardware requirements

Network computing is the most recent stage in the evolution of computer systems.

The first systems allowed batch processing in which only one user at a time could use the only available machine. This limited system became more and more unacceptable because of performance requirements. Time-sharing processing, a system to accomodate many users on a single machine, was an initial response to such problems, yet it was still to be overtaken.

The development of microprocessors and networking gave birth to individual computers (PCs and workstations) linked in a Local Area Network (LAN).

Such PC or workstation networks are called workgroups. They take advantage of a shared environment consisting mostly of printers and disks. But, as mentioned, such configurations utilize only one kind of computer : PCs or workstations.

The last step of this development is network computing . Unlike the workgroup model, the network model is based on heterogeneous computers of many brands. It distinguishes between display devices and computing devices.

A _display device_ allows the user to directly access a network, give commands to run an application, and see the results displayed. Generally, a display device can be a standard terminal, a network display station, a PC, or a workstation.

A _network display station_ is a workstation-style terminal only dedicated to locally running a server. This server displays the results of an application running on a remote host. Except for the standard terminal, these devices are bit-mapped display devices that allow graphic applications.
A bit-mapped display screen is made up of dots, or pixels (picture elements), each of which is represented in memory by one or more bits. The image on the screen may therefore be altered by changing the value of these bits.

The _computing device_ is the machine that runs the application. It can be a "super" or "mini" computer, a mainframe, a workstation or a PC, as long as it has the same communication protocol as the display devices.

The display and computing functions may be held simultaneously by the same machine.

Regardless of the fact that all machines can be of different types and can come from different vendors, with this system of networking the user can access any computer. Network computing allows an application to run on another machine, according to the user's needs and the capacities of the network devices. Many applications can run on different hosts and be displayed simultaneously on a separate device.

Network computing thus provides great flexibility. Windowing systems and user interfaces (whose concepts will be explained later) ensure this flexibility.
Moreover, windowing systems are based on the model of the desktop metaphor. It allows the user to work just as someone sitting at a desk. At a desk, one can work with several sheets of paper, each one for a specific purpose, modifying from time to time the positions of the sheets and the order in wich they are piled. One can take a new sheet and fill it or change the contents of an existing one. A sheet can be destroyed or folded and set aside.
In windowing systems, such sheets of paper are windows.

1.2. Windowing systems

This section will present certain windowing system concepts. However, the desired clearness may be lacking at times, due to both the generality in describing the various systems, and the variety of terms resulting from the various viewpoints from which the systems can be described.

1.2.1. Definition of a window

From the user's point of view, a window is a well-defined, typically rectangular section of the screen dedicated to a particular activity and containing texts and/or graphs. Many types of windows exist. Each type defines particular characteristics, among which is the activity allowed in this window.

1.2.2. Definition of a windowing system

A windowing system is a union of hardware and software components that allows windows to be displayed and manipulated on a bit-mapped device.

In the related litterature, the expression 'window system' has two meanings. An additional expression 'windowing system' will be introduced to differentiate these meanings : a window system is only a part of a windowing system and will be discussed later.

1.2.3. Types of windowing systems

Two types of windowing systems have been developped : *kernel-based* windowing systems and *network-based* (also server-based or distributed) windowing systems.

1.2.3.1. Kernel-based windowing systems

In kernel-based windowing systems, only the display device directly connected to the computer running an application can receive its output, including graphics and text information. Kernel-based windowing systems are typically used on stand-alone PCs and will not be discussed any further ; rather the focus will be on network-based windowing systems.

1.2.3.2. Network-based windowing systems

Network-based windowing systems allow any display device on the network to be the destination of an application's output. In other words, an application may be running on a computer while displaying its graphs and/or texts on a remote display device on the network.

These systems are based on a client-server model and they are built according to the layered architecture given in figure 1.1. . The client- and server sides correspond to the computing and display devices respectively. Each layer of each side is a software component of the windowing system.
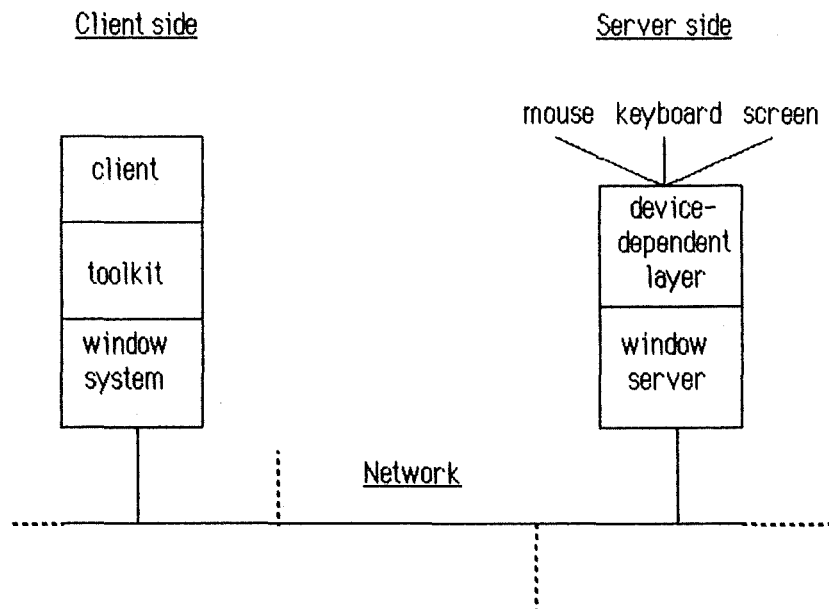
figure 1.1. client-server model of a windowing system

Server side

The server side of the model corresponds to the display device. To this device are connected two input devices the user can utilize to interact with an application. These elements are a keyboard and a pointing device which is generally a mouse (with at least one but usually two or three buttons). Attention should be paid to the expresion ' *window server* ', or ' *server* '. There are two important differences between a window server and a traditional server on a network :

1. A server in this model is not a piece of hardware but a software.
2. It does not exist on a unique place on the network but on each display device.

Moreover, the window server is always located on the display device, not on the computing device, unless both devices be the same machine.

When a device is both a computing and a display device, the application can be displayed on the machine running it, and the network-based windowing system behaves, in this case, almost like a

kernel-based one. When an application is displayed on the machine running it, clearly the server is located on the computing device, as both parts of the client-server model are merged.

The word 'server' is used in fact because it manages shared resources (the display and input devices). It receives requests which tell it to perform operations on the graph and text elements displayed on the device it controls. In the opposite direction, the server collects events related to these components and transmit them to the application that created them (an event is a kind of internal signal produced by inputs and manipulations of windows. These inputs and manipulations are possible thanks to the keyboard and the pointing device).

A window server communicates both with clients by using a common protocol, and with input devices and the screen due to a *device-dependent* layer. The device-dependent layer has to be changed when an input device is changed, but the server is always of the same type for a given window system.

Client side

The client side of the windowing system represents three levels of software components on a computing device.

The upper level is the *client* level. Two types of clients exist, applications and window managers, and are considered exactly in the same way by the server.

An *application* is written in a traditional programming language, usually the C language. The application incorporates elements from a toolkit layered on the second level or from a window system on the third level. These elements are mixed with the normal code.

A *window manager* is a client (thus software) directly based on the window system layer. Its goal is to arbitrate conflicting demands for the shared resources of the display device : screen space, mouse, keyboard. It manages the positions and sizes of all windows and interface components appearing on the screen.

The window manager determines two important things.

1. It decides whether the windows will be allowed to overlap or whether they must be tiled side by side.
2. It decides whether the keyboard focus will simply follow the pointer on the screen from window to window or whether the user must click a mouse button in a window to allow input in this window.

As a window system generally does not impose any policy for realizing the desktop metaphor, a window manager compensates for this intentional lack of rules. It can be said a window manager puts the desktop metaphor in concrete form by implementing such particular rules. An application usually gives a window manager hints to specify how it would like its windows be treated. However, the window manager is not obliged to follow them.

A _toolkit_ provides an application with elements to create an interface. A toolkit consists of three parts :

1. A set of prebuilt graphical or textual interface components often referred to as objects or widgets (the word 'widget' is probably an abbreviation of 'window object').
2. A set of routines for creating and manipulating these objects on the screen.
3. A framework, which is a more abstract mechanism, which

   a. defines the way of using objects and routines in an application code, that is to say defines a programming approach
   b. allows the creation of new widgets
   c. provides an internal dispatch loop, in order to get asynchronous inputs and events related to particular objects, and managing them.

Further chapters will describe particular toolkits in a more detailed way.

The _window system_ is the core of the windowing system. It provides a low-level set of routines used to implement the toolkits. An application can also directly call these routines, in order to access low-level mechanisms. This is a means of improving performance, for example. Such calls are sometimes necessary because no corresponding function exists in the toolkit, especially for pure graphic needs. However, using a toolkit in an application brings the advantage of consistency between all objects and their behavior, whereas when it directly calls the routines of the window system, an application must itself ensure this consistency.

These routines are directly used by applications as rarely as possible because they are really not convenient when compared to those provided by a toolkit.

They allow the communication across the network to the server in a hardware-independent and network-transparent way. They do this by conforming to the protocol defined by the window system.

1.3. Graphical user interfaces

This section presents some relatively ambiguous terminology. The terms introduced here do not refer to new concepts but rather give another, restricted point of view. Until now, we have presented the whole network environment. This section particularly deals with the description of the interface of an application running in such a system.

In a strict sense, a *Graphical User Interface* (GUI) is what the user sees and uses on the screen. According to Frank Hayes and Nick Baran in [Hay 89] , a graphical user interface is the union of the following parts :

    - a pointing device, typically a mouse
    - on-screen menus that can appear or disappear under pointing-device control
    - windows that graphically display what the computer is doing
    - icons that represent files, directories and so on
    - dialog boxes, buttons, sliders, check boxes and many other graphical objects that let the
      application be told what to do and how to do it.

The first sense of GUI is thus essentially a set of graphical objects displayed on a screen.
The precise choice of graphical objects is made by each particular GUI.
The second sense of a GUI is a structure corresponding to the client side of the client-server model of windowing systems. Figure 1.2. shows this structure.

GUI   Client side

look&feel	———————▶	client
API	———————▶	toolkit
realization	———————▶	window system
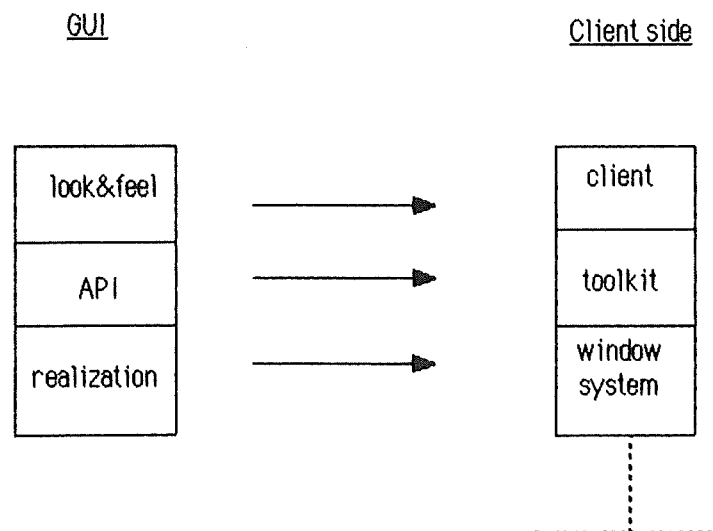
figure 1.2. structure of a GUI

Each layer of this structure caracterizes one aspect of the GUI.

A *look and feel* refers to a client. A client may only have one look and feel. This one defines the visual aspect of all objects displayed on the screen (how the user sees them), the way they can be manipulated (how the user 'feels' them), and the way they react to inputs (how they 'feel' the user). A look and feel is nothing more than a specification and for the most part, is only based upon subjective ideas and principles.

As it is a client, a window manager also has a defined look and feel. However, the look of a window manager only appears in the frame encapsulating windows created by an application. This is especially easy to see when an application with a particular look and feel is controlled by a window manager with another look and feel. In this case, it can be said which part of the interface is brought by the application and which part is brought by the window manager.

An *Application Programming Interface* (API) implements a particular look and feel by giving the applicationprogrammera set of objects which follow the rules defined by this particular look and feel and a set of routines to create and manipulate these objects.

In other words, an API is a toolkit. However, the meaning of 'API' is sometimes restricted to the set of routines.

Many APIs may exist that implement the same look and feel but a particular API only implement one look and feel.

The *realization* layer represents the window system allowing the communication with the server. A particular API is only based on one realization layer, while the latter may be the base for many APIs.

1.4. Complete example

Figure 1.3. shows a complete example of windowing architecture.

Six devices are connected in a LAN by a physical link. Device 6 is a display device, runs a window server, whereas devices 1, 3, 4, and 5 are computing devices running clients (applications and window managers). Device 2 acts at the same time as a display device and as a computing device. The logical links join computing devices and their respective display device(s). All clients of device 1 send their outputs to device 2. All clients of devices 4 and 5 do the same towards device 6, which allow clients running on different devices to be displayed on the same display device. Device 3 is connected to both display devices at the same time. That allows a client to be displayed in many places at the same time.

Three clients runs on device 1 : an application layered on a toolkit, an application directly based on the window system and a window manager. The number caracterizing a toolkit or a window manager indicates the look and feel they implement. For example, toolkit 1 allows the creation of objects with look and feel 1.

Application 3 is layered on toolkit 3 and is run and displayed on device 2. Even in such a case are the window server and the window system necessary. No particular implementation is foreseen for the case of a computer that is both the computing and the display device of an application.

Four applications are displayed on device 2 : applications 1 and 2 running on device 1, application 3 running on device 2 and application 4 running on device 3. They all are managed by window manager 1 running on device 1. These applications use different toolkits, thus different look and feel are visible at the same time on the same screen. This example illustrates the fact that an application with a particular look and feel can be managed by a window manager with another look and feel.

At the other end of the LAN, application 4 is based on toolkit 1 and is running on device 3. Application 5 is based on toolkit 2 and is running on device 4. Both applications are displayed on device 6 and are managed by window manager 2 running on device 5.
Such a configuration is not fixed but can be changed by giving other command-line parameters when invoking clients.

This figure represents a LAN. However, the same functionalities could be obtained in a Wide Area Network (WAN).
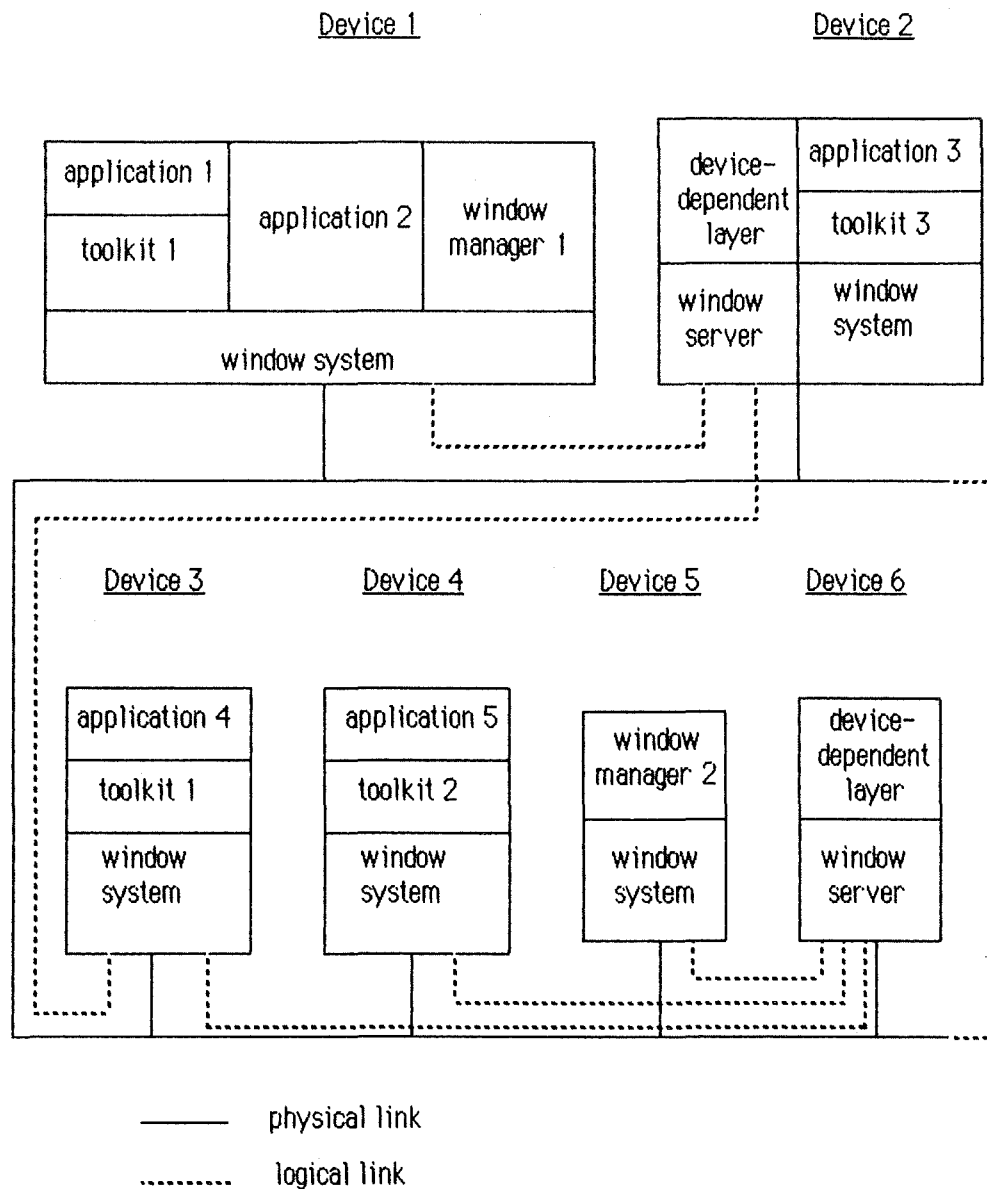
figure 1.3. complete example of windowing architecture

1.5. Historical development

The history of graphical user interfaces began fast twenty years ago, in the seventies. The only existing user interfaces available to access the capabilities of a computer was until then the simple sequential interface. Its first characteristic lies in its name : 'sequential' means that the simultaneous execution and control of many processes is not possible. It second characteristic is the support : a character-based display device.

So was the situation when the Xerox's Palo Alto Research Center (PARC) started researches grouped into many projects : Smalltalk, Star, Cedar, ... These brought the first basic ideas of GUIs.

Among them were :

- bit-mapped display devices which became workstations
- pointing device (mouse)
- windows : well-defined sections of the screen, dedicated to particular types of activities and able to be manipulated according to precise rules
- icons : small graphical representations of closed windows (temporarily not used by the user)
- direct manipulation of objects on the screen.

Sidelights on history say that Steve Jobs (from Apple) once visited PARC, saw the Star system and from it created the Macintosh in 1984. This new-style computer was one of the first after Xerox to bring on a broad public market the results of Xerox's researches. The Macintosh interface has been for a long time the GUI's reference whose definition stands for a general definition of GUIs.

As the goal of Xerox's researches was not consistency through the management of a well-defined, unique project, but explorations towards all directions, many ideas were developed (and were sometimes in conflict with each other). This gave birth to many different products from Xerox and from other companies influenced by Xerox's researches : Apple, Sun, Microsoft, to mention only a few.

1.6. Available products

Many GUI exist, as shows the following table. The most important components given here will be presented further.

Look and feel	OPEN LOOK	OPEN LOOK	OPEN LOOK	Motif	OPEN LOOK
API	Xt / Xt+	Xt / OLIT	XView	Xt / Xm	NDE
Realization	X	X	X	X	NeWS

Look and feel	OPEN LOOK	NextStep	MS-Windows	Presentation Manager
API	tNT	NextStep	MS-Windows	Presentation Manager
Realization	NeWS	Postscript	MS-Windows	Presentation Manager

1.6.1. Window systems

Two important network-based window systems are competing : X and NeWS .
They have been merged in a unique window system : X/NeWS.

1.6.1.1. The X Window System

X Window System, commonly referred to as X, is a non-proprietary system developed in collaboration with Digital Equipment Corporation and other companies by MIT's project Athena from 1984. The problem was the use of networked graphic workststions as a teaching aid for students. The presence of different hardware led to a hardware independent solution.

X takes its origins in the W windowing package developed at Stanford University. Several versions have been achieved from X version 1 (X1) to X10. X version 10 release 4 (X10R4) was the first basis for commercial products in 1986. Then other versions came, from X11R1 in September 1987 to X11R4 in January 1990.

X11 should be stable for a few years, thus allowing the development of X applications. It is moreover becoming a de facto industry standard. However, X is extensible, which means it is possible to add new primitive operations to the window system. Its code includes a mechanism for incorporating such extensions so that it will not be necessary to scan all lines of code to extend the system.

Since X11R2, X is controlled by the X consortium, formed in January 1988 : an association of computer manufacturers, software houses and universities. Software houses and universities are associate members which receive advance access to new releases.

The X Window System defines a network protocol and offers a library of low-level routines.

The X protocol

The protocol enables the communication between clients and servers via messages built on this protocol. They use this protocol even if they reside on the same machine. The X protocol distinguishes between request messages on one hand and event messages on the other hand.

Requests messages are sent by the application to the server.
An application can send a request message to the server by using routines from a library called Xlib.

Event messages are sent by the server to the application when the user interacts with the application moving the mouse, pressing a mouse button or a keyboard key or using the window manager. The server then detects that something affecting the application happened and informs it of that fact.

The X library

Xlib provides an application with a set of routines for generating and sending requests to a server so that the application programmer does not have to deal with low-level protocol details.
Xlib offers four kinds of primitives :

1. Normal calls generate most of the requests.
2. Convenience functions are a simpler and more efficient way of generating some of the requests already provided by normal calls.
3. Service functions perform local operations without using the network connection with the server. Service functions thus generate no protocol requests.
4. Informational macros and functions return information about the display's capabilities used by the application. Such routines are very important for developping portable applications.

1.6.1.2. NeWS

NeWS stands for Network Extensible Window System and has been developped by Sun. It is an operating system- and device independent system with a client-server foundation similar to that of X but it is based on Postscript.

Postscript is both a description language and a programming language.

Postscript as a description language

Postscript provides a device-independent standard for representing pages to be printed. Postscript defines graphic operators, graphic objects and an imaging model.

The graphics operators are used to create graphic objects and to control their placement.

The originality of Postscript as a description language is its paint-and-stencil imaging model.

An imaging model is a set of rules incorporated in the design of a graphic system and refers to the capabilities of this system and to the manipulation of the contents of a window. In NeWS, the imaging model is based on lines, curves and stencils rather than pixels.

The Postscript imaging model has a very abstract nature with respect to the objects it can produce. An image is built by passing some paint through a stencil before being applied to the drawing surface, as shown by figure 1.4. . The paint consists either of pure colored ink (including black and white) or of a complex texture or image. The stencil determines a possibly complex shape with lines and/or curves. This shape represents for the paint a passage through a bigger defined surface. When superimposing the paint and the stencil, only the part of the paint corresponding to the shape will be printed on the drawing surface.
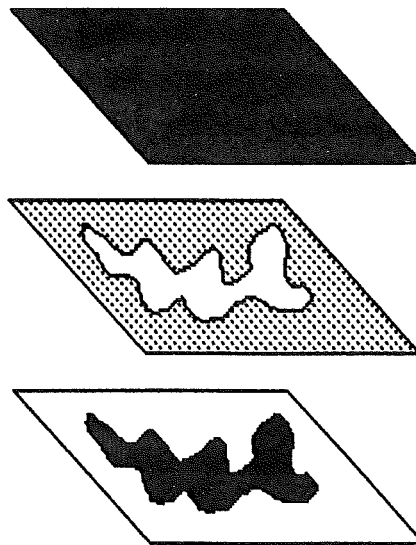
figure 1.4. the Postscript imaging model

The reader should not understand this explanation as the way of getting a printed page from a printer. This metaphor related to a concrete process only gives better chances to catch the basic idea of this imaging model.

Postscript as a programming language

Postscript allows a page description generated by an application to be run by the Postscript interpreter located on the printer-side, in order to produce the corresponding printed page.

NeWS extends the capabilities of the Postscript language so that the printer can be replaced by a screen.

This extension of the Postscript language means that instead of transmitting all the data describing a geometric figure, you only need to transmit the program creating it. Such a method has evident advantages as far as data compression is concerned.

1.6.1.3. X/NeWS

The workstation industry recognizes X and NeWS as the only windowing systems which could become standards. They were even recently merged to provide a unique server. X11/NeWS (also known as X/NeWS) has an X and a NeWS interface to communicate with both X and NeWS clients, and an underneath common layer.

1.6.2. Look and feel

Two important look and feel are in competition : Open Look and Motif.

To avoid a common mistake, the reader should keep in mind that Open Look is only the name of a particular look and feel, not of a toolkit.

Motif is less restrictive in its meaning. It first refers to a look and feel essentially based on the Presentation Manager look and feel but improved with the three-dimensional windows from the Hewlett Packard's New Wave GUI. In its second meaning, Motif refers to a X toolkit, as explained in the next section.

Other look and feel are, for example, Nextstep, Presentation Manager, MS-Windows.

1.6.3. Toolkits

On top of X has been developed the X Toolkit (with a capital 'T') known as Xt. Xt is currently part of the X standard. It defines routines : the Xt Intrinsics, and a set of widgets : the X Athena widget set (Xaw) contributed to the X community by Hewlett-Packard.

An X toolkit (note the small 't') is another set of widgets that can be used with the Intrinsics instead of Xaw. However, a few basic widgets are defined by the X Toolkit and are used with any X toolkit independently of the new set of widgets that are used. These toolkits often provide

supplementary specific routines. In the field of X toolkits, the term 'widget' is generally used rather than 'object'.
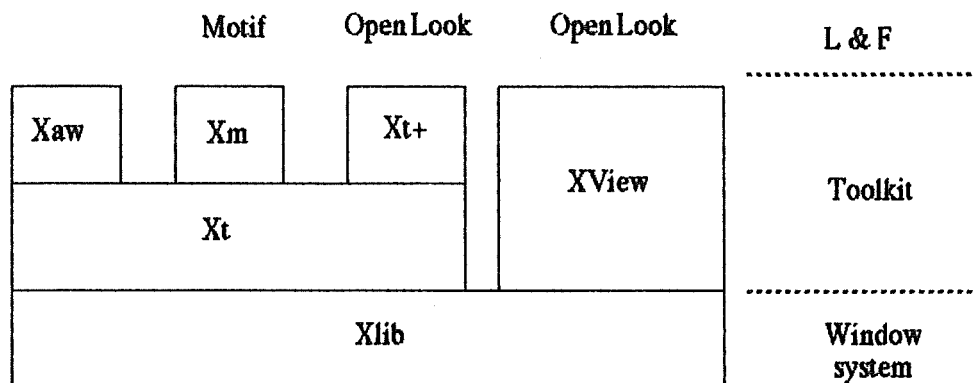
figure 1.5. major possibilities offered by the X architectures

Two significative X toolkits exist :

> Xm implements the Motif look and feel . It is mostly referred to as Motif.
>
> Xt+ implements the Open Look look and feel

Xt+ was developped by AT&T.

OLIT (Open Look Intrinsics Toolkit) is the name of the same toolkit licensed by Sun which needed an Open Look solution based on Xt for the US government.

XView implements the Open Look look and feel, too. But it is not exactly an X toolkit because it lies directly on Xlib, not on Xt. One could say that the XView 'toolkit' is equivalent to Xt and Xt+ taken together.

XView has been created by Sun to allow the migration of applications written with the SunView API in a kernel-based windowing environment (SunWindows) towards a network-based windowing one (X/NeWS).

NDE (NeWS Delopment Environment) is another Open Look toolkit from Sun. It is based on the NeWS part of X/NeWS and communicate via the Postscript language.

The NeWS toolkit (tNt) is an experimental one.

In the following chapters, we shall only consider   XView and Open Look

                                                      Motif and the Xt Intrinsics.

Chapter 2
XView and Open Look

This chapter is mainly based on [Hell 90], [SUN1 90], and [SUN2 90]. In the first part , we shall present the major characteristics of Open Look through the presentation of XView objects. The secont part will present the basic principles of XView programming.

2.1. XView packages

We said in the section 1.6.3. that XView is not exactly a toolkit. However, we will use that term from time to time, keeping in mind that former restriction.

As do all toolkits, XView has predefined objects to be used for the creation of a GUI. These objects are part of packages. A package is a set of related elements : an object, attributes and procedures.

A toolkit has some similarities with object-oriented languages and therefore the packages compose what is known as the XView class hierarchy (figure 2.1.). However, 2 packages are not part of this hierarchy : the Notifier and Notice packages.

XView distinguishes between visual and nonvisual objects.

Visual objects have their own appearance : frames, panels, windows, scrollbars are visual objects. Nonvisual objects have no own appearance but contain information used to display visual objects. Server, Screen and Fullscreen, for example, are nonvisual objects.

Because they are placed in a hierarchy, packages inherit properties from their parent class (also called superclass). Thus the root class (the Generic Object package) contains some general characteristics applicable to all objects whereas other packages define properties for themselves and for their descendants. The more a package is at a low level, the more it is specific. For example, a text object inherits general properties from the window class but also has its specific characteristics.
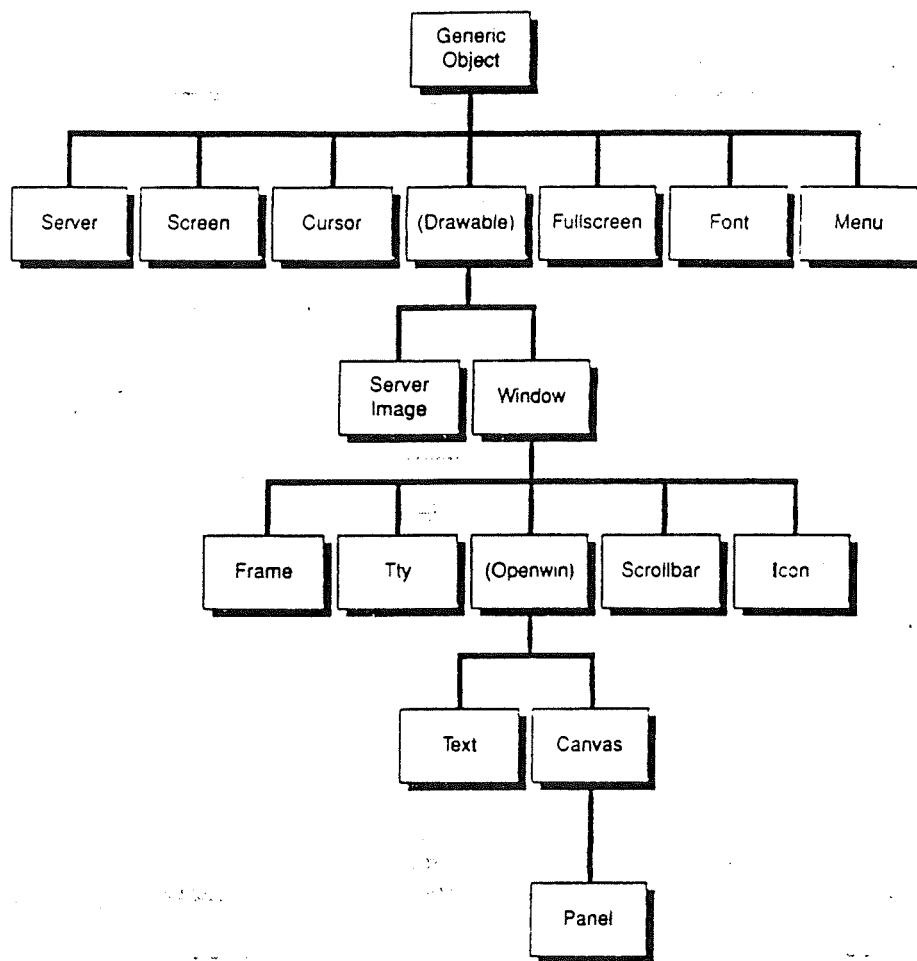
figure 2. 1. the XView class hierarchy

2.1.1. Nonvisual objects

Nonvisual objects do not contain or are not subclass of windows. They are part of the XView class hierarchy except for the Notifier and Notice packages.

These objects are mostly used internally by XView rather than directly in an application.

2.1.1.1. Generic Object

The Generic Object or XV_Object is the root object of the class hierarchy. It contains certain basic properties that all objects share. An instance of this object can never be created for itself because it has no function, but is implied in the creation of an object : after the creation of the root object, the subclass of that object is created, then the subclass of that subclass is created and so on until the object class of the type of object desired is created.

2.1.1.2. Server

The Server package is used to interact with an X server. An application begins by initializing the XView toolkit. Doing this, it especially opens a connection to the server specified in the DISPLAY environment variable of the operating system or in the command-line options if any. The initialization function returns a pointer to that server that will be used as the default server for the whole application. That server is seen by XView as an Xv_Server object. If the application must be displayed on many displays, additional connections have to be made to the corresponding servers by creating Xv_Server objects from this server package.

The first server object is thus created as a side-effect of the initialization function while the possible supplementary server objects are created by the simple creation function.

2.1.1.3. Screen

This package provides an Xv_Screen object representing the physical screen. Like the server object, this one is created by the initialization function and is then retrieved by the application. All XView objects appearing on this screen are associated with the Xv_Screen object. Because a display device may have many associated screens, many screen objects may be created, one for each screen.

Connecting to many screens should not be confused with connecting to many servers. A server is attached to a display, thus controls many screens.

2.1.1.4. Drawable

The Drawable package is not a real package in itself. It groups the Server Image and Window packages.

2.1.1.5. Fullscreen

A Fullscreen object allows an application to force the end-user to perform certain operations before being able to go on using all the capabilities of this application or of all applications displayed on the screen. For example, a user could be forced to first acknowledge the fact that an existing file already has the name he wanted to give to its own one, before giving it another name and saving it. So the Fullscreen object is used to ask the user for an immediate response or to notify him of an error that occured.

An application generally do not directly use the fullscreen package. The Fullscreen package is mostly used by the Notice package to implement its functionality.

Defining the events the user may cause and creating the fullscreen object has the simple effect of grabbing the server that is to say to temporarily stop its normal working. When the user performs one of these expected actions, the fullscreen object is destroyed and the application works normally again.

2.1.1.6. Cursor

A cursor is an image appearing on the screen to precisely locate the position of the pointer. Each window may have its own cursor appearance but generally uses a default one.

When creating a cursor, two ways are possible. The first possibility is to create within the application a server image that defines the aspect of the cursor. This server image is given to the create function as the value of a particular cursor attribute. The second possibility is to give the create function the value of another attribute. This attribute value consists of a predefined constant representing an index into an array of XView predefined glyphs (see figure 2.2.).
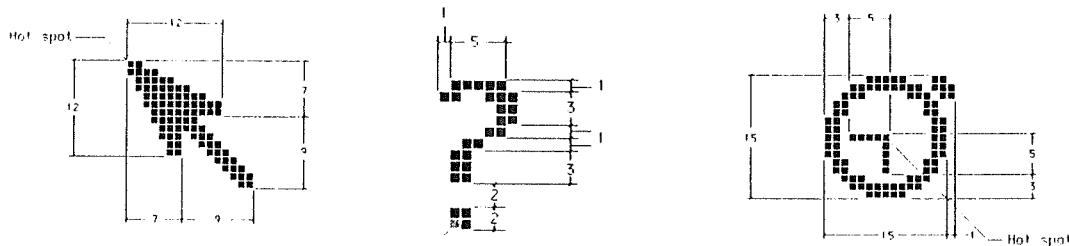
figure 2.2. cursors

2.1.1.7. Font

The Font package deals with the manipulation of fonts. A font is a character set with a particular appearance. A font is defined by its name or family (lucida, roman, courier, ...) , its style (bold, italic, bold_italic, ...) and its size or scale (10, 12, 14, 19 points).

Many XView objects use predefined fonts, complying with the Open Look specifications. These fonts have well-defined names, styles and sizes and therefore may not be changed. Many other interface components in the contrary may use whatever font the programmer wishes.

Creating an object of type Xv_Font means loading a font from the server and creating an XView font object associated with that font.

2.1.1.8. Server image

A server image is a graphic image stored on the X server. It can be used to create an icon or a cursor, for example.

2.1.1.9. Notifier

The Notifier will be described in the second part of this chapter.

2.1.2. Visual objects

2.1.2.1. Menu

A menu by itself is a windowless object. It may be attached to objects such as menu buttons, scrollbars, text subwindows, for example. The user can activate it by pressing a mouse button when the pointer is in such areas. Only at that moment, the menu is bound to a window in order to be made visible.

A menu allows the user to make a choice from different menu items. Different kinds of menu items are possible :

1. **Choice items**

   The user can select one and only one item in the menu. When the corresponding action has been performed and the menu has disappeared, that selection is completely forgotten.

figure 2.3. choice items

2. **Exclusive items**

   The user can choose one item, as he would do with choice items. The difference with choice items is that the selection of an exclusive item is retained after the menu has disappeared. Such items are used to determine non-transitory states in an application.
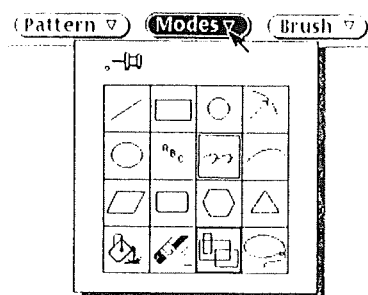
figure 2.4. exclusive items

3. Nonexclusive items

Menus that have nonexclusive items allow the user to select one or many items that will also be retained like exclusive items.
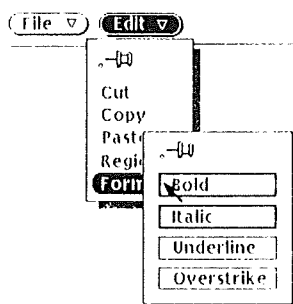
figure 2.5. nonexclusive items

All menus are popped up when activated but pop-up menus are only one type of menus among three types:

1. pop-up menus that are displayed at the pointer location when the user presses the menu button of the mouse when the pointer is in a window
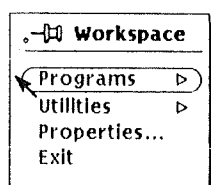
figure 2.6. pop-up menu

2. pullright menus that are displayed as a menu to the right of a menu item in a menu (see figure 2.5.)

3. pulldown menus that are displayed below a menu button on a panel (see figure 2.3.)

A pushpin can be used in some menus to pin up the menu so that it remains visible until the user closes it. (see figure 2.3.)

2.1.2.2. Window

Many objects contain windows in order to display themselves and to receive events. The window class, like the generic object class, is a hidden class : a window object is never explicitly created but an object of a subclass of the window class is created.

XView provides a set of windows that includes subwindows and frames :

1. tty (terminal emulator)
2. panels
3. canvases
4. text subwindows
5. frames

2.1.2.3. Subwindows

Subwindows never exit independently. They are always owned and maintained by a frame or another window. They may not own frames. They are constrained to fit within the borders of the frames to which they belong. Subwindows are tiled : they don't overlap. Subwindow types include

1. tty subwindows
2. panels
3. canvases
4. text subwindows

2.1.2.4. Tty

The tty subwindow emulates a standard terminal with the only difference that the number of rows and columns may vary from the 'normal' terminal.

2.1.2.5. Panel

The main function of a panel is to manage a variety of panel items (or ' controls '). It is a region of a window where controls such as buttons and settings are displayed. The panel also controls the arrangement of its controls in a horizontal or vertical fashion. Control areas within panes usually contain varied combinations of the following controls:

1. buttons
2. abbreviated (menu) buttons
3. numeric fields
4. check boxes
5. gauges

6. menu buttons
7. text fields
8. exclusive and non-exclusive choice lists
9. sliders
10. messages

The figure 2.7. shows examples of panel items defined by Open Look.
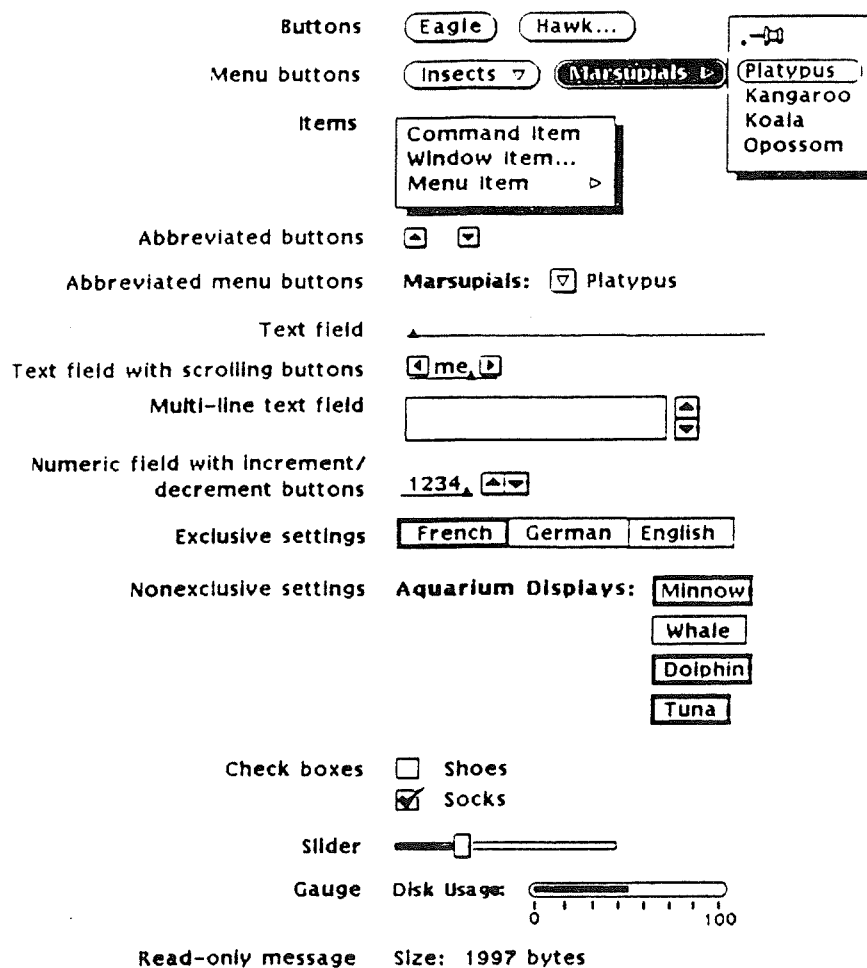
figure 2.7. panel items

Panels are used in many different contexts :

> property sheets (see figure 2.8.)
>
> notices (see figure 2.13.)
>
> menus (see figure 2.4.)
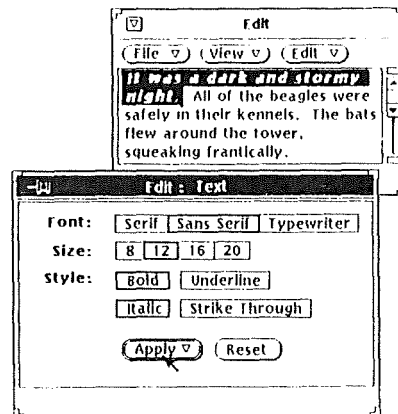
figure 2.8. property window

2.1.2.6. Textsw

The textsw package allows a user or client to display or edit a sequence of ASCII characters. It provides many text editing capabilities from the basic insertion to complex operations such as searching for and replacing a string.

figure 2.9. text subwindow

2.1.2.7. Canvas

Canvases provide drawing surfaces for the results of Xlib graphics calls. The application can draw on an area larger than the size of the visible window. The entire region is called the paint window.

The visible portion is called the view window. Views are split and joined generally by the user via the attached scrollbars.

paint window
(contains graphic)

view window
(contains no
graphic, has
scrollbars)

canvas subwindow
(displays union of view
window and paint window)

frame
(contains canvas)

figure 2.10. canvas

Three types of windows are involved with the canvas object (see figure 2.10.) :

    1. a canvas subwindow that

        is owned by a frame and

        manages one or more views

    2. one or more view windows that are the

> visible portions of the paint window

3. one or more paint windows where

> graphics and events (mouse/keyboard) take place (there is one paint window per view window)

2.1.2.8. Frame

The first XView object to be created by an application is a frame. A frame is a container for other windows. Its purpose is to manage the geometry and placement of subwindows that don't overlap and are fixed within the boundary of the frame.

The frame package provides the following capabilities

1. a communication path between the application and the window manager
2. a mechanism to receive input for the application
3. a visual container for user interface objects
4. a method to group windows with related functionality

The frame package does not manage

1. headers
2. title bars
3. footers
4. resize corners

These four elements are managed by the window manager which takes as hints the value of some attributes, for examlple : the string value of frame_label, the boolean value of frame_show_header, etc...

The frame package does not manage events either. They are managed by the windows managed by the frame.

There are two kinds of frames : base frames and pop-up frames

The base frame is the application's main frame.

Pop-up frames are typically used to perform one or more transient functions.

There are different kinds of pop-up frames:

1. <u>Command frames</u> give operands and set parameters needed for a command. They are implemented by a subframe containing only a panel. They are useful as help frames, or
   · property frames. They have a pushpin attribute.

2. <u>Help frames</u> display help text for the object under the pointer. They are implemented by a text subwindow within a subframe

3. <u>Notices</u> are used to confirm requests, to display messages and conditions (see figure 2.13.)
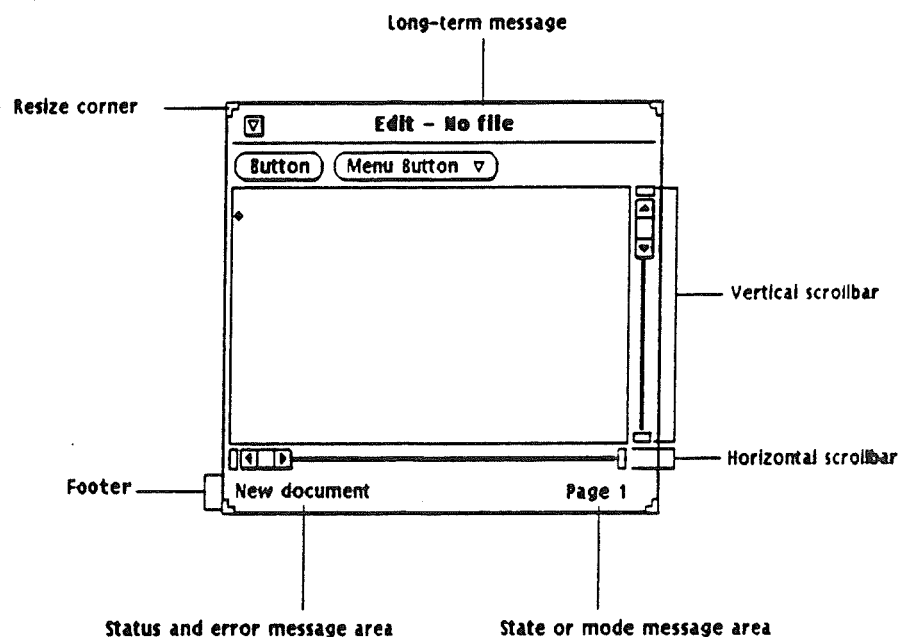
figure 2.11. a typical frame

2.1.2.9. Scrollbar

Scrollbars must be attached to a window. They are used to scroll through a document when the later is too large to be seen entirely in the window. The scrollbar package manages only the scrollbar window and does not control the window to which it is attached. Because its functionality is bound to other objects, it is sometimes wrongly considered to be a property of these objects. Scrollbars can be oriented vertically or horizontally, except in some packages (in text subwindows, they are vertical).

Scrollbars attached to canvases or text subwindows can be used to split views.

2.1.2.10. Icon

An icon is a small picture representing a closed window. A window may be closed to save space on the screen but is still active, except for the fact that it can not receive input from the user. An icon is defined by a server image, like cursors.
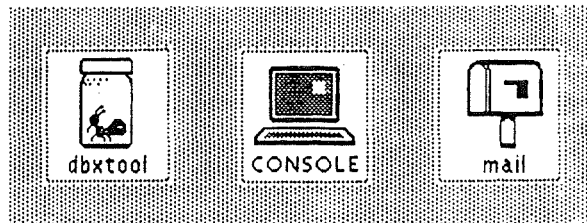
figure 2.12. icons

2.1.2.11. Openwin

Openwin is a hidden class which provides attributes for panel, canvas and text objects.

2.1.2.12. Notice

A notice is a pop-up window bound to notify the user that something went wrong and/or to ask a question that requires an immediate response. Notices are useful to confirm important operations that can not be undone.
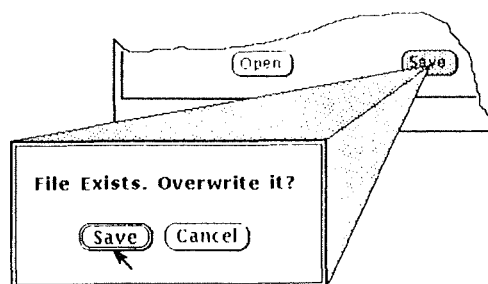
figure 2.13. notice

2.2. XView programming

XView is an attribute-value API, which means it has few function or procedure calls. The idea of the XView API is to provide a small number of functions and a large set of attributes. The functions take as arguments a list of attribute-value pairs but generally only a small part of the whole set of attributres is used for this purpose. As the length of the arguments list is variable, the last attribute-value pair is always followed by 'NULL'.

There are three categories of attributes:

1. Generic attributes are prefixed by 'xv_' and apply to all XView objects
2. Common attributes are prefixed by 'xv-' and apply to many but not all objects
3. Specific attributes are prefixed by a package name (PANEL, FRAME, ...) and apply to that package only.

The three categories of attributes may be used together in a function call.

Some attributes have a boolean value to indicate wether or not the object can cause or come under defined actions such as resizing, repainting, ... Other ones contain information needed to perform certain tasks. Other attributes caracterize the appearance of the object.

Objects from all packages can be created or manipulated using a few functions common to all packages: xv_create, xv_destroy, xv_find, xv_get, xv_set.

A few other generic functions (prefixed by 'xv_') exist. They are less important.

Other functions apply to specific packages and are prefixed by the name of the package. They are too various to be presented in a synthesized way. They deal with particular operations.

2.2.1. Xv_init

The xv_init function performs three operations.

1. It reads any argument given when invoking the application: some are XView arguments, other are application arguments. The function can behave in two ways, according to the first attribute value. In the first case, it removes all arguments that are XView-specific from the command-line and returns the latter to the application. Such a call looks like:

```
xv_init ( XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL ) ;
```

In the second case, the command-line is returned unchanged to the application that must parse it itself in order to distinguish the arguments specific to XView or to the application. In this case, xv_init is called as follows:

```
xv_init ( XV_INIT_ARGS, argc, argv, NULL ) ;
```

2. It establishes the connection with the server specified in the command-line options or with the default server. Xv_init opens a connection to only one server. If the application must be used on many displays devices, subsequent connections to the respective servers are created by creating the same number of Server objects. (After the first call to xv_init, subsequent calls to that function are ignored).

3. It initializes the Notifier.

2.2.2. Xv_create

The xv_create function creates an instance of an object and returns a handle to that object.

```
Frame frame ;
frame = xv_create ( NULL, FRAME, NULL) ;
Panel panel ;
panel = xv_create ( frame, PANEL,NULL ) ;
Panel_item button ;
button = xv_create ( panel, PANEL_BUTTON,
          PANEL_LABEL_STRING, "Quit",
          PANEL_NOTIFY_PROC, Quit,
          NULL ) ;
```

These lines of code first create the base frame of the application, then a panel in this frame and a button in this panel. The definition of xv_create is :

```
xv_create ( owner / or parent / , package, attribute-value pairs ).
```

The PANEL_NOTIFY_PROC attribute indicates which notify procedure will be executed when the button is activated.

2.2.3. Xv_destroy

The xv_destroy function destroys an instance of an object and all ones descended from it. The correct way to quit an application is thus to destroy the application's base frame and to exit the application.

That function returns the value XV_OK if no errors occurred, or XV_ERROR in the opposite case. The quit notify procedure of the section 2.2.2. could use the xv_destroy function in this way :

```
    void quit ()
{
    if ( xv_destroy ( frame ) == XV_OK )
    /*  If no problem occurred during the destruction of the application's base frame  */
        exit ( 0 );
    /*  quit the application  */
}
```

2.2.4. Xv_find

The xv_find function finds an instance of an object that meets certain criteria. If the object does not exist, it creates it. The definition of the xv_find function is the same as that of the xv_create one :

xv_find (owner [or parent] , package, attribute-value pairs).

That function returns a handle to the existing object or to the newly created one. The use of this function prevents from creating many instances of the same object. Such a constraint exists for example for fonts :

```
    Xv_Font   font ;
    font = xv_find ( frame , FONT,
                FONT_NAME,    "fixed",
                NULL ) ;
```

In the example above, a font named "fixed" (and existing on almost all servers) is created as child of the base frame of the application.

2.2.5. Xv_get

The xv_get function gets the value of one or many attribute(s) of an object. Its definition is :

xv_get (object , attribute)

The function returns a value that is an opaque data type, as attributes may be of different types. That is why that value must be forced to the correct type by writing the latter before the function name.

In the following example, the xv_get function is used to retrieve the value of the label of the button created in section 2.2.2.

```
char *label ;
```

```
label = (char *) xv_get ( button , PANEL_LABEL_STRING ) ;
```

2.2.6. Xv_set

The xv_set function sets the value of one or many attribute(s) of an object

```
xv_set (frame,
        FRAME_LABEL,              "test",
        FRAME_SHOW_LABEL,  TRUE,
        FRAME_NO_CONFIRM,  TRUE,
        NULL ) ;
```

This function call sets three attributes of the frame created in section 2.2.2. .

2.2.7. The Notifier

When programming a user interface with traditional tools, the programmer has to provide for a main loop collecting all input from the user. Such a task imposes to test all kinds of events from the keyboard or from the mouse. Erroneous and correct input must be distinguished and treated by the application.

The Notifier is a process consisting of a loop dealing with the dispatching of events such as keystrokes and mouse movements. The main control loop resides in the Notifier, not in the application. Each component of an application receives only the events the user has directed towards it. . This way of doing is called event-driven programming (see figure 2. 14.).
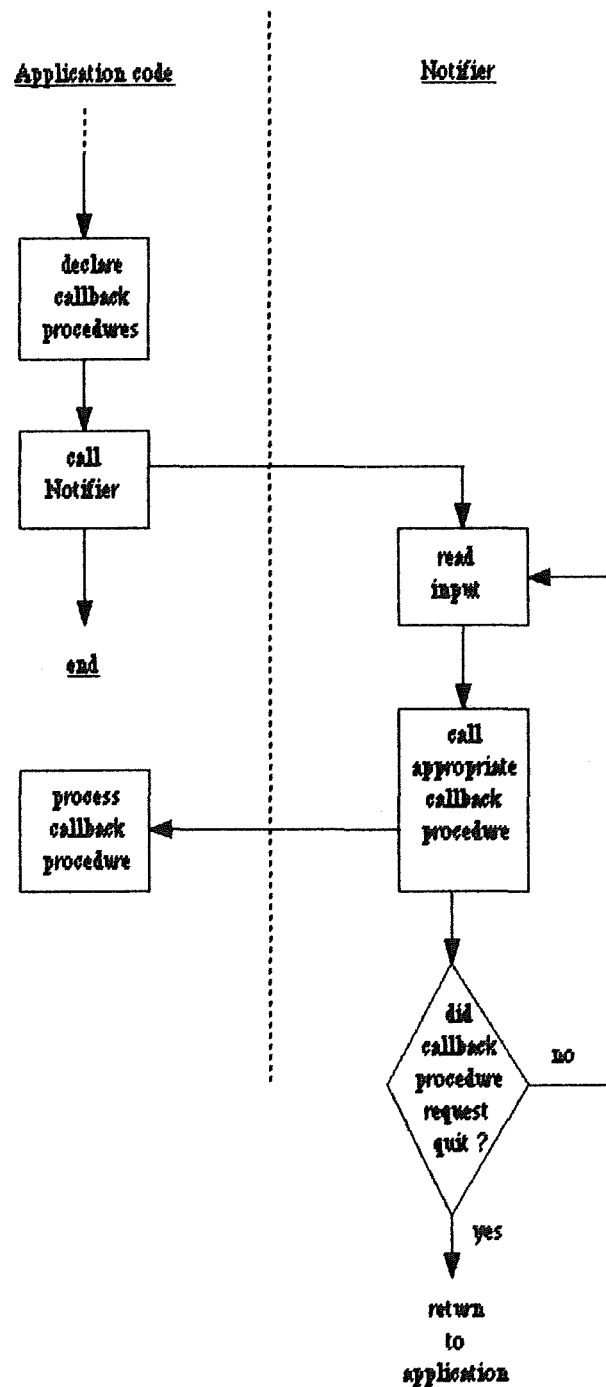
figure 2.14. event-driven programming

2.2.8. Structure of XView applications

The general structure of XView programs is :

1. Include header files

Two types of include statements are generally used. The order in which they appear is important because variables and types defined in files of the first type may be used in files of the other type. The first type of header files groups general files used in traditional C programs :

```
#include < stdio.h >
#include < string.h >
```

The second type groups XView header files. The first file contains declarations used by all packages:

```
#include < xview/xview.h>
```

The following files refer to particular packages, for example :

```
#include < xview/frame.h >
#include < xview/panel.h >
#include < xview/textsw.h >
```

2. Initialize XView using xv_init
3. Create a top-level window (FRAME) to manage subwindows and other objects
4. Create the children objects tree
5. Call xv_main_loop to start the dispatching of events

Chapter 3
OSF/Motif and the Xt Intrinsics

The gist of the third chapter originates from the OSF/Motif manuals : [OSF1 90], [OSF2 90], [OSF3 90], and [OSF4 90] and from some X Toolkit manuals : [Nye2 90], [Vol5 90], and [You 89].

As we did in the second chapter, we will present here most of the Motif widgets. Moreover, as widgets are only one side of the toolkit, we will present the programming approach of the Xt Intrinsics, too.

3.1. Motif widgets

Motif defines a lot of widget and gadget classes and allows the creation of new widget classes. The meaning of 'widget' has already been explained in the first chapter : a widget is a graphical component of a user interface. At first sight, gadgets are improved forms of widgets. They will be presented later in this chapter.

A widget class consists of the procedures and data applicable to all widgets belonging to that class. These procedures and data can be inherited by subclasses. A widget is an instance of a widget class.

A *Resource* acts as an attribute of a widget. Resources determine the appearance and functionality of widgets. They are either specific to a particular widget class or directly inherited from the superclass. This superclass may inherit resources from its superclass and transmit them to its child(ren).

If a class has few resources, they will be described.

The basic class hierarchy structure is given below. The Core, Composite and Constraint classes are provided by the X Toolkit. The Object and RectObj classes are superclasses for windowless

widgets. The whole Core class provides support for windowed widgets. Composite widgets are containers for any number of child widgets. Constraint widgets maintain additional state data for their children, as for example, constraints on the child's geometry.
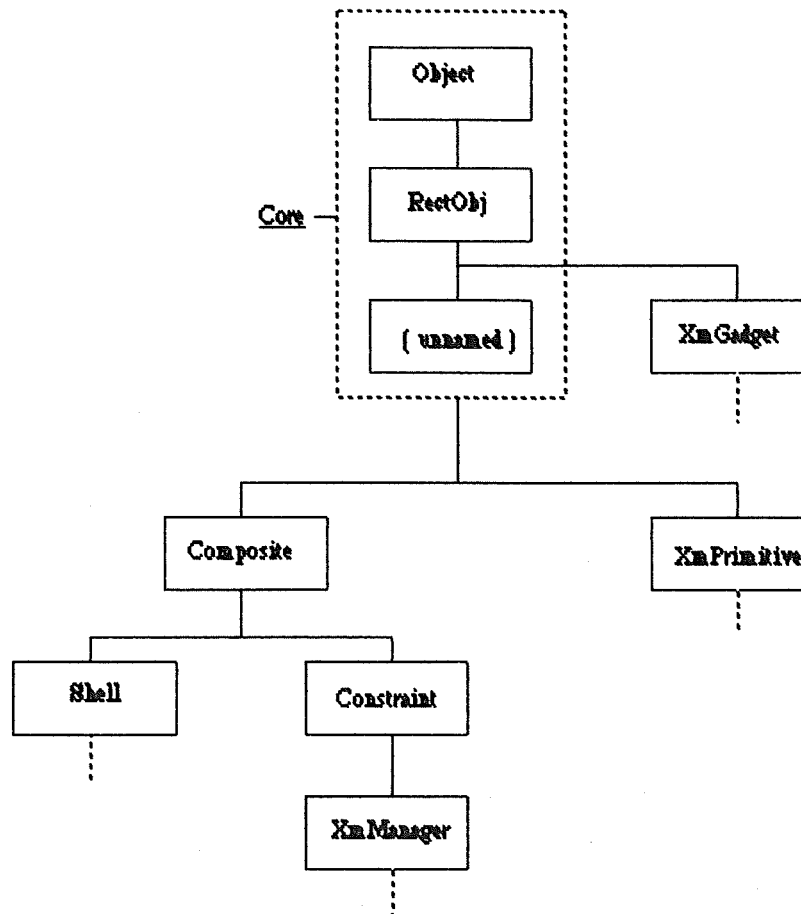
figure 3.1. basic widget class hierarchy

All Motif widget classes are grouped into six categories :

1. Shell widget classes
2. Display widget classes
3. Container widget classes
4. Dialog widget classes
5. Gadget classes
6. Menu widget classes

3.1.1. Shell widgets

The term 'Shell' both represents one of the five categories of widget classes and a widget class of that category. Widgets of the Shell category constitute the interface between the window manager and other widgets. Each window displayed on the screen is based on a Shell widget of the Shell category. Shell widgets are invisible. Many different Shell classes have been designed, according to the child widgets they accept.
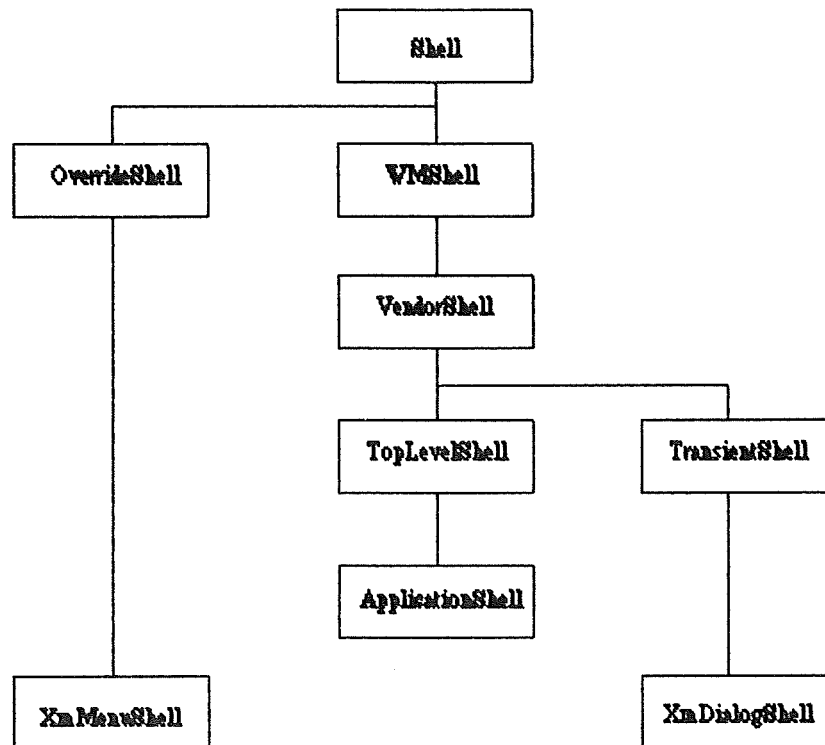
figure 3.2. Shell widget classes

Two important types of Shell widgets exist : private and public ones. Shell, WMShell and VendorShell widgets are private while the others are public. Private widgets may not be instantiated. They are internal and are not offered to the interface designer. These widgets just transmit their resources to subclasses. On the contrary, public Shell widgets may be used as normal widgets (except for the fact that they are invisible).

DialogShell widgets stand as an exception because they are semi-public. A DialogShell widget is always created as part of a set of widgets, either directly by the programmer, as a public widget, or internally by functions, as a private widget.

The X Toolkit provides seven Shell classes, while Motif defines three such classes. The Xt Shell classes are : Shell, OverrideShell, WMShell, VendorShell, TransientShell, TopLevelShell and

ApplicationShell. The Motif-specific Shell classes are : VendorShell, XmMenuShell and XmDialogShell.

3.1.1.1. Shell

Shell is the base Shell class. It provides resources for all other Shell classes. This top-level widget only accepts one child.

3.1.1.2. OverrideShell

OverrideShell is a special Shell class bound to be ignored by the window manager and therefore is used to create pop-up menus. OverrideShell adds no specific resources to the resources inherited from its superclasses. It only changes the default value of two resources inherited from the Shell class :

1. *XmNoverrideRedirect* is a boolean indicating if the shell is a temporary widget to be ignored by the window manager. Its value in the Shell class is False and it switches to True in the OverrideShell class.

2. *XmNsaveUnder* is a boolean indicating if the contents of this widget instance that appear on the screen should be saved when particular conditions are satisfied. Its value changes from False to True, too.

3.1.1.3. WMShell

WMShell defines resources to be used by the window manager protocol.

3.1.1.4. VendorShell

VendorShell contains resources used by a specific window manager. Foreseen by Xt, this class is modified according to the vendor-specific window manager.

3.1.1.5. TransientShell

TransientShell is used for shell widgets that are managed by the window manager but cannot be iconified separately from the parent window.
This class only defines one new resource :

XmNtransientFor indicates the widget for which this TransientShell widget behaves like a pop-up child.

As does OverrideShell, TransientShell changes the *XmNsaveUnder* shell resource value from False to True. Moreover, the value of the *XmNtransient* resource inherited from WMShell becomes True for TransientShell. That means that a widget instance of that class is transient and must therefore be considered in a special way by the window manager.

3.1.1.6. TopLevelShell

TopLevelShell is used to create a top-level window to which corresponds a set of windows. It introduces three specific resources:

    1. *XmNiconic* is a boolean that tells the window manager if the application must start as an icon or not.

    2. *XmNiconName* specifies the name to be displayed in that icon.

    3. *XmNiconNameEncoding* specifies a property type that represents the encoding of the XmNiconName string.

3.1.1.7. ApplicationShell

An application normally have one toplevel (root) window of class ApplicationShell. Possible supplementary toplevel windows are of class TopLevelShell and are created using another routine. They can be considered the root of a second widget tree of the application.
ApplicationShell provides two specific resources:

    1. *XmNargc* counts the number of arguments given to the application by the user
    2. *XmNargv* contains these arguments

3.1.1.8. XmMenuShell

XmMenuShell is the parent of menu panes and therefore is the basis for creating pop-up and pulldown menus.
It has a specific resource:

XmNdefaultFontList specifies a font list for its children. This list is only used when the child widget has no own font list.

XmMenuShell also overrides the *XmNallowShellResize* resource inherited from the Shell class. That resource deals with geometry requests from children of the widget. Its value changes from False to True.

3.1.1.9. XmDialogShell

XmDialogShell is the Shell class used to create dialog windows. It has no specific resource but changes the value of the *XmNdeleteResponse* resource inherited from VendorShell. This resource indicates what should be done when a delete message from the window manager is received.

3.1.2. Display widgets
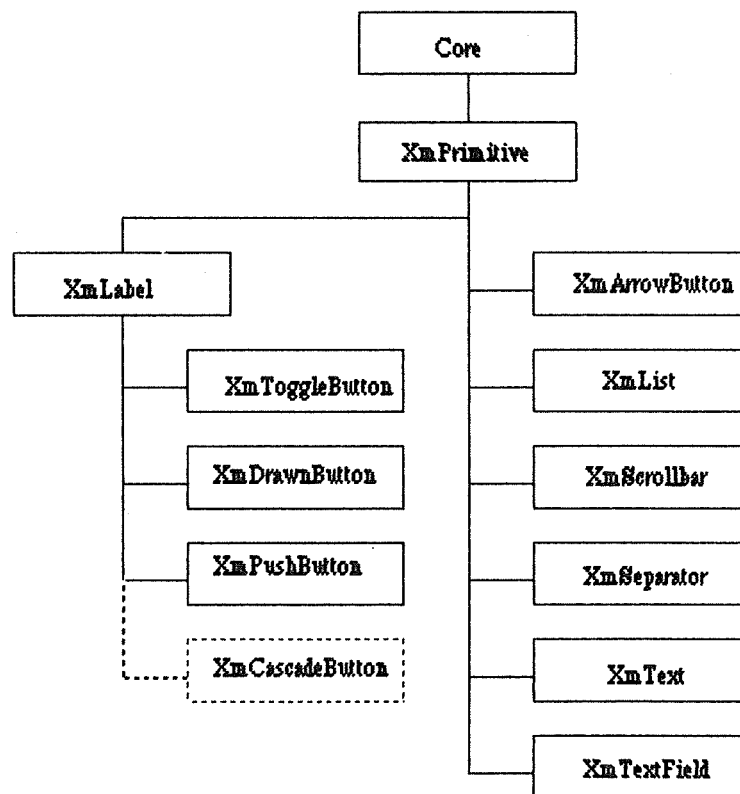
figure 3.3. display widget classes

Motif uses particular denominations. All widget classes presented in figure 3.3. are display widget classes except for the XmCascadeButton. However, XmPrimitive and all classes situated below it are primitive classes, including XmCascadeButton.

3.1.2.1. Core

The Core class is an Xt superclass that provides common resources needed by all other classes, such as x and y locations, height, width, window border width and so on.

3.1.2.2. XmPrimitive

XmPrimitive is a supporting superclass, too. Its resources essentially concern border drawing and highlighting.

3.1.2.3. XmArrowButton

The XmArrowButton widget is a directional (left, right, up or down) arrow surrounded by a border shadow. The selection of this button can make the shadow move to give the appearance that the arrow button has been pressed. When the button is unselected, it seems to be released.
The specific resources of this class refer to the direction of the arrow and the clicking and callback mechanisms.
Figure 3.4. shows a window with four ArrowButton widgets.
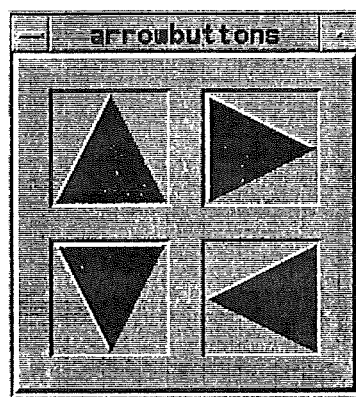
figure 3.4. ArrowButton widgets

3.1.2.4. XmPushButton

The difference between XmArrowButton and XmPushButton is the text label or the pixmap that replaces the arrow. Both types of buttons are used to invoke actions (for example, run cancel, stop, ...)

However, the default behavior of a PushButton used in a menu depends of the type of this menu. Examples of PushButton widgets can be seen in figure 3.15. .

3.1.2.5. XmDrawnButton

The XmDrawnButton widget consists of a directional arrow surrounded by a border shadow. It can give the appearance to be pushed or released, too.

3.1.2.6. XmLabel

As an instantiable widget, XmLabel can contain text or graphics.

As a superclass, XmLabel provides resources for button subclasses such as CascadeButton, DrawnButton, PushButton and ToggleButton.

3.1.2.7. XmScrollBar

A ScrollBar widget is always combined with a widget containing data too large to be viewed in its entirety. The viewable portion of the data is called the working area. ScrollBars can be placed horizontally or vertically. A window may also have one ScrollBar of each type.

A ScrollBar consists of two arrows placed at each end of a rectangle. The rectangle is called the scroll region and contains a smaller one called the slider. The data is scrolled either by clicking on an arrow, selecting on the scroll region or dragging the slider. When an arrow is selected, the slider within the scroll region is moved one step in the direction of the arrow. If the mouse button is held down, the slider continues to move at a constant rate. The ratio of the slider size to the scroll region size represents the ratio of the size of the visible data to the total size of the data.

Examples of ScrollBar widgets can be seen in figure 3.6. .

3.1.2.8. XmList

XmList allows the selection of one or more items from a list of choices. The number of visible choices is variable and a ScrollBar can be added.

Such List widgets are presented in figure 3.5.

3.1.2.9. XmSeparator

The Separator widget is a separation line between widgets. It can be placed horizontally or vertically.

figure 3.5. List widgets

3.1.2.10. XmText

The Text widget provides a single-line or multiline text editor with basic editing functionalities such as creating or editing file, cutting and pasting text, and so on.
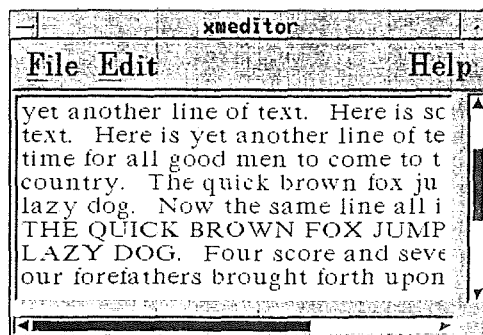
figure 3.6. Text widget

3.1.2.11. XmTextField

TextField does the same as Text for single-line texts.

3.1.2.12. XmToggleButton

The ToggleButton widget consists of a text or graphics face with an indicator placed to the left of the text or graphics. Toggle buttons are used for setting non-transitory data in an application. Diamond-shaped indicators only allow one choice to be selected while square indicators allow nonexclusive choices. The selected or unselected states of the button are indicated by an empty or filled indicator as shows figure 3.7. .
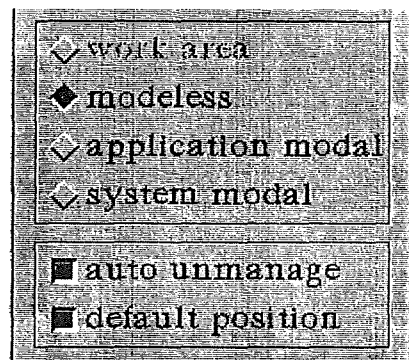
figure 3.7. ToggleButtons

3.1.3. Container widgets

Container widgets are Composite widgets that provide applications with general layout functionalities. As Composite widgets, container widgets can have children.

All classes presented in figure 3.8. are container classes except for XmBulletinBoard. The ten classes are manager classes.

3.1.3.1. XmManager

The XmManager class is never instantiated as a widget. It is a superclass for other widget classes. It supports the visual resources, graphics contexts, ... necessary for the graphics mechanisms.
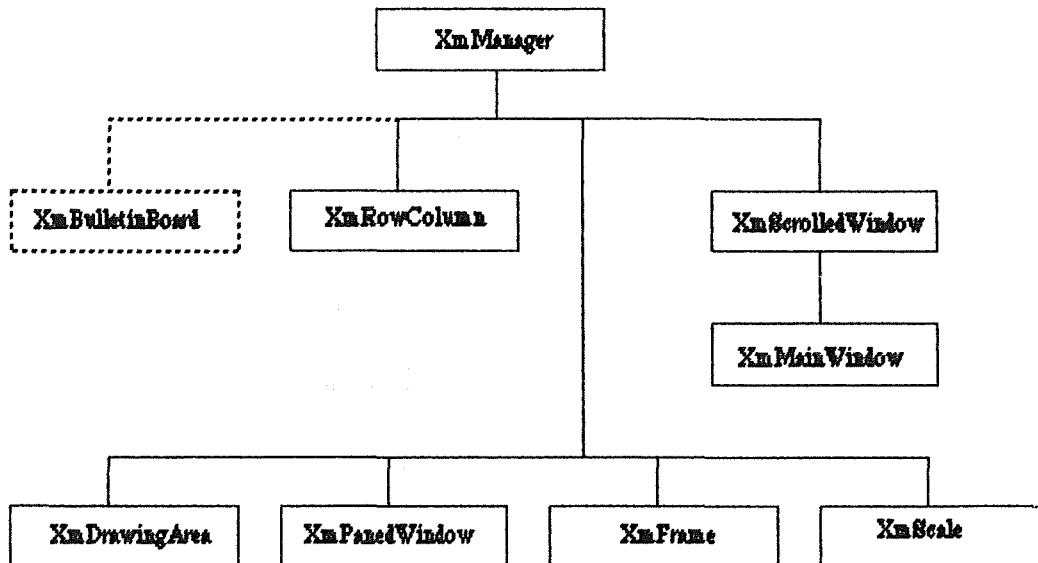
figure 3.8. container widget classes

3.1.3.2. XmDrawingArea

The DrawingArea widget easily adapts to a variety of purposes, for example displaying graphics.

3.1.3.3. XmFrame

The XmFrame widget is used to enclose a single child within a border drawn by XmFrame.

3.1.3.4. XmMainWindow

XmMainWindow can be used as the primary window of an application. This widget can display a menu bar, a command window, a work region, and scrollbars but these areas are optional.

3.1.3.5. XmRowColumn

The RowColumn widget is a general purpose manager that can contain any widget as a child. It is the basis for menus.

A RowColumn does not need to know how its children behave.

The children widgets can be laid out in rows or columns. Moreover, their layout must be one of the following :

1. The children are packed together into rows or columns (see figure 3.9.)
2. Each child is placed in an identically sized box producing a symmetrical look

3. A specific layout (the current x and y positions of the children determine the location)

A RowColumn must be created as a child of a Frame in order to have a three-dimensional shadow.
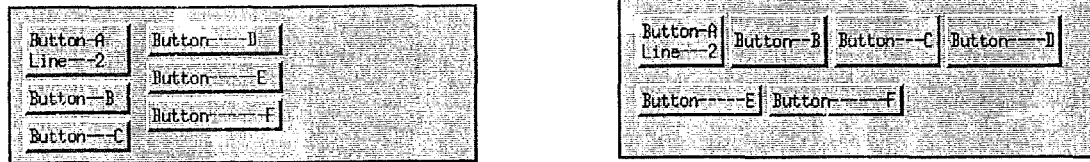
figure 3.9. RowColumn widget

3.1.3.6. XmScale

A Scale can be either input /output or only output.
It allows

- an application to indicate a value from a range of values, in the second case (only output)
- a user to input or modify a value from the same range, by adjusting a slider to a position along a line, in both cases
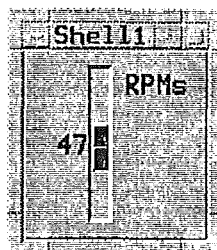
figure 3.10. Scale widget

3.1.3.7. XmScrolledWindow

The ScrolledWindow widget combines one or more scrollbars and a viewing area to implement a visible window onto another larger data display. The visible part of the window can be scrolled through the larger display by using scrollbars.

3.1.3.8. XmPanedWindow

A PanedWindow lays out children in vertically tiled format. Children appear from top to bottom. The first child inserted appears at the top of the PanedWindow and the last child at the bottom. The PanedWindow widget grows to match the width of the widest child and all other children are forced to this width.

Maximum and minimum sizes can be specified for each pane containing a child. The height of each pane can be changed at run-time within these limits, by dragging a small square box placed at the bottom of the pane it modifies.

Figure 3.11. shows a PanedWindow before and after such a modification.
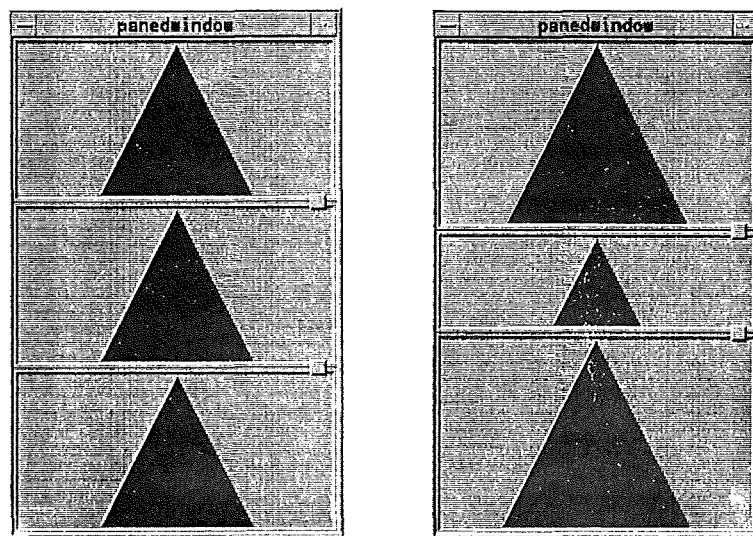
figure 3.11 PanedWindow

3.1.4. Dialog widgets

The dialog widgets are used to create Dialog widgets, also called Dialogs. (Note the small and capitals letters).

A dialog widget is one of the seven widgets listed hereafter.

A Dialog includes a DialogShell, a BulletinBoard, (or a subclass of BulletinBoard, or another container widget) and various children such as Label, PushButton, Text.
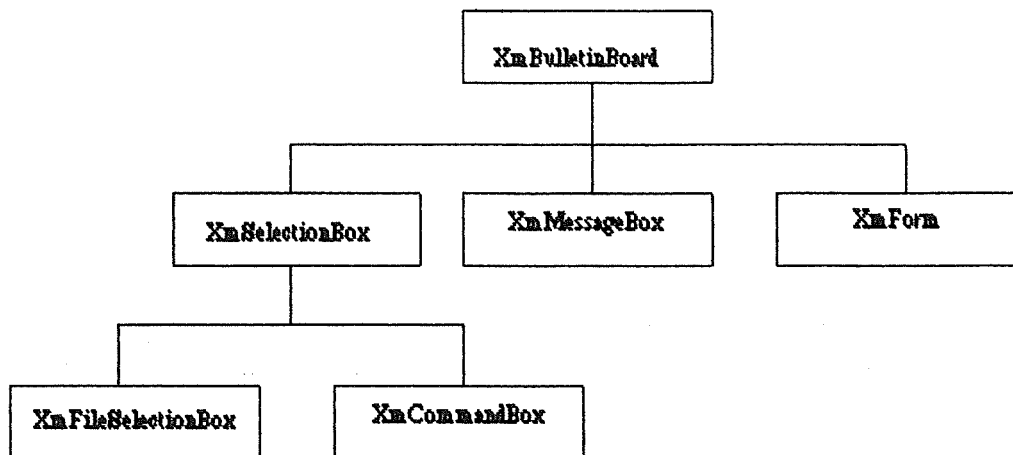
figure 3.12. Dialog widget classes

Dialogs are used for interaction tasks such as displaying messages, setting properties and providing selection from a list of items. They normally ask a question or give the user some information requiring a response. A Dialog can be modal or modeless. A modal Dialog stops the work session and asks the user for some input. A modeless Dialog waits for input from the user but does not interrupt interaction with any application. The modal character may be defined in many ways by setting the *XmNdialogStyle* resource of BulletinBoard to the appropriate value out of three :

1. *XmDialog_System_Modal* : means that the Dialog must be responded to before any other interaction in any application.

2. *XmDialog_Application_Modal* : means that the Dialog must be responded to before some other interactions in ancestors of the widget.

3. *XmDialog_Full_Application_Modal* : means that the Dialog must be responded to before some other interactions in the application.

3.1.4.1. XmDialogShell

The XmDialogShell widget has been presented as a widget of the Shell category.
However, as it is the basis for all Dialogs, it can be seen in a sense like a dialog widget, too.

3.1.4.2. XmBulletinBoard

BulletinBoard provides simple geometry management for children widgets. It does not impose the position of its children but may refuse geometry requests that would make children overlap. BulletinBoard is a base widget for most Dialogs but is also used as a general container widget.

3.1.4.3. XmForm

The Form widget provides a layout language used to establish spatial relationships between its children. Specific resources are used to specify attachments of the child's sides to the Form's sides. Form maintains these relationships when it is resized, new children are added or when its children are resized, unmanaged or destroyed.

Figure 3.13. gives an example. The first ArrowButton on the left has been positionned within the Frame widget by means of three constraints :

1. The top of the ArrowButton is set to 20 pixels under the top of the Form.
2. The distance between the left sides of the Form and the ArrowButton is ten percent of the total width of the Form.
3. The distance between the left side of the Form and the right side of the ArrowButton is thirty percent of the total width of the Form.

As the bottom side of the ArrowButton has not been constrained, the height of that widget is maintained when the Form is resized. Anyway, the left and right sides move in order to maintain the constraints.
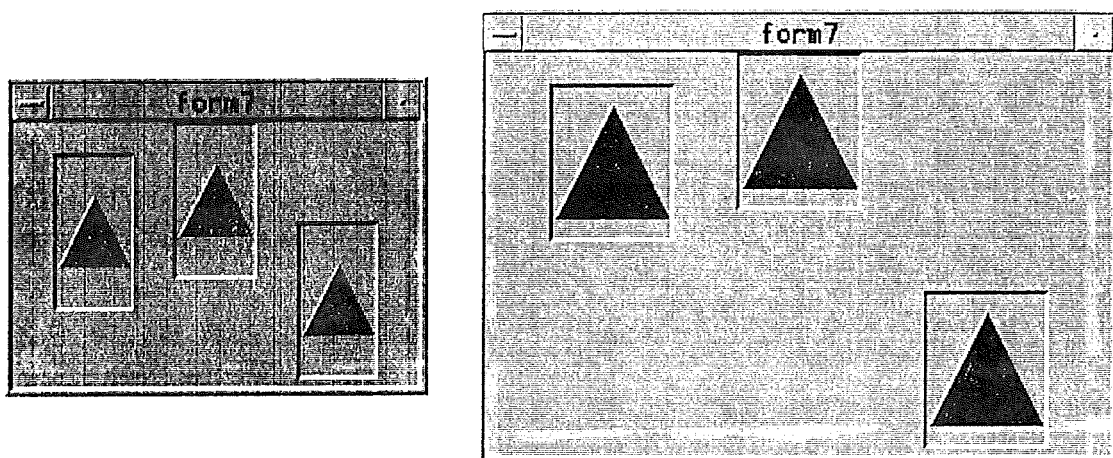
figure 3.13. Form widget

3.1.4.4. XmMessageBox

MessageBox is the base widget for providing information to the user. Three buttons are available : OK, CANCEL, HELP but are not mandatory. This widget contains a message symbol and the message itself.

3.1.4.5. XmSelectionBox

The SelectionBox widget is used to select an item from a list of choices. It contains a message, an editable text field and a scrolling list of choices. Four buttons are available : OK, CANCEL, APPLY, HELP.

3.1.4.6. XmCommandBox

In the OSF/Motif documentation, this widget is sometimes called XmCommand, too.

XmCommand displays a command line input text field, a command line prompt and a command history mechanism. It is similar to a SelectionBox but can also record selections in a history region. This history region is accessible and items can be selected from it.

Figure 3.14. shows a CommandBox with a scrolled history region.
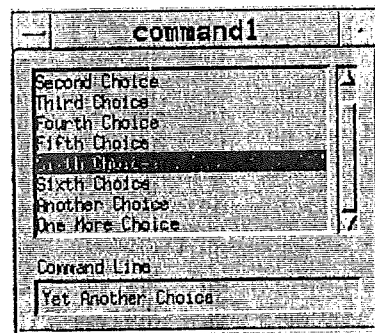
figure 3.14. Command widget

3.1.4.7. XmFileSelectionBox

A FileSlectionBox gives the user a way of selecting a file in any directory or subdirectory. It has five main areas :

1. An input text field for displaying and editing a directory mask used to select the files to be displayed

2. A scrollable list of filenames

3. A scrollable list of subdirectories

4. An input text field for displaying and editing a filename

5. A group of PushButtons : OK, FILTER, CANCEL, HELP

3.1.4.8. Dialog convenience functions

A Dialog can be created either by creating each of its widgets in turn or by using a Dialog convenience function.

Dialog convenience functions allow the creation of group of widgets or gadgets by making just one function call. A convenience function creates a predetermined set of widgets and returns the parent widget's identifier. Dialog functions are of the form :

XmCreate< *Dialog*>Dialog.

A Dialog convenience function instantiates a dialog widget as a child of a DialogShell to create a convenience Dialog. For example, the QuestionDialog convenience function creates a DialogShell and a MessageBox and PushButtons as children of the DialogShell.

The following six convenience Dialogs (and functions) are based on the MessageBox :

1. (XmCreate)WorkingDialog : tells the user a time-consuming operation is executing and gives him the opportunity for cancelling it. The default symbol is a square with an hourglass in it.

2. (XmCreate)WarningDialog : warns the user of the consequences of an action and gives him the opportunity for cancelling it. The default symbol is an exclamation mark. (see figure 3.16.)
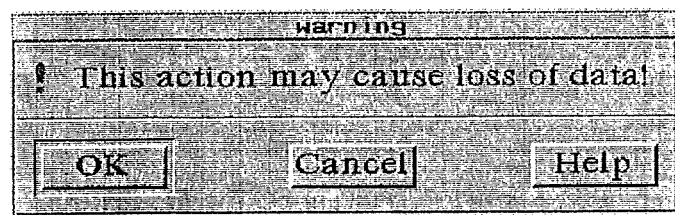
figure 3.15. WarningDialog widget

3. (XmCreate)QuestionDialog : gets an answer from the user. The default symbol is a question mark. (see figure 3.15.)
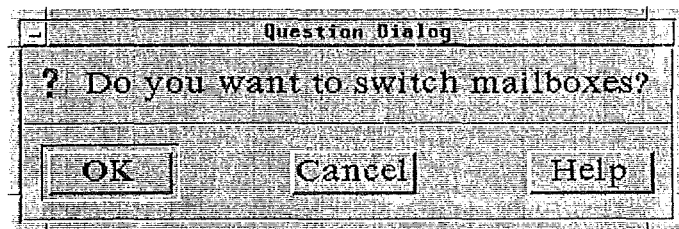
figure3.16. QuestionDialog widget

4. (XmCreate)InformationDialog : gives information to the user. The default symbol is a square with an ' i ' in it.

5. (XmCreate)ErrorDialog : warns the user of an invalid or potentially dangerous condition. The default symbol is an hexagon with a hand inside.

6. (XmCreate)MessageDialog : gives information to the user. There is no default symbol.

Motif determines the number of PushButtons and the symbol to display for each Dialog, while the message explaining the error , asking the question, or giving an information is given by the application.

The other convenience Dialogs (and functions) are :

1. (XmCreate)FileSelectionBoxDialog is based on FileSelectionBox and allows the user to select a file. (see figure 3.17.)
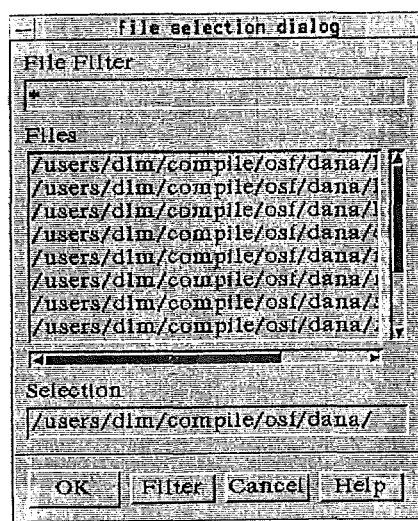
figure3.17. FileSelectionBox widget

2. (XmCreate)BulletinBoardDialog is based on BulletinBoard and is used for interactions not supported by the standard Dialogs.

3. (XmCreate)SelectionBoxDialog is based on SelectionBox and allows the user to get a selection from a list.

4. (XmCreate)FormDialog is based on Form and is used for interactions not supported by the standard Dialogs.

5. (XmCreate)PromptDialog is based on SelectionBox and asks the user for a text input.

3.1.5. Gadgets

Gadgets provide for the most part the same functionalities as the equivalent widgets. The main reason to define gadgets is to improve performance, both in execution time and data space. This applies to both the application and server processes and minimizes the amount of lost functionality. The difference is so important between widgets and gadgets that these should be used whenever possible.

Gadgets can be understood as windowless widgets. They don't have any of the visual resources found in the XmPrimitive class.
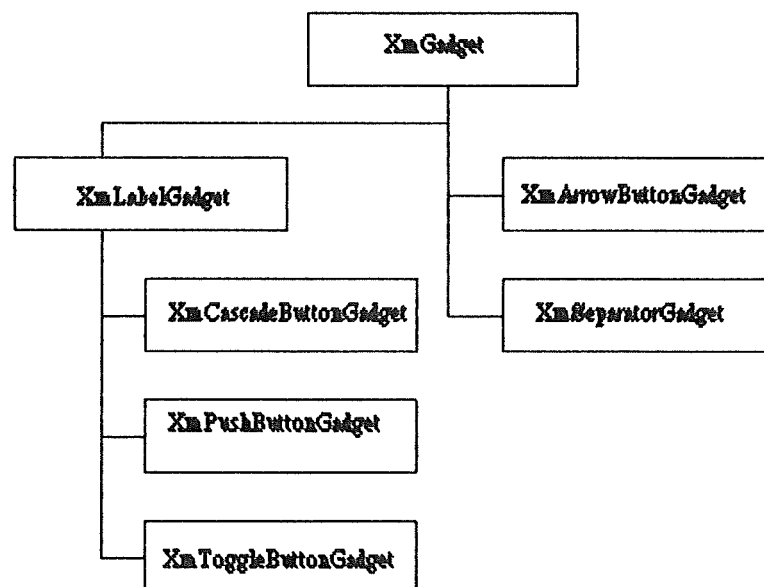
figure 3.18. Gadget classes

Motif provides the following gadgets :

1. XmGadget (the superclass for other gadget classes)
2. XmArrowButtonGadget
3. XmSeparatorGadget

4. XmLabelGadget

5. XmCascadeButtonGadget

6. XmPushButtonGadget

7. XmToggleButtonGadget

3.1.6. Menu widgets

A menu system is a combination of widgets that produce the visual and interactive behavior of a menu. The Motif menu system designates the set of all such menu systems.
Motif provides three types of menu systems :

1. pop-up menu systems

2. pulldown menu systems

3. option menu systems

All menu systems are based on the RowColumn widget.

3.1.6.1. Pop-up menu systems

A pop-up menu system simply appears at the pointer location when required by the user. It consists of a MenuPane generally containing PushButton, ToggleButton and/or CascadeButton widgets.
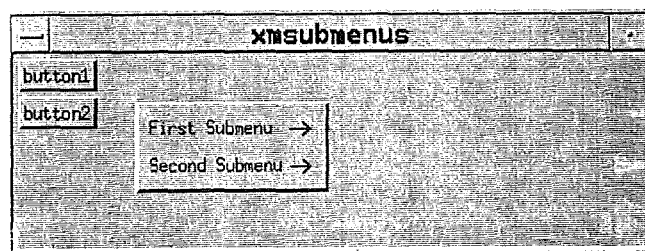
figure 3.19. top-level of a pop-up menu system
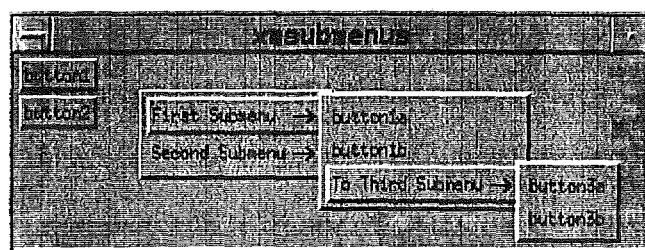
figure 3.20. complete pop-up menu system

In figure 3.19. , the top-level of the pop-up menu contains two CascadeButtons giving access when required to pulldown cascading submenus in figure 3.20.

3.1.6.2. Pulldown menu systems

A pulldown menu system consists of a MenuBar containing CascadeButtons. To each CascadeButton is attached a pulldown MenuPane. This MenuPane may contain PushButtons, ToggleButtons and/or CascadeButtons. Like in pop-up menu systems, PushButtons are used to invoke actions (cut, run, save, ...) ToggleButons set data and CascadeButtons can give access to pulldown submenu panes.
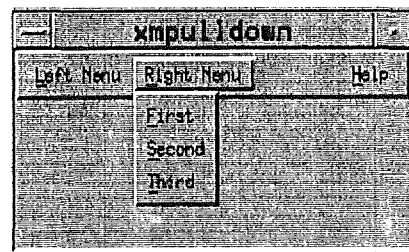
figure 3.21. pulldown menu system

3.1.6.3. Option menu systems

An option menu system consists of one or more Labels describing the sets of options. A CascadeButton is placed to the right of each Label. It can give access to a pulldown MenuPane containing PushButtons and permanently contains a Label indicating the most recent option selected in the pulldown menu. Figure 3.22. shows the top-level of such a menu system and figure 3.23. shows the whole first option menu.

figure 3.22. top-level of an option menu system
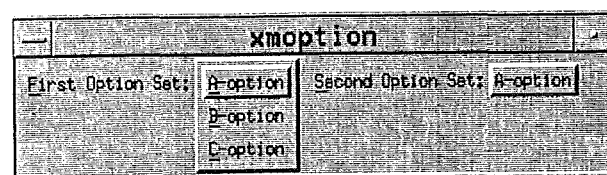
figure 3.23. option menu system

3.1.6.4. The RowColumn widget

The RowColumn widget is the basis for most of the menu system components. It can behave like a menu bar, a pulldown menu pane, a pop-up menu pane or an option menu, according to the values given to some of its resources. Convenience functions have been provided to easily create these special versions of the RowColumn widget. When necessary, they first create a MenuShell widget, too. The Motif menu system is composed of the following widgets and convenience functions:

1. XmRowColumn (widget)
2. XmCreateMenuBar (convenience function)
3. XmCreateOptionMenu (convenience function)
4. XmCreatePulldownMenu (convenience function)
5. XmCreatePopMenu (convenience function)
6. XmMenuShell (widget)

7. XmCascadeButton (widget and gadget)
8. XmSeparator (widget and gadget)
9. XmLabel (widget and gadget)
10. XmToggleButton (widget and gadget)
11. XmPushButton (widget and gadget)

3.2. Structure of a Motif application

This section explains the steps to follow to write a Motif application on the base of the C language:

1. Include header files
2. Initialize the Xt Intrinsics
3. Create the widgets
4. Realize the widgets
5. Enter the main loop

3.2.1. Include header files

Three types of include statements are generally used. The order in which they appear is important because variables and types defined in files of the first type may be used in files of both other types. Similarly, variables and types defined in files of the second type may be used in files of the third type.

The first type of header files groups general files used in traditional C programs:

```
#include < stdio.h >
#include < string.h >
```

The second type refers to the X Toolkit. We explained in section 1.6.3. that an X toolkit is composed of a set of routines, the Xt Intrinsics, and a specific set of widgets (the Motif widget set in the case of the Motif toolkit). In order to use these routines, the following statement must be found in the code (remember that the X Toolkit, thus the Intrinsics, are part of the X standard and are to be found in the X11 directory) :

#include <X11/Intrinsics.h >

The first two parts of the set of include statements are virtually the same for all applications. The third one is the more likely to change. It refers to the Motif toolkit. Only the first header file of this type, Xm.h, is mandatory in all cases because it contains definitions for all Motif widgets. So the next statement looks like

#include <Xm/Xm.h >

No particular order must be taken into account for the following lines. One include statement must be added for each widget class used in the program, no matter if the widget is used one or many times.
These statements look like

#include <Xm/ widget.h >

For example :   #include < Xm/BulletinB.h > includes the header file for the BulletinBoard widget.

3.2.2. Initialize the Xt Intrinsics

It should now have become clear to the reader that a Motif application is first of all an Xt application, even if it is a particular one depending of the widget that is used. That is why the object of the initialization are the Xt Intrinsics and not the Motif toolkit.

The initialization always occurs before any other call to an Xt function. It is performed by the *XtAppInitialize* function. That function requires many arguments and creates an application context, establishes the connection with the server, analyses the command line given to call the application, and returns a shell widget to be used as the root widget of the application. A shell widget accepts one child and is the interface between that child and the window manager.

The next two lines are an example of initialization.

Widget Phonebook:

**

Phonebook = XtAppInitialize (&Appc , "test" , 0 , 0 , &argc , argv , 0 , al , 0) ;

Phonebook is declared of type Widget, thus could be any widget. However, once it has been assigned a type, it can not change any more.

a. &Appc represents a pointer to an application context. Application contexts enable the coexistence of several logical applications in a single address space.

b. "test" gives the class name of the application.

c. The next two arguments are the options and number of options indicating how to parse the command line. The initialization function uses and then removes all arguments it can recognize in the command line. It returns the application a command line with application-specific arguments but no X or Xt specific arguments any more.

d. &argc is a pointer to the number of arguments in the command line.

e. argv is the command line itself. argc and argv will be modified according to the third and fourth arguments.

f. The seventh argument gives resources to be used if the resource file of the application can not be read.

g. al is an array of resources characterizing the Shell widget to create.

h. The last argument indicates how many arguments have to be taken into account in al.

XtAppInitialize is not the only initialization function.
The XtInitialize function is a more simple function that calls three other Intrinsics functions :

XtToolkitInitialize just initializes the internals of the Xt Intrinsics

XtOpenDisplay opens a connection with the display server

XtAppCreateShell creates a top-level shell for the application.

Instead of using the single XtInitialize function, the application may call these three functions separately. In this case, if an application context is created (by a call to *XtCreateApplicationContext* , it is used as an argument of the XtOpenDisplay and XtAppCreateShell. If no application context has been defined, a default one is used.

So, many ways are possible to initialize the Intrinsics. Calling a single function (XtInitialize or XtAppInitialize) is an easier and probably a more efficient way to do it than using many calls to Xt

functions. However, in this latter case, some customization is possible, as for example the choice of the precise shell widget to use as the parent widget for the application.

3.2.3. Create the widgets

The widgets created at this step will be the children and descendants of the top-level widget created by the initialization function. The creation of a widget involves three parts:

1. set up the arguments of the widget
2. create and manage the widget
3. add the callback routines for each widget

3.2.3.1. Set up the arguments of the widget

The arguments are pairs of resource names and values. They define characteristics of the widget : location, size, functionalities,...

The *XtSetArg* function is used to build an array with all arguments referring to the widget to create:

```
Arg al (10) ;


XtSetArg (al (0) , XmNx , 234) ;
XtSetArg (al (1) , XmNy , 35) ;
XtSetArg (al (2) , XmNlabelString , str1] ;
```

The array al contains ten elements, each of which is of type *Arg*. The predefined Arg type consists of a string (corresponding to the resource name) and a value (corresponding to the resource value). After the three calls to XtSetArg, the first three elements of al are three pairs of resource names and values : the x and y locations and the string that will be displayed in the widget. The str1 string variable must be of a particular type : the *XmString* type.

At this point, al may be given as an argument to the create function of the widget.

Let us now suppose that the programmer want , for any reason, to remove the set up of the XmNy resource. That implies that all subsequent statements will have to receive another ordering number. In the code given above, the third one will change from ... (al (2) , ...) to ... (al (1) , ...).

Another way of setting the arguments makes it easier to add or delete calls to XtSetArg.

```
Arg al (20) ;
Integer n=0 ;
```

```
XtSetArg (al (n) , XmNx , 234) ; n++ ;
XtSetArg (al (n) , XmNy , 35) ; n++ ;
XtSetArg (al (n) , XmNlabelString , str1) ; n++ ;
```

The idea is to use an integer variable index for the array, instead of a constant.
Removing the second statement now causes no problem any more.

3.2.3.2. Create and manage the widgets

All Motif widgets can be created by means of a Motif convenience function or an Xt function.

1. *Motif convenience functions*

The Motif convenience functions look like :

```
XmCreate widget ( parent , name , arguments , number of arguments ) ;
```

Keeping in mind the preceding sections, let us consider the following example :

```
Widget MB ;
MB = XmCreateMenubar ( Phonebook , "Menbar" , NULL , 0 ) ;
```

A MenuBar is created as a child widget in the Phonebook Shell. MB is its name. The widget is created only with default resource values as no arguments are given to the creation function. As the value of the fourth argument is 0, replacing the third one by al (the array containing the arguments that describe the MenuBar widget) would produce the same result.

When necessary, the Motif convenience functions create a parent Shell widget from the Shell category as the parent of the desired widget.

2. *The Xt create function*

The Xt function used to create a widget is :

```
XtCreateWidget ( name , widget class name , parent , arguments , number of arguments ) ;
```

The arguments are the same as those required by a Motif convenience function, except for the widget class name. The information the latter brings to the Xt function is already available through the name itself of the Motif function.

3. *Manage the widgets*

Both XmCreate *widget* and XtCreateWidget functions create an unmanaged widget. A widget must be managed by its parent. That means for example that the widget's parent must manage the size and location of the widget and control input to the widget. Children can be laid out in rows and columns while others will be grouped in scrollable lists.

To tell the widget's parent it must manage its child, the application must call the Xt function *XtManageChild*(widget) . In this function, the widget given as argument is supposed to have been created either by an Xm or by the Xt function.

XtManageChild is the only function that can manage a widget created by a Motif convenience funtion.

The Intrinsics offer two other functions to manage widgets :

1. *XtCreateManagedWidget*calls XtCreateWidget and XtManageChild

2. *XtManageChildren*(list of widgets , number of widgets) takes as arguments widgets created by XtCreateWidget.

The use of XtCreateManagedWidget is an advantage for the programmer. However, each time a new managed child is added to the list of managed children of a widget, the latter must perform the whole geometry management again. So, when many children of the same widget must be created and managed, it is a more efficient way of doing it to create unmanaged children that will simultaneously be managed by XtManageChildren.

3.2.3.3. Add the callback routines for each widget.

Using a toolkit in an application allow to separate the code creating the user interface from the application code itself. That means that each of these two parts can be created or modified without looking at the other part.

However, the interface and application codes need to communicate during the execution of the whole code. The mechanism mostly used to link these two parts is called 'callback mechanism' and uses callback procedures. Another mechanism is based on 'actions'. It will not be described here.

Many widgets define one or more callback resources. A callback resource gives the list of callback procedures that will be executed under precise conditions. For example, the ArrowButon widget defines the following ones :

1. *XmNactivateCallback*: list of callback procedures called when the ArrowButton is activated. Activating a widget means pressing and releasing a defined mouse button while the pointer is inside the widget.

2. *XmNarmCallback*: list of callback procedures called when the ArrowButton is armed. Arming a widget means pressing a defined mouse button while the pointer is inside the widget.

3. *XmNdisarmCallback*: list of callback procedures called when the ArrowButton is disarmed. Disarming a widget means pressing a defined mouse button while the pointer is inside the widget (same definition as XmNarmCallback).

A callback procedure can be added to the list for a precise callback by means of the Xt function :

    XtAddCallback (widget , callback resource , callback procedure , client data)

For example, if quit_button is a PushButton,

    XtAddCallback (quit_button , XmNactivateCallback , quit_proc , data)

means that clicking on quit_button will call quit_proc with the data argument.
Callback procedures must be defined this way (example of the quit-proc procedure) :

    quit_proc (quit_button , client_data , call_data)

The first argument is the widget that caused the procedure to be called. The second argument is the value passed as the last argument of XtAddCallback. The third argument is a widget-specific data that the widget passes to the application. However, not all widgets define such a value, in which case, call_data is set to NULL.

3.2.3.4. Realize the widgets

Once the widgets have been created and managed, they have to be realized to be visible. The Xt function

    XtRealizeWidget (widget)

realizes (makes visible) the widget given as argument and all its descendants. This widget is thus normally the root widget created by the initialization function.

3.2.3.5. Enter the main loop

Entering the main loop means calling *XtMainLoop*. If an application context has been defined, that function is replaced by *XtAppMainLoop*(application_context).

This loop is an Xt function indefinitely collecting events and dispatching them to the widgets. These in turn will call callback procedures. The mechanism is very similar to that of the XView Notifier.

Events are caused by keystrokes, mouse buttons or pointer movements.

Chapter 4
An application

This chapter is dedicated to a particular application : an internal phone book for BIM, the firm where the traineeship related to this thesis took place.

The main goal of this application is to provide a support for the study and the use of the two toolkits presented in the previous chapters. In fact, the choice of the application has no importance for itself. First of all, we present the list of the minimal functionalities to be implemented by that application and which components can be used, in order to have a base of comparison. We then describe the existing internal phone book at the time of the traineeship. In the following section, the application itself is described and we show how we expect to use the selected components. After that, the algorithms of the XView and the Motif versions are presented. However, because of the very important amount of time needed to master them, the specific implementation problems they brought in addition to the typical difficulties of the C language could not be resolved in time. That's why neither the XView version nor the Motif version of the application are fully implemented (see annexes).

Anyway, their developments are significant enough to allow the presentation of the algorithms for the two versions.

The chapter ends with the definition of a generic language bound to describe algorithms independently of any of both toolkits.

4.1. Selection of toolkit components

Confronted to a wide set of objects and widgets, we decided to group them according to their functionalities. We distinguish seven groups : container components, text capabilities, graphics capabilities, menus, scrolling capabilities, commands and choices, and informations. The choice of those groups and their number (seven) is quite arbitrary and depends on no particular criteria. However, we consider that these objects constitute the basic elements that can be found in an user interface.

We selected in each group a few objects of both toolkits to be used in a specific implementation. They are presented in the sections 4.1.1. to 4.1.7. .

After a generic description of the application, we explain which components of the selection will be used for each function of the application.

Then we present the specific algorithms and we show how we use the selected components. When necessary, we show that the choice was not judicious and that we had to use another object(s) or widget(s).

4.1.1. Container components

Container components are used to enclose and manage children objects.

XView : Frame (for one or more children)
  Panel (to accept panel items)
Motif : XmFrame (for a single child)
  XmMainWindow (as primary application's window)
  XmPanedWindow (for vertically tiled children)
  XmForm (maintains relationalships between children when resizing or adding children)
  XmBulletinBoard (does not force positioning of children)

4.1.2. Text capabilities

We consider here texts as inputs from the user.

XView : Textsw
Motif : XmTextField (for single-line fields)
  XmText (for multiline fields)

4.1.3. Graphics capabilities

These components are given for information only ; they will not be used.

XView : Canvas
Motif : XmDrawingArea
  XmLabel (for texts or graphics)

4.1.4. Menus

XView : Menu (includes pop-up, pulldown and pullright menus and pushpins)

Motif : XmRowColumn (to be instanciated to one of the following types : pop-up menu, pulldown
menu, option menu or menubar)

4.1.5. Scrolling capabilities

XView : Scrollbar (can be used to split views in canvases)

Motif : XmScrollBar

XmScrolledWindow

XmSelection Box (includes a scrolling list of choices)

4.1.6. Commands and choices

XView : Panel (with particular items)

Ttysw (terminal emulator)

Motif : XmCommand   XmFileSelectionBox

XmPushButton   XmScale

XmDrawnButton   XmSelectionBox

XmList   XmToggleButton

4.1.7. Informations

XView : Textsw

Notice

Motif : XmScale

XmMessageBox

4.2. Description of the existing internal phone book

BIM is located on three sites. Part of the staff work at the site 'Castle' in Everberg and about the
same number of persons work at the site 'Horizon' (about 70 persons) . Only a few people (about
10 persons) work at the site 'Two lions' which is very close to the Castle.

Internal phone numbers at BIM can be found in two printed lists. The first list relates to 'Horizon'
and is structured as follows :

1. name

2. first name

3. internal phone number (three-digit numbers)

4. initials

The second list concerns all BIM and contains :

1. name
2. first name
3. location character ('H' for 'Horizon'

                              'L' for 'Two Lions '

                  no character for Castle')
4. internal phone number
5. initials

Phone numbers in the first list only allow communications inside the site. This is due to the fact that you can't directly call someone working in one of the two other sites, but you have to use a public phone number to call the secretary that will dispatch the call.

In the second list, phone numbers to Horizon are different from those of the first list : they are four-digit numbers.

Both lists are regularly brought up to date, printed, and distributed to all people.

As the first list is included in the second one, problems of consistency could appear. This is more likely to happen if different persons update these lists.

A phone book application could clear such problems and could contain other interesting information.

4.3. An internal phone book application

4.3.1. Information manipulated by the application

The database should contain the following data for all persons working at BIM :

1. name
2. firstname
3. initials: the first letter of the name followed by the first letter of the christian name
4. phone number : all phone numbers are three-digit numbers except those that are used to access the site 'Horizon' from the two others. Thus two phone numbers will be given for each person working at 'Horizon' : a three-digit one to be used inside the site and a four digit number to be used from the two other sites.
5. location: the location is indicated by a character : 'H' for Horizon, 'L' for Two Lions and no character for the Castle.

6. <u>workgroup</u> : internal division of the firm ; examples of workgroups are ungETeam (Unix group expert team), sdg (software development group), Sybase.

7. <u>length of service</u>

8. <u>responsibilities</u> or function in the company

9. activities ; <u>specialization</u>

4.3.2. Description of the application

The application will provide two kinds of operations on the database : consulting and updating.

4.3.2.1. Consulting

The database will be consulted according to one of three keys : a name, a first name, or a specialization. The usefulness of the name or of the first name as a key is quite clear. Using the specialization as a key allows to give the user the list of all persons likely to solve his problems in a precise field.

When the user has introduced such a key, the information related to all persons whose data match the key is displayed. Complete or cut information can be provided. The complete information consists of the nine fields of data described in section 4.3.1. The cut information only relates to the fields 1 to 4. When the user gives a key value that corresponds to nobody, he is notified of that fact.

Another way of consulting the database is to ask for a list of informations. Two lists are available. One list provides all information contained in the database, that is to say all fields of data for all persons. That list thus provides a sequential access to the whole database. The second list is composed of all activities or specializations mentioned in the fields number nine of the database. That list is useful to find the exact denomination of a specialization before using it as a key.

4.3.2.2. Updating

The application manipulates data which are clearly subject to changes. Therefore, it should provide authorized users a way to update the database. The three traditional functions are offered : the user can insert, modify, or delete data.

A password is not planned yet because the application will first be designed for a restricted use, not for a public one (all employees). Even in the latter case, it does not matter if some information is wrong because the aim is to test such things as : ease of use, missing or unnecessary functionalities, presentation of the windows, of the information, ...

The pair (name , first name) is considered as the identifier of a person.

4.3.2.3. Quitting

For people who have never heard about a window manager, the application should give the user an explicit and easy way of quitting it. Moreover, the user should be able to confirm or deny his choice.

4.3.3. Implementation of the application

As was section 4.1., this section was written before programming the application. That is why some components may be not well chosen.

4.3.3.1. The main window

The three groups of operations of section 4.3.2. are typically represented in an application's main window by three distinct buttons. The separation of these three elements clearly shows that the three operations or groups of operations are completely different.

We shall use an XView frame and a panel for one version and a Motif MainWindow and a BulletinBoard for the other. The buttons will be panel item buttons for XView and PushButtons for Motif. They will contain a corresponding label (an attribute of the XView button and a Label widget).

With these elements, the application's main window should look like figure 4.1..
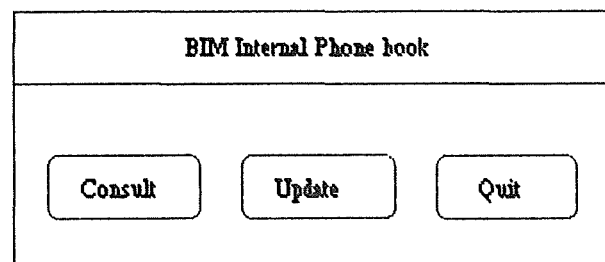
figure 4.1. the main window

4.3.3.2. The Consult menu

Whereas the three groups of operations are always shown in the main window, the different operations themselves are hidden and displayed as menus when required. That avoids confusing the user that would otherwise have too many possibilities presented at the same time.

The button labeled 'Consult' brings up a pulldown menu when activated. The Consult menu consists of the options 'Keys' and 'Lists' which both refer to pullright menus (see figure 4.2.).

The objects that will be used for the Consult menu and submenus are the XView menu and the MotifRowColumn with the correct attribute or resource values.
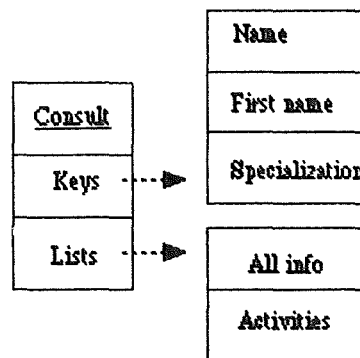
figure 4.2. the Consult menu

4.3.3.2.1. The Keys submenu

- Each option 'Name', 'First name', and 'Specialization' in the Keys menu gives rise to the same scenario:

1. A dialog window (figure 4.3.) is created and asks the user for a name, a first name or a specialization as a search key. The user has two choices. The first one deals with complete or cut information. The second one allows him either to trigger the search with the given key or to quit the option back to the main window, making the dialog window disappear. An 'Erase' button can be used to set the input text field to blank.

figure 4.3. a dialog window

To build this dialog window, we shall use the following components :

Elements	XView	Motif
window	frame and panel	BulletinBoard
'Consult : Name'	frame attribute	XmLabel
buttons	panel items (buttons)	PushButtons
choice	panel item (exclusive choice)	ToggleButtons
input field	panel item (text field)	XmTextField

2.1. If the name, first name or specialization does not exist in the database,

    a. the dialog window remains unchanged and

    b. an error window appears (figure 4.4.) . No further use of the application is possible until the 'OK' button is selected, which causes the error window to disappear. The input field in the dialog window is set to blank, ready for a new request.

The name **********
does not exist

OK
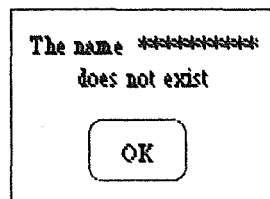
figure 4.4. an error window

We shall use an XView notice or a Motif MessageBox for the error window.

2.2. If the key refers to one or more persons, the complete information or the cut information is displayed in an information window (figure 4.5.) provided with three buttons. The buttons labeled 'Next' and 'Previous' then allow the navigation through the list of selected persons, in order to display the information related to another person. Figure 4.5 presents a complete information window. A cut information window would only contain 4 data fields and would be smaller. The dialog window is still available, with an input field set to blank. In all cases, once the dialog window has been created, it remains visible until being released by the 'Quit' button.

No new components are used here for the information window : except for the ToggleButtons, we shall use the same objects as those of the dialog window.

4.3.3.2.2. The Lists submenu

Two sorts of lists can be asked for :

1. List of all information contained in the database

2. List of all activities mentioned in the database in the fields number nine

The option 'All info' directly gives rise to a complete information window similar to that presented in the figure 4.5. That information window allows the sequential access to the whole database.

The second list is useful when the user don't exactly know the precise denomination of an activity. That list is presented in an information window provided with a 'Quit' button and a scrollbar (figure 4.6.).

figure 4.5. a complete information window

figure 4.6. an information window for the activities

This information window for the activities is provided with an XView scrollbar or a Motif ScrollBar. The textual part of the window is an XView Textsw or a Motif Text.

4.3.3.3. The Update menu

The Update menu is based on the XView menu package or the Motif RowColumn like the Consult menu.

figure 4.7. the Update menu

All update items ('Insert', 'Modify' and 'Delete') of the Update menu make an update window appear (figure 4.8.). Such a window is provided with a 'Check' button, an 'Erase button', a 'Quit' button and one update button corresponding to the item selected in the update menu. It is also provided with nine labels placed before the nine input text fields of data already described.

The 'Check' button can be used to make the application detecting wether a person already exists in the database. Such a way of doing is not mandatory but can save the user the trouble of needlessly typing in all other related information.

Of course, after using that button, the user still needs to use the update button to perform the desired operation

The user must explicitly dismiss the Update window so that multiple updates are performed in an easier way : you do not need to select an item in the update menu each time you wish to update the database.

4.3.3.3.1. Inserting

The user first gives the name and first name, then he possibly gives the other information and press the insert button.

If the pair (name , first name) already exists in the database, an error window pops up and the user has to try with another name or to quit. Otherwise, he types in all information.

figure 4.8. an update window

The Update window requires the same objects as the information window (figure 4.5.).

4.3.3.3.2. Modifying

The user gives the name and first name of the person whose data are not corect any more and press the modify button.

If the pair (name , first name) does not exist in the database, an error window pops up and the user has to try with another name or to quit.

If it exists, all information is displayed in the same window and the user may edit it.

4.3.3.3.3. Deleting

The user only needs to give the name and first name of the person whose data must disappear from the list and press the delete button.

If the pair (name , first name) does not exist in the database, an error window pops up and the user has to try with another name or to quit.

If it exists, all information related to that person is deleted.

The error windows discussed in the last three sections will be XView notices or Motif MessageBox.

4.3.3.4. Quitting

The activation of the third button of the main window lets a confirmation window appear (figure 4.9.). No further use of the application is then possible until one of two buttons is selected : 'Quit' or 'Cancel'.

Do you want
to quit ?

Quit   Cancel
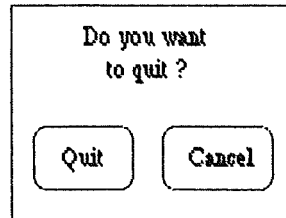
figure 4.9. the confirmation window

This confirmation window will be an XView notice or a Motif MessageBox.

4.4. The Motif version of the application

This section presents and comments the algorithms of the internal phone book application written with Motif.

The source file of the application begins with the declarations of the include files referring to the widgets used.

Many widgets are then declared as global variables as do other data strucures.

All normal and callback procedures are declared at a global level, too.

The main function of that C program is then defined.

4.4.1. Algorithm of main

The function main creates the main window of the application, its menus and its submenus, opens the application database and gives the control to the event dispatcher.

In section 4.3.3.1., we planned to use a MainWindow and a BulletinBoard to accept PushButtons that would give access to the Consult and Update menus. CascadeButtons must be used instead of them because PushButtons can not access menus. (But they can be used as menu items to perform actions). As CascadeButtons have to be placed in a menu pane or in a MenuBar, and because a MenuBar can be a child of a Shell, we do not use the MainWindow and the BulletinBoard.

These problems are the typical difficulties we encountered when programming with Motif and XView. Most of them are not mentioned in the manuals and we often had to try many possibilities to find something correct.

We thus place three CascadeButtons in a MenuBar and the latter is a child of the top-level Shell. The CascadeButtons call menus. The third one ('Quit') calls no menu. It can not be replaced by a PushButton because a MenuBar only accepts CascadeButtons.

Instead of using a RowColumn, we prefer a convenience function for its simplicity (XmCreatePullDownMenu)

Figure 4.10. presents the tree of widgets created and their callback procedures.

It is divided into three parts to be more readable.
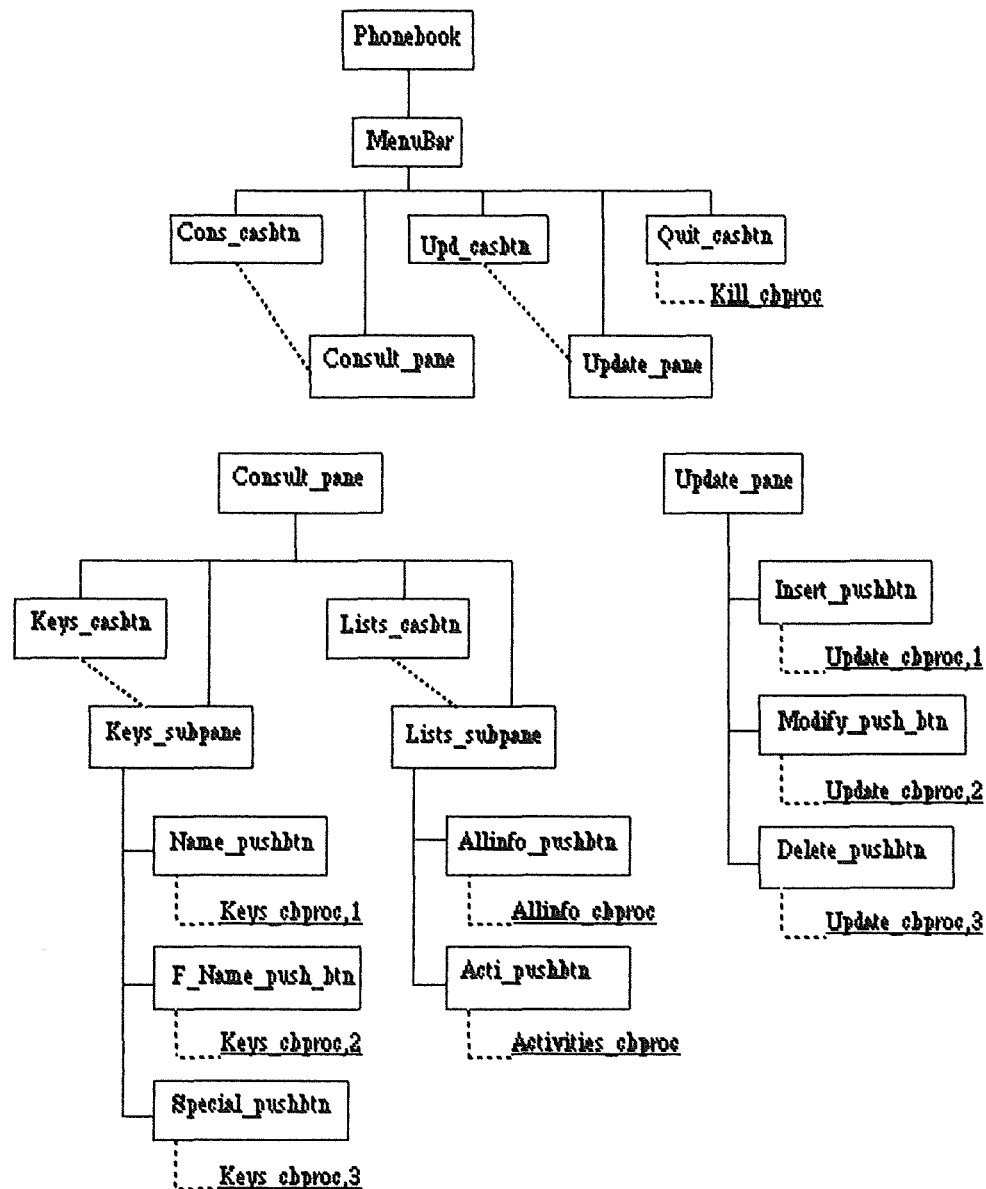
figure 4.10. widgets of the application's main window

create a top-level shell (Phonebook) by means of an initialization function

create a menu bar (MenuBar) in Phonebook

create two menu panes (Consult_pane and Update_pane) as children of MenuBar

create two cascade buttons (Cons_casbtn and Upd_casbtn) as children of MenuBar. Those
buttons are linked respectively with Consult_pane and Update_pane so that their
activation pulls down the corresponding menu.

create a cascade button (Quit_casbtn) as child of MenuBar and declare its callback routine
Kill_cbproc, to be called with the argument Phonebook.

create two pulldown submenu panes (Keys_subpane and Lists_subpane) as children of
Consult_pane

create two cascade buttons (Keys_casbtn and Lists_casbtn) as children of Consult_pane.
These buttons are linked respectively with Keys_subpane and Lists_subpane, so that
their activation pulls right the corresponding submenu.

create three push buttons (Name_pushbtn, F_Name_pushbtn, Special_pushbtn) as
children of Keys_subpane and declare their callback routine Keys_cbproc, to be
called with the respective argument value 1, 2 or 3.

create two push buttons (Allinfo_pushbtn and Acti_pushbtn) as children of Lists_subpane
and declare their respective callback routine Allinfo_proc and Activities_proc

create three push buttons (Insert_pushbtn, Modify_pushbtn and Delete_pushbtn) as
children of Update_pane and declare their callback routine Update_cbproc, to be
called with the respective argument value 1, 2 or 3.

call a function (Init) that opens the file Phonefile standing for the database or creating it if
it does not exist

call the function MainLoop.

The so-called link between a cascade button and its associated menu is created by giving the
reference of that menu pane as the value of the resource XmNsubmenuId of the cascade button.

4.4.2. Algorithm of Kill_cbproc

The function Kill_cbproc creates a confirmation window used to confirm or cancel a request to quit
the application.

As foreseen in section 4.3.3.4., the confirmation window is based on a MessageBox. However, we
use a convenience Dialog based on that widget, for its simplicity (see section 3.1.4.8.).

create a question dialog box with a label 'Do you want to quit ?', a Cancel button and an
'OK' button with the callback routine Kill_ok_cbproc

4.4.3. Algorithm of Kill_ok_cbproc

This procedure only consists of a function call.

       exit the application

4.4.4. Algorithm of Keys_cbproc

The function Keys_cbproc creates a dialog window as presented in figure 4.3. . It receives a value indicating the type of search key : 1 for 'name', 2 for 'first name', 3 for 'specialization'.

The ToggleButtons and the label 'Consult : Name' are not implemented. The three other widgets are those that were selected in section 4.3.3.2.1. . However, we need an additional Label widget to place before the input text field.

Figure 4.11. presents the tree of widgets created and their callback procedures.

It is followed by the algorithm.
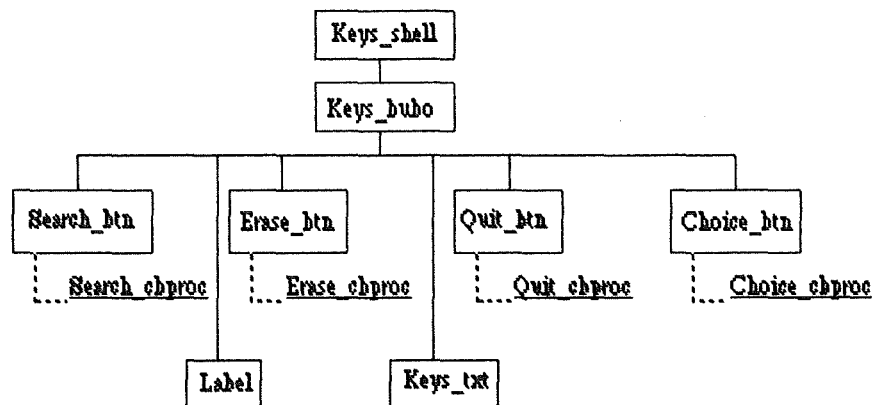
figure 4.11. widgets of a dialog window

create a shell : Keys_shell

create a bulletin board (Keys_bubo) in Keys_shell

create a push button (Search_btn) in Keys_bubo and declare its associated callback routine Search_cbproc, to be called with an argument value corresponding to the type of search key

create a push button (Erase_btn) in Keys_bubo and declare its callback routine : Erase_cbproc, to be called with the argument value 1

create a push button (Quit_btn) in Keys_bubo and declare its associated callback routine : Quit_cbproc, to be called with the argument Keys_shell

create toggle buttons (Choice_btn) for the complete information/cut information choice in

Keys_bubo and declare its callback routine Choice_cbproc

create the input text field (Keys_txt) in Keys_bubo

if the argument value passed to Keys_cbproc = 1

then create the input label (Label) in Keys_bubo, with the value "Name"

if the argument value passed to Keys_cbproc = 2

then create the input label (Label) in Keys_bubo, with the value "First Name"

if the argument value passed to Keys_cbproc = 3

then create the input label (Label) in Keys_bubo, with the value "Specialization"

4.4.5. Algorithm of Search_cbproc

The function Search_cbproc takes as argument the type of search key and searches for all persons whose information match the search key.

The same remark as in section 4.4.2. applies here about the error window : we use a convenience Dialog (XmCreateMessageDialog) based on XmMessageBox (see section 3.1.4.8.).

get the value of the input text field Keys_txt (that is to say the search key) in the character
array Search_data

open Phonefile

read first line

while not EOF

read the following 8 lines

if   (the search key is a name and line 1 equals to Search_data)

or (the search key is a first name and line 2 equals to Search_data)

or (the search key is a specialization and line 9 equals to Search_data)

then copy the nine lines in the table Info

read next line

close Phonefile

if no data matching Search_data has been found in Phonefile

then create a message dialog box Error with the label 'Does not exist' and an 'OK'
button

else unmanage Keys_shell

call Keys_info_proc

4.4.6. Algorithm of Keys_info_proc

The function Keys_info_proc creates a complete or cut information window.

A BulletinBoard, PushButtons, and text fields are used as foreseen for an information window.

Label widgets have been added to the list of widgets, as in Keys_cbproc.

Figure 4. 12. presents the tree of widgets created and their callback procedures.

It is followed by the algorithm.
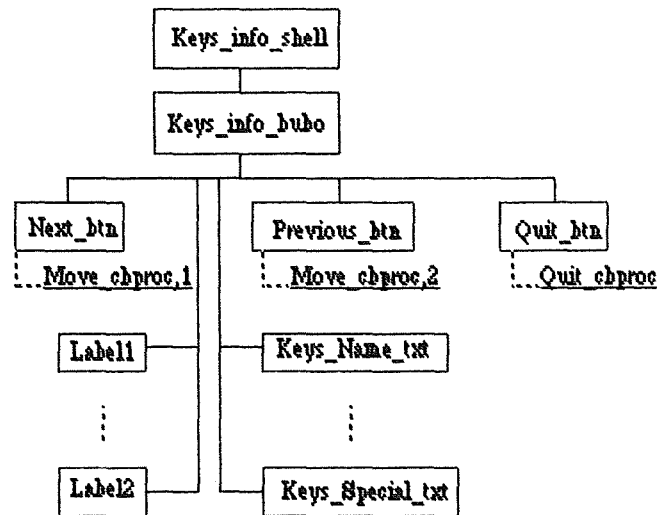
figure 4. 12. widgets of an information window

create a shell : Keys_info_shell

create a bulletin board (Keys_info_bubo) in Keys_info_shell

create a push button (Next_btn) in Keys_info_bubo and declare its callback routine
Move_cbproc to be called with the argument value 1

create a push button (Previous_btn) in Keys_info_bubo and declare its callback routine
Move_cbproc to be called with the argument value 2

create a push button (Quit_btn) in Keys_info_bubo and declare its associated callback
routine Quit_cbproc to be called with the argument Keys_info_shell

if complete_info is true

    then create nine labels (Label1 to Label9) in Keys_info_bubo corresponding to the
nine fields of data already described

        create nine input text fields (Keys_Name_txt to Keys_Special_txt) in
Keys_info_bubo corresponding to the nine fields of data already described

        set the value of these nine input text fields to the first nine lines of the table Info
created in Search_cbproc

    else create four labels (Label1 to Label4) in Keys_info_bubo corresponding to the
four fields of data already described

        create four input text fields (Keys_Name_txt to Keys_Phone_txt) in
Keys_info_bubo corresponding to the four fields of data already described

set the value of these four input text fields to the first four lines of the table Info
createdinSearch_cbproc

4.4.7. Algorithm of Move_cbproc

The function Move_cbproc displays the complete or cut information of another person in Keys_bubo according to the button that called it.

if the value received as argument = 1 and the index of the table Info is less than (size of the
table) - 1
then increment the index of the table
if the value received as argument = 2 and the index of the table is greater than 0
then decrement the index of the table
if the index has been modified
if complete_info is true
then set the value of the nine input text fields Keys_Name_txt to Keys_Special_txt
to the nine values corresponding to the index in the table Info
else set the value of the four input text fields Keys_Name_txt to Keys_Phone_txt
to the four values corresponding to the index in the table Info

4.4.8. Algorithm of Quit_cbproc

The function Quit_cbproc creates a confirmation window used to confirm or cancel a request to kill a window of the application, except the main window.

As foreseen in section 4.3.3.4., the confirmation window is based on a MessageBox. However, we use a convenience Dialog based on that widget (see section 3.1.4.8.).

create a question dialog box with a label 'Do you want to quit ?' a Cancel button and an
'OK' button with the callback routine Quit_ok_cbproc, called with the window to
kill as argument

4.4.9. Algorithm of Quit_ok_cbproc

This procedure only consists of a function call.

destroy the window passed as argument

4.4.10. Algorithm of Erase_cbproc

The function Erase_cbproc can be called from the dialog window created by Keys_cbproc or from the update window created by Update_cbproc.

    if the value received as argument = 1
      then set Keys_txt to blank
    if the value received as argument = 2
        then set the nine input text fields of Update_bubo to blank

4.4.11. Algorithm of Choice_cbproc

    if the Complete info option has been selected
      then set Complete_info to true
    if Cut info option has been selected
      then set Complete_info to false

4.4.12. Algorithm of Allinfo_proc

The function Allinfo_proc copies the file Phonefile into the table Info and calls Keys_info_proc to make its content displayed.

We use here a convenience Dialog (XmCreateMessageDialog) based on XmMessageBox to create an error window.

    openPhonefile
    read first line
    while not EOF
      read the following 8 lines
      copy the nine lines in the table Info
      read next line
    closePhonefile
    if Phonefile is empty
      then create a message dialog box Check_msg with the label 'The phone book is empty'
        and with an 'OK' button
      else call Keys_info_proc

4.4.13. Algorithm of Activities_proc

The function Activities_proc copies all activities found in the line 9 for each person into the table Activities and calls Special_info_proc to make them displayed.

The same remark as in section 4.4.12. applies here about the error window.

> openPhonefile
>
> read first line
>
> while not EOF
>
> > read the following 8 lines
> >
> > extract the specialization(s) mentioned in the line 9 and copy them into the table
> > > Activities
> >
> > read next line
>
> closePhonefile
>
> if the table is empty
>
> > then create a message dialog box Check_msg with the label 'No specialization' and an
> > > 'OK' button
> >
> > else call Special_info_proc

4.4.14. Algorithm of Special_info_proc

The function Special_info_proc creates an information window as presented in figure 4.6. .

In this procedure, we would use a ScrolledWindow instead of a Text provided with a ScrollBar.
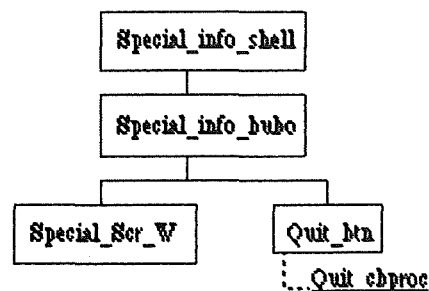
figure 4.13. widgets of an information window for the activities

> create a shell : Special_info_shell
>
> create a bulletin board (Special_info_bubo) in Special_info_shell

create a push button (Quit_btn) in Special_info_bubo and declare its callback routine :

Quit_cbproc, to be called with the argument Special_info_shell

create a scrolled window (Special_Scr_W) in Special_info_bubo

set the value of the text Special_info_txt to the table Activities created in Activities_proc

4.4.15. Algorithm of Update_cbproc

The function Update_cbproc creates an update window as presented in figure 4.8. .

Label widgets (placed before the text fields) are used in addition to those defined in section 4.3.3.3. .
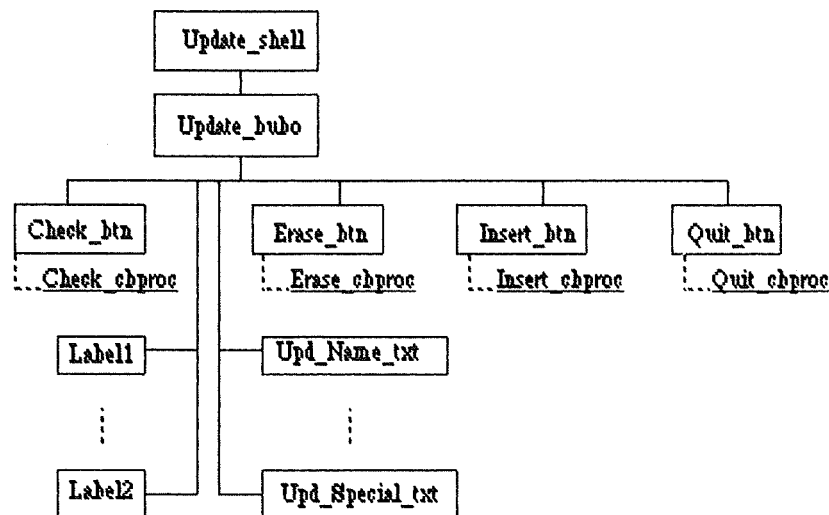
figure 4.14. widgets of an update window

create a shell : Update_shell

create a bulletin board (Update_bubo) in Update_shell

create a push button (Check_btn) in Update_bubo and declare its callback routine
Check_cbproc

create a push button (Erase_btn) in Update_bubo and declare its callback routine
Erase_cbproc to be called with the argument value 2

create a push button (Quit_btn) in Update_bubo and declare its callback routine
Quit_cbproc to be called with the argument Update_shell

if the value received as argument by Update_cbproc = 1

then create a push (Insert_btn) button in Update_bubo and declare its callback routine
Insert_cbproc

if the value received as argument by Update_cbproc = 2

> then create a push button (Modify_btn) in Update_bubo and declare its callback routine Modify_cbproc

if the value received as argument by Update_cbproc = 3

> then create a push button (Delete_btn) in Update_bubo and declare its callback routine Delete_cbproc

create nine labels (Label1 to Label9) in Update_bubo corresponding to the nine fields of data already described

create nine input text fields (Upd_Name_txt to Upd_Special_txt) in Update_bubo corresponding to the nine fields of data already described

4.4.16. Algorithm of Check_cbproc

The function Check_cbproc tests if the person identified by the name and first name given in the update window already exists in Phonefile or not.

Once again, a MessageBox is created via a convenience Dialog.

> get the value of Upd_Name_txt and Upd_F_Name_txt
> if these values exist consecutively in Phonefile
>> then set the value of a string to 'Exists'
> if these values do not exist consecutively in Phonefile
>> then set the value of a string to 'Does not exist'
> create a message dialog box with a label which value is set to the preceding string and with an 'OK' button

4.4.17. Algorithm of Insert_cbproc

The function Insert_cbproc inserts the values of the nine input text fields of the update window into Phonefile, unless the informatuion of the person identified by the name and first name is already in Phonefile.

> get the value of the nine input text fields Upd_Name_txt to Upd_F_Name_txt
> if the values of Upd_Name_txt to Upd_F_Name_txt exist consecutively in Phonefile
>> then set the value of a string to 'Exists'
>>> create a message dialog box with a label which value is set to that preceding string and with an 'OK' button
> if the values of Upd_Name_txt to Upd_F_Name_txt do not exist consecutively in Phonefile
>> then insert the values of the nine input text fields into the file Phonefile

<u>4.4.18. Algorithm of Modify_cbproc</u>

The name and algorithm of the function Modify_cbproc are meaningful enough. Moreover, the modification becomes an insertion if the pair (name , first name) does not exist in Phonefile.

> get the value of the nine input text fields Upd_Name_txt to Upd_F_Name_txt
> insert the values of these nine input text fields into the file Phonefile

<u>4.4.19. Algorithm of Delete_cbproc</u>

The function Delete_cbproc is symmetrical to the function Insert_cbproc.

> get the value of the two input text fields Upd_Name_txt and Upd_F_Name_txt
> if the values of Upd_Name_txt to Upd_F_Name_txt do not exist consecutively in Phonefile
> > then set the value of a string to 'Does not exist'
> > > create a message dialog box with a label which value is set to that preceding string
> > > and an 'OK' button
> if the values of Upd_Name_txt to Upd_F_Name_txt exist consecutively in Phonefile
> > then delete the nine corresponding lines in Phonefile

4.5. The XView version of the application

This section presents and comments the algorithms of the internal phone book application written with XView.

However, most of the algorithms that follow have been deduced from a few ones, as the XView application is not fully implemented.

The text file of the application begins with the declarations of the include files referring to the packages used.

Many objects are then declared as global variables as do other data structures.

All normal and callback procedures are declared at a global level, too.

The main function of that C program is then defined.

<u>4.5.1. Algorithm of main</u>

The function main creates the main window of the application, its menus and its submenus, opens the application database and gives the control to the Notifier. The attribute XV_KEY_DATA is used

in an object to associate data with that object. The notify procedure of that object automatically receives the handle to the object and the associated data can be retrieved by a call to xv_get.

As foreseen in section 4.3.3.1., we use a frame, a panel, panel buttons and menus to create the main window of the application.
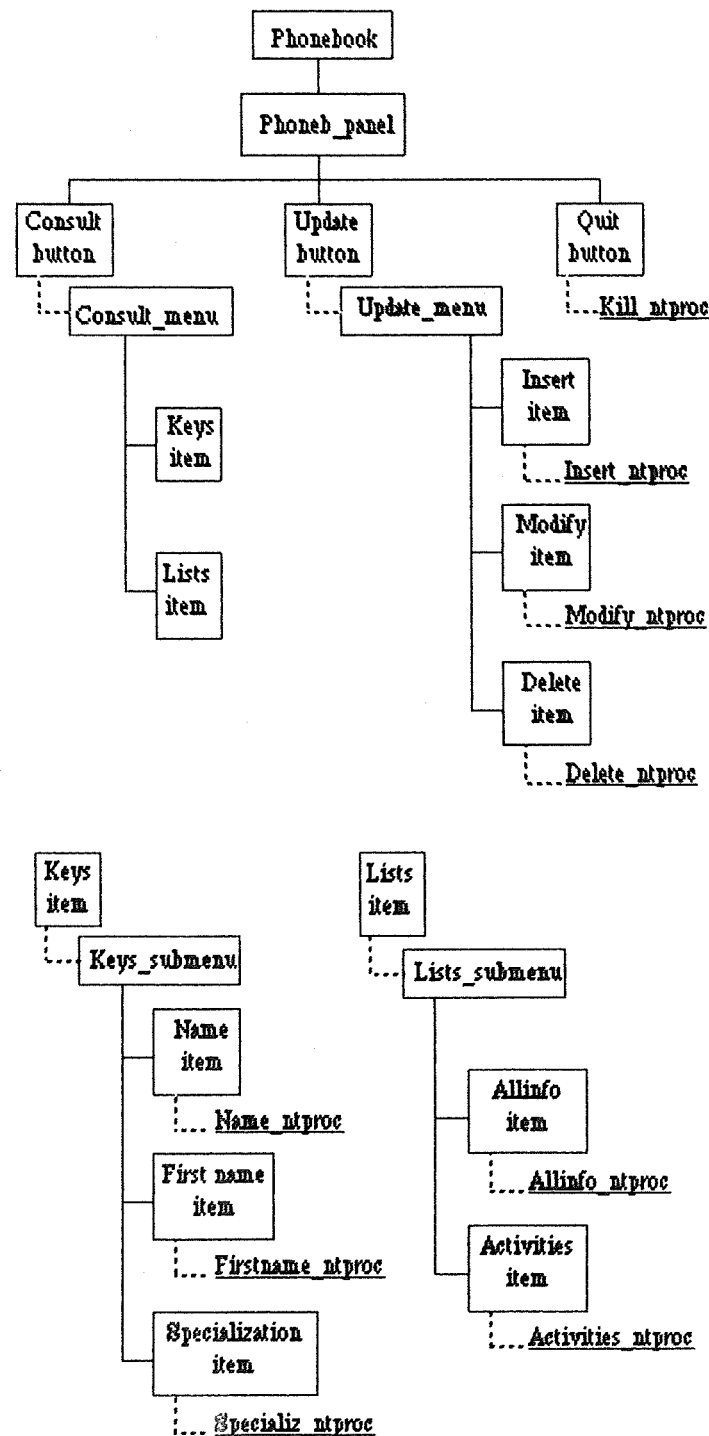
figure 4.15. the objects of the application's main window

call the initialization function

create a base frame : Phonebook

create a submenu (Keys_submenu) with three items and declare the procedures they call :
'Name' and Name_proc, 'First name' and Firstname_proc, 'Specialization' and
Specializ_proc

create a submenu (Lists_submenu) with two items and declare the procedures they call :
'Allinfo' and Allinfo_proc, 'Activities' and Activities_proc.

create a menu (Consult_menu) with two items and declare their respective pullrignt
menus: 'Keys' and Keys_submenu , 'Lists' and Lists_submenu

create a menu (Update_menu) with three items and declare the procedures they call :
'Insert' and Insert_proc, 'Modify' and Modify_proc, 'Delete' and Delete_proc

create a panel (Phoneb_panel) within Phonebook

create a button with the label 'Consult' in Phoneb_panel and declare its menu :
Consult_menu

create a button with the label 'Update' in Phoneb_panel and declare its menu :
Update_menu

create a button with the label 'Quit' in Phoneb_panel and declare its callback routine :
Kill_ntproc

call a function (Init) that opens the file Phonefile standing for the database or creating it if
it does not exist

call the function mainloop .

4.5.2. Algorithm of Name_proc

call the function Keys_proc with the argument value 1

4.5.3. Algorithm of Firstname_proc

call the function Keys_proc with the argument value 2

4.5.4. Algorithm of Specializ_proc

call the function Keys_proc with the argument value 3

4.5.5. Algorithm of Keys_proc

The function keysproc creates a dialog window as presented in figure 4.3. . It receives a value
indicating the type of search key : 1 for 'Name', 2 for 'First name', 3 for 'Specialization'.

All objects precised in section 4.3.3.2.1. are implemented, except for the exclusive choice.
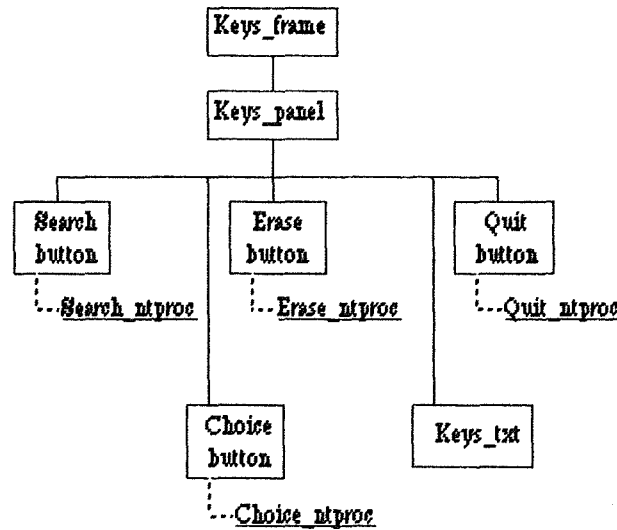
figure 4.16. the objects of a dialog window

create a frame Keys_frame

create a panel (Keys_panel) in Keys_frame

create a button with the label 'Search' in Keys_panel, declare its notify procedure Search_ntproc and set the attribute XV_KEY_DATA of that button to the value indicating the type of search key

create a button with the label 'Erase' in Keys_panel, declare its notify procedure Erase_ntproc and set the attribute XV_KEY_DATA of that button to 1

create a button with the label 'Quit' in Keys_panel, declare its notify procedure Quit_ntproc and set the attribute XV_KEY_DATA of that button to the value Keys_frame

create the complete information/cut information choice in Keys_panel and declare its callbackroutine Choice_ntproc

create an input text (Keys_txt) in Keys_panel

if the value received as argument by Keys_proc = 1 set the value of the text label to 'Name'

if the value received as argument by Keys_proc = 2 set the value of the text label to 'First name'

if the value received as argument by Keys_proc = 3 set the value of the text label to 'Specialization'

4.5.6. Algorithm of Search_ntproc

The function Search_ntproc searches for all persons whose information match the search key.

get in the character array Search_data the value of the input text field Keys_txt (that is to say the search key)

get in Search_key the value of the attribute XV_KEY_DATA of the button that called Search_cbproc

open Phonefile

read first line

while not EOF

read the following 8 lines

    if   (Search_key = 1 and line 1 equals to Search_data)

        or (Search_key = 2 and line 2 equals to Search_data)

        or (Search_key = 3 and line 9 equals to Search_data)

    then copy the nine lines in the table Info

    read next line

closePhonefile

if no data matching Search_data has been found in Phonefile

    then create a notice Error with the label 'Does not exist' and an 'OK' button

    else destroy Keys_frame

        call Keys_info_proc

4.5.7. Algorithm of Keys_info_proc

The function Keys_info_proc creates a complete or cut information window.

The objects created in an information window are shown in figure 4.17.
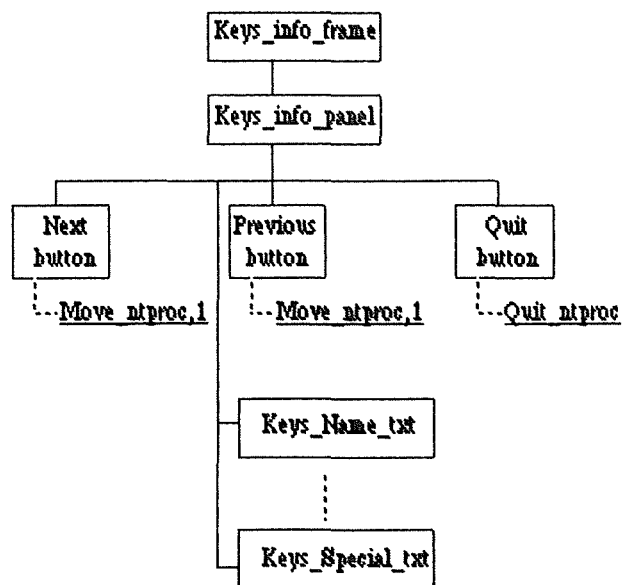
figure 4.17. the objects of an information window

create a frame : Keys_info_frame

create a panel (Keys_info_panel) in Keys_info_frame

create a button (with the label 'Next') in Keys_info_panel, declare its notify procedure
Move_ntproc and set the attribute XV_KEY_DATA of that button to 1

create a button (with the label 'Previous') in Keys_info_panel, declare its notify procedure
Move_ntproc and set the attribute XV_KEY_DATA of that button to 2

create a button (with the label 'Quit') in Keys_info_panel, declare its notify procedure
Quit_ntproc and set the attribute XV_KEY_DATA to the value Keys_info_frame

if complete_info is true

then create nine input text fields (Keys_Name_txt to Keys_Special_txt) in
Keys_info_panel corresponding to the nine fields of data already
described

set the labels of these nine input text fields to the values 'Name' to
'Specialization'

set the value of these nine input text fields to the first nine lines of the table Info
created in Search_ntproc

else create four input text fields (Keys_Name_txt to Keys_Phone_txt) in
Keys_info_panel corresponding to the four fields of data already
described

set the labels of these four input text fields to the values 'Name' to 'Phone
number'

set the value of these four input text fields to the first four lines of the table Info
created in Search_cbproc

4.5.8. Algorithm of Move_ntproc

The function Move_ntproc displays the complete or cut information of another person in
Keys_info_panel according to the button that called it : the 'Next' or the 'Previous' button

get in Direction the value of the attribute XV_KEY_DATA of the button that called
Move_ntprocif Direction = 1 and the index of the table Info is less than (size of the
table) - 1

then increment the index of the table

if Direction = 2 and the index of the table is greater than 0

then decrement the index of the table

if the index has been modified

then if Complete_info is true

then set the value of the nine input text fields Keys_Name_txt to Keys_Special_txt to the nine values corresponding to the index in the table Info

else set the value of the four input text fields Keys_Name_txt to Keys_Phone_tx to the four values corresponding to the index in the table Info

4.5.9. Algorithm of Erase_ntproc

The function Erase_ntproc can be called from the dialog window created by Keys_proc or from the update window created by Update_proc.

get in Window the value of the attribute XV_KEY_DATA of the button that called Erase_ntproc

if Window = 1

then set Keys_txt to blank

if Window = 2

then set the nine input text fields of Update_panel to blank

4.5.10. Algorithm of Quit_ntproc

The function Quit_ntproc creates a confirmation window used to confirm or cancel a request to kill a window of the application, except the main window.

create a notice with a label 'Do you want to quit ?' a 'Cancel' button and an 'OK' button

if the 'OK' button is selected

then get in Window the value of the attribute XV_KEY_DATA of the button that called Quit_ntproc

destroy Window

4.5.11. Algorithm of Choice_ntproc

if the Complete info option has been selected

then set Complete_info to true

if Cut info option has been selected

then set Complete_info to false

4.5.12. Algorithm of Allinfo_proc

The function Allinfo_proc copies the file Phonefile into the table Info and calls Keys_info_proc to make its content displayed.

```
openPhonefile
read first line
while not EOF
        read the following 8 lines
        copy the nine lines in the table Info
        read next line
closePhonefile
if Phonefile is empty
    then create a notice with the label 'The phone book is empty' and an 'OK' button
    else call Keys_info_proc
```

4.5.13. Algorithm of Activities_proc

The function Activities_proc copies all activities found in the line 9 for each person into the table Activities and calls Special_info_proc to make them displayed.

```
openPhonefile
read first line
while not EOF
        read the following 8 lines
        extract the specialization(s) mentioned in the line 9 and copy them into the table
                Activities
        read next line
closePhonefile
if the table is empty
    then create a notice with the label 'No specialization' and an 'OK' button
    else call Special_info_proc
```

4.5.14. Algorithm of Special_info_proc

The function Special_info_proc creates an information window as presented in figure 4.6.
The tree of objects of that window is given in figure 4.18.
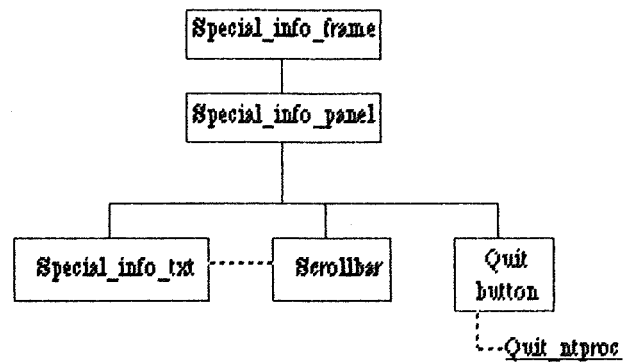
figure 4.18. the objects of an information window for the activities

create a frame Special_info_frame

create a panel (Special_info_panel) in Special_info_frame

create a button with the label 'Quit' in Special_info_panel, declare its notify procedure

        Quit_ntproc and set the attribute XV_KEY_DATA to the value Special_info_frame

create a text subwindow Special_info_txt in Special_info_panel

create a scrollbar attached to Special_info_txt

set the value of Special_info_txt to the value of the table Activities created in

        Activities_proc

4.5.15. Algorithm of Insert_proc

call Update_proc with the argument value 1

4.5.16. Algorithm of Modify_proc

call Update_proc with the argument value 2

4.5.17. Algorithm of Delete_proc

call Update_proc with the argument value 3

4.5.18. Algorithm of Update_proc

The function Update_proc creates an Update window as presented in figure 4.8. .
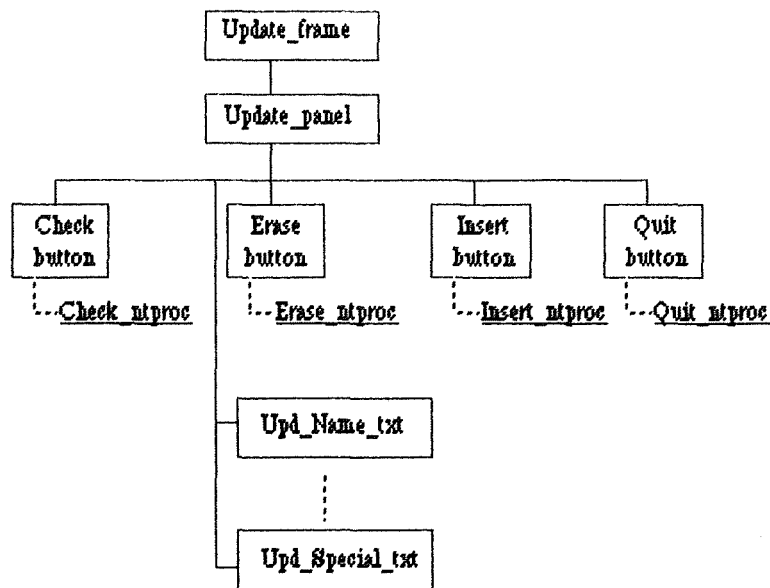
figure 4.19. the objects of an update window

create a frame Update_frame

create a panel Update_panel

create a button with the label 'Check' in Update_panel and declare its notify procedure Check_ntproc

create a button with the label 'Erase' in Update_panel, declare its notify procedure Erase_ntproc and set the attribute XV_KEY_DATA of that button to the value 2

create a button with the label 'Quit' in Update_panel, declare its notify procedure Quit_ntproc and set the attribute XV_KEY_DATA of that button to the value Update_frame

if the value received as argument by Update_ntproc = 1

then create a button with the label 'Insert' in Update_panel and declare its notify procedure Insert_ntproc

if the value received as argument by Update_ntproc = 2

then create a button with the label 'Modify' in Update_panel and declare its notify procedure Modify_ntproc

if the value received as argument by Update_ntproc = 3

then create a button with the label 'Delete' in Update_panel and declare its notify procedure Delete_ntproc

create nine input text fields (Upd_Name_txt to Upd_Special_txt) in Update_panel corresponding to the nine fields of data already described

set the labels of these nine input text fields to the values 'Name' to 'Specialization'

4.5.19. Algorithm of Check_ntproc

The function Check_ntproc tests if the person identified by the name and first name given in the update window already exists in Phonefile or not.

> get the value of Upd_Name_txt and Upd_F_Name_txt
> if these values exist consecutively in Phonefile
> > then set the value of a string to 'Exists'
> if these values do not exist consecutively in Phonefile
> > then set the value of a string to 'Does not exist'
> create a notice with a label which value is set to the preceding string and with an 'OK'
> > button

4.5.20. Algorithm of Insert_ntproc

The function Insert_ntproc inserts the values of the nine input text fields of the update window into Phonefile, unless the information of the person identified by the name and first name is already in Phonefile.

> get the value of the nine input text fields Upd_Name_txt to Upd_F_Name_txt
> if the values of Upd_Name_txt to Upd_F_Name_txt exist consecutively in Phonefile
> > then set the value of a string to 'Exists'
> > > create a notice with a label which value is set to that preceding string and an 'OK'
> > > button
> if the values of Upd_Name_txt to Upd_F_Name_txt do not exist consecutively in Phonefile
> > then insert the values of the nine input text fields into the file Phonefile

4.5.21. Algorithm of Modify_ntproc

The name and algorithm of the function Modify_ntproc are meaningful enough. Moreover, the modification becomes an insertion if the pair (name , first name) does not exist in Phonefile.

> get the value of the nine input text fields Upd_Name_txt to Upd_F_Name_txt
> insert the values of these nine input text fields into the file Phonefile

4.5.22. Algorithm of Delete_ntproc

The function Delete_ntproc is symmetrical to the function Insert_ntproc.

get the value of the two input text fields Upd_Name_txt and Upd_F_Name_txt

if the values of Upd_Name_txt to Upd_F_Name_txt do not exist consecutively in Phonefile

    then set the value of a string to 'Does not exist'

        create a notice with a label which value is set to that preceding string and an 'OK' button

if the values of Upd_Name_txt to Upd_F_Name_txt exist consecutively in Phonefile

    then delete the nine corresponding lines in Phonefile

4.5.23. Algorithm of Kill_ntproc

The function Kill_ntproc creates a confirmation window used to confirm or cancel a request to quit the application.

create a notice with a label 'Do you want to quit ?' a 'Cancel' button and an 'OK' button

if the 'OK' button is selected

    then exit the application

4.6. Comparison and generalization of the algorithms of the two versions

This section only generalizes those algorithms that deal with graphical objects, with the aim of producing toolkit-independent algorithms to be instanciated to a Motif or an XView application. The generic algorithms presented here will sometimes look like an XView or a Motif one, but there is often no other way of expressing them.

We shall use three specific terms to write generic algorithms : 'create', 'attach', and 'refer'.

The term 'create' represents the creation of an object or a group of objects. It includes the management or the realization of these objects when necessary. That term thus stands for xv_create(), for XmCreate *Widget ()* , for XtCreateManagedWidget(), or for XtCreateWidget() followed by XtManageChild().

The term 'attach' represents a link between objects. This link can be a parental link, or a functional link. A parental link exists for example between a button and the container object that is its parent and contains it. A functional link can be created between a button or an item, and a menu by means of a resource or an attribute. A scrollbar can be attached to a window, too.

The term 'refer' introduces the callback procedure of an object and possibly gives the value with which that procedure is called. That value is an argument value for Motif and an XV_KEY_DATA value for XView.

Generic algorithms written with these three 'functions' will always need to be transformed, completed or reduced by the programmer in order to be instanciated to one of the two toolkits, according to the comments given for each of them.

4.6.1. The main function

The initialization and the creation of the main window (a toplevel shell) are performed by Motif in one function call while XView needs two statements.

The menus and submenus of the main window are created in the Motif program in a top-down way: from the menubar to the push buttons of the submenus. The components of these menus conform in that way to the normal order of creation of widgets : a widget is created, then another one is created as a child of the first widget, considered as the parent widget.
However, a particularity of Motif menu systems has to be shown. As Keys_casbtn gives access to the submenu Keys (containing Name_pushbthn, F_name_pushbtn and Special_pushbtn), one could think that Keys_subpane is a child widget of Keys_casbtn. In fact, Keys_subpane is a child widget of Consult_pane as do Keys_casbtn and a functional link exists between Keys_casbtn and Keys_subpane.
The same remark can be made for other components (see figure 4.10.).

XView reverses the order of creation of those objects. This is due to the fact that submenus are referenced by attributes of the menus. Thus submenus must be created before menus.

Another difference lies in the fact that Motif places the three top-level buttons of the menu in a specific menu component(MenuBar) while XView places them in a general object (Phoneb_panel) able to contain many kinds of panel items. (see figure. 2.7.).

Such considerations lead us to give the following generic algorithm of the function main :

        create the submenu Keys
              refer the procedure Keys with the appropriate value
        create the submenu Lists
              refer the procedure All_info
              refer the procedure Activities

```
create the menu Consult_menu
        attach the submenu Keys to the menu Consult_menu
        attach the submenu Lists to the menu Consult_menu
create the menu Update_menu
        refer the procedure Update_proc with the appropriate value
create the button Consult_btn
        attach the menu Consult_menu to the button Consult_btn
create the button Update_btn
        attach the menu Update_menu to the button Update_btn
create the button Quit_btn
        refer the procedure Quit_proc with the appropriate value
create the board Main
        attach the button Consult_btn to the board Main
        attach the button Update_btn to the board Main
        attach the button Quit_btn to the board Main
create the window Main
        attach the board Main to the window Main
```

However, that algorithm is tightly bound to the XView algorithm because of the order of creation of the objects.

In some cases, two specific algorithms (for XView and for Motif) of the same procedure differ too much to allow the definition of a generic algorithm. Such a situation occurs in our application with the main function, where the main window is created. The menus and submenus attached to that window are created in different ways, as explained above.

In such a case, instead of giving the programmer the order of creation of the objects, we give him a representation of the final set of widgets with their relationships, as in figure 4.20. . From that figure, he then decides which objects and which functions he will use.
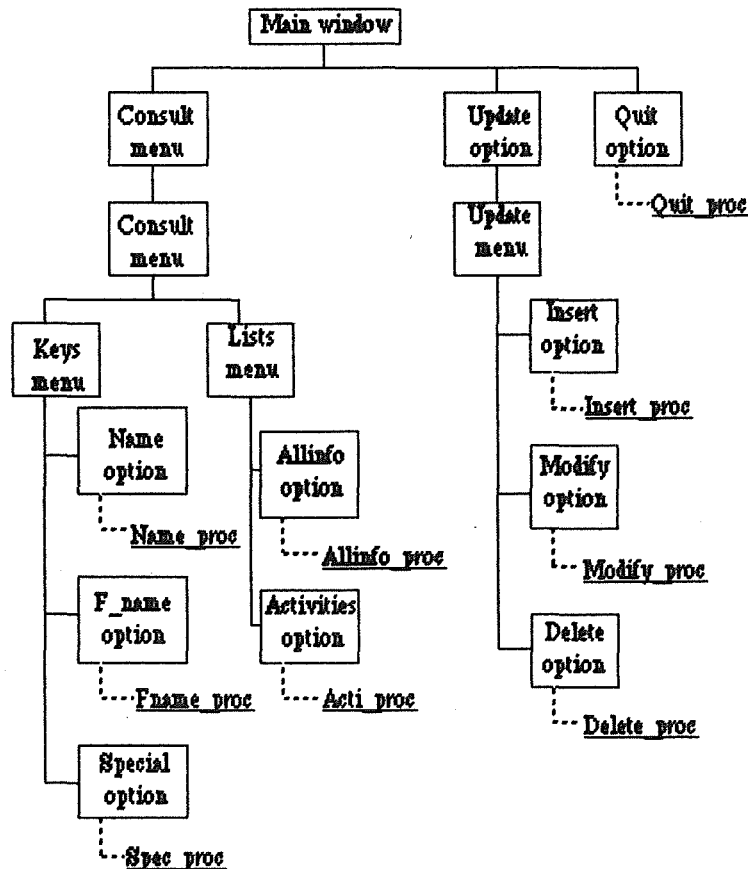
figure 4.20. generic representation of the main window

The use of such generic representations can be generalized to all algorithms, even if a simple generic algorithm would be sufficient.

4.6.2. The Keys submenu

As XView allows no data to be directly associated with a notify procedure, another mechanism must be used to transmit a value to a procedure : the XV_KEY_DATA attribute. However, this attribute can not be used with the menu items because the latter are themselves defined like attributes of the menu. So, whereas Motif calls a general procedure (Keys_cbproc) with an argument value indicating the type of search key, XView must go through specific procedures (Name_proc, Firstname_proc and Specializ_proc) to transmit that key value.

Thus the difference between the two implementations of the submenu Keys is the set of three intermediary specific procedures needed by XView, but the procedures Keys_cbproc for Motif and Keys_proc for XView themselves are very similar. The former creates a bulletin board in a shell and the latter creates the equivalent objects : a panel in a frame. The bulletin board and the panel both accept buttons and text fields. For the Motif toolkit, the labels placed before the input text fields and these text fields are diferent widgets. But for XView, such a label is only an attribute of a text field.

The generic algorithm for the creation of the dialog window will be :

create a window

create a support for other objects

create a Search button

    refer the Search procedure with the appropriate value

create an Erase button

    refer the Erase procedure with the appropriate value

create a Quit button

    refer the Quit procedure with the appropriate value

create a Complete/Cut information choice

    refer the Choice procedure with the appropriate value

create an input textfield

if the search key is a name

    then set the label of the text field to the value 'Name'

if the search key is a first name

    then set the label of the text field to the value 'First name'

if the search key is a specialization

    then set the label of the text field to the value 'Specialization'

4.6.3. The Search procedure

The procedures Search_cbproc and Search_ntproc differs in three points :

1. The procedure Search_ntproc needs to get the value of the attribute XV_KEY_DATA of the button that called this procedure, while the procedure Search_cbproc automatically receives it as argument.

2. If no desired data has been found in Phonefile, Search_cbproc creates a message box, while Search_ntproc creates a notice.

3. If some data has been found, a shell is destroyed in the Motif version, and a frame is destroyed in the XView version.

4.6.4. The Keys_info_proc procedure

Once again, we find in the two versions of the same procedure some similarities. On one side, a bulletin board is created in a shell and on the other side, a panel is created in a frame. A few buttons are created and declare their callback or notify procedures. The neccesary values passed to these

procedures are either argument values (Motif) or values for the attribute XV_KEY_DATA (XView). The difference between the text fields of the two toolkits still appears in these procedures. The generic algorithm is :

create a window

create a support for other objects

create a Next button

refer the Next procedure with the appropriate value

create a Previous button

refer the Previous procedure with the appropriate value

create a Quit button

refer the Quit procedure with the appropriate value

if Complete_info is true

then create the nine text fields with the corresponding labels

set the value of the text fields to the value of the table Info created in the

procedure Search

else create the four text fields with the corresponding labels

set the value of the text fields to the value of the table Info created in the

procedure Search

4.6.5. The Move and Erase procedures

The only difference between the two versions of the procedures Move and Erase is the use of the attribute XV_KEY_DATA.

4.6.6. The Quit procedure

The difference between Quit_cbproc and Quit_ntproc is the creation of a question dialog box in the former and the creation of a notice in the latter, in addition to the use of the attribute XV_KEY_DATA.

Moreover, XView does not need to call another function to kill a window when this request has been confirmed by the user.

4.6.7. The Kill procedure

The same remarks can be made for the procedure Kill than for the procedure Quit, except for the fact that no attribute XV_KEY_DATA is used because that procedure only deals with the main window Phonebook.

4.6.8. The Allinfo_proc and Activities_proc procedures

The two procedures Allinfo_proc and the two procedures Activities_proc differ in one point : the creation of a message dialog box or of a notice.

4.6.9. The Special_info_proc procedure

The procedures Special_info_proc presents the typical differences : a bulletin board is created as child widget of a shell on one side, a panel is created as a child object of a frame on the other side. The use of the attribute XV_KEY_DATA is still to observe. A particularity must be mentioned : Motif offers a scrolled text widget but the equivalent XView object does not exist. It must be created by attaching a scrollbar to a text object. Keeping that fact in mind, the generic algorithm is :

create a window
create a support for other objects
create a Quit button
    refer the Quit procedure with the appropriate value
create a scrolled text
set the value of the text to the value of the table Activities created in the procedure
    Activities_proc

4.6.10. The Update menu

The menu Update calls for the same remarks than the submenu Keys. Because the XView menu items 'Insert' 'Modify' and 'Delete' declare notify procedures that can not directly receive arguments, three intermediary specific routines are used to acces the procedure Update with the correct value of argument. The generic algorithm of the procedure Update is :

create a window
create a support for other objects
create a Check button
    refer the Check procedure with the appropriate value
create an Erase button
    refer the Erase procedure with the appropriate value
create a Quit button
    refer the Quit procedure with the appropriate value
if the value received as argument = 1
    then create an Insert button

refer the Insert procedure
if the value received as argument = 2
then create a Modify button
refer the Modify procedure
if the value received as argument = 3
then create a Delete button
refer the Delete procedure
create the nine text fields with the corresponding labels

4.6.11. The Check, Insert, Modify and Delete procedures

The two versions of the procedures Check, Insert, Modify and Delete only differ in one point : one creates a message dialog box and the other creates a notice.

Chapter 5
Comparison of XView and Motif

The fifth chapter goes on one step beyond the mere presentation of the two toolkits. Its aim is to draw a parallel between most of their elements. However, even if a widget is said to be the equivalent of a package, they always differ in some way.

The structure of this chapter is partially based on the seven groups we presented in section 4.1. : container components, text capabilities, graphics capabilities, menus, scrolling capabilities, commands and choices, and informations. We also discuss miscellaneous components and functions.

5.1. Container components

In a few words, we can say that Motif offers a greater choice of components than XView. However, this choice can sometimes confuse the programmer. The two main components are the XView frame and panel and the Motif TopLevelShell and BulletinBoard.

With the expression 'container component', we mean the first or second object created at the creation of a window, in order to enclose and manage children objects.
The first object created to obtain a window is a frame (XView) or a widget of the category Shell (Motif).
For the user, a frame is a visible object that can be manipulated, whereas a shell is an invisible widget. From the programmer point of view, a frame is also different from a shell. XView only has one type of frame while Motif offers ApplicationShell and TopLevelShell. ApplicationShell is created by the initialization function and TopLevelShell is used for subsequent toplevel shells. The classes XmMenuShell and XmDialogShell should be related to other XView packages.
Thus an XView programmer has no choice whereas a programmer new to Motif could hesitate between all types of shells, including those that are not mentioned here.

Frames and shells themselves often need another container component that will typically accept command objects such as buttons, choices, ...

The XView package Panel defines a container (panel) and all the panel items. These panel items must be explicitly positioned in the panel.

Motif proposes many widgets. The principal ones are listed below :

1. XmFrame accepts a single child
2. XmMainWindow can be used as the primary application's window. It is provided with optional elements : a menu bar, a command window, a work region and scrollbars.
3. XmPanedWindow is used for vertically tiled children
4. XmBulletinBoard does not force positioning of children
5. XmRowColumn specify the layout of its children in a vertically or horizontally fashion. It is also the basis of menu systems.
6. XmForm is an interesting widget because of its definition of spatial relationships

As far as the user is concerned, two Motif widgets present interesting properties : the PanedWindow and Form widgets. The different panes of a PanedWindow can be resized, as explained in section 3.1.3.8. When resized, a Form maintains the special relationships between its children (see section 3.1.4.3.).

No XView object has such possibilities.

For the programmer, the use of a panel is quite easy : it can only be a child of a frame. When the programmer has to deal with Motif widgets, some problems may arise. Some combinations of widgets are not possible or do not produce the expected effect.

5.2. Text capabilities

Motif and XView are quite similar as far as text capabilities are concerned : they both offer a multiline and a single-line text components.

Few XView or Motif components are directly concerned with texts. XView defines the package Textsw that corresponds to the XmText widget. They are both provided with basic editing capabilities and a vertical scrollbar. The XmText widget has a supplementary horizontal scrollbar while the XView text subwindow offers a few other menu options. Both XmText and Textsw are concerned with multiline fields.

Single-line text fields are handled by the Panel_text panel item of XView and by the XmTextField widget.

A particularity for the programmer : the XView panel item is easier to use because it incorporates a label attribute while the Motif equivalent is a label widget distinct from the XmTextField widget. Moreover, the user will appreciate the scrolling buttons that appear at each end of the XView text field (panel item) when the input text is too long for that object.

5.3. Graphics capabilities

The XmDrawingArea widget seems to be the only one concerned with the display of graphical output. The package that corresponds to it is Canvas.

5.4. Menus

XView provides an easy-to-use menu package whereas Motif menu systems are quite complicated. However, except for the Motif option menu, their possibilities are not really different.

In order to allow the creation of menus, Motif defines two classes of widgets (XmMenuShell and XmRowColumn) and many convenience functions. The programmer should be able to use these two basic widgets or the convenience functions to create menu systems.

The XView programmer necessarily has to consider the Menu package.

As explained in section 4.5.1. , programmers will immediately see the main difference between the XView and Motif ways of creating menus : the order of creation of the elements of the menu is not the same.

Another difference lies in the fact that menu items are declared in XView as attributes of the menu pane while Motif declares them as separate widgets. The consequence of this is that the XView code is more compact than its Motif counterpart. However, Motif allows the callback procedure of a menu item to be called with a defined value, which is impossible with XView.

From a user point of view, both toolkits offer pop-up, pulldown and pullright menus. In addition to them, Motif also has an option menu while XView proposes exclusive and non-exclusive choices in menus. XView also allows the use of pushpins to fix menu panes on the screen, and to dismiss them when they are not used any more.

It is not clear whether Motif also offers exclusive and non-exclusive choices in menus.

5.5. Scrolling capabilities

It must be noted about scrolling capabilities that Motif offers more built-in components provided with scrollbars than XView.

XView provides the Scrollbar package. That object is independent and must be attached to canvases or text subwindows. It can be used to split them in two or more work regions.

The scrollable panel package is a panel with a scrollbar already attached to it. Such a panel can accept more items then it could do otherwise.

The XmScrollBar widget is the Motif equivalent of the XView scrollbar. However, a difference exist in the way they can be used. Both components look the same except for the fact that the slider of the XView scrollbar is directly surrounded by two buttons attached to it and containing an arrow. Pressing a defined mouse button while the pointer is over one of these buttons makes the slider and the two buttons move until the mouse button is released. This way of moving the slider is not possible with XmScrollBar. Moreover, during this movement, the pointer stays on the arrow button that is to say near the slider and the other button. This allows a quick change of direction.

The XmScrolledWindow is used to frame other components. It is provided with scrollbars in order to scroll the visible area. The XmSelectionBox and XmFileSelectionBox widgets are other built-in applications of scrolling facilities.

The XmList widget can be made scrollable when necessary. The XView counterpart is implemented by the Panel_list panel item.

5.6. Commands and choices

Commands and choices are not easy to compare because of the variety of objects and the fact that all components do not necessarily exist in both toolkits.

What we mean by 'commands and choices' is illustrated by the following Motif widgets :

       XmCommand
       XmPushButton
       XmArrowButton
       XmScale
       XmToggleButton

Except the Tty subwindow bound to receive commands like a standard terminal, XView provides no equivalent for XmCommand.

The XmPushButton plays the same role as the panel_button panel item.

The XmScale widget can be used in input/output or output only mode. The first case is handled in XView by a slider while a gauge acts as an XmScale in output only mode. (As output component, the gauge is considered as an information component).

The XmToggleButton widget is represented in XView by exclusive and non-exclusive choices.

The equivalent of XmArrowButton is an abbreviated button.

XView offers a few objects that do not exist in Motif (see section 2.7.) :

Numeric fields can be provided with increment/decrement buttons. Theses buttons are used to quickly increment or decrement the numeric input field until the desired value be reached.

Check boxes are only another presentation for the nonexclusive choices.

As it can be seen, both toolkits offer objects that do not exist in the other one but XView has more of them.

5.7. Informations

XView provide information to the user by means of two panel items (gauges and read-only messages) and notices (see figure 2.13.).

The Motif equivalent of the gauge is the XmScale widget discussed in section 5.6.

The XmMessageBox widget and its associated Dialogs (sections 3.1.4.4. and 3.1.4.8.) provide the same functionalities as the XView notice. However, they are more adapted to particular situations.

5.8. Miscellaneous components

Other Motif widgets are presented in chapter 3 and have no direct counterpart.

The shell classes can not be compared in a one-to-one way with XView objects.

The XmSeparator widget is a pretty way of separating widgets but is not an essential (mandatory) element in windows.

Some XView packages are specific to that toolkit, too.

The Generic Object package is equivalent to the Core class. The server, screen, fullscreen, cursor, font, server image and notifier packages have no equivalent in terms of Motif widgets classes.

The existence of them (at least the server and notifier packages) is to be related to the fact that Motif relies on the Xt Intrinsics while XView has to include such functionalities.

Motif apparently provides no equivalent for the Tty and icon packages of XView.

5.9. Functions

The most important XView and Xt functions related to the creation, manipulation and destruction of objects and widgets have been presented in order to make the reader understand how the toolkits are used to display graphical components on a screen. Clearly, not al functions could be discussed. Many other functions exist : functions specific to particular packages in the case of XView, Xt functions, and Motif convenience functions.

Mainly three differences must be brought up when comparing the two sets of functions.

1. The initialization of the toolkit and the creation of the main window are performed by Motif in one function call while XView needs two statements. As the creation of the main window always follows the initialization of the toolkit, the Xt/Motif solution seems to be better.

2. When all objects have been created, both toolkits call a main loop function. In addition to that, a Motif program also needs to manage the widgets (possibly by using a unique function call with XtCreateManagedWidget), and to realize them (by realizing the main window). The XView toolkit does not have 'manage', unmanage', or 'realize' function. However, besides the xv_destroy function, an attribute (xv_show) allows an object to be displayed or undisplayed.

3. Motif convenience functions have no equivalent in XView. With a unique function call, complex graphical components built with many widgets can be created.

5.10. Conclusion

Although XView and Motif present evident similarities due to their intrinsic nature of X based toolkit, the reader will have understood that there are also many differences between them.

Programs written with XView or Motif typically consist of a main function creating the main window of the application and its children directly accessible. That function ends with the call to the main loop function. Callback procedures referred to by objects created in the main function are defined after the main function. They consist either of pure application code (without any use of the X, Xt, Motif, or XView libraries), or create new windows.

Whereas XView is an attribute-value toolkit, Motif offers more widgets. Moreover, it sometimes provides many widgets for almost identical uses. For example, many Motif Dialogs correspond to the XView notice.

From a general point of view, Motif seems to privilege the initiative of the programmer by an enlarged choice of widgets and functions. One consequence can be a more performant resulting application. However, that advantage can only be reached by experienced programmers.

In comparison to Motif, XView appears to be more concise and more easy to learn. XView also implements a more pleasant look and feel.

Finally, it is not possible to say which toolkit should be preferred. XView must be used to easily migrate SunView applications to the X world. Motif can be chosen when an Xt based solution is needed.

Conclusion

As conclusion to this thesis, we shall stand back from the work that led to that report. Before the traineeship that was the support for this thesis, we almost knew nothing about windowing systems and graphical user interfaces. The first goal was to learn to use such systems on workstations in an Unix environment. Then we began to refresh our knowledge of the C programming language which is used in XView and Motif.

With this prerequisite knowledge, we began to discover and study the Motif language, as the simultaneous study of the two toolkits was not to advice.

A perfect comparison would require to almost master XView and Motif. However, reaching such a level needs much time. In [Mik, 90], page 44, Alan Gibbons, an industry consultant for AT&T Bell Labs says about the learning curve of X programming :

' *Even for programmers already experienced in Unix and C, the average seems to be about three to six months for the basics, and even longer for learning about Motif, Open Look, or other toolkits* '.

These considerations should explain some incomplete aspects in the presentation or comparison of the two toolkits.

In spite of that, we reached many goals.

We presented a synthesis about windowing systems and a description of the two toolkits. In the introduction, we said that XView and Motif are competing at two levels : the programming approach and the look and feel. As foreseen, we especially considered the first aspect. We also discussed at times the user point of view.

An important part of this thesis is devoted to a generic language describing the creation of user interfaces in a toolkit-independent way. A few concepts have been introduced and we showed that some difficulties in describing an interface in that way can be solved by using figures.

In the last chapter, we presented a comparison of XView objects and Motif widgets that is essentially based on the effective applications that we wrote and run.

Bibliography

[Hal 87]   Hal L. Stern, Comparison of Window Systems, *BYTE* , November
           1987, pp 265-272.

[Hay 89]   Frank Hayes and Nick Baran, A Guide to GUIs, *BYTE* , July 1989, pp
           250-257.

[Hell 90] The Definitive Guides to the X Window System, O'Reilly & Associates,
           Inc : Volume seven :   Dan Heller, XView Programming Manual   *An
           OPEN LOOK Toolkit for XII*, 1990.

[Hoe 88]   Tony Hoeber, Open Look Design Goals,   *Sun Technology*   Autumn
           1988, pp 63-75.

[Hoe 91]   Tony Hoeber, Face to face with Open Look, *BYTE* December 1988, pp
           286-296.

[Jon 89]   Oliver Jones, Introduction to the X Window System, Prentice-Hall, 1989.

[Ker 78]   Brian W. Kernighan, Dennis M. Ritchie : The C Programing language,
           Prentice-Hall , 1978.

[Man 87]   NeWS Manual, Sun Microsystems, 1987.

[Mar 88]   Ralph R. Swick and Mark S. Ackerman Project Athena MIT, The X
           Toolkit : More Bricks for Building User-Interfaces -or- Widgets For Hire,
           to be presented at the *winter 1988 Usenix*, Dallas, Texas.

[Mik 90]   Steven Mikes, The realities of X, *Unix World Connectivity* 1990, pp 43-
           46.

[Nye1 90] The Definitive Guides to the X Window System, O'Reilly & Associates,
           Inc : Volume one : Adrian Nye, Xlib Programming Manual *for version 11*
           1990.

[Nye2 90] The Definitive Guides to the X Window System, O'Reilly & Associates,
           Inc : Volume four : Adrian Nye, and Tim O'Reilly, X Toolkit Intrinsics
           Programming Manual *for version 11*, 1990.

[OSF1 90] Open Software Foundation, OSF/Motif Programmer's Guide, Revision
           1.1, 1990.

[OSF2 90] Open Software Foundation, OSF/Motif Programmer's Reference, Revision 1.1, 1990.

[OSF3 90] Open Software Foundation, OSF/Motif Style Guide, Revision 1.1, 1990.

[OSF4 90] Open Software Foundation, OSF/Motif User's Guide, Revision 1.1, 1990.

[Ove 87] NeWS Technical Overview, Sun Microsystems, 1987.

[Pap 90] Sun Technology Papers, Sun Microsystems, 1990, pp 195-224.

[Post] Postscript Tutorial and Cookbook (Internal documentation of BIM).

[Rad 88] David Radoff, A new look for Unix, Unix World July 1988 pp 66-70.

[Rei 88] The Definitive Guides to the X Window System, O'Reilly & Associates, Inc : Volume three : Tim O'Reilly, Valerie Quercia, and Linda Lamb, X Window System User's Guide, 1988.

[Sch 90] Chris Schoettle Unix System Laboratories Europe, Open Look - A Consistent Approach to a GUI Architecture, EUUGN Vol. 10 No. 3 Autumn 1990, pp 66-70.

[SUN1 90] OPEN LOOK Graphical User Interface Application Style Guidelines, Sun Microsystems Inc and AT&T, Addison Wesley Publishing Company Inc, 1990.

[SUN2 90] OPEN LOOK Graphical User Interface Functional Specification, Sun Microsystems Inc and AT&T, Addison Wesley Publishing Company Inc, 1989.

[Ver 90] Alain Vermeiren, The X Window System and NCD X Terminals, BIM NETWORK PRODUCTS & SERVICES, July 1990.

[Vol5 90] The Definitive Guides to the X Window System, O'Reilly & Associates, Inc : Volume five : X Toolkit Intrinsics Reference Manual for version 11, 1990.

[Whi 90] Open Windows Version 2 White Papers, July 1990, Sun Microsystems, Inc.

[You 89] Douglas A. Young, X Window Systems, Programming and Applications with Xt, Prentice-Hall, 1989.

ANNEXES

A1. Motif Application Code

1 DECLARATIONS

```c
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <Xm/CascadeB.h>
#include <Xm/BulletinB.h>
#include <Xm/MessageB.h>
#include <Xm/RowColumn.h>
#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/Text.h>
#include <Xm/PanedW.h>
#include <Xm/ScrollBar.h>
#include <Xm/Label.h>
#include <Xm/List.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>

XtAppContext    AppC;
Widget  Phonebook,

 Keys_shell,Keys_txt,

 Keys_display_shell,
 Keys_Name_txt, Keys_F_Name_txt, Keys_Initials_txt,
 Keys_Phone_txt, Keys_Location_txt, Keys_Workgroup_txt,
 Keys_Service_txt, Keys_Respons_txt,
 Keys_Special_txt,

 Special_info_shell,

 Update_shell,
 Upd_Name_txt, Upd_F_Name_txt,  Upd_Initials_txt,
 Upd_Phone_txt, Upd_Location_txt, Upd_Workgroup_txt,
 Upd_Service_txt,Upd_Respons_txt, Upd_Special_txt;

Arg al[10];
int ac;

FILE *fopen(),*fp1,*fp2;

char line1[80],line2[80],line3[80],line4[80],line5[80],
     line6[80],line7[80],line8[80],line9[80],line[80];

char name[80],f_name[80],Search_data[80];

char text_value[5000];

int nbmax,nelem;

struct Check_data_type
{
 char name[80];
 char f_name[80];
} Check_data,Delete_data;
```

```
struct Update_data_type
{
 char name[80];
 char f_name[80];
 char initials[80];
 char phone[80];
 char location[80];
 char workgroup[80];
 char service[80];
 char respons[80];
 char special[80];
} Insert_data,Modify_data;

char table[200][9][80];

void Init();
void Keys_cbproc();
void Set_info_cbproc();
void Search_cbproc();
void GetString();
void Adapt1();
int  Find();
void Settable();
void Erase_cbproc();
void Keys_display_info();
void Move_cbproc();
void Lists_cbproc();
void Allinfo_proc();
void Activities_proc();
void Special_info_proc
void Adapt2();
void Update_cbproc();
void Check_cbproc();
int  Check();
void Insert_cbproc();
void Insert();
void Modify_cbproc();
void Modify();
void Delete_cbproc();
void Delete_proc();
void Kill_cbproc();
void Kill_ok_cbproc();
void Quit_cbproc();
void Quit_ok_cbproc();

/****************************************************************************************/
```

2 main(argc,argv)

```
main(argc,argv)
 int argc;
 char *argv[];
{
 Widget  Menubar, Cons_casbtn, Consult_pane, Keys_casbtn,
  Keys_subpane, Name_pushbtn, F_Name_pushbtn, Special_pushbtn,
  Lists_casbtn, Lists_subpane, Allinfo_pushbtn,Acti_pushbtn,
  Upd_casbtn, Update_pane, Insert_pushbtn, Modify_pushbtn,
  Delete_pushbtn, Quit_casbtn;

        Phonebook = XtInitialize("test","test_class",NULL,0,&argc,argv);

 Init();

/* create a menubar in the main window */
```

```
Menubar=XmCreateMenuBar(Phonebook,"Menubar",NULL,0);
XtManageChild(Menubar);

/* create 2 pulldown menu panes attached to the menubar */

Consult_pane=XmCreatePulldownMenu(Menubar,"Consult_pane",NULL,0);
Update_pane =XmCreatePulldownMenu(Menubar,"Update_pane",NULL,0);
XtManageChild(Consult_pane);
XtManageChild(Update_pane);

/* create 2 cascade buttons : Consult and Update                */
/*           1 push    button   : Quit              in the menubar */
ac = 0;
XtSetArg(al[ac],XmNsubMenuId,Consult_pane); ac++;
Cons_casbtn=XmCreateCascadeButton(Menubar,"Consult",al,ac);
XtManageChild(Cons_casbtn);

ac = 0;
XtSetArg(al[ac],XmNsubMenuId,Update_pane); ac++;
Upd_casbtn=XmCreateCascadeButton(Menubar,"Update",al,ac);
XtManageChild(Upd_casbtn);

ac = 0;
Quit_casbtn=XmCreateCascadeButton(Menubar,"Quit",al,ac);
XtAddCallback(Quit_casbtn,XmNactivateCallback,Kill_cbproc,Phonebook);
XtManageChild(Quit_casbtn);

/* create 2 pulldown submenus in Consult_pane */

Keys_subpane=XmCreatePulldownMenu(Consult_pane,"Keys_subpane",NULL,0);
Lists_subpane=XmCreatePulldownMenu(Consult_pane,"Lists_subpane",NULL,0);
XtManageChild(Keys_subpane);
XtManageChild(Lists_subpane);

/* create 2 cascade buttons in  Consult_pane : Keys and Lists */

ac = 0;
XtSetArg(al[ac],XmNsubMenuId,Keys_subpane); ac++;
Keys_casbtn=XmCreateCascadeButton(Consult_pane,"Keys",al,ac);
XtManageChild(Keys_casbtn);

ac = 0;
XtSetArg(al[ac],XmNsubMenuId,Lists_subpane); ac++;
Lists_casbtn=XmCreateCascadeButton(Consult_pane,"Lists",al,ac);
XtManageChild(Lists_casbtn);

/* create 3 push buttons in Keys_subpane */

ac = 0;
Name_pushbtn=XmCreatePushButton(Keys_subpane,"Name",al,ac);
XtAddCallback(Name_pushbtn,XmNactivateCallback,Keys_cbproc,1);
XtManageChild(Name_pushbtn);

ac = 0;
F_Name_pushbtn=XmCreatePushButton(Keys_subpane,"First name",al,ac);
XtAddCallback(F_Name_pushbtn,XmNactivateCallback,Keys_cbproc,2);
XtManageChild(F_Name_pushbtn);

ac = 0;
Special_pushbtn=XmCreatePushButton(Keys_subpane,"Specialization",
        al,ac);
XtAddCallback(Special_pushbtn,XmNactivateCallback,Keys_cbproc,3);
XtManageChild(Special_pushbtn);

/* create 2 push buttons in Lists_subpane */

ac = 0;
Allinfo_pushbtn=XmCreatePushButton(Lists_subpane,"All info",al,ac);
XtAddCallback(Allinfo_pushbtn,XmNactivateCallback,Lists_cbproc,1);
XtManageChild(Allinfo_pushbtn);
```

```
ac = 0;
Acti_pushbtn=XmCreatePushButton(Lists_subpane,"Activities",al,ac);
XtAddCallback(Acti_pushbtn,XmNactivateCallback,Lists_cbproc,2);
XtManageChild(Acti_pushbtn);

/* create 3 push buttons in  Update_pane : Insert, Modify and Delete */

ac = 0;
Insert_pushbtn=XmCreatePushButton(Update_pane,"Insert ",al,ac);
XtAddCallback(Insert_pushbtn,XmNactivateCallback,Update_cbproc,1);
XtManageChild(Insert_pushbtn);

ac = 0;
Modify_pushbtn=XmCreatePushButton(Update_pane,"Modify ",al,ac);
XtAddCallback(Modify_pushbtn,XmNactivateCallback,Update_cbproc,2);
XtManageChild(Modify_pushbtn);

ac = 0;
Delete_pushbtn=XmCreatePushButton(Update_pane,"Delete ",al,ac);
XtAddCallback(Delete_pushbtn,XmNactivateCallback,Update_cbproc,3);
XtManageChild(Delete_pushbtn);

/* main loop */

XtRealizeWidget(Phonebook);
XtMainLoop();

} /* end of main */
```

void Init( )

```
void Init()
{
 fp1=fopen("phonefile","r");
 if (fp1==NULL)
   {
    fp1=fopen("phonefile","w");
    fclose(fp1);
   };

} /* end of Init */
```

void Keys_cbproc(w,client_data,call_data)

```
void Keys_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 Widget  Keys_bubo, Search_btn, Btn1, Btn2,
  Clear_btn, Quit_btn, Label,  RowCol;
 int keys;

 if (client_data==1) keys=1;
 if (client_data==2) keys=2;
 if (client_data==3) keys=3;

/* create a shell in Keys_cbproc */

ac = 0;
XtSetArg(al[ac],XmNheight,120); ac++;
XtSetArg(al[ac],XmNwidth, 340); ac++;
Keys_shell=XtCreateManagedWidget("Keys_shell",
    topLevelShellWidgetClass,Phonebook,al,ac);
```

```
/* create the form in Keys_cbproc */

Keys_bubo=XtCreateManagedWidget("Keys_form",
    xmBulletinBoardWidgetClass,Keys_shell,NULL,0);

/* create the search button in Keys_cbproc */

ac = 0;
XtSetArg(al[ac],XmNlabelString,XmStringCreate("Search",
    XmSTRING_DEFAULT_CHARSET)); ac++;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,20); ac++;
Search_btn=XtCreateManagedWidget("Search_btn",
    xmPushButtonWidgetClass,Keys_bubo,al,ac);
XtAddCallback(Search_btn,XmNactivateCallback,Search_cbproc,keys);

/* create the erase button in Keys_cbproc */

ac = 0;
XtSetArg(al[ac],XmNlabelString,XmStringCreate("Erase ",
    XmSTRING_DEFAULT_CHARSET)); ac++;
XtSetArg(al[ac],XmNx,70); ac++;
XtSetArg(al[ac],XmNy,20); ac++;
Erase_btn=XtCreateManagedWidget("Erase_btn",
    xmPushButtonWidgetClass,Keys_bubo,al,ac);
XtAddCallback(Erase_btn,XmNactivateCallback,Erase_cbproc,1);

/* create the quit button in Keys_cbproc */

ac = 0;
XtSetArg(al[ac],XmNlabelString,XmStringCreate(" Quit ",
    XmSTRING_DEFAULT_CHARSET)); ac++;
XtSetArg(al[ac],XmNx,130); ac++;
XtSetArg(al[ac],XmNy,20); ac++;
Quit_btn=XtCreateManagedWidget("Quit_btn",
    xmPushButtonWidgetClass,Keys_bubo,al,ac);
XtAddCallback(Quit_btn,XmNactivateCallback,Quit_cbproc,Keys_shell);

/* create the complete/cut info choice in Keys_cbproc */

/* XtSetArg(al[0],XmNx,190);
XtSetArg(al[1],XmNy,20);
XtSetArg(al[2],XmNradioAlwaysOne,True);
XtSetArg(al[3],XmNradioBehavior,True);
RowCol=XtCreateManagedWidget("RowCol",xmRowColumnWidgetClass,
      Keys_bubo,al,4);
ac = 0;
XtSetArg(al[ac],XmNx,190); ac++;
XtSetArg(al[ac],XmNy,30); ac++;
XtSetArg(al[ac],XmNset,True); ac++;
Btn1=XtCreateManagedWidget("Cut info     ",
    xmToggleButtonWidgetClass,RowCol,al,ac);
ac = 0;
XtSetArg(al[ac],XmNx,190); ac++;
XtSetArg(al[ac],XmNy,40); ac++;
XtSetArg(al[ac],XmNset,False); ac++;
Btn2=XtCreateManagedWidget("Complete info",
    xmToggleButtonWidgetClass,RowCol,al,ac); */

/* create the input label in Keys_cbproc */
```

```
ac = 0;
XtSetArg(al[ac],XmNx,10); ac++; /* common to  */
XtSetArg(al[ac],XmNy,81); ac++;/* all labels */
if (keys==1)
 {
         XtSetArg(al[ac],XmNlabelString,
             XmStringCreate("Name",XmSTRING_DEFAULT_CHARSET));
  ac++;
 }
if (keys==2)
 {
         XtSetArg(al[ac],XmNlabelString,
             XmStringCreate("First Name",XmSTRING_DEFAULT_CHARSET));
  ac++;
 }
if (keys==3)
 {
         XtSetArg(al[ac],XmNlabelString,
   XmStringCreate("Specialization",
      XmSTRING_DEFAULT_CHARSET));
  ac++;
 }
Label=XtCreateManagedWidget("Label",xmLabelWidgetClass,Keys_bubo,al,ac);

/* create the input text field in Keys_cbproc */

        ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNy,75); ac++;
XtSetArg(al[ac],XmNcolumns,30); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNmaxLength,80); ac++;
XtSetArg(al[ac],XmNeditMode,XmSINGLE_LINE_EDIT); ac++;
Keys_txt=XtCreateManagedWidget("Keys_txt",
    xmTextWidgetClass,Keys_bubo,al,ac);

} /* end of Keys_cbproc */
```

void Set_info_cbproc(w,client_data,call_data)

```
void Set_info_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
} /* end of Set_info_cbproc */
```

void Search_cbproc(w,client_data,call_data)

```
void Search_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 Widget Check_msg;
 int i,f;
 XmString str1,str2;

        GetString(Search_data,Keys_txt);
```

```c
fp1=fopen("phonefile","r");
i=0;
while(fgets(line1,80,fp1)!=NULL)
  {
  fgets(line2,80,fp1);
  fgets(line3,80,fp1);
  fgets(line4,80,fp1);
  fgets(line5,80,fp1);
  fgets(line6,80,fp1);
  fgets(line7,80,fp1);
  fgets(line8,80,fp1);
  fgets(line9,80,fp1);
  if (((client_data==1)&&(strcmp(line1,Search_data)==0))
     ||((client_data==2)&&(strcmp(line2,Search_data)==0))
     ||((client_data==3)&&((f=Find(line9,Search_data))==1)))
    {
    Settable(i,0,line1);
    Settable(i,1,line2);
    Settable(i,2,line3);
    Settable(i,3,line4);
    Settable(i,4,line5);
    Settable(i,5,line6);
    Settable(i,6,line7);
    Settable(i,7,line8);
    Settable(i,8,line9);
    i++;
    };
  };
fclose(fp1);

nbmax=i;
if (nbmax==0)
  {
  ac = 0;
  str1=XmStringCreate("Does not exist",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNdialogType,XmDIALOG_MESSAGE); ac++;
  XtSetArg(al[ac],XmNmessageString,str1); ac++;
          str2=XmStringCreate("Continue", XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNokLabelString,str2); ac++;
    XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL);
    ac++;

  Check_msg=XmCreateMessageDialog(Phonebook,"Check_msg",al,ac);
  XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
      XmDIALOG_HELP_BUTTON));
  XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
      XmDIALOG_CANCEL_BUTTON));
  XtManageChild(Check_msg);
  };

if (nbmax>200)
  nbmax=200;
if (nbmax>0)
  {
  XtUnmanageChild(Keys_shell);
  nelem=0;
  Keys_display_info();
  }

} /* end of Search_cbproc */
```

void GetString(ptr,w)

```
void GetString(ptr,w)
 char *ptr;
 Widget w;
{
 char *temp1,*temp2;
 int i;

 temp2 = ptr;
 temp1 = XmTextGetString(w);
 for (i=1;i<=80;i++)
   *ptr++ = *temp1++;
 Adapt1(temp2);

} /* end of GetString */
```

void Adapt1(data_pointer)

```
void Adapt1(data_pointer)
 char *data_pointer;
{
 int i;

 i=1;
 while ((*data_pointer!='\0')
      &&(*data_pointer!='\n')
      &&(i<=79))
   {
    data_pointer++;
    i++;
   }
 *data_pointer++='\n';
 *data_pointer='\0';

}. /* end of Adapt1 */
```

2   int Find(str,substr)

```
int Find(str,substr)
 char *str;
 char *substr;
{
 int found;

 if (*str==*substr)
      if (*str=='\0')
          found=1;
      else {
      str++;
      substr++;
      found=Find(str,substr);
    }
 else if (*str=='\0')
   found=0;
      else {
      str++;
      found=Find(str,substr);
    };
```

```
   return(found);
} /* end of Find */
```

3 void Settable(i,j,line)

```
void Settable(i,j,line)
 int i;
 int j;
 char *line;
{
 int k;

 k=0;
 while (*line!='\0')
    if (*line!='\n')
    table[i][j][k++] = *(line++);
    else line++;
 table[i][j][k++]='\0';
 printf("%s\n",table[i][j]);

} /* end of Settable */
```

4 void Erase_cbproc(w,client_data,call_data)

```
void Erase_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 if (client_data==1)
   {
    XmTextSetString(Keys_txt,"");
   };
 if (client_data==2)
   {
    XmTextSetString(Upd_Name_txt,"");
    XmTextSetString(Upd_F_Name_txt,"");
    XmTextSetString(Upd_Initials_txt,"");
    XmTextSetString(Upd_Phone_txt,"");
    XmTextSetString(Upd_Location_txt,"");
    XmTextSetString(Upd_Workgroup_txt,"");
    XmTextSetString(Upd_Service_txt,"");
    XmTextSetString(Upd_Respons_txt,"");
    XmTextSetString(Upd_Special_txt,"");
   };

} /* end of Erase_cbproc */
```

5 void Keys_display_info( )

```
void Keys_display_info()
{
 Widget Keys_info_bubo,Next_btn,Previous_btn,Quit_btn,
        Label1,Label2,Label3,Label4,Label5,Label6,Label7,Label8,Label9;

/* create a shell in Keys_display_info */
```

```
ac = 0;
XtSetArg(al[ac],XmNheight,470); ac++;
XtSetArg(al[ac],XmNwidth,650); ac++;
Keys_display_shell=XtCreateManagedWidget("Keys_display_shell",
    topLevelShellWidgetClass,Phonebook,al,ac);

/* create the bulletin board in Keys_display_info */

Keys_info_bubo=XtCreateManagedWidget("Keys_info_bubo",
  xmBulletinBoardWidgetClass,Keys_display_shell,NULL,0);

/* create the Next button in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNlabelString,XmStringCreate("Next",
    XmSTRING_DEFAULT_CHARSET)); ac++;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,20); ac++;
Next_btn=XtCreateManagedWidget("Next_btn",
    xmPushButtonWidgetClass,Keys_info_bubo,al,ac);
XtAddCallback(Next_btn,XmNactivateCallback,Move_cbproc,1);

/* create the Previous button in Keys_display_info */

  ac = 0;
  XtSetArg(al[ac],XmNlabelString,XmStringCreate("Previous",
      XmSTRING_DEFAULT_CHARSET)); ac++;
  XtSetArg(al[ac],XmNx,60); ac++;
  XtSetArg(al[ac],XmNy,20); ac++;
  Previous_btn=XtCreateManagedWidget("Previous_btn",
      xmPushButtonWidgetClass,Keys_info_bubo,al,ac);
  XtAddCallback(Previous_btn,XmNactivateCallback,Move_cbproc,2);

/* create the quit button in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNlabelString,XmStringCreate(" Quit ",
    XmSTRING_DEFAULT_CHARSET)); ac++;
XtSetArg(al[ac],XmNx,155); ac++;
XtSetArg(al[ac],XmNy,20); ac++;
Quit_btn=XtCreateManagedWidget("Quit_btn",
    xmPushButtonWidgetClass,Keys_info_bubo,al,ac);
XtAddCallback(Quit_btn,XmNactivateCallback,Quit_cbproc,
        Keys_display_shell);
/* create the name label in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++; /* common to all labels */
XtSetArg(al[ac],XmNy,76); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Name",XmSTRING_DEFAULT_CHARSET)); ac++;
Label1=XtCreateManagedWidget("Label1",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);
/* create the first name label in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++; /* common to all labels */
XtSetArg(al[ac],XmNy,116); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("First Name",XmSTRING_DEFAULT_CHARSET));
ac++;
Label2=XtCreateManagedWidget("Label2",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);
/* create the initials label in Keys_display_info */
```

```
ac = 0;
XtSetArg(al[ac],XmNx,10); ac++; /* common to all labels */
XtSetArg(al[ac],XmNy,156); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Initials",XmSTRING_DEFAULT_CHARSET));
ac++;
Label3=XtCreateManagedWidget("Label3",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);
/* create the phone label in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,196); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Phone number",
     XmSTRING_DEFAULT_CHARSET)); ac++;
Label4=XtCreateManagedWidget("Label4",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);

/* create the location label in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,236); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Location",XmSTRING_DEFAULT_CHARSET));
ac++;
Label5=XtCreateManagedWidget("Label5",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);

/* create the workgroup label in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,276); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Workgroup",XmSTRING_DEFAULT_CHARSET));
ac++;
Label6=XtCreateManagedWidget("Label6",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);

/* create the service label in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,316); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Service",XmSTRING_DEFAULT_CHARSET));
ac++;
Label7=XtCreateManagedWidget("Label7",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);

/* create the responsibilities label in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,356); ac++;
        XtSetArg(al[ac],XmNlabelString,
        XmStringCreate("Responsibilities",XmSTRING_DEFAULT_CHARSET)); ac++;
Label8=XtCreateManagedWidget("Label8",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);

/* create the specialization label in Keys_display_info */
```

```
ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,396); ac++;
        XtSetArg(al[ac],XmNlabelString,
                  XmStringCreate("Specialization",
    XmSTRING_DEFAULT_CHARSET)); ac++;
Label9=XtCreateManagedWidget("Label9",xmLabelWidgetClass,
        Keys_info_bubo,al,ac);

/* create the name input text field in Keys_display_info */


ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,70); ac++;
Keys_Name_txt=XtCreateManagedWidget("Keys_Name_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the first name input text field in Keys_display_info */
ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,110); ac++;
Keys_F_Name_txt=XtCreateManagedWidget("Keys_F_Name_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the initials input text field in Keys_display_info */
ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,150); ac++;
Keys_Initials_txt=XtCreateManagedWidget("Keys_Initials_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the phone input text field in Keys_display_info */
ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,190); ac++;
Keys_Phone_txt=XtCreateManagedWidget("Keys_Phone_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the location input text field in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,230); ac++;
Keys_Location_txt=XtCreateManagedWidget("Keys_Location_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the workgroup input text field in Keys_display_info */
```

```
ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,270); ac++;
Keys_Workgroup_txt=XtCreateManagedWidget("Keys_Workgroup_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the service input text field in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,310); ac++;
Keys_Service_txt=XtCreateManagedWidget("Keys_Service_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the responsibilities input text field in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;

XtSetArg(al[ac],XmNy,350); ac++;
Keys_Respons_txt=XtCreateManagedWidget("Keys_Respons_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* create the specialization input text field in Keys_display_info */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
XtSetArg(al[ac],XmNeditable,False); ac++;
XtSetArg(al[ac],XmNy,390); ac++;
Keys_Special_txt=XtCreateManagedWidget("Keys_Special_txt",
    xmTextWidgetClass,Keys_info_bubo,al,ac);

/* set the value of the strings */

XmTextSetString(Keys_Name_txt,table[0][0]);
XmTextSetString(Keys_F_Name_txt,table[0][1]);
XmTextSetString(Keys_Initials_txt,table[0][2]);
XmTextSetString(Keys_Phone_txt,table[0][3]);
XmTextSetString(Keys_Location_txt,table[0][4]);
XmTextSetString(Keys_Workgroup_txt,table[0][5]);
XmTextSetString(Keys_Service_txt,table[0][6]);
XmTextSetString(Keys_Respons_txt,table[0][7]);
XmTextSetString(Keys_Special_txt,table[0][8]);

} /* end of Keys_display_info */
```

6 void Move_cbproc(w,client_data,call_data)

```
void Move_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 int changed;

 changed=0;
        if ((nelem<nbmax-1)&&(client_data==1))
   {
    nelem++;
    changed=1;
   };
 if ((nelem>0)&&(client_data==2))
   {
    nelem--;
    changed=1;
   };
 if (changed==1)
   {
   XmTextSetString(Keys_Name_txt,table[nelem][0]);
   XmTextSetString(Keys_F_Name_txt,table[nelem][1]);
   XmTextSetString(Keys_Initials_txt,table[nelem][2]);
   XmTextSetString(Keys_Phone_txt,table[nelem][3]);
   XmTextSetString(Keys_Location_txt,table[nelem][4]);
   XmTextSetString(Keys_Workgroup_txt,table[nelem][5]);
   XmTextSetString(Keys_Service_txt,table[nelem][6]);
   XmTextSetString(Keys_Respons_txt,table[nelem][7]);
   XmTextSetString(Keys_Special_txt,table[nelem][8]);
   }
} /* end of Move */
```

7 void Lists_cbproc(w,client_data,call_data)

```
void Lists_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 if (client_data==1) Allinfo_proc();
 if (client_data==2) Activities_proc();

} /* end of Lists_cbproc */
```

8 void Allinfo_proc( )

```
void Allinfo_proc()
{
 Widget Check_msg;
 int i;
 XmString str1,str2;
```

```
fp1=fopen("phonefile","r");
i=0;
while(fgets(line1,80,fp1)!=NULL)
  {
   fgets(line2,80,fp1);
   fgets(line3,80,fp1);
   fgets(line4,80,fp1);
   fgets(line5,80,fp1);
   fgets(line6,80,fp1);
   fgets(line7,80,fp1);
   fgets(line8,80,fp1);
   fgets(line9,80,fp1);

   Settable(i,0,line1);
   Settable(i,1,line2);
   Settable(i,2,line3);
   Settable(i,3,line4);
   Settable(i,4,line5);
   Settable(i,5,line6);
   Settable(i,6,line7);
   Settable(i,7,line8);
   Settable(i,8,line9);

   i++;
  };
fclose(fp1);

nbmax=i;
if (nbmax==0)
  {
   str1=XmStringCreate("The phonebook is empty",
      XmSTRING_DEFAULT_CHARSET);
   ac = 0;
   XtSetArg(al[ac],XmNdialogType,XmDIALOG_MESSAGE); ac++;
   XtSetArg(al[ac],XmNmessageString,str1); ac++;
          str2=XmStringCreate("Continue", XmSTRING_DEFAULT_CHARSET);
   XtSetArg(al[ac],XmNokLabelString,str2); ac++;
     XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL);
   ac++;

   Check_msg=XmCreateMessageDialog(Phonebook,"Check_msg",al,ac);
   XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
      XmDIALOG_HELP_BUTTON));
   XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
      XmDIALOG_CANCEL_BUTTON));
   XtManageChild(Check_msg);
  };

if (nbmax>200)
  nbmax=200;
if (nbmax>0)
  {
   Keys_display_info();
  }

} /* end of Allinfo_proc */
```

9 void Activities_proc( )

```c
void Activities_proc()
{
Widget Check_msg;
XmString str1,str2;
char *value;
int i,j;

i=0;
value=text_value;
fp1=fopen("phonefile","r");
while((fgets(line1,80,fp1)!=NULL)&&(i<=4998))
  {
    fgets(line2,80,fp1);
    fgets(line3,80,fp1);
    fgets(line4,80,fp1);
    fgets(line5,80,fp1);
    fgets(line6,80,fp1);
    fgets(line7,80,fp1);
    fgets(line8,80,fp1);
    fgets(line9,80,fp1);

    Adapt2(line9);

    j=0;
    while((line9[j]!='\n')&&(i<=4998))
    {
      if ((line9[j]==',')||(line9[j]==';'))
          *value++='\n';
      else *value++=line9[j];
      i++;
      j++;
    };
  *value++='\n';
  i++;

  };
*value='\0';
fclose(fp1);

if (i==0)
  {
    str1=XmStringCreate("No specialization",
       XmSTRING_DEFAULT_CHARSET);
    ac = 0;
    XtSetArg(al[ac],XmNdialogType,XmDIALOG_MESSAGE); ac++;
    XtSetArg(al[ac],XmNmessageString,str1); ac++;
          str2=XmStringCreate("Continue", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(al[ac],XmNokLabelString,str2); ac++;
      XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL);
    ac++;

    Check_msg=XmCreateMessageDialog(Phonebook,"Check_msg",al,ac);
    XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
       XmDIALOG_HELP_BUTTON));
    XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
       XmDIALOG_CANCEL_BUTTON));
    XtManageChild(Check_msg);
  };

if (i>0)
  Keys_display_special();

} /* end of Activities_proc */
```

10 void Adapt2(data_pointer)

```
void Adapt2(data_pointer)
 char *data_pointer;
{
 while ((*data_pointer!='\0')
       &&(*data_pointer!='\n'))
    data_pointer++;
 *data_pointer='\n';

} /* end of Adapt2 */
```


11 void Special_info_proc( )

```
void Special_info_proc()
{
 Widget Special_info_bubo, Quit_btn,
        Special_info_txt;

/* create a shell in Special_info_proc */

 ac = 0;
 XtSetArg(al[ac],XmNheight,200); ac++;
 XtSetArg(al[ac],XmNwidth,225); ac++;
 Special_info_shell=XtCreateManagedWidget("Special_info_shell",
    topLevelShellWidgetClass,Phonebook,al,ac);

/* create the bulletin board in Special_info_proc */

 Special_info_bubo=XtCreateManagedWidget("Special_info_bubo",
    xmBulletinBoardWidgetClass,Special_info_shell,NULL,0);

/* create the quit button in Special_info_proc */

 ac = 0;
 XtSetArg(al[ac],XmNy,10); ac++;
 XtSetArg(al[ac],XmNx,30); ac++;
 XtSetArg(al[ac],XmNlabelString,XmStringCreate(" Quit ",
    XmSTRING_DEFAULT_CHARSET)); ac++;
 Quit_btn=XtCreateManagedWidget("Quit_btn",
    xmPushButtonWidgetClass,Special_info_bubo,al,ac);
 XtAddCallback(Quit_btn,XmNactivateCallback,Quit_cbproc,
        Special_info_shell);

/* create a text in Special_info_proc */

 ac = 0;
 XtSetArg(al[ac],XmNy,40); ac++;
 XtSetArg(al[ac],XmNx,30); ac++;
 XtSetArg(al[ac],XmNheight,150); ac++;
 XtSetArg(al[ac],XmNwidth,180); ac++;
 XtSetArg(al[ac],XmNscrollBarPlacement,XmBOTTOM_LEFT); ac++;
        XtSetArg(al[ac],XmNeditMode,XmMULTI_LINE_EDIT); ac++;
 Special_info_txt=XmCreateScrolledText(Special_info_bubo,
        "Special_info_txt",al,ac);
 XtManageChild(Special_info_txt);

 XmTextSetEditable(Special_info_txt,False);
 XmTextSetString(Special_info_txt,text_value);

} /* end of Special_info_proc */
```

12   void Update_cbproc(w,client_data,call_data)

```
void Update_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 Widget  Update_bubo, Check_btn, Insert_btn, Modify_btn,
  Delete_btn, Clear_btn, Quit_btn,
  Label1,Label2,Label3,Label4,Label5,Label6,Label7,Label8,Label9;

/* create a shell in Update_cbproc */

 ac = 0;
 XtSetArg(al[ac],XmNheight,470); ac++;
 XtSetArg(al[ac],XmNwidth,650); ac++;
 Update_shell=XtCreateManagedWidget("Update_shell",
    topLevelShellWidgetClass,Phonebook,al,ac);

/* create the bulletin board in Update_cbproc */

 Update_bubo=XtCreateManagedWidget("Update_bubo",
    xmBulletinBoardWidgetClass,Update_shell,NULL,0);

/* create the check button in Update_cbproc */

 ac = 0;
 XtSetArg(al[ac],XmNy,20); ac++;

 XtSetArg(al[ac],XmNx,10); ac++;
 XtSetArg(al[ac],XmNlabelString,XmStringCreate("Check",
    XmSTRING_DEFAULT_CHARSET)); ac++;
 Check_btn=XtCreateManagedWidget("Check_btn",
    xmPushButtonWidgetClass,Update_bubo,al,ac);
 XtAddCallback(Check_btn,XmNactivateCallback,Check_cbproc,1);

/* create the insert button in Update_cbproc */

 ac = 0;
 XtSetArg(al[ac],XmNy,20); ac++;
 XtSetArg(al[ac],XmNx,60); ac++; /* common to all update buttons */

 if (client_data==1)
 {
  XtSetArg(al[ac],XmNlabelString,XmStringCreate("Insert",
     XmSTRING_DEFAULT_CHARSET)); ac++;
  Insert_btn=XtCreateManagedWidget("Insert_btn",
     xmPushButtonWidgetClass,Update_bubo,al,ac);
  XtAddCallback(Insert_btn,XmNactivateCallback,Insert_cbproc,1);
 }

/* create the modify button in Update_cbproc */

 if (client_data==2)
 {
  XtSetArg(al[ac],XmNlabelString,XmStringCreate("Modify",
     XmSTRING_DEFAULT_CHARSET)); ac++;
  Modify_btn=XtCreateManagedWidget("Modify_btn",
     xmPushButtonWidgetClass,Update_bubo,al,ac);
  XtAddCallback(Modify_btn,XmNactivateCallback,Modify_cbproc,1);
 }

/* create the delete button in Update_cbproc */
```

```
if (client_data==3)
{
 XtSetArg(al[ac],XmNlabelString,XmStringCreate("Delete",
    XmSTRING_DEFAULT_CHARSET)); ac++;
 Delete_btn=XtCreateManagedWidget("Delete_btn",
    xmPushButtonWidgetClass,Update_bubo,al,ac);
 XtAddCallback(Delete_btn,XmNactivateCallback,Delete_cbproc,1);
}
/* create the erase button in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNy,20); ac++;
XtSetArg(al[ac],XmNx,60); ac++; /* common to all update buttons */
XtSetArg(al[ac],XmNx,115); ac++;
XtSetArg(al[ac],XmNlabelString,XmStringCreate("Erase ",
    XmSTRING_DEFAULT_CHARSET)); ac++;
Erase_btn=XtCreateManagedWidget("Erase_btn",
    xmPushButtonWidgetClass,Update_bubo,al,ac);
XtAddCallback(Erase_btn,XmNactivateCallback,Erase_cbproc,2);

/* create the quit button in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNy,20); ac++;
XtSetArg(al[ac],XmNx,60); ac++; /* common to all update buttons */
XtSetArg(al[ac],XmNx,185); ac++;
XtSetArg(al[ac],XmNlabelString,XmStringCreate(" Quit ",
    XmSTRING_DEFAULT_CHARSET)); ac++;
Quit_btn=XtCreateManagedWidget("Quit_btn",
    xmPushButtonWidgetClass,Update_bubo,al,ac);
XtAddCallback(Quit_btn,XmNactivateCallback,Quit_cbproc,Update_shell);

/* create the name label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,76); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Name",XmSTRING_DEFAULT_CHARSET)); ac++;
Label1=XtCreateManagedWidget("Label1",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the first name label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,116); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("First Name",XmSTRING_DEFAULT_CHARSET));
ac++;
Label2=XtCreateManagedWidget("Label2",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the initials label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,156); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Initials",XmSTRING_DEFAULT_CHARSET));
ac++;
Label3=XtCreateManagedWidget("Label3",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the phone label in Update_cbproc */
```

```
ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,196); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Phone number",
    XmSTRING_DEFAULT_CHARSET)); ac++;
Label4=XtCreateManagedWidget("Label4",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the location label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,236); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Location",XmSTRING_DEFAULT_CHARSET));
ac++;
Label5=XtCreateManagedWidget("Label5",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the workgroup label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,276); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Workgroup",XmSTRING_DEFAULT_CHARSET));
ac++;
Label6=XtCreateManagedWidget("Label6",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the service label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,316); ac++;
        XtSetArg(al[ac],XmNlabelString,
                XmStringCreate("Service",XmSTRING_DEFAULT_CHARSET));
ac++;
Label7=XtCreateManagedWidget("Label7",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the responsibilities label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,356); ac++;
        XtSetArg(al[ac],XmNlabelString,
        XmStringCreate("Responsibilities",XmSTRING_DEFAULT_CHARSET)); ac++;
Label8=XtCreateManagedWidget("Label8",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the specialization label in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,10); ac++;
XtSetArg(al[ac],XmNy,396); ac++;
        XtSetArg(al[ac],
XmNlabelString,XmStringCreate("Specialization",
    XmSTRING_DEFAULT_CHARSET)); ac++;
Label9=XtCreateManagedWidget("Label9",xmLabelWidgetClass,Update_bubo,
        al,ac);
/* create the name input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,70); ac++;
Upd_Name_txt=XtCreateManagedWidget("Upd_Name_txt",
    xmTextWidgetClass,Update_bubo,al,ac);
```

```
/* create the first name input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,110); ac++;
 Upd_F_Name_txt=XtCreateManagedWidget("Upd_F_Name_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

/* create the initials input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,150); ac++;
Upd_Initials_txt=XtCreateManagedWidget("Upd_Initials_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

/* create the phone input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,190); ac++;
Upd_Phone_txt=XtCreateManagedWidget("Upd_Phone_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

/* create the location input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,230); ac++;
Upd_Location_txt=XtCreateManagedWidget("Upd_Location_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

/* create the workgroup input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,270); ac++;
Upd_Workgroup_txt=XtCreateManagedWidget("Upd_Workgroup_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

/* create the service input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,310); ac++;
Upd_Service_txt=XtCreateManagedWidget("Upd_Service_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

/* create the responsibilities input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;
```

```
XtSetArg(al[ac],XmNy,350); ac++;
Upd_Respons_txt=XtCreateManagedWidget("Upd_Respons_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

/* create the specialization input text field in Update_cbproc */

ac = 0;
XtSetArg(al[ac],XmNx,125); ac++;
XtSetArg(al[ac],XmNcolumns,80); ac++;
XtSetArg(al[ac],XmNshadowThickness,1); ac++;

XtSetArg(al[ac],XmNy,390); ac++;
Upd_Special_txt=XtCreateManagedWidget("Upd_Special_txt",
    xmTextWidgetClass,Update_bubo,al,ac);

} /* end of Update_cbproc */
```

13 void Check_cbproc(w,client_data,call_data)

```
void Check_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 Widget Check_msg;
 int exist;
 XmString str1,str2;

 exist=Check();

 if (exist==0)str1=XmStringCreate("Does not exist",
     XmSTRING_DEFAULT_CHARSET);
 if (exist==1)str1=XmStringCreate("Exists",XmSTRING_DEFAULT_CHARSET);
 if (exist==-1)str1=XmStringCreate("Not allowed",
     XmSTRING_DEFAULT_CHARSET);

 ac = 0;
 XtSetArg(al[ac],XmNdialogType,XmDIALOG_MESSAGE); ac++;
 XtSetArg(al[ac],XmNmessageString,str1); ac++;
 str2=XmStringCreate("Continue", XmSTRING_DEFAULT_CHARSET);
 XtSetArg(al[ac],XmNokLabelString,str2); ac++;
 XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL); ac++;

 Check_msg=XmCreateMessageDialog(Phonebook,"Check_msg",al,ac);
 XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
     XmDIALOG_HELP_BUTTON));

 XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
     XmDIALOG_CANCEL_BUTTON));
 XtManageChild(Check_msg);

} /* end of Check_cbproc */
```

14 int Check( )

```
int Check()
{
 int i,exist;

 GetString(Check_data.name,Upd_Name_txt);
 GetString(Check_data.f_name,Upd_F_Name_txt);

 fp1=fopen("phonefile","r");
 exist=0;
```

```
while(   (exist==0)
      && (fgets(name,80,fp1)!=NULL)
      && (fgets(f_name,80,fp1)!=NULL)  )
{
 if  ((strcmp(name,Check_data.name)==0)
    && (strcmp(f_name,Check_data.f_name)==0))
   {
    exist=1;
   }
 else for (i=1;i<=7;i++) fgets(line,80,fp1);
};

fclose(fp1);

if ((strcmp("\n",Check_data.name)==0)
   ||(strcmp("\n",Check_data.f_name)==0).) exist = -1;

return(exist);

} /* end of Check */
```

15   void Insert_cbproc(w,client_data,call_data)

```
void Insert_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 Widget Check_msg;
 XmString str1,str2;
 int exist;

 GetString(Insert_data.name,Upd_Name_txt);
 GetString(Insert_data.f_name,Upd_F_Name_txt);
 GetString(Insert_data.initials,Upd_Initials_txt);
 GetString(Insert_data.phone,Upd_Phone_txt);
 GetString(Insert_data.location,Upd_Location_txt);
 GetString(Insert_data.workgroup,Upd_Workgroup_txt);
 GetString(Insert_data.service,Upd_Service_txt);
 GetString(Insert_data.respons,Upd_Respons_txt);
 GetString(Insert_data.special,Upd_Special_txt);

 exist=Check();
 if (exist==0) Insert();
 if (exist==1)
   {
    str1=XmStringCreate("Exists",XmSTRING_DEFAULT_CHARSET);
    ac = 0;
    XtSetArg(al[ac],XmNdialogType,XmDIALOG_MESSAGE); ac++;
    XtSetArg(al[ac],XmNmessageString,str1); ac++;
    str2=XmStringCreate("Continue", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(al[ac],XmNokLabelString,str2); ac++;
    XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL);
    ac++;

    Check_msg=XmCreateMessageDialog(Phonebook,"Check_msg",al,ac);
    XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
        XmDIALOG_HELP_BUTTON));
    XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
       XmDIALOG_CANCEL_BUTTON));
    XtManageChild(Check_msg);
   }

} /* end of Insert_cbproc */
```

```
void Insert()
{
        int inserted;

        fp1=fopen("phonefile","r");
        fp2=fopen("phonetemp","w");

        inserted=0;

        while (fgets(line1,80,fp1)!=NULL)
                {
                fgets(line2,80,fp1);
                fgets(line3,80,fp1);
                fgets(line4,80,fp1);
                fgets(line5,80,fp1);
                fgets(line6,80,fp1);
                fgets(line7,80,fp1);
                fgets(line8,80,fp1);
                fgets(line9,80,fp1);

                if ((((strcmp(line1,Insert_data.name  )>=0)
                   &&(strcmp(line2,Insert_data.f_name )>0))

                   || ((strcmp(line1,Insert_data.name    )>0)
                   &&(strcmp(line2,Insert_data.f_name)>=0)))



                &&(inserted==0))
                        {
                        fputs(Insert_data.name,fp2);
                        fputs(Insert_data.f_name,fp2);
                        fputs(Insert_data.initials,fp2);
                        fputs(Insert_data.phone,fp2);
                        fputs(Insert_data.location,fp2);
                        fputs(Insert_data.workgroup,fp2);
                        fputs(Insert_data.service,fp2);
                        fputs(Insert_data.respons,fp2);
                        fputs(Insert_data.special,fp2);
                        inserted=1;
                        };
                fputs(line1,fp2);
                fputs(line2,fp2);
                fputs(line3,fp2);
                fputs(line4,fp2);
                fputs(line5,fp2);
                fputs(line6,fp2);
                fputs(line7,fp2);
                fputs(line8,fp2);
                fputs(line9,fp2);
                };

        if  (inserted==0)
                {
                fputs(Insert_data.name,fp2);
                fputs(Insert_data.f_name,fp2);
                fputs(Insert_data.initials,fp2);
                fputs(Insert_data.phone,fp2);
                fputs(Insert_data.location,fp2);
                fputs(Insert_data.workgroup,fp2);
                fputs(Insert_data.service,fp2);
                fputs(Insert_data.respons,fp2);
                fputs(Insert_data.special,fp2);
                inserted=1;
                };
```

```
      fclose(fp1);
      fclose(fp2);

      system("cp phonetemp phonefile");

} /* end of Insert */
```

17   void Modify_cbproc(w,client_data,call_data)

```
void Modify_cbproc(w,client_data,call_data)
Widget w;
int client_data;
XmAnyCallbackStruct *call_data;
{
Widget Check_msg;
XmString str1,str2;
int exist;

GetString(Modify_data.name,Upd_Name_txt);
GetString(Modify_data.f_name,Upd_F_Name_txt);
GetString(Modify_data.initials,Upd_Initials_txt);
GetString(Modify_data.phone,Upd_Phone_txt);
GetString(Modify_data.location,Upd_Location_txt);
GetString(Modify_data.workgroup,Upd_Workgroup_txt);
GetString(Modify_data.service,Upd_Service_txt);
GetString(Modify_data.respons,Upd_Respons_txt);
GetString(Modify_data.special,Upd_Special_txt);

exist=Check();
if (exist==1) Modify();
if (exist==0)
  {
    str1=XmStringCreate("Does not exist",XmSTRING_DEFAULT_CHARSET);
    ac = 0;
    XtSetArg(al[ac],XmNdialogType,XmDIALOG_MESSAGE); ac++;
    XtSetArg(al[ac],XmNmessageString,str1); ac++;
    str2=XmStringCreate("Continue", XmSTRING_DEFAULT_CHARSET);
    XtSetArg(al[ac],XmNokLabelString,str2); ac++;
    XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL);
    ac++;

    Check_msg=XmCreateMessageDialog(Phonebook,"Check_msg",al,ac);
    XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
        XmDIALOG_HELP_BUTTON));
    XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
        XmDIALOG_CANCEL_BUTTON));
    XtManageChild(Check_msg);
  }

} /* end of Modify_cbproc */
```

18   void Modify( )

```
void Modify()
{
int modified;

fp1=fopen("phonefile","r");
fp2=fopen("phonetemp","w");

modified=0;
```

```
while (fgets(line1,80,fp1)!=NULL)
  {
   fgets(line2,80,fp1);
   fgets(line3,80,fp1);
   fgets(line4,80,fp1);
   fgets(line5,80,fp1);
   fgets(line6,80,fp1);
   fgets(line7,80,fp1);
   fgets(line8,80,fp1);
   fgets(line9,80,fp1);

   if  ((strcmp(line1,Modify_data.name)==0)
       &&(strcmp(line2,Modify_data.f_name)==0)
       &&(modified==0))
   {
    fputs(Modify_data.name,fp2);
    fputs(Modify_data.f_name,fp2);
    fputs(Modify_data.initials,fp2);
    fputs(Modify_data.phone,fp2);
    fputs(Modify_data.location,fp2);
    fputs(Modify_data.workgroup,fp2);
    fputs(Modify_data.service,fp2);
    fputs(Modify_data.respons,fp2);
    fputs(Modify_data.special,fp2);
    modified=1;
   }
   else  {
    fputs(line1,fp2);
    fputs(line2,fp2);
    fputs(line3,fp2);
    fputs(line4,fp2);
    fputs(line5,fp2);
    fputs(line6,fp2);
    fputs(line7,fp2);
    fputs(line8,fp2);
    fputs(line9,fp2);
   };
  };

 fclose(fp1);
 fclose(fp2);

 system("cp phonetemp phonefile");

} /* end of Modify */
```

19 void Delete_cbproc(w,client_data,call_data)

```
void Delete_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 Widget Check_msg;
 XmString str1,str2;
 int exist;

 GetString(Delete_data.name,Upd_Name_txt);
 GetString(Delete_data.f_name,Upd_F_Name_txt);
```

```
    exist=Check();
    if (exist==1) Delete_proc();
    if (exist==0)
      {
        str1=XmStringCreate("Does not exist",XmSTRING_DEFAULT_CHARSET);
        ac = 0;
        XtSetArg(al[ac],XmNdialogType,XmDIALOG_MESSAGE); ac++;
        XtSetArg(al[ac],XmNmessageString,str1); ac++;
        str2=XmStringCreate("Continue", XmSTRING_DEFAULT_CHARSET);
        XtSetArg(al[ac],XmNokLabelString,str2); ac++;
        XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL);
        ac++;

        Check_msg=XmCreateMessageDialog(Phonebook,"Check_msg",al,ac);
        XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
            XmDIALOG_HELP_BUTTON));
        XtUnmanageChild(XmMessageBoxGetChild(Check_msg,
            XmDIALOG_CANCEL_BUTTON));
        XtManageChild(Check_msg);
      }

} /* end of Delete_cbproc */
```

20 void Delete_proc( )

```
void Delete_proc()
{
 int deleted;

 fp1=fopen("phonefile","r");
 fp2=fopen("phonetemp","w");

 deleted=0;

 while (fgets(line1,80,fp1)!=NULL)
   {
     fgets(line2,80,fp1);
     fgets(line3,80,fp1);
     fgets(line4,80,fp1);
     fgets(line5,80,fp1);
     fgets(line6,80,fp1);
     fgets(line7,80,fp1);
     fgets(line8,80,fp1);
     fgets(line9,80,fp1);

     if  ((strcmp(line1,Delete_data.name)!=0)
        ||(strcmp(line2,Delete_data.f_name)!=0)
        ||(deleted==1))
     {
       fputs(line1,fp2);
       fputs(line2,fp2);
       fputs(line3,fp2);
       fputs(line4,fp2);
       fputs(line5,fp2);
       fputs(line6,fp2);
       fputs(line7,fp2);
       fputs(line8,fp2);
       fputs(line9,fp2);
     }
     else deleted=1;
   };

 fclose(fp1);
 fclose(fp2);
```

```
system("cp phonetemp phonefile");
} /* end of Delete */
```

21 void Kill_cbproc(w,client_data,call_data)

```
void Kill_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 Widget Kill_confirm;
 XmString str1,str2;

 ac = 0;
 XtSetArg(al[ac],XmNdialogType,XmDIALOG_QUESTION); ac++;

 str1=XmStringCreate("Do you want to quit ?",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(al[ac],XmNmessageString,str1); ac++;

 str2=XmStringCreate("Quit", XmSTRING_DEFAULT_CHARSET);
 XtSetArg(al[ac],XmNokLabelString,str2); ac++;

 XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL); ac++;
 XtSetArg(al[ac],XmNokCallback,Kill_ok_cbproc); ac++;

 Kill_confirm=XmCreateQuestionDialog(Phonebook,"Kill_confirm",al,ac);
 XtUnmanageChild(XmMessageBoxGetChild(Kill_confirm,
      XmDIALOG_HELP_BUTTON));
 XtManageChild(Kill_confirm);

} /* end of Kill_cbproc */
```

22 void Kill_ok_cbproc(w,client_data,call_data)

```
void Kill_ok_cbproc(w,client_data,call_data)
 Widget w;
 int client_data;
 XmAnyCallbackStruct *call_data;
{
 exit(0);

} /* end of Kill_ok_cbproc */
```

23 void Quit_cbproc(w,w_to_kill,call_data)

```
void Quit_cbproc(w,w_to_kill,call_data)
 Widget w;
 Widget w_to_kill;
 XmAnyCallbackStruct *call_data;
{
 Widget Quit_confirm;
 XmString str1,str2;

 ac = 0;
 XtSetArg(al[ac],XmNdialogType,XmDIALOG_QUESTION); ac++;

 str1=XmStringCreate("Do you want to quit ?",XmSTRING_DEFAULT_CHARSET);
 XtSetArg(al[ac],XmNmessageString,str1); ac++;
```

```
str2=XmStringCreate("Quit", XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNokLabelString,str2); ac++;

XtSetArg(al[ac],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL); ac++;

Quit_confirm=XmCreateQuestionDialog(Phonebook,"Quit_confirm",al,ac);
XtUnmanageChild(XmMessageBoxGetChild(Quit_confirm,
     XmDIALOG_HELP_BUTTON));
         XtAddCallback(XmMessageBoxGetChild(Quit_confirm,XmDIALOG_OK_BUTTON),
         XmNactivateCallback,
         Quit_ok_cbproc,
         w_to_kill);
XtManageChild(Quit_confirm);

} /* end of Quit_cbproc */
```

24   void Quit_ok_cbproc(w,w_to_kill,call_data)

```
void Quit_ok_cbproc(w,w_to_kill,call_data)
 Widget w;
 Widget w_to_kill;
 XmAnyCallbackStruct *call_data;
{
 XtDestroyWidget(w_to_kill);

} /* end of Quit_ok_cbproc */

/* END OF PROGRAM PHONEBOOK */
```

A2. XView Application Code

1 DECLARATIONS

```c
#include <string.h>
#include <stdio.h>
#include <xview/xview.h>
#include <xview/frame.h>
#include <xview/panel.h>
#include <xview/textsw.h>
#include <xview/notice.h>

void Init();
void Name_proc();
void Firstname_proc();
void Specializ_proc();
Keysproc();
void Search_ntproc();
void Search();
void Adapt1();
int Find();
void Settable();
void Erase_ntproc();
void Quit_ntproc();
Allinfo_proc();
void Quit_all_ntproc();

void Activities_proc();
void Insert_proc();
void Modify_proc();
void Delete_proc();
void Next_ntproc();
void Previous_ntproc();

static Frame name_frame = 0 ;
static Panel_item name_tpi ;

static Frame fname_frame = 0 ;
static Panel_item fname_tpi ;

static Frame spec_frame = 0 ;
static Panel_item spec_tpi ;

static Frame allinfo_frame = 0 ;
static Panel_item allinfo_name_tpi ;
static Panel_item allinfo_fname_tpi ;
static Panel_item allinfo_init_tpi ;
static Panel_item allinfo_phnum_tpi ;
static Panel_item allinfo_loc_tpi ;
static Panel_item allinfo_wg_tpi ;
static Panel_item allinfo_ser_tpi ;
static Panel_item allinfo_resp_tpi ;
static Panel_item allinfo_spec_tpi ;

static Frame acti_frame = 0 ;

static Frame insert_frame = 0 ;
static Panel_item insert_name_tpi ;
static Panel_item insert_fname_tpi ;
static Panel_item insert_init_tpi ;
static Panel_item insert_phnum_tpi ;
static Panel_item insert_loc_tpi ;
static Panel_item insert_wg_tpi ;
static Panel_item insert_ser_tpi ;
static Panel_item insert_resp_tpi ;
static Panel_item insert_spec_tpi ;
```

```
static Frame modify_frame = 0 ;
static Panel_item modify_name_tpi ;
static Panel_item modify_fname_tpi ;
static Panel_item modify_init_tpi ;
static Panel_item modify_phnum_tpi ;
static Panel_item modify_loc_tpi ;
static Panel_item modify_wg_tpi ;
static Panel_item modify_ser_tpi ;
static Panel_item modify_resp_tpi ;
static Panel_item modify_spec_tpi ;

static Frame delete_frame = 0 ;
static Panel_item delete_name_tpi ;
static Panel_item delete_fname_tpi ;
static Panel_item delete_init_tpi ;
static Panel_item delete_phnum_tpi ;
static Panel_item delete_loc_tpi ;
static Panel_item delete_wg_tpi ;
static Panel_item delete_ser_tpi ;
static Panel_item delete_resp_tpi ;
static Panel_item delete_spec_tpi ;

char table[3][200][9][30];

char line1[30],line2[30],line3[30],line4[30],line5[30],
     line6[30],line7[30],line8[30],line9[30],line[30];

int nbmax[3],nelem[3];

FILE *fopen(),*fp1,*fp2;
```

2 main (argc,argv)

```
main (argc,argv)
 int argc;
 char *argv[];
{
 Frame mainframe;
 Panel panel;
 Menu menuconsult,menulists,menukeys,menuupdate;

 xv_init (XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

 mainframe = (Frame)xv_create (NULL,FRAME,
   FRAME_LABEL, "BIM Internal phone book",
   NULL);

 menukeys = (Menu)xv_create (NULL,MENU,
   MENU_TITLE_ITEM, "Keys",
   MENU_ACTION_ITEM, "Name", Name_proc,
   MENU_ACTION_ITEM, "First name", Firstname_proc,
   MENU_ACTION_ITEM, "Specialization", Specializ_proc,
   NULL);

 menulists = (Menu)xv_create (NULL,MENU,
   MENU_TITLE_ITEM, "Lists",
   MENU_ACTION_ITEM, "All info", Allinfo_proc,
   MENU_ACTION_ITEM, "Activities", Activities_proc,
   NULL);

 menuconsult = (Menu)xv_create (NULL, MENU,
   MENU_TITLE_ITEM, "Consult",
   MENU_PULLRIGHT_ITEM, "Keys", menukeys,
   MENU_PULLRIGHT_ITEM, "Lists", menulists,
   NULL);
```

```
menuupdate = (Menu)xv_create (NULL,MENU,
  MENU_TITLE_ITEM, "Update",
  MENU_ACTION_ITEM, "Insert", Insert_proc,
  MENU_ACTION_ITEM, "Modify", Modify_proc,
  MENU_ACTION_ITEM, "Delete", Delete_proc,
  NULL);

panel = (Panel)xv_create (mainframe, PANEL, NULL);

(void) xv_create (panel,PANEL_BUTTON,
  PANEL_LABEL_STRING,  "Consult",
  PANEL_ITEM_MENU, menuconsult,
  NULL);

(void) xv_create (panel,PANEL_BUTTON,
  PANEL_LABEL_STRING,  "Update",
  PANEL_ITEM_MENU,  menuupdate,
  NULL);

(void) xv_create (panel,PANEL_BUTTON,
  PANEL_LABEL_STRING,  "Quit",
  PANEL_NOTIFY_PROC,  Quit_all_ntproc,
  NULL);

      window_fit(panel);
      window_fit(mainframe);

Init;

xv_main_loop (mainframe);

} /* end of main */
```

3 void Init( )

```
void Init()
{
    fp1=fopen("phobefile","r");
    if (fp1==NULL)
        {
  fp1=fopen("phonefile","w");
  fclose(fp1);
      };
} /* end of Init */
```

4 void Name_proc( )

```
void Name_proc()
{ Keys_proc(1);
}
```

5 void Firstname_proc( )

```
void Firstname_proc()
{ Keys_proc(2);
}
```

6 void Specializ_proc( )

```
void Specializ_proc()
{ Keys_proc(3);
}
```

7 Keys_proc (type)

```
Keys_proc(type)
 int type;
{
        Panel panel;
        Frame frame;
 Panel_item tpi;
        if ((type==1) && (name_frame!=0))
{
            xv_set(name_frame,XV_SHOW,TRUE);
            return(0);
        }

        if ((type==2) && (fname_frame!=0))
{
            xv_set(fname_frame,XV_SHOW,TRUE);
            return(0);
        }

        if ((type==3) && (spec_frame!=0))
{
            xv_set(spec_frame,XV_SHOW,TRUE);
            return(0);
        }
 frame = (Frame)xv_create (NULL,FRAME,
  XV_SHOW, TRUE,
                XV_KEY_DATA, 22, type,
  NULL);

 panel = (Panel)xv_create (frame,PANEL,
                XV_KEY_DATA, 22, type,
  NULL);

 xv_create (panel, PANEL_BUTTON,
  PANEL_LABEL_STRING, "Search",
  PANEL_NOTIFY_PROC, Search-ntproc,
  PANEL_ITEM_X,   20,
  PANEL_ITEM_Y,   20,
                XV_KEY_DATA, 22, type,
  NULL);
 xv_create (panel, PANEL_BUTTON,
  PANEL_LABEL_STRING, "Erase ",
  PANEL_NOTIFY_PROC, Erase-ntproc,
  PANEL_ITEM_X,   90,
  PANEL_ITEM_Y,   20,
                XV_KEY_DATA, 22, type,
  NULL);
 xv_create (panel, PANEL_BUTTON,
  PANEL_LABEL_STRING, " Quit ",
  PANEL_NOTIFY_PROC, Quit_ntproc,
  PANEL_ITEM_X,   160,
  PANEL_ITEM_Y,   20,
                XV_KEY_DATA, 22, type,
  NULL);
```

4

```
tpi = xv_create (panel, PANEL_TEXT,
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   60,
               XV_KEY_DATA, 22, type,
 NULL);

        switch(type)
{
        case 1 /* name */ :
                xv_set(frame,FRAME_LABEL,"Consult : Name",NULL);
                xv_set(tpi,PANEL_LABEL_STRING,"Name : ",NULL);
                name_frame = frame ;
                name_tpi = tpi ;
                break;
        case 2 /* firstname */ :
                xv_set(frame,FRAME_LABEL,"Consult : Firstname",NULL);
                xv_set(tpi,PANEL_LABEL_STRING,"Firstname : ",NULL);
                fname_frame = frame ;
                fname_tpi = tpi ;
                break;
        case 3 /* specialization */ :
                xv_set(frame,FRAME_LABEL,"Consult : Specialization",NULL);
                xv_set(tpi,PANEL_LABEL_STRING,"Specialization : ",NULL);
                spec_frame = frame ;
                spec_tpi = tpi ;
                break;
        }
        window_fit(panel);
        window_fit(frame);

} /* end of Keys_proc */
```

8   void Search-ntproc(item, event)

```
void Search-ntproc(item, event)
   Panel_item      item;
   Event           *event;
{
 Frame frame;
 Panel panel;
 Panel_item tpi;
 int type,result;
 char *key;

   type = xv_get(item,XV_KEY_DATA,22);

 switch(type)
 {
        case 1 /* name */ : key = (char *)xv_get(name_tpi,PANEL_VALUE);
        search(key,1); break;
        case 2 /* firstname */ : key = (char *)xv_get(fname_tpi,PANEL_VALUE);
            search(key,2);break;
        case 3 /* specialization */ : key= (char *)xv_get(spec_tpi,PANEL_VALUE);
            search(key,3); break;
 }

 panel = (Panel)xv_get(item,PANEL_PARENT_PANEL);

 if (nbmax[type]==0)
   result = notice_prompt(panel,NULL,
   NOTICE_FOCUS_XY,   event_x(event),event_y(event),
   NOTICE_MESSAGE_STRINGS,  "No such value",NULL,
   NOTICE_BUTTON_YES,  "Continue",
   NULL);
```

```
    if (nbmax[type]>200)
     nbmax[type]=200;
    if (nbmax[type]>0)
     {
       nelem[type]=0;
     }
} /* end of Search_ntproc */
```

9 void Search(key,type)

```
void Search(key,type)
 char *key;
 int type;
{
 int i,f;

 fpl=fopen("phonefile","r");
 i=0;
 while(fgets(line1,80,fpl)!=NULL)
   {
     Adapt1(line1);

     fgets(line2,80,fpl);
     Adapt1(line2);

     fgets(line3,80,fpl);
     Adapt1(line3);

     fgets(line4,80,fpl);
     Adapt1(line4);

     fgets(line5,80,fpl);
     Adapt1(line5);

     fgets(line6,80,fpl);
     Adapt1(line6);

     fgets(line7,80,fpl);
     Adapt1(line7);

     fgets(line8,80,fpl);
     Adapt1(line8);

     fgets(line9,80,fpl);
     Adapt1(line9);

     if (((type==1)&&(strcmp(line1,key)==0))
        ||((type==2)&&(strcmp(line2,key)==0))
        ||((type==3)&&((f=Find(line9,key))==1)))
     {
       Settable(type,i,0,line1);
       Settable(type,i,1,line2);
       Settable(type,i,2,line3);
       Settable(type,i,3,line4);
       Settable(type,i,4,line5);
       Settable(type,i,5,line6);
       Settable(type,i,6,line7);
       Settable(type,i,7,line8);
       Settable(type,i,8,line9);
       i++;
     };
   };
 fclose(fpl);
 nbmax[type]=i;

} /* end of Search */
```

10 void Adapt1(data_pointer)

```
void Adapt1(data_pointer)
 char *data_pointer;
{
 int i;

 i=1;
 while ((*data_pointer!='\0')
      &&(*data_pointer!='\n')
      &&(i<31))
   {
   data_pointer++;
   i++;
   }.
 *data_pointer='\0';

} /* end of Adapt1 */
```

11 int Find(str,substr)

```
int Find(str,substr)
 char *str;
 char *substr;
{
 int found;

 if (*str==*substr)
     if (*str=='\0')
         return(1);
      else {
      str++;
      substr++;
      found=Find(str,substr);
      }
 else if ((*str=='\0')||(*substr=='\0'))
         return(0);
      else {
      str++;
      found=Find(str,substr);
      };

} /* end of Find */
```

12 void Settable(t,i,j,line)

```
void Settable(t,i,j,line)
 int t;
 int i;
 int j;
 char *line;
{
 int k;

 k=0;
 while (*line!='\0')
  table[t][i][j][k++]=*line++;
 table[t][i][j][k++]='\0';

} /* end of Settable */
```

13 void Erase_ntproc(button, event)

```
void Erase_ntproc(button, event)
   Panel_item       button;
   Event            *event;
{
 int type;

    type = xv_get(button,XV_KEY_DATA,22);

 switch(type)
 {
        case 1 /* name */ : xv_set(name_tpi,PANEL_VALUE,"",NULL);break;
        case 2 /* firstname */ : xv_set(fname_tpi,PANEL_VALUE,"",NULL);break;
        case 3 /* specialization */:xv_set(spec_tpi,PANEL_VALUE,"",NULL);break;
        case 6 /* insert */ :
   xv_set(insert_name_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_fname_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_init_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_phnum_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_loc_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_wg_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_ser_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_resp_tpi,PANEL_VALUE,"",NULL);
   xv_set(insert_spec_tpi,PANEL_VALUE,"",NULL);break;
        case 7 /* modify */ :
   xv_set(modify_name_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_fname_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_init_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_phnum_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_loc_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_wg_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_ser_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_resp_tpi,PANEL_VALUE,"",NULL);
   xv_set(modify_spec_tpi,PANEL_VALUE,"",NULL);break;
        case 8 /* delete */ :
   xv_set(delete_name_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_fname_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_init_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_phnum_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_loc_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_wg_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_ser_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_resp_tpi,PANEL_VALUE,"",NULL);
   xv_set(delete_spec_tpi,PANEL_VALUE,"",NULL);break;
 }
} /* end of Erase_ntproc */
```

14 void Quit_ntproc(button, event)

```
void Quit_ntproc(button, event)
   Panel_item       button;
   Event            *event;
{
 int type;
   int action ;
```

```
        type = xv_get(button,XV_KEY_DATA,22);
        switch(type)
    {
        case 1 : xv_destroy_safe(name_frame); name_frame = 0 ; break ;
        case 2 : xv_destroy_safe(fname_frame); fname_frame = 0 ;break ;
        case 3 : xv_destroy_safe(spec_frame);spec_frame=0; break ;
        case 4 : xv_destroy_safe(allinfo_frame);allinfo_frame=0; break ;
        case 5 : xv_destroy_safe(acti_frame);acti_frame=0; break ;
        case 6 : xv_destroy_safe(insert_frame);insert_frame=0; break ;
        case 7 : xv_destroy_safe(modify_frame);modify_frame=0; break ;
        case 8 : xv_destroy_safe(delete_frame);delete_frame=0; break ;
        }

} /* end of Quit_ntproc */
```

15 Allinfo_proc( )

```
Allinfo_proc()
{
 Panel panel;

        if (allinfo_frame!=0)
    {
            xv_set(allinfo_frame,XV_SHOW,TRUE);
            return(0);
        }
 allinfo_frame = (Frame)xv_create (NULL,FRAME,
  FRAME_LABEL, "Consult : All information",
  XV_SHOW, TRUE,
                XV_KEY_DATA, 22, 4,
  NULL);

 panel = (Panel)xv_create (allinfo_frame,PANEL,
                XV_KEY_DATA, 22, 4,
  NULL);

 xv_create (panel, PANEL_BUTTON,
  PANEL_LABEL_STRING, "Next",
  PANEL_NOTIFY_PROC, Next_ntproc,
  PANEL_ITEM_X,   20,
  PANEL_ITEM_Y,   20,
                XV_KEY_DATA, 22, 4,
  NULL);
 xv_create (panel, PANEL_BUTTON,
  PANEL_LABEL_STRING, "Previous ",
  PANEL_NOTIFY_PROC, Previous_ntproc,
  PANEL_ITEM_X,   80,
  PANEL_ITEM_Y,   20,
                XV_KEY_DATA, 22, 4,
  NULL);
 xv_create (panel, PANEL_BUTTON,
  PANEL_LABEL_STRING, " Quit ",
  PANEL_NOTIFY_PROC, Quit_ntproc,
  PANEL_ITEM_X,   185,
  PANEL_ITEM_Y,   20,
                XV_KEY_DATA, 22, 4,
  NULL);
```

```
allinfo_name_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "Name              ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   60,
            XV_KEY_DATA, 22, 4,
 NULL);

allinfo_fname_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "First name       ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   90,
            XV_KEY_DATA, 22, 4,
 NULL);

allinfo_init_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "Initials         ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   120,
            XV_KEY_DATA, 22, 4,
 NULL);

allinfo_phnum_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "Phone number      ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,.  150,
            XV_KEY_DATA, 22, 4,
 NULL);

allinfo_loc_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "Location          ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   180,
            XV_KEY_DATA, 22, 4,
 NULL);

allinfo_wg_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "Workgroup         ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   210,
            XV_KEY_DATA, 22, 4,
 NULL);

allinfo_ser_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "Service           ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   240,
            XV_KEY_DATA, 22, 4,
 NULL);

allinfo_resp_tpi = xv_create (panel, PANEL_TEXT,
 PANEL_LABEL_STRING, "Responsibilities  ",
 PANEL_VALUE_DISPLAY_LENGTH, 30,
 PANEL_ITEM_X,   20,
 PANEL_ITEM_Y,   270,
            XV_KEY_DATA, 22, 4,
 NULL);
```

```
allinfo_spec_tpi = xv_create (panel, PANEL_TEXT,
  PANEL_LABEL_STRING, "Specialization    ",
  PANEL_VALUE_DISPLAY_LENGTH, 30,
  PANEL_ITEM_X,   20,
  PANEL_ITEM_Y,   300,
              XV_KEY_DATA, 22, 4,
  NULL);

        window_fit(panel);
        window_fit(allinfo_frame);

} /* end of Allinfo_proc */
```

16 void Activities_proc( )

```
void Activities_proc()
{
}
```

17 void Insert_ntproc( )

```
void Insert_ntproc()
{
}
```

18 void Modify_ntproc( )

```
void Modify_ntproc()
{
}
```

19 void Delete_ntproc ( )

```
void Delete_ntproc()
{
}
```

20 void Next_ntproc( )

```
void next_ntproc()
{
}
```

21 void Previous_ntproc( )

```
void Previous_ntproc()
{
}
```

22   void Quit_all_nt( )

```
void Quit_all_nt()
{
 exit(0);
}
/* END OF PROGRAM PHONEBOOK */
```