NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN ABSTRACT INTERACTIVE GRAPHICS INTERFACE
FOR THE IBM/PC AND MACINTOSH

by

Liang, Ko-Hsin

June 1988

Thesis Advisor: Daniel Davis

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (*If applicable*) Code 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000	7b ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (*If applicable*)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (*Include Security Classification*)

AN ABSTRACT INTERACTIVE GRAPHICS INTERFACE FOR THE IBM/PC AND MACINTOSH

12 PERSONAL AUTHOR(S)
Liang, Ko-Hsin

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 June	181

16 SUPPLEMENTARY NOTATION
The views expresses in this thesis are thoses of the author abd do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	

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

Different computer systems have different programming environments in spite of their similar capabilities. GEM and Macintosh software systems both provide an operating environment in which the users can ulitize all kinds of functions and routines to produce a user-friendly application program. Unfortunately, the programmers have to repeat the learning procedure and recode the source works if for some reason the application program is needed to run on both IBM PC and Macintosh microcomputers. In this thesis, a common interface is provided for programmers to reduce duplicated efforts and hopefully to get the same effect both operating environments.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Daniel Davis	22b TELEPHONE (*Include Area Code*) (408)646-3091 22c OFFICE SYMBOL Code 52Dv

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

Approved for public release; distribution is unlimited.

An Abstract Interactive Graphics Interface for the IBM/PC and Macintosh

by

Liang, Ko-Hsin
Lieutenant, Taiwan Navy
B.S., Chinese Naval Academy, 1984

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1988

ABSTRACT

Different computer systems have different programming environments in spite of their similar capabilities. GEM and Macintosh software system both provide an operating environment in which the users can utilize all kinds of functions and routines to produce a user-friendly application program. Unfortunately, the programmers have to repeat the learning procedure and recode the source works if for some reason the application program is needed to run on both IBM PC and Macintosh microcomputers. In this thesis, a common interface is provided for programmers to reduce duplicated efforts and hopefully to get the same effect in both operating environments.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Different computer systems have different programming environments in spite of their similar capabilities. Some system functions, utilized through programming language compilers, work in the environments supported by software production, or by system hardware. Although the environments of software development support similar algorithms and tools, they usually make software programmers write another program to obtain the same result from different computer systems. There is no standard interface for the various workstation (SUN, APOLLO, etc.) systems.

A. PURPOSE OF THESIS

In this thesis, a common interface for a graphic software environment is established to create systematical functions which can be used for two different personal computing systems: Apple's Macintosh* and IBM PC series. The most important method used here to obtain this common interface is the Abstract Specification of data types, also named Abstract Data Type, consisting of a set of instances and a set of primitive operations which provide the only means for creating and interacting with the instances. The advantages of the Abstract Data Type, such as precise specification, modularity, and information hiding, can be very helpful for implementing the interfaces easily and with less errors. [Ref. 4, p. 18-19]

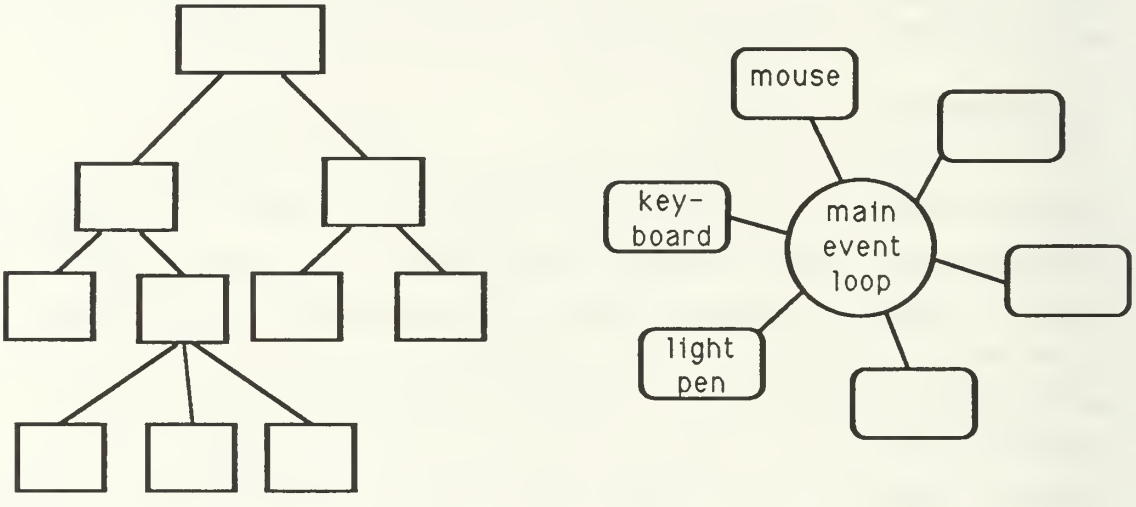
The development environment selected here is a graphics based software system that supports both window management and a menu driven style. The system is more user friendly and it becomes a definite trend toward the development of computer workstation systems because using the visual effects of graphics can generally communicate information more effectively than text. Menu displays save people the trouble of remembering many complex operation commands. The structure for user friendly system is different from the traditional structure of software (see Figure 1). The traditional software system is a kind of hierarchical structure that needs top-down approach to implement a program. The user friendly system needs a circular polling devices like mouse, keyboard, floppy disk drive, etc.

---

* Macintosh is a trademark of Apple Computer, Inc.

B. TOOLS

The primary development compiler and system language in this thesis is the C language. The C language is used primarily because it is easily ported to new systems and it allows the user to access his resources directly. For the Macintosh computer, LightspeedC™ (by THINK Technologies, Inc.) is used, and LATTICE C™ is used for the IBM PC computer with GEM (Graphics Environment Manager) which will be described later.



Traditional structure of software Structure for user friendly system

Figure 1 Different Structure of Software Systems

II. PROGRAMMING ENVIRONMENT (User Interface Technology)

The operation and control procedures should be simple for the user to use the computer comfortably. A user-friendly system should provide all the information needed by the user in a graphics display. These graphic displays are referred to here as **desktop**. On the desktop, the user can slide documents around, organize work in folders, throw things away, or obtain new work—simply by moving the mouse and pressing the mouse button. The Macintosh Operating System supports such an operating and programming environment on Macintosh Computer [Ref. 2], and GEM provides a comparable environment for the IBM PC. GEM, developed by Digital Research, Inc.(DRI), is an operating environment which is similar to an operating system [Ref. 1]. Whereas an operating system allows the program to utilize console and disk devices in a standard manner, the GEM operating environment allows the GEM programmer to control a number of graphics devices and develop application interfaces in a consistent and standard fashion [Ref. 1]. So, these two environments allow a variety of high-level functions access to peripheral graphic devices and whose purpose is to make it easier for the application programmer to develop software that is both efficient and easy to use. In fact, the developed software is very similar to the window-type structure used in Macintosh software system, which is rather user-friendly in today's software development. Figure 2 shows the relationship between the application program, the user, and the computer [Ref. 1, p 4].

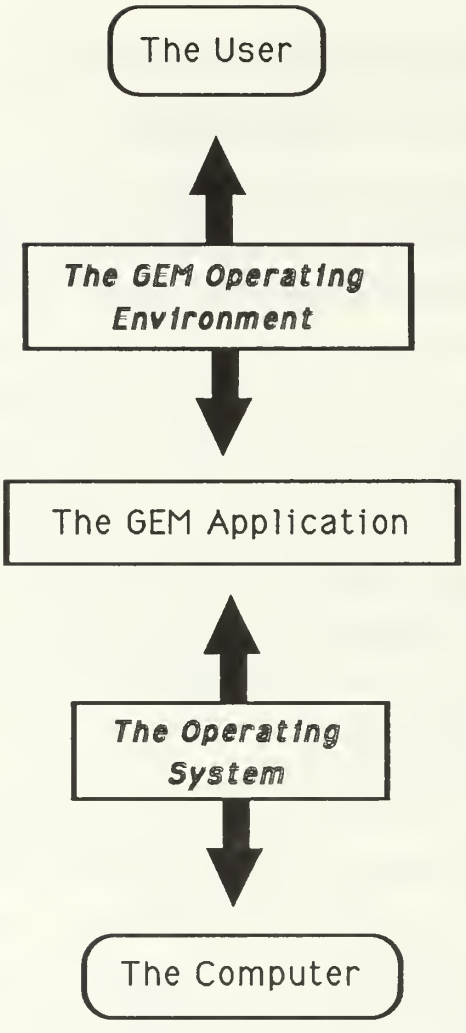


Figure 2 The Role of the GEM Operating Environment

With the GEM functions, the application program can control many devices manipulated by the user including the keyboard, the mouse, the screen, the printer, and the plotter [Ref. 1]. GEM is very similar to an operating system in that it allows the user to write programs without having to worry about what kind of mouse is attached to the computer, what resolution the screen has, or whether the computer's monitor is color or monochrome [Ref. 1].

Another example of a programming environment is the Operating System and the User Interface Toolbox in Macintosh [Ref. 2]. The application program will always call the routines which mostly are part of either the Operating System or the User Interface Toolbox and in the Macintosh ROM. The Operating System is at the lowest level; it does basic tasks such as input and output, memory management, and interrupt handling. The User Interface Toolbox is a level above the Operating System; it helps you implement the standard

Macintosh user interface in the application program [Ref. 3]. The user interface is the most important part of the user friendly computer system. In plain English, an interface is a junction or boundary where two things meet. In computerese, it refers to the set of rules and conventions by which one part of an organized system communicates with another. Whenever two components of the system come together, they exchange information by way of an interface [Ref. 3].

GEM and Macintosh software system both provide an operating environment in which the users can utilize all kinds of functions and routines to produce a user-friendly application program. Unfortunately, the programmers have to repeat the learning procedure and recode the source works if for some reason the application program is needed to run on both IBM PC and Macintosh microcomputers. In this thesis, a common interface is provided for programmers to reduce duplicated efforts and hopefully to get the same effect in both operating environments. The relationship between this common interface, the user, and the computers is shown in Figure 3.

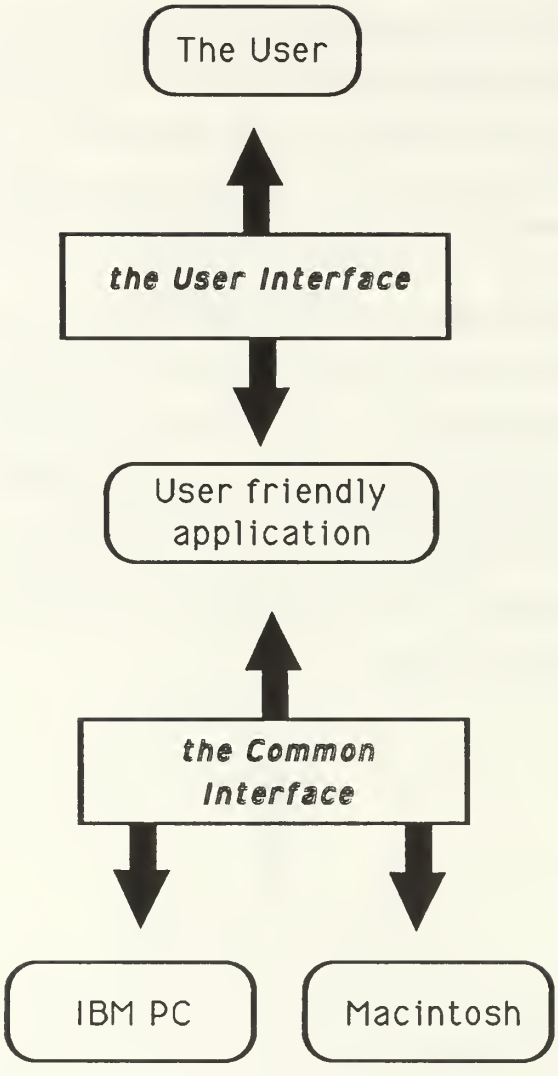


Figure 3 The Role of Common Interface

III. OVERVIEW OF GEM

The common interface mentioned last section actually consists of one interface with two drivers, one on IBM PC and the other on the Macintosh. It can be extended to any other mini- or microcomputers which provide a similar operating environment and a window and menu style structure. Before introducing the details of the common interface, the components of the GEM software environment will be described.

GEM consists of two major functional units: the Application Environment Services (AES) and the Virtual Device Interface (VDI); both provide a set of function libraries as a graphic interface [Ref. 1]. To build a typical GEM application, the user could implement the data fork and resource fork separately: the former basically consists of a set of procedures in the language that the program is written; the latter represents the menu bar and its associated submenus, form alerts, and dialogs created by another GEM application, known as the Resource Construction Set (RCS), which is provided by DRI. The RCS allows the programmer to construct the images, dialogs, and alerts that your application uses before any application code is written [Ref. 1]. GEM also provides some routines which build and deal with resources of application. It is less complicated when some important messages need to be modified without changing the application codes. This is a very important concept of establishing resources of a program because it saves the programmer a considerable amount of time and energy, when making complicated programming changes of some graphic structure. Thus, the application program is more flexible to change.

A. The Role of AES

The GEM AES provides routines which can be utilized to build the desktop and are organized in sets of related functions called *libraries* [Ref. 1]. For example, all the routines that manipulate windows are collected and form the Window Library of the AES, and all of the event routines form the Event Library, and so on [Ref. 1]. So, the AES represents a set of tools which can be useful when writing the first GEM application, the desktop, and in developing the common interface. AES includes a limited multitasking kernel, a screen Manager, and 11 libraries: Application, Event, Menu, Object, Form, Graphics, File Selector, Scrap, Window, Resource, and Shell. The GEM kernel is a limited multitasking system in that it can only handle five tasks: three desk accessory programs, one application, and the Screen Manager [Ref. 1]. Actually the Screen Manager is an internal task for event messages reporting to the AES event function. The GEM AES Event Library provides the

foundation that governs all user input in a GEM application. These input actions could be keyboard interrupts, mouse movement, mouse button changes, timer expiration, and messages in which some of them need the application to respond when receiving related events [Ref. 1].*

B. The Role of VDI

The purpose of the GEM VDI is to allow the user to control many different graphic devices with the same functions. The user can use the drawing routines to draw circles without considering what kind of output device will be used. This is very important because unlike IBM PC, Macintosh has more strict input and output constrains on hardware. IBM PC has a huge market share in the world and thousands of manufacturers who provide various competitive peripheral devices. Therefore, portability becomes indispensable for GEM. The VDI not only has a collection of drawing functions which can implement various shapes including points, markers, lines, polylines, graphics text, rectangle, and so on, but also control functions which open and close workstations (and virtual workstations) [Ref. 1].

* The details of all functions of other libraries can be found in the Programmer's Guide To GEM by Balma and Fitler (1986).

IV. OVERVIEW OF MACINTOSH

The Macintosh personal computer is designed in the way that the user can learn and use easily. Its revolutionary user interface distinguishes the Macintosh from other personal computers. Since the user interface acts as a good friend, it helps the user to communicate with the Macintosh comfortably. Everything on a Macintosh screen is displayed graphically; the Macintosh has no text mode. Generally speaking, the function sets are more detailed and includes more categories than GEM. All these functions are built into every Macintosh in ROM (read-only memory). The ROM can be divided into three parts: the Macintosh Operating System, which handles low-level tasks such as memory management, disk input/output, and serial communications; the QuickDraw graphics routines, which are responsible for everything displayed on the screen; and the User Interface Toolbox, which implements the higher-level constructs of the user interface, such as windows and menus [Ref. 3, p. 2]. The routines are divided according to function in Macintosh and and are called "managers" [Ref. 2, p. I-9]. Figure 4 shows the whole function distribution in the Macintosh [Ref. 2, p. I-10].

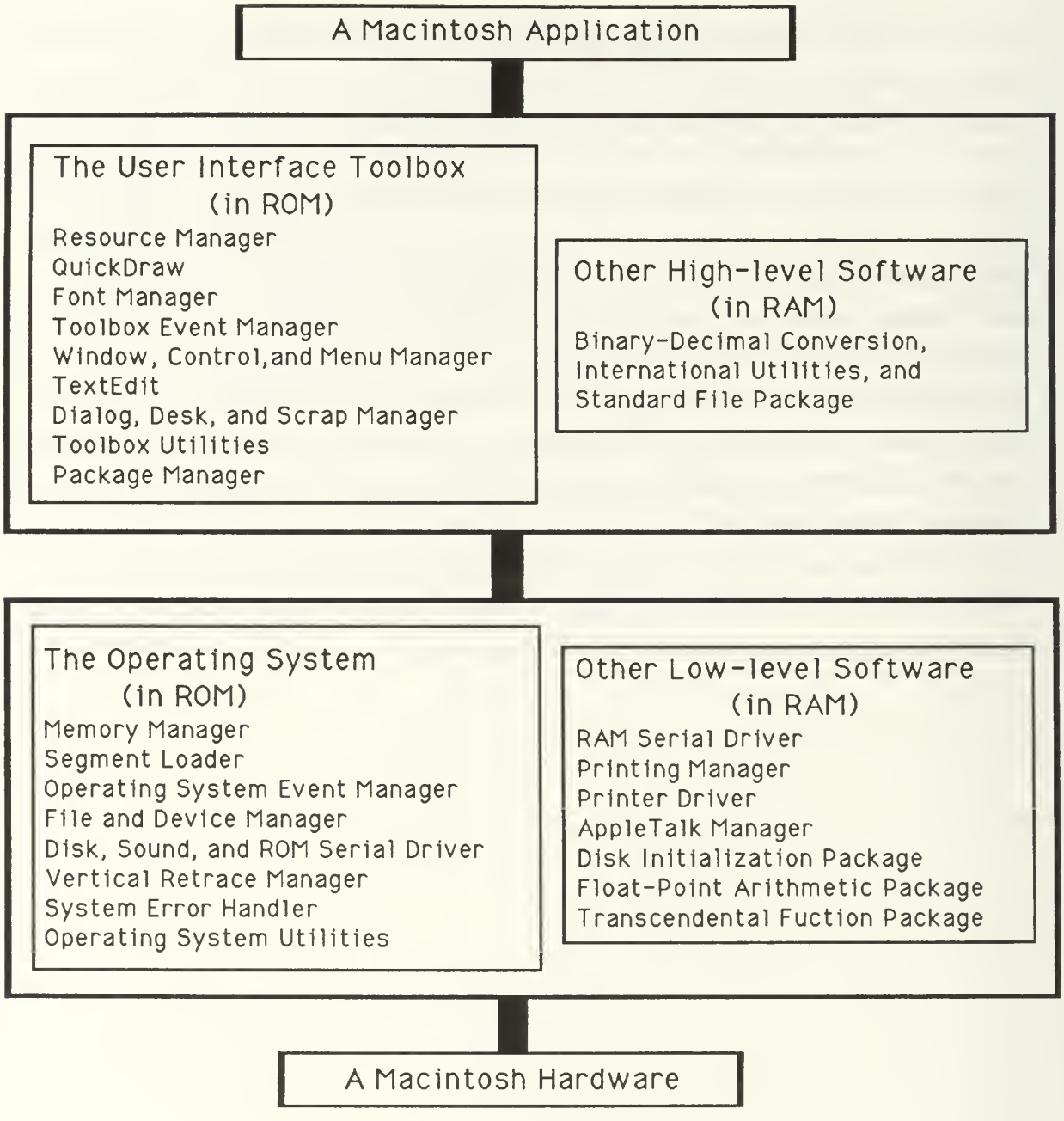


Figure 4 Overview of Macintosh

The Macintosh Toolbox also includes the Resource Manager which serves keep the data of an application separate from its code, making the data easier to modify and easier to share among applications. The Macintosh Resource Manager also supports more resource

types and more specified details than GEM. To manage and process the resource information, many utilities are available from the public domain [Ref. 2].

Before the Macintosh II come out, some routines in QuickDraw also enabled applications to do color drawing, including eight different colors, on color output devices. All nonwhite colors will appear as black on black-and-white output devices. In Macintosh II, more sophisticated color drawing routines are supported with 2^{32} colors.

Anyone who's used a Macintosh knows all about windows. The application displays all the information in the windows to the user, and the user tells the program what to do by clicking the mouse or hitting the keyboard. There can be any number of windows on the screen, and they can overlap in any order. Two different windows, the application window and the system window, both have their own characteristics to perform different tasks [Ref. 3].

Most of the time, the menu bar appears at the top of the screen, listing the titles of the available menus. One of the user's response to the program is to issue a command from an menu item under the title. Also, menus can be of various types in Macintosh to behave in certain standard ways. General speaking, the Macintosh Operating System and User Interface Toolbox provide a more complete function set of facilities for working with the User Interface than GEM does with its Operating Evironment [Ref. 3]. For the same reason, it is also more complicated.

V. DESIGN OF THE COMMON INTERFACE

Before starting to implementing the common interface, we have to design what functions are required to provide the useraccess to the common interface, and we have to design common interface functions that both Macintosh and GEM can support. Basically the common interface is general purpose and should be extendable. Some special functions can be done by several algorithms and we need to think about possible procedures that can finish specified task, like window update and redraw, and slao be compatible to different computers. Both GEM and Macintosh have detailed functions that may work in different ways, but their basic view of the user interface is similar. When we select the common portions of the functions, we may reduce function performance, but we also simplify the interface. Figure 5 shows the relationship of the Macintosh user interface, the common interface, and the GEM operating environment.

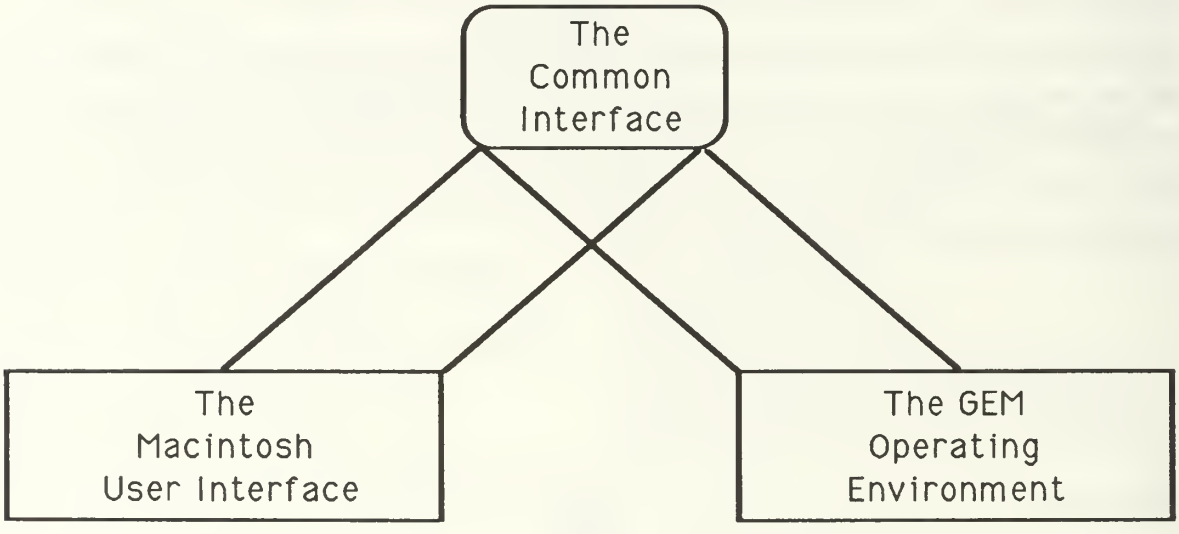


Figure 5 the relationship of all the interfaces.

To create the basic user-friendly interface, i.e., providing the complete graphical functions, at least four libraries must be built: the menu library, the primitive object library, the window library, and the event library.

A. Design Methodology and Abstract Data Type

When we decide to build a common interface which can perform graphic functions and window style, the most important consideration is the structure of this common interface and how to make the common interface easier to use. The structure of the

common interface can be divided into four libraries which can be implemented independently. Every library groups those functions which are relative to themselves. Also in every library, the functions can be futher divided into several subgroups according to their tasks.

Before introducing the details of the libraries, we will discuss the design methodology and abstract data types of the common interface. In all of the primitive drawing objects, the rectangle acts a very important role in the common interface. When a circle, an ellipse, an arc, or a round rectangle are drawn, a rectangle is always needed for setting the drawing boundary and calculating the outline of the specified object. In GEM and Macintosh, they use different data structures to create a rectangle. GEM uses a top left point , a width and a height to specify a rectangle and Macintosh uses two points: a top left point and a bottom right point. It sounds tricky for us to decide which data structure is better. However, we are not going to worry about the data types of rectangle or point when using the concept of Abstract Data Type. We think about a rectangle in an abstract way. A rectangle consists of four points connected with four outer lines, and a point consists of two coordinate values, a horizontal and a vertical value, but not all the data are necessary to create a rectangle on the screen. In using the concept of Abstract Data Type, we simply design a set of functions that perform all the operations on a rectangle and achieve information hiding of the data structure. The programmer can do whatever he wants with a rectangle by utilizing these functions. Several representations, including the GEM and Macintosh ones, can be used to represent the rectangle and still support the rectangle functions defined in the abstract data type. The functions act like guards or interfaces that surround and protect the data structure in the center (see Figure 6). Obviously, the different data structures on the Macintosh and GEM are irrelevant. We follow the same design methodology of abstraction and information hiding on all the functions in the interface.

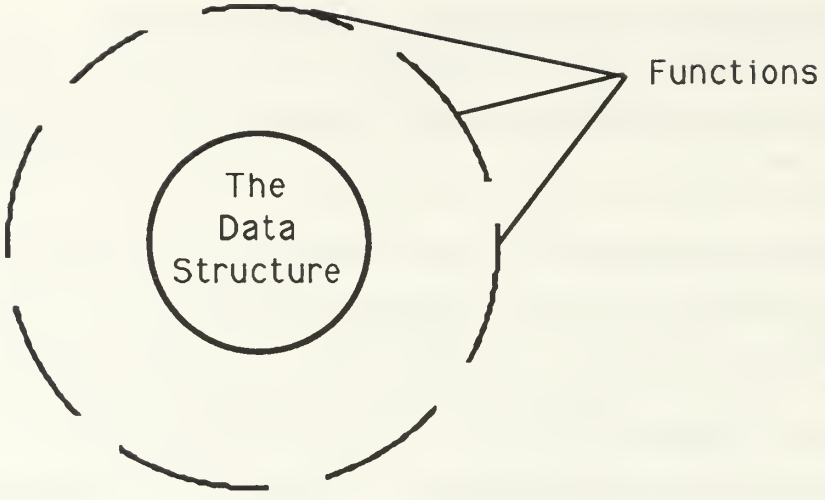


Figure 6 The Data Structure and Functions in Abstract Data Type

After implementing the design, the programmer won't need to manipulate the point and rectangle directly because he can utilize the functions which are provided by the common interface to deal with the rectangle or point. However, we still need to select a data type. In C, they are defined below:

```
typedef struct
  {
  Int    v,h;
  } Point ;

typedef struct
  {
  Point       topLeft;
  Point       botRight;
  } Rect ;

#define top         topLeft.v
#define left        topLeft.h
#define bottom      botRight.v
#define right       botRight.h
```

When the programmer wants to write an application program, the organization of the program becomes easy by using the concept of the common interface. In fact, the Event Library which provides the function that always generates fixed messages, or events, has

made the program only need to take care of the events. By notification of these events, the program can receive commands from the menu selection or handle the variation of the mouse button and movement issued by the user.

All the necessary definitions used by the common interface are put in the file "ASBIND1.H" for both Macintosh and GEM. To make sure that the program runs well, the programmer better selects the relative one when compiling the program. Similarly, the programmer might have a data type that is similar to the common interface to keep track of all the background information. The most obvious example in the DEMO program which will be introduced in next chapter, is the usage of the 'Winlist' structure which retains all the useful information about a window.

B. Design of the Primitive Object Library

The Primitive Object Library supports the manipulation of the primitive objects of the abstract specification — the Point and the Rectangle. As background, the graphic display device is subdivided into discrete areas known as pixels. As far as the graphic device is concerned, pixels are the smallest unit of manipulation. Reference to particular pixels on the abstract screen are via an imposed coordinate system. The origin or (0, 0) pixel is located at the upper left corner of the screen. In the Abstract Specification, there is a one to one mapping between points and pixels. A point is defined by specifying its horizontal and vertical displacement from the origin of a graphic environment. However, these displacements are relative to a particular window environment in which the point is used. Rectangles are defined by specifying the top left and bottom right corners of the rectangle (see Figure 7) [Ref. 2, p. I-140].

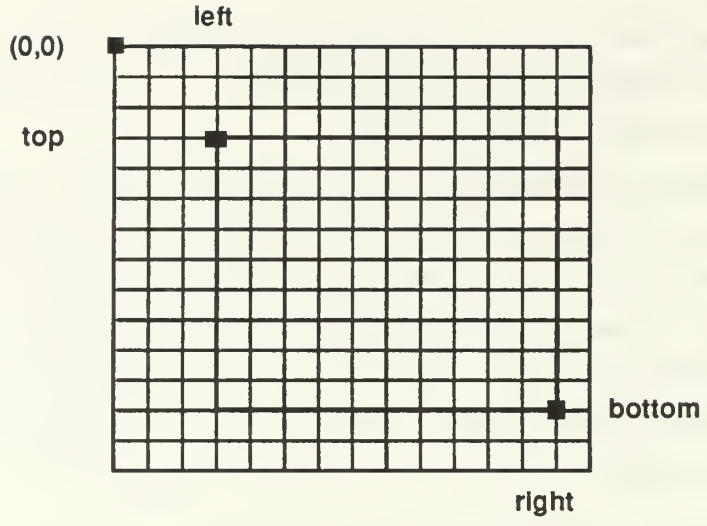


Figure 7 A Rectangle and the Origin

In the following description of the Primitive Object Library, as well as the other libraries, we will explain all functions in the C language style with their parameters. In the Primitive Object Library, the whole functions can be classified into three sets: the Point set, the Rectangle set, and the Point and Rectangle translation set.

1. The Point Set

As mentioned before, a point is specified by two integers which are coordinate values. We need enough functions to calculate or transfer the data type about point and integer. Some C compilers, because they do not allow passing **structs** as arguments, require the address of the Point to be passed instead. There are five functions in the Point set.

a. Set point by integers

Given two integers which represent the X and Y coordinate (the horizontal and vertical positions of the point respectively), the function returns a point.

State	set_point (x, y, pt)		
Input:	Int	x, y	the value of the X and Y coordinate respectively.
Output:	Point	*pt	the returned point.

b. Get X coordinate value from point

Function which returns the horizontal coordinate value of the input point.

Int	ret_val = get_x_coord (pt)		
Input:	Point	*pt	the given point.

Output:	Int	ret_val	the X coordinate value.

c. Get Y coordinate value from point

Function which returns the vertical coordinate value of the input point.

Int	ret_val = get_y_coord (pt)		
Input:	Point	*pt	the given point.
Output:	Int	ret_val	the Y coordinate value.

d. Test two equal points

Function which determines if the two input points are the same point.

Bool	ret_val = equalpt (p1, p2)		
Input:	Point	*p1, *p2	the two given points.
Output:	Bool	ret_val	TRUE, if p1 and p2 are the same point.
			FALSE, if not.

e. Copy point

Function which copies the source point into the destination point.

State	copypt (source, dest)		
Input:	Point	*source	the given source point.
Output:	Point	*dest	the returned destination point.

2. The Rectangle Set

Some functions, which pertain to the calculation of two rectangles, belong to this category.

a. Set intersection of rectangles

Function which determines the rectangle which is formed by the intersection of the two input rectangles. If the intersection is empty, the rectangle returned will be defined by a top left and bottom right point of (0, 0).

State	set_insect_rect (r1, r2, rint)		
Input:	Rect	*r1, *r2	the given two rectangle.
Output:	Rect	*rint	the returned rectangle of intersection.

b. Test intersection of rectangles

Function which determines whether the two input rectangles intersect.

Bool	ret_val = insect_rect (r1, r2)		
Input:	Rect	*r1, *r2	the given two rectangle.
Output:	Bool	ret_val	TRUE, if r1 and r2 intersect. FALSE, if not.

c. Test equal rectangles

Function which determines if the two input rectangles are the same rectangle.

Bool	ret_val = equalrect (r1, r2)		
Input:	Rect	*r1, *r2	the given two rectangle.
Output:	Bool	ret_val	TRUE, if r1 and r2 are the same. FALSE, if not.

d. Copy rectangles

Function which copies the source rectangle into the destination rectangle.

State	copypt (source, dest)		
Input:	Rect	*source	the given source rectangle.
Output:	Rect	*dest	the returned destination rectangle.

3. The Point and Rectangle Translation Set

A rectangle is specified by two points. We need the Point and the Rectangle have enough operations to cover the information exchange. For example, type transfer from points to the rectangle or from the rectangle to the points.

a. Set rectangle by points

Function which, given two points, determines the smallest rectangle that those points could define and sets the top left and bottom right points of the output rectangle to correspond to that rectangle.

State	set_rect (p1, p2, r)		
Input:	Point	*p1, *p2	the given two points.
Output:	Rect	*r	the returned rectangle.

b. Get the top left point from rectangle

Function which returns the top left point of the input rectangle.

State	set_topLeft (r, p)		
Input:	Rect	*r	the given rectangle.
Output:	Point	*p	the returned top left point.

c. Get the bottom right point from rectangle

Function which returns the bottom right point of the input rectangle.

State	set_botRight (r, p)		
Input:	Rect	*r	the given rectangle.
Output:	Point	*p	the returned bottom right point.

d. Test point in rectangle

Function which determines if the input point is within or on the border of the input rectangle.

Bool	ret_val = pt_in_rect (p, r)		
Input:	Point	*p	the given point.
	Rect	*r	the given rectangle.
Output:	Bool	ret_val	TRUE, if point p in rectangle r. FALSE, if not.

C. Design of the Event Library

Whenever the user presses the mouse button, or types on the keyboard, the application program is notified by means of an event. In the Abstract Specification, all events represent the user's actions. Not only the user can generate an event, also the event can generate another event. For instance, when the user drags a window away, the uncovered original region may need to be updated, and a redraw event is issued by the program. In Macintosh, more complicated events are also provided like disk-inserted event, network event, and device driver event, but there are only fundamental events in GEM. There are eight events that are summarized for the event function to meet the basic interface requirement. Two other mouse functions which relate to events are also included in the Event Library.

1. Get event function

Function which senses user interaction with the program, determines the type of interaction, and reports the user interaction to the program via the message globe data item. At present there are eight different types of events which are reported to the program:

a. Activate event

A notification that the user pressed the mouse button while the cursor was over an inactive window (requesting) to make that window active, and the application has to reorder the windows.

b. Redraw event

A notification that the work area of one of the windows present on the screen has been disturbed or exposed and must be rewritten.

c. Close window event

A notification that the user has pressed the mouse in the close box of the active window (if present).

d. Mouse down event

A notification that the user has pressed the mouse button in the working area of the active window.

e. Keyboard event

A notification that the user has typed the keyboard.

f. Mouse up event

A notification that the user has released the mouse.

g. Menu selection event

A notification that the user has selected a menu item.

h. Scroll bar event

A notification that the user has pressed the mouse in some part of the scroll bar.

Two functions are taken care of automatically by this routine : changing the size of a window in response to the user dragging in the window's grow box and moving a window in response to a user dragging in the title bar of the window.

State get_event ()

Input: none.

Output: 11 messages, in the following, are declared in "ASBIND1.H" file.

• EVTTYPE: always has a value to represent an event. There are 8 kinds of events that their program codes are shown below. The coming event always appends some relative and useful information, also shown behind the event, which can tell the programmer more details about the event.

- REDRAW with EVTWINDOW, EVTRECT.
- TOPPED with EVTWINDOW, ENTPOINT, EVTMOD.
- CLOSEWIN with none.
- SCROLLBAR with EVTSCRPART, EVTSCRMOVE, EVTSCRPOSN.
- MOUSEDOWN with EVTWINDOW, EVTPOINT, EVTMOD.
- KEYBOARD with EVTKEY, EVTMOD.
- MOUSEUP with EVTWINDOW, EVTPOINT, EVTMOD.
- MENUHIT with EVTMTITLE, EVTMITEM.

• EVTWINDOW: the returned window ID.

• EVTRECT: the rectangular area that needs redrawn.

• EVTPOINT: the cursor position when the event happened.

• EVTSCRPART: the scroll bar position report which the possible value is

V_PAGEUP, V_PAGEDOWN, V_ROWUP, V_ROWDOWN,
H_PAGEUP, H_PAGEDOWN, H_ROWUP, H_ROWDOWN,
V_THUMB, H_THUMB.

• EVTSCRPOSN: the scroll bar current setting. The minimam value of any scroll bar is zero, and the maximam one is 1000.

- EVTSCRMOVE: the difference that current setting minus the old one.
- EVTKEY: the input ASCII code.
- EVTMOD: the states of the modifier keys.
- EVTMTITLE: the selected menu title.
- EVTMITEM: the selected menu item.

2. Get mouse location

This function gets the current mouse position and outputs it in the local coordinate system of the specified window.

State get_mouse(Id, pt)

Input:	Int	Id	the given window ID.
Output	Point	*pt	the returned point of the mouse position.

3. Test mouse button

This is the function we use to get the state of the mouse up or down. It's useful when the user presses and moves the mouse as an action.

Bool ret_val = mouse_up()

Input: none.

Output:	Bool	ret_val	FALSE, if mouse button is pressed.
			TRUE, if not.

E. Design of the Window Library

1. The Structure of Window

In the Abstract Specification, all objects (points, rectangles, etc) are defined in relation to the window which happens to be active at the time. The window, as an object consists of two basic regions, a structure region and a content region. The structure region contains the following objects (see Figure 7 about window structure):

a. Title bar

Bar at the top of the window containing the window's title.

b. Move area

Lined area of the title bar which can be clicked in to move the window. Normally the move area is the same as the title bar area.

c. Close box

White rectangle which when clicked in, signals that the user desires to close the window.

d. Scroll bar:

Bars on the right and bottom of the window, used to signal the user's desire to move the window's contents up, down or side to side.

e. Grow box:

Area at bottom right of window, which when clicked and dragged around, changes the size of the window.

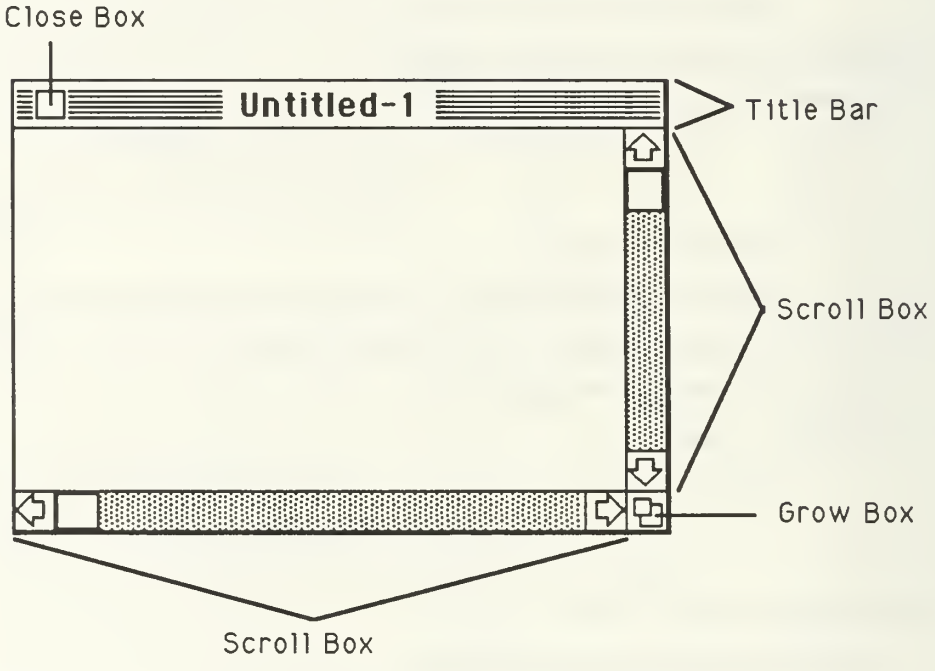


Figure 8 An active window

The remaining portion in the center of the window is the content region. This region can be thought of as an independent screen with its own local coordinate system whose origin (0,0) is at the top left corner of the content region. The basic system window, the desktop, is slightly different, having only a menu bar area at the top and then its content region. At any one time, there is a window which is "on top" of the screen. This window is the active window. All drawing activities take place in the active window except in the case of an update, in which case it takes place in the window specified.

All windows once allocated are managed by the window ID number which is assigned at the time of creation. To prevent out of memory errors, the maximum number of available windows is eight. In general, the programmer has to keep the ID information while manipulating multiple windows.

2. The Window Function Set

All functions used to manipulate windows and other relative objects, can be divided into six parts.

a. Window Manipulation

There are several basic functions here to manipulate windows as a whole entity.

i. Create a new window

Function which allocates space for a new window and displays it as the active window. The programmer can create new windows with different optional properties such as vertical and horizontal scroll bars, close box, grow box, etc.

Window_id ret_val = set_new_window (InitRect, Partspec, Title, is_Visible)

Input:	Rect	*InitRect	a rectangle, given in global coordinates, determines the window's size and location.
	Bit16	Partspec	specifies which optional parts of the window are to be included (see below).

Parts (optional): defined in "ASBIND1.H" file.

W_NAME	include a title bar;
W_CLOSE	include a close box;
W_SIZE	include a size box;
W_HSCROLL	include a horizontal scroll bar;
W_VSCROLL	include a vertical scroll bar.

(To include more than one option, pass a bitwise OR of any combination of above)

	Char	Title	address of string to be used as a title for the window.
	Bool	is_Visible	TRUE, if the window is to be displayed; FALSE, if not.
Output:	Window_id	ret_val	the identifier of the new window.

ii. Show window

Function which draws an invisible but previously defined window onto the screen. This window becomes the active window.

State show_window (Id)

Input:	Window_id	Id	the given window identifier.
Output	none.		

iii. Hide window

Function which removes the specified window from the screen without deallocating it.

State	hide_window (Id)		
Input:	Window_id	Id	the given window identifier.
Output	none.		

iv. Activate window

Function which causes the specified window to become the active window. It causes any window (but the desktop with a ID number of 0) to be moved to the top and a new background will be drawn in, however, the contents will not be automatically redrawn.

State	activate_win (Id)		
Input:	Window_id	Id	the given window identifier.
Output	none.		

v. Close window

Function which closes and permanently deallocates the specified window.

State	close_window (Id)		
Input:	Window_id	Id	the given window identifier.
Output	none.		

vi. Update window

Function which sets the system into the update window mode, drawing will be limited to the visible region of the window to be updated (as identified by the ID number input) to the function. When given a rectangular area to update, the function will return the intersection between that area and one of the rectangles which define the visible area of the wondow to be updated. In the Macintosh, the update event happens window by window (in front to back order). In GEM, when a REDRAW event is issued, a rectangle list, divided from the screen and window, is built for the program to update. Thus, the programmer always needs to pass the EVTRECT to this function.

Bool	rec_val = update_win (ID, Up_rct, Dr_rct)		
Input	Window_id	ID	the ID of the window that will be updated.
	Rect	*Up_rct	the rectangle to be updated.
Output:	Rect	*Dr_rect	the intersection of update rectangle and visible region.
	Bool	ret_val	TRUE, if need updat. FALSE, if not.

vii. Update next window

To solve the update problem of GEM and Macintosh, this function moderates the conceptual difference between two computers and completes the update mission without having much redundant work. In Macintosh, this function does nothing and always returns false. But, it is still useful for GEM.

Bool	rec_val = next_update (Up_rct, Dr_rct)		
Input	Rect	*Up_rct	the rectangle to be updated.
Output:	Rect	*Dr_rect	the intersection of update rectangle and visible region.
	Bool	ret_val	TRUE, if next update is necessary; FALSE, if not.

viii. End updating window

Function to ends the update mode and restore the clip area to match the active (topmost) window. The programmer always has to call update_win() when receiving a REDRAW event, and call end_update() at the end of update.

State	end_update ()
Input:	none.
Output:	none.

ix. Find active window

Function which shows the identifier of the active window.

Window_id	ret_val = get_active()		
Input:	none.		
Output	Window_id	ret_val	the returned window ID.

b. Scroll Bar Manipulation

In fact, scroll bar is part of control facilities of a window to adjust the viewing position of the document of the window. The scroll bar is divided into five parts to perform different functions. The up and down arrows scroll the window's content a line at a time. The paging up and down regions scroll a page at a time. The thumb can be dragged to any position in the scroll bar, to scroll to a corresponding position within the document (see Figure 9) [Ref. 2]. Six functions are shown below.

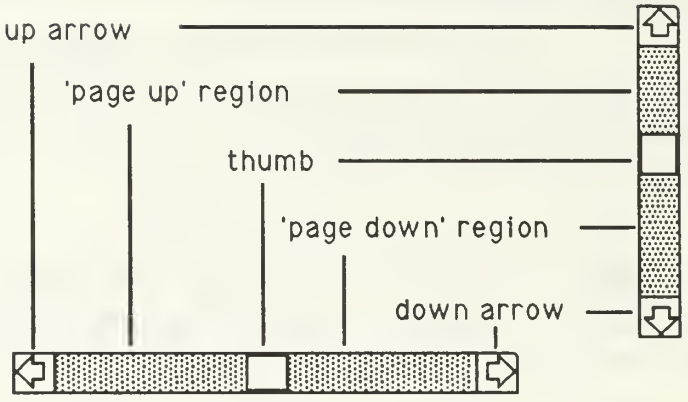


Figure 9 Parts of a scroll bar

i. Horizontal content scrolling

Function which scrolls the content area of the active window by the specified number of pixels. If the number is positive, the region will move to the left, and to the right if negative. The returned rectangle, which was previously covered, will show up now and should be passed for update.

State	hscroll (num, Up_rect)		
Input:	Int	num	the pixel number to scroll
Output	Rect	*Up_rect	return the rectangle which will be updated

ii. Vertical content scrolling

Function which scrolls the content area of the active window by the specified number of pixels. If the number is positive, the region will move up, and down if negative. The returned rectangle, which was previously covered, will show up now and should be passed for update.

State	vscroll (num, Up_rect)		
Input:	Int	num	the pixel number to scroll
Output	Rect	*Up_rect	return the rectangle which will be updated

iii. Set horizontal scroll bar value

Function which sets the value of the horizontal scroll bar of the active window.

State	set_hscroll (val)		
Input:	Int	val	new horizontal scroll bar setting.
Output:	none.		

iv. Set vertical scroll bar value

Function which sets the value of the vertical scroll bar of the active window.

State set_vscroll (val)

Input: Int val new vertical scroll bar setting.

Output: none.

v. Get horizontal scroll bar value

Function which returns the horizontal scroll bar value.

Int ret_val = get_hscroll ()

Input: none.

Output: Int ret_val the returned horizontal scroll bar value.

vi. Get horizontal scroll bar value

Function which returns the horizontal scroll bar value.

Int ret_val = get_vscroll ()

Input: none.

Output: Int ret_val the returned vertical scroll bar value.

c. Drawing Background Manipulation

Abstraction Specification of graphic objects has three different kinds of characteristic: background pattern, mode, and color. Pattern includes black, dark gray, gray, light gray, and white. Mode includes replace, transparent, xor, and reverse transparent. Color includes light and dark which both include white, black, red, green, blue, cyan, yellow, and magenta.

i. Set pattern

Function which sets the pattern to be used to draw and fill in shape.

State set_pattern (newpattern)

Input: Pattern_id newpattern the given pattern ID.

Output: none.

ii. Set mode

Function which sets the global mode for drawing onto the screen.

State set_xfer_mode (newmode)

Input: Pattern_id newmode the given transfer mode ID.

Output: none.

iii. Set color

Function which sets the global color for drawing.

State set_color (newcolor)

Input: Color_id newcolor the given color ID.

Output: none.

iv. Get pattern

Function which retuens the identifier of the drawing pattern.

Pattern_id ret_val = get_pattern ()

Input: none.

Output: Pattern_id ret_val the returned pattern ID.

v. Get mode

Function which retuens the identifier of the drawing transfer mode.

Mode_id ret_val = get_xfer_mode ()

Input: none.

Output: Mode_id ret_val the returned transfer mode ID.

vi. Get color

Function which retuens the identifier of the drawing color.

Color_id ret_val = get_color ()

Input: none.

Output: Color_id ret_val the returned color ID.

d. Drawing Object Manipulation

Drawing functions are the most important part of the Abstract Specification. The reason we put this function in the Window Library is all object drawing routines happen in a window. All of the drawing happens in the active window with the current setting of pen mode, pen pattern, and color. The coordinates of the input point or rectangle are assumed to be relative to the top left corner of the active window's work area. There are five kinds of object supported in the Abstract Specification : line, rectangle, ellipse, arc and round rectangle.

i. Draw a line

Function which draws a line in the currently active window.

State drawline (St_pt, End_pt)

Input: Point *St_pt the starting point.

Point *End_pt the ending point.

Output: none.

ii. Draw a rectangle

Function which draws the outline of a rectangle in the active window.

State drawrect (In_rect)

Input: Rect *In_rect the given rectangle.

Output: none.

iii. Draw an ellipse

Function which draws the outline of an ellipse within the specified rectangular area of the active window.

State drawellipse (In_rect)

Input: Rect *In_rect the given rectangle.

Output: none.

iv. Draw an arc

Function which draws the outline of an elliptical arc between the two input angles within the specified rectangular area of the active window.

State drawarc (R, begang, endang)

Input: Rect *R the given rectangle.

Int begang the starting angle.

Int endang the ending angle.

Output: none.

v. Draw a round rectangle

Function which draws the outline of an round rectangle within the specified rectangular area of the active window.

State drawrndrct (In_rect)

Input: Rect *In_rect the given rectangle.

Output: none.

vi. Fill a rectangle

Function which fills the outline of a rectangle in the active window.

State fillrect (In_rect)

Input: Rect *In_rect the given rectangle.

Output: none.

vii. Fill an ellipse

Function which fills the outline of an ellipse within the specified rectangular area of the active window.

State fillellipse (In_rect)

Input: Rect *In_rect the given rectangle.

Output: none.

viii. Fill an arc

Function which fills the outline of an elliptical arc between the two input angles within the specified rectangular area of the active window.

State fillarc (R, begang, endang)

Input: Rect *R the given rectangle.

Int begang the starting angle.

Int endang the ending angle.

Output: none.

ix. Fill a round rectangle

Function which fills the outline of an round rectangle within the specified rectangular area of the active window.

State fillrndrct (In_rect)

Input: Rect *In_rect the given rectangle.

Output: none.

e. Text Manipulation

In the Abstract Specification, only a few functions are available for the basic manipulation of text.

i. Set text pen position

Function which sets the location of the next character to be drawn in the active window (location of text pen in window local coordinates).

State txtpen (inpt)

Input: Point *inpt the given text location.

Output: none.

ii. Get text pen position

Function which returns the location of the text pen for the currently active window (in window local coordinates).

State set_txtpen (pen)

Input: none.

Output: Point *pen the returned text location.

iii. Write string

Function which draws a string into the active window at the current location of its text pen.

State drawstring (strptr)

Input: Char *strptr the string which will be drawn.

Output: none.

iv. Write character

Function which draws a character at the current location of the active window's text pen.

State drawstring (inchr)

Input: Char inchr the character which will be drawn.

Output: none.

v. Get character width

Functions return the current character width.

Int ret_val = get_wchar ()

Input: none.

Output Int ret_val the character width.

vi. Get character height

Functions return the current character height.

Int ret_val = get_hchar ()

Input: none.

Output Int ret_val the character height.

f. System Manipulation

The programmer needs to call sys_init() and sys_end(), which will be described below, at the beginning and end of the program respectively.

i. System initialization

Function to initialize the system to run the Abstract Specification Interface.

State sys_init ()

Input: none.

Output: none.

ii. Exit application program

Function which returns all allocated resources to the system at the end of the program.

State sys_end ()

Input: none.

Output: none.

E. Design of the Menu Library

Menu selection is a method used to issue a command to the application program. This is one of the most important and user-friendly characteristics of the common interface, the user just moves and clicks the mouse around the screen to control the application program without typing the keyboard. GEM menus are known as drop-down menus because when the user moves the mouse over the menu bar, the GEM Screen Manager drops the entire menu down onto the screen. In contrast, the Macintosh uses pull-down menus, which work by having the user click on the desired menu title, and, holding the button down, move through the menu highlighting each pointed-at item. By releasing the button the user selects the last highlighted item. Thus, on the Macintosh, the menu is displayed as long as the button remains depressed, whereas GEM menus are visible until the user moves the mouse out of the menu, either into another menu or to another part of a screen. GEM menus are also different in that the mouse button is used to select a menu item. As shown in Figure 10, the application highlights the title and displays the menu items [Ref. 2, p. I-52].

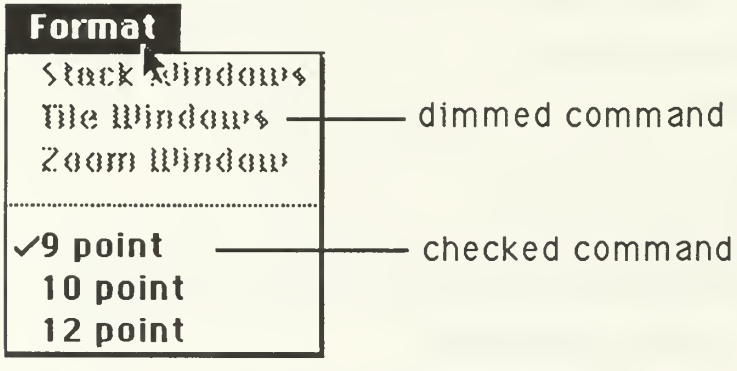


Figure 10 Menu

The GEM Menu Library only provides the fundamental functions required, but the Macintosh Menu Manager includes the complete works of the menu functions. To collect the necessary set, there are five basic menu routines that are chosen for performing menu functions.

1. Menu bar initialization

This function always has to be called by the programmer at the beginning of the application to show the menu bar. Here we need the resource file name prepared in advance. So before passing the resource file name to this function, the programmer must

utilizes the respective resource maker utility program supported from DRI and Apple Computer, Inc., to edit the menu resource for the application program* .

State init_menu (filename, barid)

Input: char *filename the resource file name

Menu_id barid the menu ID specified by resource utility.

Output: none.

2. Menu item enable

To make sure the user can issue the proper commands, the application program may only allow certain commands to be selectable. This function corresponds to the menu item disable which will be mentioned next.

State item_enble (menunum, itemnum)

Input: Int menunum the menu title number

Int itemnum the menu item number

Output: none.

3. Menu item disable

In some specified situation, some unacceptable or unnecessary commands must be disabled. A disabled item cannot be chosen; it appears dimmed in the menu and is not highlighted when the cursor moves over it. You can change the enabled or disabled state of a menu item with this and the last function.

State item_disable (menunum, itemnum)

Input: Int menunum the menu title number

Int itemnum the menu item number

Output: none.

4. Set menu item check mark

The programmer can place a check mark to the left of the text of the menu item. This action can clearly tell the user which command is working or what state is presenting. With this function, the programmer can set or clear the check mark.

State item_mark (menunum, itemnum, mark)

Input: Int menunum the menu title number

Int itemnum the menu item number

---

* There are several utilities available, include RMaker, ResTool, and ResEdit, for the Macintosh computer. Also, GEM has the Resource Construction Set supported by DRI for the same purpose.

	Bool	mark	if TRUE, then a check mark will appear each subsequent time the menu is pulled down. If FALSE, then remove the check mark from the menu item.

Output: none.

5. Menu title highlighting

When an item is selected, the menu title in the menu bar remains highlighted until the command has completed execution. So after the menu is selected, the application should perform the chosen task and then call this function to unhighlight the chosen menu title. The programmer can also use this function to highlight the menu title. Since only one menu title can be highlighted at a time, it unhighlights any previously highlighted menu title.

State menu_hilight (menunum, hilight)

Input:	Int	menunum	the menu title number
	Bool	hilight	if TRUE, then hilight the title of given menu. If FALSE, unhilight the chosen menu title.

Output: none.

VI. IMPLEMENTATION

In this section we will discuss some details of implementing the common interface and the testing of a demonstration program. This mini interface actually provided only basic functions for building an application program. To fully utilize the available functions, we have to introduce all the other necessary features and properties of the programming environment. First, we will examine the designing of the data structures, then all the functional abilities of these libraries. In the view of a design task, the design of the data structures should be put last . But the whole design is actually digested in the GEM and Macintosh programming environments to get a feasible intersection. So, here we just use the data structure to establish the direction of the implementation work.

There are several special data structures and defined constant data which were designed for the Event and Window Library to be utilized by the programmer. The following descriptions show some detail notes about the Abstract Specification of implementation:

- **The window type and limitation**
  - —The maximum number that an application can open at a time is limited under seven to prevent out of memory. Because we have to control our own window by the data structure which summarized from the GEM and Macintosh, and specify the number of windows during the compile time.
  - —Every window has its own identifier instead of a window pointer to its location.
  - —The scroll bar is regarded as part of the window structure. The thumb value is between zero and 1000.
  - —The Desktop on the screen has the window identifier value zero. When an invalid window happens in any function its identifier value is -1.
  - —A newly created window has options to include title, close box, grow box, the horizontal and/or vertical scroll bar.

- **The event structure**
  - —the notification of of a event is always accompanied by different information which depends on the event. A keyboard event comes with the key stroke and the state of the modifier keys. Menu selection events come with the selected menu title and item. Redraw event comes with the window and rectangle which needs redrawn. Mouse down event comes with the mouse down window,

cursor point, and the state of modifier keys. Update, close window event comes with the window where the event happened. Scroll bar event comes with the specified part of scroll bar, and the new thumb position.

—all events are enqueued into an internal first in first out data structure.

—all windows can always be dragged or sized, but the actions might generate redraw events depending on whether the hidden parts of inactive window appear.

- **The graphic object structure**

—the graphic objects can be line, rectangle, ellipse, arc and round rectangle. Except line object, the other objects can be drawn either outline only or with pattern in.

—the background of all objects include color, pattern, and mode that they all can be represented by the specified identifier value.

- **The basic structure of a standard program**

—the DEMO program shown in appendix has a basic structure and it can be a good reference for the programmer. Normally, the programmer takes responsibility of the content of the include file and resource file for the apllication.

—the programmer should always include the "ASBIND.H" and "ASFBIND.H" files. The ASBIND.H comprises the binding data type of Abstract Specification, the ASFBIND.H includes all the binding functions call of Abstract Specification.

--the following flow chart shows the basic style of an application.

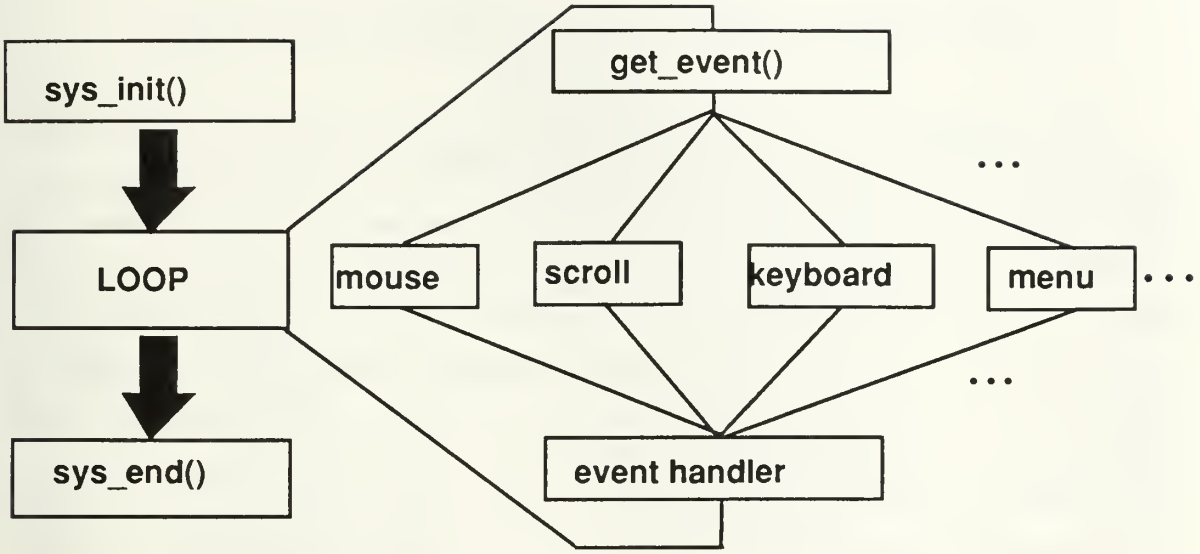


Figure 11 The Basic Structure of an Application Program

For the purpose of understanding how this Abstract Specification of the common interface will work, there is a demo program in appendix illustrating the basic graphic application and how those functions can be applied by the user.

VII. CONCLUSION AND RECOMMENDATION

In the beginning of this paper we mentioned that the purpose of this common interface is to make the same source code run on an IBM PC under the GEM environment and the Apple Macintosh with the same effect. This idea could be used to improve the portability of many applications since the application could be separated into system dependent and system independent (common interface) routines. We have achieved this purpose since the source code that uses the common interface of either system is independent. When the programmer want to run the same result on other different machines, he can just rewrite the system dependent part. In this thesis, we just prove that it is possible to support the common interface (system dependent part) to the programmer and save duplicated works. Clearly only one drawing demo program cannot prove that the common interface will work correctly when further used in other more sophisticated application program, but it does prove the feasibility of the idea. Of course, in the intersection of the GEM and Macintosh we lose some of their origional powerful abilities, but if the system dependent part of an application can expand and provide other functions, then the concept of the common interface could be an important idea.

For further study, we recommend that this idea be examined on other systems. For example, can the same abstract design be implemented on top of X-Windows on Unix, or MS Windows on MS-DOS? Such an effort would lead to a better understanding of this type of interface.

APPENDIX A

Demo program listing

```
/*-------------------------------------------------------------------*/
/*                          DEMO.C                                   */
/*-------------------------------------------------------------------*/

#include        "asbind.h"
#include        "asfbind.h"
#include        "demo.h"

#define         SINGSCR     20
#define         PAGES       4
#define         COLPAGE     20

#define         NUMDR       100

char    *Title[MAXNUMWIN] = begin
                        "DrawWindow1",
                        "DrawWindow2",
                        "DrawWindow3",
                        "DrawWindow4",
                        "DrawWindow5",
                        "DrawWindow6",
                        "DrawWindow7",
                end;

typedef         struct  drstr
begin
        Rect            drrct;
        Color_id        drcol;
        Mode_id         drmo;
        Pattern_id      drpat;
        int             drshp;
        Bool            drfill;
end drstr;

typedef         struct  winstr
begin
        Window_id       winid;
        drstr           Drawn[NUMDR];
        Int             drawcnt;
        Color_id        wincol;
        Mode_id         winmode;
        Pattern_id      winpat;
        Bool            doline;
        Bool            dofill;
        Bool            dodark;
        Int             selPat;
        Int             selCol;
        Int             selMod;
        Int             Shape;
```

```
        Bool            Created;
        Bool            Visible;
end     winstr;

        Point           Tl,Br;
        Rect            winrect;

        winstr          Winlist[MAXNUMWIN];
        Int             Lastactive;

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

        Int
Findindex(Id)

        Window_id       Id;

begin
        Int     I;

        if (Id == DESK_WIN)
                return(INVALID);

        for(I = 0; ((I < MAXNUMWIN) && (Winlist[I].winid != Id)); I++);

        return(I);

end

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/
        Void
ResetMenus(oldind,Index)

        Int     oldind;
        Int     Index;

begin


        if (oldind == INVALID)
                oldind = Lastactive;


                        /* handle drawing menu */

        if (Winlist[Index].dofill != Winlist[oldind].dofill)
        begin
                if (Winlist[oldind].dofill)
                begin
                        item_mark(MNDRAW,ITOUTLN,TRUE);
                        item_mark(MNDRAW,ITFILL,FALSE);
                end
```

```
        else
        begin
                item_mark(MNDRAW,ITOUTLN,FALSE);
                item_mark(MNDRAW,ITFILL,TRUE);
        end

end

item_mark(MNDRAW,Winlist[oldind].Shape,FALSE);
item_mark(MNDRAW,Winlist[Index].Shape,TRUE);

if (Winlist[Index].doline != Winlist[oldind].doline)
begin
        if (Winlist[oldind].doline)
        begin
                item_enable(MNDRAW,ITOUTLN);
                item_enable(MNDRAW,ITFILL);
                item_enable(MNDRAW,ITRECT);
                item_enable(MNDRAW,ITELLIP);
                item_enable(MNDRAW,ITARC90);
                item_enable(MNDRAW,ITARC180);
                item_enable(MNDRAW,ITARC270);
                item_enable(MNDRAW,ITRNDRCT);
                item_mark(MNDRAW,ITSHAPE,TRUE);
                item_mark(MNDRAW,ITLINE,FALSE);
        end

        else
        begin
                item_disable(MNDRAW,ITOUTLN);
                item_disable(MNDRAW,ITFILL);
                item_disable(MNDRAW,ITRECT);
                item_disable(MNDRAW,ITELLIP);
                item_disable(MNDRAW,ITARC90);
                item_disable(MNDRAW,ITARC180);
                item_disable(MNDRAW,ITARC270);
                item_disable(MNDRAW,ITRNDRCT);
                item_mark(MNDRAW,ITSHAPE,FALSE);
                item_mark(MNDRAW,ITLINE,TRUE);
        end

end

                        /* handle mode menu  */

item_mark(MNMODE,Winlist[oldind].selMod,FALSE);
item_mark(MNMODE,Winlist[Index].selMod,TRUE);

                        /* handle color menu  */

if (Winlist[Index].dodark != Winlist[oldind].dodark)
begin
        if (Winlist[oldind].dodark)
```

```
                begin
                        item_mark(MNCOLOR,ITLIGHT,TRUE);
                        item_mark(MNCOLOR,ITDARK,FALSE);
                end

                else
                begin
                        item_mark(MNCOLOR,ITDARK,TRUE);
                        item_mark(MNCOLOR,ITLIGHT,FALSE);
                end

        end

        item_mark(MNCOLOR,Winlist[oldind].selCol,FALSE);
        item_mark(MNCOLOR,Winlist[Index].selCol,TRUE);

        item_mark(MNPATTRN,Winlist[oldind].selPat,FALSE);
        item_mark(MNPATTRN,Winlist[Index].selPat,TRUE);

end



/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

DoScroll(part,newposn,amtmove)

        int     part,newposn,amtmove;

begin
        int             numscr,oldh,oldv,newh,newv,pixperscr;
        Rect            uprect;
        Window_id       Active;

        Active = get_active();
        numscr = 1;
        pixperscr = (SINGSCR * PAGES * COLPAGE) / MAXSCR;

        switch (part)
        begin

        case H_PAGEDOWN:
                numscr = COLPAGE;

        case H_ROWDOWN:
        begin
                numscr *= SINGSCR;
                newh = numscr / pixperscr;
                oldh = get_hscroll();
                newh += oldh;

                if (newh > MAXSCR)
                begin
```

```
                newh = MAXSCR;
                numscr = (newh - oldh) * pixperscr;
        end

        hscroll(numscr,&uprect);
        set_hscroll(newh);
        DoUpdate(Active,&uprect);
        break;
end;

case H_PAGEUP:
        numscr = COLPAGE;

case H_ROWUP:
begin
        numscr *= (- SINGSCR);
        newh = numscr / pixperscr;
        oldh = get_hscroll();
        newh += oldh;

        if (newh < 0)
        begin
                newh = 0;
                numscr = (newh - oldh) * pixperscr;
        end

        hscroll(numscr,&uprect);
        set_hscroll(newh);
        DoUpdate(Active,&uprect);
        break;
end;

case V_PAGEDOWN:
        numscr = COLPAGE;

case V_ROWDOWN:
begin
        numscr *= SINGSCR;
        newv = numscr / pixperscr;
        oldv = get_vscroll();
        newv += oldv;

        if (newv > MAXSCR)
        begin
                newv = MAXSCR;
                numscr = (newv - oldv) * pixperscr;
        end

        vscroll(numscr,&uprect);
        set_vscroll(newv);
        DoUpdate(Active,&uprect);
        break;
end;
```

```
        case V_PAGEUP:
                numscr = COLPAGE;

        case V_ROWUP:
        begin
                numscr *= (- SINGSCR);
                newv = numscr / pixperscr;
                oldv = get_vscroll();
                newv += oldv;

                if (newv < 0)
                begin
                        newv = 0;
                        numscr = (newv - oldv) * pixperscr;
                end

                vscroll(numscr,&uprect);
                set_vscroll(newv);
                DoUpdate(Active,&uprect);
                break;
        end;

        case H_THUMB:
        begin
                numscr = pixperscr * amtmove;
                hscroll(numscr,&uprect);
                set_hscroll(newposn);
                DoUpdate(Active,&uprect);
                break;
        end;

        case V_THUMB:
        begin
                numscr = pixperscr * amtmove;
                vscroll(numscr,&uprect);
                set_vscroll(newposn);
                DoUpdate(Active,&uprect);
                break;
        end;

        default:        break;

        end

end


/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

DoUpdate(Id,uprect)

        Window_id   Id;
        Rect        *uprect;
```

```
begin
        Bool            Flag;
        Rect            Dummy;
        int             I;
        int             tmpshape;
        Bool            tmpdofill;
        Int             Winindex;
        Window_id       tmpActive;


        Winindex = Findindex(Id);

        Flag = update_win(Id,uprect,&Dummy);
        tmpshape = Winlist[Winindex].Shape;
        tmpdofill = Winlist[Winindex].dofill;

        while (Flag)
        begin

                for(I = 0;I < Winlist[Winindex].drawcnt;I++)
                begin
                        Winlist[Winindex].dofill =
                                Winlist[Winindex].Drawn[I].drfill;
                        Winlist[Winindex].Shape =
                                Winlist[Winindex].Drawn[I].drshp;
                        set_xfer_mode(Winlist[Winindex].Drawn[I].drmo);
                        set_pattern(Winlist[Winindex].Drawn[I].drpat);
                        set_color(Winlist[Winindex].Drawn[I].drcol);
                        DrawShape(&(Winlist[Winindex].Drawn[I].drrct));
                end

                Flag = next_update(uprect,&Dummy);
        end

        Winlist[Winindex].dofill = tmpdofill;
        Winlist[Winindex].Shape = tmpshape;
        set_xfer_mode(Winlist[Winindex].winmode);
        set_pattern(Winlist[Winindex].winpat);
        set_color(Winlist[Winindex].wincol);

        end_update();
end

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

DrawShape(rct)

        Rect    *rct;

begin

        Int             Index;
```

```
Window_id     Active;

Active = get_active();
Index = Findindex(Active);

if (Winlist[Index].dofill)
begin
        switch (Winlist[Index].Shape)
        begin

                case ITRECT:
                begin
                        fillrect(rct);
                        break;
                end;

                case ITELLIP:
                begin
                        fillellipse(rct);
                        break;
                end;

                case ITARC90:
                begin
                        fillarc(rct,0,900);
                        break;
                end;

                case ITARC180:
                begin
                        fillarc(rct,0,1800);
                        break;
                end;

                case ITARC270:
                begin
                        fillarc(rct,0,2700);
                        break;
                end;

                case ITRNDRCT:
                begin
                        fillrndrct(rct);
                        break;
                end;

                default:        break;

        end

end

else
begin
```

```
            switch (Winlist[Index].Shape)
            begin

                case ITRECT:
                begin
                        drawrect(rct);
                        break;
                end;

                case ITELLIP:
                begin
                        drawellipse(rct);
                        break;
                end;

                case ITARC90:
                begin
                        drawarc(rct,0,900);
                        break;
                end;

                case ITARC180:
                begin
                        drawarc(rct,0,1800);
                        break;
                end;

                case ITARC270:
                begin
                        drawarc(rct,0,2700);
                        break;
                end;

                case ITRNDRCT:
                begin
                        drawrndrct(rct);
                        break;
                end;

                default:        break;

            end

        end

end

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

DoMouseDown(p1,mod)

        Point   *p1;
```

```
        int     mod;

begin

        Point           p2,p3;
        Rect            tempr;
        Color_id        tempcol;
        Mode_id                 tempmode;
        Pattern_id      temppat;
        Int             Index;
        Int             Drcnt;
        Window_id       Active;


        Active = get_active();
        Index = Findindex(Active);

        if (!Winlist[Index].doline)
        begin

                copypt(*p1,&p2);
                tempmode = get_xfer_mode();
                set_xfer_mode(XOR);
                tempcol = get_color();
                set_color(LTBLACK);
                temppat = get_pattern();
                set_pattern(HATCH);

                set_rect(p1,&p2,&tempr);
                drawrect(&tempr);

                while (!mouse_up())
                begin
                        get_mouse(Winlist[Index].winid,&p3);

                        if (!equalpt(&p2,&p3))
                        begin
                                drawrect(&tempr);
                                set_rect(p1,&p3,&tempr);
                                drawrect(&tempr);
                                copypt(p3,&p2);
                        end
                end

                drawrect(&tempr);
                set_xfer_mode(tempmode);
                set_color(tempcol);
                set_pattern(temppat);

                if (!equalpt(p1,&p2))
                begin
                        DrawShape(&tempr);
                        Drcnt = Winlist[Index].drawcnt;
```

```
                    copyrect(tempr,
                            &(Winlist[Index].Drawn[Drcnt].drrct));

                    Winlist[Index].Drawn[Drcnt].drfill =
                            Winlist[Index].dofill;

                    Winlist[Index].Drawn[Drcnt].drshp =
                            Winlist[Index].Shape;

                    Winlist[Index].Drawn[Drcnt].drmo = tempmode;
                    Winlist[Index].Drawn[Drcnt].drpat = temppat;
                    Winlist[Index].Drawn[Drcnt].drcol = tempcol;
                    Winlist[Index].drawcnt =
                            (Winlist[Index].drawcnt + 1) % NUMDR;
            end

            else
                    txtpen(p1);

        end

        else
        begin
            copypt(*p1,&p2);

            while (!mouse_up())
            begin
                    get_mouse(Winlist[Index].winid,&p3);

                    if (!equalpt(&p2,&p3))
                    begin
                            drawline(&p2,&p3);
                            copypt(p3,&p2);
                    end
            end

        end
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
        Void
ChDraw(itemnum)

        int     itemnum;

begin
        Int             Index;
        Window_id       Active;

        Active = get_active();

        if (Active == DESK_WIN)
                return;
```

```
Index = Findindex(Active);

switch (itemnum)
begin

        case ITOUTLN:
        begin
                Winlist[Index].dofill = FALSE;
                item_mark(MNDRAW,ITOUTLN,TRUE);
                item_mark(MNDRAW,ITFILL,FALSE);
                break;
        end;

        case ITFILL:
        begin
                Winlist[Index].dofill = TRUE;
                item_mark(MNDRAW,ITOUTLN,FALSE);
                item_mark(MNDRAW,ITFILL,TRUE);
                break;
        end;

        case ITSHAPE:
        begin
                if (Winlist[Index].doline)
                begin
                        Winlist[Index].doline = FALSE;
                        item_enable(MNDRAW,ITOUTLN);
                        item_enable(MNDRAW,ITFILL);
                        item_enable(MNDRAW,ITRECT);
                        item_enable(MNDRAW,ITELLIP);
                        item_enable(MNDRAW,ITARC90);
                        item_enable(MNDRAW,ITARC180);
                        item_enable(MNDRAW,ITARC270);
                        item_enable(MNDRAW,ITRNDRCT);
                        item_mark(MNDRAW,ITSHAPE,TRUE);
                        item_mark(MNDRAW,ITLINE,FALSE);
                end
                break;
        end;

        case ITLINE:
        begin
                if (!Winlist[Index].doline)
                begin
                        Winlist[Index].doline = TRUE;
                        item_disable(MNDRAW,ITOUTLN);
                        item_disable(MNDRAW,ITFILL);
                        item_disable(MNDRAW,ITRECT);
                        item_disable(MNDRAW,ITELLIP);
                        item_disable(MNDRAW,ITARC90);
                        item_disable(MNDRAW,ITARC180);
                        item_disable(MNDRAW,ITARC270);
                        item_disable(MNDRAW,ITRNDRCT);
```

```
                        item_mark(MNDRAW,ITSHAPE,FALSE);
                        item_mark(MNDRAW,ITLINE,TRUE);
                end
                break;
        end;

        default:
        begin
                item_mark(MNDRAW,Winlist[Index].Shape,FALSE);
                item_mark(MNDRAW,itemnum,TRUE);
                Winlist[Index].Shape = itemnum;
                break;
        end;

    end

end


/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
ChMode(itemnum)

    int     itemnum;

begin
    Int             Index;
    Window_id       Active;

    Active = get_active();

    if (Active == DESK_WIN)
        return;

    Index = Findindex(Active);

    switch (itemnum)
    begin

        case ITREPLCE:
        begin
            set_xfer_mode(REPLACE);
            break;
        end;

        case ITTRANS:
        begin
            set_xfer_mode(TRANSPAR);
            break;
        end;

        case ITXOR:
        begin
            set_xfer_mode(XOR);
```

```
                break;
        end;

        case ITREVTR:
        begin
                set_xfer_mode(REVTRANS);
                break;
        end;

        default:        break;

    end

    item_mark(MNMODE,Winlist[Index].selMod,FALSE);
    item_mark(MNMODE,itemnum,TRUE);

    Winlist[Index].selMod = itemnum;
    Winlist[Index].winmode = get_xfer_mode();

end


/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/

ChColor(itemnum)

    int     itemnum;

begin

    int             darkinc;
    int             tempc;
    Int             Index;
    Window_id       Active;

    Active = get_active();

    if (Active == DESK_WIN)
        return;


    Index = Findindex(Active);

    if (Winlist[Index].dodark)
        darkinc = DKWHITE;
    else
        darkinc = 0;


    switch (itemnum)
    begin

        case ITDARK:
```

```
begin
        item_mark(MNCOLOR,itemnum,TRUE);
        item_mark(MNCOLOR,ITLIGHT,FALSE);
        Winlist[Index].dodark = TRUE;
        tempc = get_color();
        tempc = (tempc % DKWHITE) + DKWHITE;
        set_color(tempc);
        break;
end;

case ITLIGHT:
begin
        item_mark(MNCOLOR,itemnum,TRUE);
        item_mark(MNCOLOR,ITDARK,FALSE);
        Winlist[Index].dodark = FALSE;
        tempc = get_color();
        tempc = (tempc % DKWHITE);
        set_color(tempc);
        break;
end;

case ITBLACK:
begin
        set_color(LTBLACK + darkinc);
        break;
end;

case ITWHITE:
begin
        set_color(LTWHITE + darkinc);
        break;
end;

case ITRED:
begin
        set_color(LTRED + darkinc);
        break;
end;

case ITGREEN:
begin
        set_color(LTGREEN + darkinc);
        break;
end;

case ITBLUE:
begin
        set_color(LTBLUE + darkinc);
        break;
end;

case ITCYAN:
begin
        set_color(LTCYAN + darkinc);
```

```
                break;
        end;

        case ITYELLOW:
        begin
                set_color(LTYELLOW + darkinc);
                break;
        end;

        case ITMAGENT:
        begin
                set_color(LTMAGENTA + darkinc);
                break;
        end;

        default:        break;

    end

    if ((itemnum != ITDARK) && (itemnum != ITLIGHT))
    begin
        item_mark(MNCOLOR,Winlist[Index].selCol,FALSE);
        item_mark(MNCOLOR,itemnum,TRUE);
        Winlist[Index].selCol = itemnum;
    end

    Winlist[Index].wincol = get_color();
end

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

ChPattern(itemnum)

    int     itemnum;

begin
    Int             Index;
    Window_id       Active;

    Active = get_active();

    if (Active == DESK_WIN)
        return;

    Index = Findindex(Active);

    switch (itemnum)
    begin

        case ITSOLID:
        begin
            set_pattern(SOLID);
            break;
```

```
                end;

                case ITHVYHT:
                begin
                        set_pattern(HEAVYHATCH);
                        break;
                end;

                case ITHATCH:
                begin
                        set_pattern(HATCH);
                        break;
                end;

                case ITLTHAT:
                begin
                        set_pattern(LTHATCH);
                        break;
                end;

                case ITEMPTY:
                begin
                        set_pattern(EMPTY);
                        break;
                end;

                default:        break;

        end

        item_mark(MNPATTRN,Winlist[Index].selPat,FALSE);
        item_mark(MNPATTRN,itemnum,TRUE);

        Winlist[Index].selPat = itemnum;
        Winlist[Index].winpat = get_pattern();

end


/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

        Void
ChWin(itemnum)

        Int     itemnum;

begin
        Int             oldIndex;
        Int             Index;
        Window_id       active;

        switch (itemnum)
        begin
```

```
        case ITWIN1:
        begin
                Index = 0;
                break;
        end;

        case ITWIN2:
        begin
                Index = 1;
                break;
        end;

        case ITWIN3:
        begin
                Index = 2;
                break;
        end;

        case ITWIN4:
        begin
                Index = 3;
                break;
        end;

        case ITWIN5:
        begin
                Index = 4;
                break;
        end;

        case ITWIN6:
        begin
                Index = 5;
                break;
        end;

        case ITWIN7:
        begin
                Index = 6;
                break;
        end;

        default:
        begin
                Index = INVALID;
                return;
                break;
        end;

end

if (Winlist[Index].Created)
begin
```

```
        if (Winlist[Index].Visible)
        begin
                hide_window(Winlist[Index].winid);
                Winlist[Index].Visible = FALSE;
                oldIndex = Index;

                active = get_active();
                Index = Findindex(active);

                if (active != DESK_WIN)
                        ResetMenus(oldIndex,Index);
                else
                        Lastactive = oldIndex;

                item_mark(MNWIN,itemnum,FALSE);
        end

        else
        begin
                active = get_active();
                oldIndex = Findindex(active);
                show_window(Winlist[Index].winid);
                ResetMenus(oldIndex,Index);
                Winlist[Index].Visible = TRUE;
                item_mark(MNWIN,itemnum,TRUE);
        end

end

else
begin
        active = get_active();
        oldIndex = Findindex(active);

        Winlist[Index].winid = set_new_window(&winrect,
                W_NAME | W_SIZE | W_CLOSE | W_HSCROLL | W_VSCROLL,
                Title[Index],TRUE);

        Winlist[Index].wincol = LTBLACK;
        Winlist[Index].winpat = SOLID;
        Winlist[Index].winmode = REPLACE;
        Winlist[Index].Shape = ITRECT;

        Winlist[Index].selPat = ITSOLID;
        Winlist[Index].selCol = ITBLACK;
        Winlist[Index].selMod = ITREPLCE;

        Winlist[Index].dodark = TRUE;
        Winlist[Index].dofill = FALSE;
        Winlist[Index].doline = FALSE;

        Winlist[Index].Created = TRUE;
        Winlist[Index].Visible = TRUE;
```

```
                Winlist[Index].drawcnt = 0;

                ResetMenus(oldIndex,Index);
                item_mark(MNWIN,itemnum,TRUE);

        end

end

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

DoMenu(menunum,itemnum)

        int     menunum,itemnum;

begin
        switch (menunum)
        begin

                case MNDRAW:
                begin
                        ChDraw(itemnum);
                        menu_hilight(MNDRAW,FALSE);
                        break;
                end;

                case MNMODE:
                begin
                        ChMode(itemnum);
                        menu_hilight(MNMODE,FALSE);
                        break;
                end;

                case MNCOLOR:
                begin
                        ChColor(itemnum);
                        menu_hilight(MNCOLOR,FALSE);
                        break;
                end;

                case MNPATTRN:
                begin
                        ChPattern(itemnum);
                        menu_hilight(MNPATTRN,FALSE);
                        break;
                end;

                case MNWIN:
                begin
                        ChWin(itemnum);
                        menu_hilight(MNWIN,FALSE);
                        break;
                end;
```

```
                default:        break;

        end

end


/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
        Void
DoKey(inchr,inmod)

        Char    inchr;
        Int     inmod;

begin
        Int     width;
        Int     height;
        Point   Penloc;

        switch (inchr)
        begin

                case CARR_RET:
                begin
                        height = get_hchar();
                        set_txtpen(&Penloc);
                        Penloc.h = 0;
                        Penloc.v += height;
                        txtpen(&Penloc);
                        break;
                end;

                case BACK_SP:
                begin
                        width = get_wchar();
                        set_txtpen(&Penloc);
                        Penloc.h -= width;
                        txtpen(&Penloc);
                        drawchar(BLANK);
                        txtpen(&Penloc);
                        break;
                end;

                default:
                begin
                        if ((inchr >= BLANK) && (inchr <= '~'))
                                drawchar(inchr);
                        break;
                end;

        end
```

```
end

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

evtloop()

begin

        Bool    Stop;
        Int     Index;
        Int     oldIndex;

        Stop = FALSE;

        while (!Stop)
        begin

                get_event();

                switch (EVTTYPE)
                begin

                        case CLOSEWIN:
                        begin
                                Stop = TRUE;
                                break;
                        end;

                        case SCROLLBAR:
                        begin
                                DoScroll(EVTSCRPART,EVTSCRPOSN,
                                    EVTSCRMOVE);
                                break;
                        end;

                        case KEYBOARD:
                        begin
                                DoKey(EVTKEY,EVTMOD);
                                break;
                        end;

                        case TOPPED:
                        begin
                                if (EVTWINDOW != DESK_WIN)
                                begin
                                        oldIndex = Findindex(get_active());
                                        Index = Findindex(EVTWINDOW);
                                        ResetMenus(oldIndex,Index);
                                        activate_win(EVTWINDOW);
                                end
                                break;
                        end;
```

```
                    case MOUSEUP:       break;

                    case MOUSEDOWN:
                    begin
                            if(EVTWINDOW != DESK_WIN)
                                    DoMouseDown(&(EVPOINT),EVTMOD);
                            break;
                    end;

                    case REDRAW:
                    begin
                            DoUpdate(EVTWINDOW,&EVTRECT);
                            break;
                    end;

                    case MENUHIT:
                    begin
                            DoMenu(EVTMTITLE,EVTMITEM);
                            break;
                    end;

                    default:        break;

            end
        end
end

/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/


ASMAIN()

begin

        sys_init();
        init_menu("TEST5.RSC",TEST5BAR);

        set_point(10,10,&Tl);
        set_point(300,300,&Br);
        set_rect(&Tl,&Br,&winrect);

        Winlist[0].winid = set_new_window(&winrect,
                        W_NAME | W_SIZE | W_CLOSE | W_HSCROLL | W_VSCROLL,
                        Title[0],TRUE);

        set_color(LTBLACK);
        set_pattern(SOLID);
        set_xfer_mode(REPLACE);

        Winlist[0].wincol = LTBLACK;
        Winlist[0].winpat = SOLID;
        Winlist[0].winmode = REPLACE;
```

```
        Winlist[0].Shape = ITRECT;

        Winlist[0].selPat = ITSOLID;
        Winlist[0].selCol = ITBLACK;
        Winlist[0].selMod = ITREPLCE;

        Winlist[0].dodark = TRUE;
        Winlist[0].dofill = FALSE;
        Winlist[0].doline = FALSE;

        Winlist[0].Created = TRUE;
        Winlist[0].Visible = TRUE;
        Winlist[0].drawcnt = 0;

        evtloop();
        sys_end();

end
```

```
/*-------------------------------------------------------------------*/
/*                          ASFBIND.H                                */
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/* Interface Specifications for functions used to initialize the     */
/* interface.                                                        */
/*-------------------------------------------------------------------*/


                        /* sys_init()                                */
extern  State
        sys_init();

                        /* sys_end()                                 */
extern  State
        sys_end();



/*-------------------------------------------------------------------*/
/* Interface Specifications for functions used to manipulate the     */
/* primitive data type point (in file Asprim.c).                     */
/*-------------------------------------------------------------------*/


                        /* set_point(Horiz,Vert,&DestPoint)          */
extern  State
        set_point();

                        /* Horiz = get_x_coord(&InputPoint)          */
extern  Int
        get_x_coord();

                        /* Vertical = get_y_coord(&InputPoint)       */
extern  Int
        get_y_coord();

                        /* Flag = equalpt(&Point1,&Point2)           */
extern  Bool
        equalpt();

                        /* copypt(&SourcePoint,&DestPoint)           */
extern  State
        copypt();


/*-------------------------------------------------------------------*/
/* Interface Specifications for functions used to manipulate the     */
/* primitive data type rectangle (in file Asprim.c).                 */
/*-------------------------------------------------------------------*/
```

```
                        /* set_rect(&Point1,&Point2,&DestRect)      */
extern  State
        set_rect();

                        /* set_topLeft(&SourceRect,&DestPoint)      */
extern  State
        set_topLeft();

                        /* set_botRight(&SourceRect,&DestPoint)     */
extern  State
        set_botRight();

                        /* Flag = pt_in_rect(&QPoint,&TgtRect)      */
extern  Bool
        pt_in_rect();

                        /* set_insect_rect(&Rect1,&Rect2,&DestRect) */
extern  State
        set_insect_rect();

                        /* Flag = insect_rect(&Rect1,&Rect2)        */
extern  Bool
        insect_rect();

                        /* Flag = equalrect(&Rect1,&Rect2)          */
extern  Bool
        equalrect();

                        /* copyrect(&SourceRect,&DestRect)                  */
extern  State
        copyrect();


/*-------------------------------------------------------------------*/
/* Interface Specifications for functions used to manipulate   */
/* windows as a whole entity. (in file Aswin.c).               */
/*-------------------------------------------------------------------*/


                        /* set_new_window(&DefRect,Partspec,Titlestr,   */
                        /*              Visible)                        */
extern  Window_id
        set_new_window();

                        /* activate_win(WindowId)                       */
extern  State
        activate_win();

                        /* hide_window(WindowId)                        */
extern  State
        hide_window();
```

```
                        /* show_window(WindowId)                    */
extern  State
        show_window();

                        /* close_window(WindowId)                   */
extern  State
        close_window();

                        /* Flag = update_win(WindowId,&UpdRect,&InctRect)*/
extern  Bool
        update_win();

                        /*Flag = next_update(WindowId,&UpdRect,&InctRect)*/
extern  Bool
        next_update();

                        /* end_update()                             */
extern  State
        end_update();

                        /* WindowId = get_active()                  */
extern  Window_id
        get_active();


/*-------------------------------------------------------------------------------*/
/* Interface Specifications for functions used to manipulate the scroll          */
/* bar portions of windows. (in file Aswin.c).                                   */
/*-------------------------------------------------------------------------------*/


                        /* hscroll(NumberPixels,&UpdRect)           */
extern  State
        hscroll();

                        /* vscroll(NumberPixels,&UpdRect)           */
extern  State
        vscroll();

                        /* set_hscroll(Value)                       */
extern  State
        set_hscroll();

                        /* set_vscroll(Value)                       */
extern  State
        set_vscroll();

                        /* Value = get_hscroll()                    */
extern  Int
        get_hscroll();

                        /* Value = get_vscroll()                    */
extern  Int
```

```
        get_vscroll();

/*-------------------------------------------------------------------*/
/* Interface Specifications for functions used to manipulate the*/
/* drawing environment of windows. (in file Aswin.c).        */
/*-------------------------------------------------------------------*/


                        /* set_xfer_mode(NewModeId)              */
extern  State
        set_xfer_mode();

                        /* set_pattern(NewPatternId)             */
extern  State
        set_pattern();

                        /* set_color(NewColorId)                 */
extern  State
        set_color();

                        /* ColorId = get_color()                 */
extern  Color_id
        get_color();

                        /* ModeId = get_xfer_mode()              */
extern  Mode_id
        get_xfer_mode();

                        /* Pattern_id = get_pattern()            */
extern  Pattern_id
        get_pattern();


/*-------------------------------------------------------------------------*/
/* Interface Specifications for functions used for drawing graphic    */
/* objects into windows. (in file Aswin.c).                           */
/*-------------------------------------------------------------------------*/



                        /* drawline(&StartPoint,&EndPoint)       */
extern  State
        drawline();

                        /* drawrect(&InputRect)                  */
extern  State
        drawrect();

                        /* drawellipse(&InputRect)               */
extern  State
        drawellipse();

                        /* drawarc(&InputRect,BeginAng,EndAng)   */
```

```
extern  State
        drawarc();

                        /* drawrndrect(&InputRect)                  */
extern  State
        drawrndrect();

                        /* fillrect(&InputRect)                     */
extern  State
        fillrect();

                        /* fillellipse(&InputRect)                  */
extern  State
        fillellipse();

                        /* fillarc(&InputRect,BeginAng,EndAng)      */
extern  State
        fillarc();

                        /* fillrndrect(&InputRect)                  */
extern  State
        fillrndrect();


/*-------------------------------------------------------------------*/
/* Interface Specifications for functions used for text              */
/* manipulation within windows. (in file Aswin.c).                   */
/*-------------------------------------------------------------------*/



                        /* txtpen(&InputPoint)                      */
extern  State
        txtpen();

                        /* set_txtpen(&DestPoint)                   */
extern  State
        set_txtpen();

                        /* drawstring(&String)                      */
extern  State
        drawstring();

                        /* drawchar(Character)                      */
extern  State
        drawchar();

                        /* CharWidth = get_wchar()                  */
extern  Int
        get_wchar();

                        /* CharHeight = get_hchar()                 */
extern  Int
        get_hchar();
```

```
/*-------------------------------------------------------------------------*/
/* Interface Specifications for functions used to manipulate menus         */
/* (in file Asmenu.c).                                                     */
/*-------------------------------------------------------------------------*/


                        /* init_menu(&ResourceName,MenuBarId)   */
extern  State
        init_menu();

                        /* item_enable(MenuNumber,ItemNumber)   */
extern  State
        item_enable();

                        /* item_disable(MenuNumber,ItemNumber)  */
extern  State
        item_disable();

                        /* item_mark(MenuNum,ItemNum,ToMark)    */
extern  State
        item_check();

                        /* menu_hilight(MenuNum,ToHilight)      */
extern  State
        menu_hilight();


/*-------------------------------------------------------------------------*/
/* Interface Specifications for event manager functions (in file           */
/* Asevt.c).                                                               */
/*-------------------------------------------------------------------------*/


                        /* get_event()                          */
extern  State
        get_event();

                        /* get_mouse(WindowId,&DestPoint)       */
extern  State
        get_mouse();

                        /* Flag = mouse_up()                    */
extern  Bool
        mouse_up();
```

APPENDIX B

Mac implementation of Common Interface

```
/*-------------------------------------------------------------------*/
/*                          ASBIND.H  (for Demo.c use)    */
/*-------------------------------------------------------------------*/

#include "MacTypes.h"

#define     begin       {
#define     end         }


typedef     int         Bool;
typedef     int         Int;
typedef     char        Char;
typedef     long        Long;
typedef     unsigned int    Bit16;

#define     State       void
#define     Void        void

typedef     int         Pattern_id;
typedef     int         Mode_id;
typedef     int         Color_id;
typedef     int         Window_id;

#define     W_NAME          0X0009
#define     W_CLOSE         0X0002
#define     W_SIZE          0x0020
#define     W_HSCROLL       0x0E00
#define     W_VSCROLL       0x01C0

#define     INVAL_WIN       -1
#define     DESK_WIN        0
#define     MAXNUMWIN       7

#define     SOLID           1
#define     HEAVYHATCH      2
#define     HATCH           3
#define     LTHATCH         4
#define     EMPTY           5

#define     LTWHITE         0
#define     LTBLACK         1
#define     LTRED           2
#define     LTGREEN         3
#define     LTBLUE          4
#define     LTCYAN          5
#define     LTYELLOW        6
#define     LTMAGENTA       7
#define     DKWHITE         8
```

```
#define     DKBLACK         9
#define     DKRED           10
#define     DKGREEN         11
#define     DKBLUE          12
#define     DKCYAN          13
#define     DKYELLOW        14
#define     DKMAGENTA       15

#define     REPLACE         1
#define     TRANSPAR        2
#define     XOR             3
#define     REVTRANS        4

#define     FALSE           0x0000
#define     TRUE            0x0001

#define     POINTER(x)      (int)(x)
#define     ASMAIN()        main()

typedef     struct  Evtmsg
begin
        int             type;
        Window_id       winid;
        Rect            evrec;
        Point           evpoint;
        int             scrpart;
        int             scrposn;
        int             scrmoved;
        char            keystroke;
        int             mod;
        int             mtitle;
        int             mitem;
end             Evtmsg;

extern  Evtmsg          Message;

#define     EVTTYPE         Message.type
#define     EVTWINDOW       Message.winid
#define     EVTRECT         Message.evrec
#define     EVPOINT         Message.evpoint
#define     EVTSCRPART      Message.scrpart
#define     EVTSCRPOSN      Message.scrposn
#define     EVTSCRMOVE      Message.scrmoved
#define     EVTKEY          Message.keystroke
#define     EVTMOD          Message.mod
#define     EVTMTITLE       Message.mtitle
#define     EVTMITEM        Message.mitem


#define     REDRAW          0
#define     TOPPED          1
#define     CLOSEWIN        2
#define     SCROLLBAR       3
#define     MOUSEDOWN       4
```

```
#define     KEYBOARD        5
#define     MOUSEUP         6
#define     MENUHIT         7

#define     V_PAGEUP        0
#define     V_PAGEDOWN      1
#define     V_ROWUP         2
#define     V_ROWDOWN       3
#define     H_PAGEUP        4
#define     H_PAGEDOWN      5
#define     H_ROWUP         6
#define     H_ROWDOWN       7
#define     V_THUMB         8
#define     H_THUMB         9

#define     MINSCR          0
#define     MAXSCR          1000

#define     DESKMENU        32767

#define     NUL_CHR         '\0'
#define     CARR_RET        0x0D
#define     BACK_SP         0x08
#define     BLANK           0x20
```

```
/*-------------------------------------------------------------------*/
/*                        DEMO.H  (for Demo.c use)                   */
/*-------------------------------------------------------------------*/

#define     INVALID         -1

#define     TEST5BAR        12

#define     MNDRAW          13
#define     ITOUTLN         1
#define     ITFILL          2
#define     ITRECT          4
#define     ITELLIP         5
#define     ITARC90         6
#define     ITARC180        7
#define     ITARC270        8
#define     ITRNDRCT        9
#define     ITSHAPE         11
#define     ITLINE          12

#define     MNMODE          14
#define     ITREPLCE        1
#define     ITTRANS         2
#define     ITXOR           3
#define     ITREVTR         4

#define     MNCOLOR         15
#define     ITDARK          1
#define     ITLIGHT         2
#define     ITBLACK         4
#define     ITWHITE         5
#define     ITRED           6
#define     ITGREEN         7
#define     ITBLUE          8
#define     ITCYAN          9
#define     ITYELLOW        10
#define     ITMAGENT        11

#define     MNPATTRN        16
#define     ITSOLID         1
#define     ITHVYHT         2
#define     ITHATCH         3
#define     ITLTHAT         4
#define     ITEMPTY         5

#define     MNWIN           20
#define     ITWIN1          1
#define     ITWIN2          2
#define     ITWIN3          3
#define     ITWIN4          4
#define     ITWIN5          5
#define     ITWIN6          6
#define     ITWIN7          7
```

```
/*-------------------------------------------------------------------*/
/*                          ASPRIM.C  (for Demo.c use)               */
/*-------------------------------------------------------------------*/

#include        "Asbind1.h"

/*-------------------------------------------------------------------*/
/* get_x_coord: Function which returns the horizontal               */
/*        coordinate of the input point pt.                         */
/*-------------------------------------------------------------------*/

Int     get_x_coord(pt)

        Point   *pt;

begin
        return((*pt).h);
end

/*-------------------------------------------------------------------*/
/* get_y_coord: Function which returns the vertical coordinate      */
/*         of the input point pt.                                    */
/*-------------------------------------------------------------------*/

Int     get_y_coord(pt)

        Point   *pt;

begin
        return((*pt).v);
end

/*-------------------------------------------------------------------*/
/* set_topLeft: Function which returns the top left point of the    */
/*        input rectangle r as p.                                   */
/*-------------------------------------------------------------------*/

State   set_topLeft(r,p)

        Rect    *r;
        Point   *p;

begin
        (*p).h = (*r).left;
        (*p).v = (*r).top;
end

/*-------------------------------------------------------------------*/
/* set_botRight: Function which returns the bottom right            */
/*        point of the inputrectangle r as p.                       */
/*-------------------------------------------------------------------*/

State   set_botRight(r,p)
```

```
        Rect    *r;
        Point   *p;

begin
        (*p).h = (*r).right;
        (*p).v = (*r).bottom;
end

/*-------------------------------------------------------------------------*/
/* pt_in_rect: Function which determines if the input point p is  */
/*       within or on the border of the input rectangle r.         */
/*-------------------------------------------------------------------------*/

Bool    pt_in_rect(p,r)

        Point   *p;
        Rect    *r;

begin
        return(PtInRect(*p, r));
end

/*-----------------------------------------------------------------------------------*/
/* set_insect_rect: Function which determines the rectangle which                    */
/*       is formed by the intersection of the input rectangles r1 and r2.  The       */
/*       resulting rectangle is returned in rint.  If the intersection is non-       */
/*       empty, the rectangle returned in rint will be defined by  top left and     */
/*       bottom right points of (0,0).                                               */
/*-----------------------------------------------------------------------------------*/

State   set_insect_rect(r1,r2,rint)

        Rect    *r1, *r2, *rint;

begin
        SectRect(r1, r2, rint);
end

/*-------------------------------------------------------------------------*/
/* insect_rect: Function which determines if the input                     */
/*       rectangles r1 and r2 intersect.                                   */
/*-------------------------------------------------------------------------*/

Bool    insect_rect(r1,r2)

        Rect            *r1, *r2;

begin
        Rect            *rint;

        return(SectRect(r1, r2, rint));
end

/*-------------------------------------------------------------------------*/
```

```
/* equalrect: Function which determines if the two input          */
/*       rectangles are the same rectangle.                       */
/*-------------------------------------------------------------------*/

Bool    equalrect(r1,r2)

        Rect            *r1, *r2;

begin
        return(EqualRect(r1, r2));
end

/*-------------------------------------------------------------------*/
/* equalpt: Function which determines if the two input points     */
/*       are the same point.                                      */
/*-------------------------------------------------------------------*/

Bool    equalpt(p1,p2)

        Point   *p1, *p2;

begin
        return(EqualPt(*p1,*p2));
end
/*-------------------------------------------------------------------*/
/* copypt: Function which copies the source point into the    */
/*       destination point.                                       */
/*-------------------------------------------------------------------*/

State   copypt (source,dest)

        Point   source, *dest;

begin
        (*dest).h = source.h;
        (*dest).v = source.v;
end


/*-------------------------------------------------------------------*/
/* copyrect: Function which copies the source rectangle       */
/*       into the destination rectangle.                          */
/*-------------------------------------------------------------------*/

State   copyrect (source,dest)

        Rect            source,*dest;

begin
        (*dest).left = source.left;
        (*dest).top = source.top;
        (*dest).right = source.right;
        (*dest).bottom = source.bottom;
end
```

```
/*-------------------------------------------------------------------------*/
/* set_point: Given two integers which represent the x and y               */
/*       coordinates (the new horizontal and vertical positions            */
/*       of the point), the function returns a modified point.             */
/*-------------------------------------------------------------------------*/

State   set_point(x,y,pt)

        int             x,y;
        Point   *pt;

begin
        SetPt(pt,x,y);
end

/*-------------------------------------------------------------------------------------*/
/* set_rect: Function which, given two points, determines the smallest                 */
/*       rectangle that those points could define and sets the top left and            */
/*       bottom right points of the output rectangle r to correspond to that           */
/*       rectangle.                                                                    */
/*-------------------------------------------------------------------------------------*/

State   set_rect(p1,p2,r)

        Point   *p1,*p2;
        Rect            *r;

begin
                                        /* case 1 p2 is to the right and below p1 */
        if (((*p2).h >= (*p1).h) && ((*p2).v >= (*p1).v))
                SetRect(r,(*p1).h, (*p1).v, (*p2).h, (*p2).v);

                                        /* case 2 p1 is to the right and below p2 */
        else if (((*p1).h >= (*p2).h) && ((*p1).v >= (*p2).v))
                SetRect(r, (*p2).h, (*p2).v, (*p1).h, (*p1).v);

                                        /* case 3 p1 is to the right and above p2 */
        else if (((*p1).h <= (*p2).h) && ((*p1).v >= (*p2).v))
                SetRect(r,(*p1).h, (*p2).v, (*p2).h, (*p1).v);

                                        /* case 4 p2 is to the right and above p1 */
        else if (((*p2).h <= (*p1).h) && ((*p2).v >= (*p1).v))
                SetRect(r,(*p2).h, (*p1).v, (*p1).h, (*p2).v);
end
```

```
/*-------------------------------------------------------------------*/
/*                          ASEVT.C                                  */
/*-------------------------------------------------------------------*/

#include "asevti.c"

State   get_event()
begin

        EventRecord             myEvent;
        PenState                thePen;

        evtstop = false;
        while (!evtstop) begin

                SystemTask();
                GetNextEvent(everyEvent, &myEvent);
                switch(myEvent.what) begin
                case mouseDown:
                        MDEvent(myEvent);
                        break;

                case autoKey:
                case keyDown:
                        EVTTYPE = KEYBOARD;
                        EVTKEY = (char)(0x7F & LoWord(myEvent.message));
                        EVTMOD = myEvent.modifiers;
                        evtstop = true;
                        break;
                case updateEvt:
                        EVTWINDOW = GetWRefCon(myEvent.message);

                        SetRect(&EVTRECT,0,0,0,0);
                        EVTTYPE = REDRAW;
                        evtstop = true;

                        SetPort(myEvent.message);
                        GetPenState(&thePen);
                        PenMode(patCopy);
                        PenPat(black);

                        SetOrigin(WindList[EVTWINDOW].Wholewin.top,
                                WindList[EVTWINDOW].Wholewin.left);
                        ClipRect(&(WindList[EVTWINDOW].Wholewin));

                        if ((WindList[EVTWINDOW].Parts & W_SIZE) == W_SIZE)
                                DrawGrowIcon(WindList[EVTWINDOW].Winhandle);

                        DrawControls(myEvent.message);

                        SetOrigin(WindList[EVTWINDOW].Workwin.top,
                                WindList[EVTWINDOW].Workwin.left);
                        ClipRect(&(WindList[EVTWINDOW].Workwin));
```

```
                        SetPenState(&thePen);

                        SetPort(WindList[Active_win].Winhandle);

                        break;
                default:break;
                end
        end
end

/*-------------------------------------------------------------------------------------*/
/* get_mouse: Function which gets the current mouse position and outputs   */
/*       it in the local coordinate system of the window specified by Id.  */
/*-------------------------------------------------------------------------------------*/

State   get_mouse(Id,pt)

        Int             Id;
        Point   *pt;

begin

        GrafPtr         tempport;

        GetPort(&tempport);
        SetPort(WindList[Id].Winhandle);
        GetMouse(pt);
        SetPort(tempport);
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

Bool    mouse_up()

begin
        return(!Button());
end
```

```
/*-----------------------------------------------------------------*/
/*                          ASEVTI.C                               */
/*-----------------------------------------------------------------*/

#include "Windecl.h"

static  Bool            evtstop;
        Evtmsg          Message;
extern  Window_id       Active_win;    /* index of active window      */
extern  Winrec          WindList[MAXNUMREC];

State   MDEvent(event)

        EventRecord             event;

begin
        WindowPtr               MyWindow;
        Window_id               winID;
        Int                     location;
        GrafPtr                 tempport;
        ControlHandle           whscroll;
        Int                     part,modpart,hval,vval;
        Rect                    arect,brect;
        Long                    amtmove;
        Long                    menuresp;

        location = FindWindow(event.where,&MyWindow);

        if (MyWindow != NIL)
                EVTWINDOW = GetWRefCon(MyWindow);
        else
                EVTWINDOW = 0;
        if ((EVTWINDOW != Active_win)&&(location != inMenuBar)) begin
                GetPort(&tempport);
                SetPort(MyWindow);
                GlobalToLocal(&(event.where));

                EVTTYPE = TOPPED;
                EVTMOD = event.modifiers;
                copypt(event.where,&EVPOINT);
                SetPort(tempport);
                evtstop = true;
        end

        else
        begin
                switch (location)
                begin
                case    inMenuBar:
                        menuresp = MenuSelect(event.where);
                        EVTMTITLE = HiWord(menuresp);
                        EVTMITEM = LoWord(menuresp);
                        EVTTYPE = MENUHIT;
                        evtstop = true;
```

```
        break;

case    inContent:
        GetPort(&tempport);
        SetPort(MyWindow);

        copypt(event.where, &EVPOINT);
        SetOrigin(0,0);
        ClipRect(&(WindList[EVTWINDOW].Wholewin));

        GlobalToLocal(&(event.where));
        part = FindControl(event.where,MyWindow,&whscroll);

        if (part == 0) begin
                SetOrigin(WindList[EVTWINDOW].Workwin.left,
                        WindList[EVTWINDOW].Workwin.top);
                ClipRect(&(WindList[EVTWINDOW].Workwin));
                GlobalToLocal(&EVPOINT);
                EVTTYPE = MOUSEDOWN;
                EVTMOD = event.modifiers;
                SetPort(tempport);
                evtstop = TRUE;
        end

        else if ((whscroll == WindList[EVTWINDOW].Hscrhandle) ||
                (whscroll == WindList[EVTWINDOW].Vscrhandle)) begin

                EVTTYPE = SCROLLBAR;
                hval = GetCtlValue(WindList[EVTWINDOW].Hscrhandle);
                vval = GetCtlValue(WindList[EVTWINDOW].Vscrhandle);

                modpart = TrackControl(whscroll,event.where,0);

                if (modpart == part)
                begin
                  if (whscroll == WindList[EVTWINDOW].Vscrhandle)
                  begin

                                switch (modpart)
                                begin

                                case inPageUp:
                                        EVTSCRPART = V_PAGEUP;
                                        break;

                                case inPageDown:
                                        EVTSCRPART = V_PAGEDOWN;
                                        break;


                                case inUpButton:
                                        EVTSCRPART = V_ROWUP;
                                        break;
```

```
                case inDownButton:
                        EVTSCRPART = V_ROWDOWN;
                        break;

                case inThumb:
                        EVTSCRPART = V_THUMB;
                        break;


                default: break;
        end

        EVTSCRMOVE =
                GetCtlValue(WindList[EVTWINDOW].Vscrhandle)
                                                        - vval;
        EVTSCRPOSN =
                GetCtlValue(WindList[EVTWINDOW].Vscrhandle);
        SetCtlValue(WindList[EVTWINDOW].Vscrhandle,vval);
end

else
begin

        switch (modpart)
        begin

                case inPageUp:
                        EVTSCRPART = H_PAGEUP;
                        break;

                case inPageDown:
                        EVTSCRPART = H_PAGEDOWN;
                        break;

                case inUpButton:
                        EVTSCRPART = H_ROWUP;
                        break;

                case inDownButton:
                        EVTSCRPART = H_ROWDOWN;
                        break;

                case inThumb:
                        EVTSCRPART = H_THUMB;
                        break;

                default: break;
        end

        EVTSCRMOVE =
                GetCtlValue(WindList[EVTWINDOW].Hscrhandle)
                                                        - hval;
```

```
                    EVTSCRPOSN =
                        GetCtlValue(WindList[EVTWINDOW].Hscrhandle);
                    SetCtlValue(WindList[EVTWINDOW].Hscrhandle,hval);
                end

        end     /* if */

        SetOrigin(WindList[EVTWINDOW].Workwin.left,

        WindList[EVTWINDOW].Workwin.top);
        ClipRect(&(WindList[EVTWINDOW].Workwin));
        SetPort(tempport);
        evtstop = TRUE;
end             /* else */

break;

case inDrag:
        if (EVTWINDOW == Active_win)
        begin
                SetRect(&brect,-32000,20,32000,32000);
                DragWindow(MyWindow,event.where,&brect);

                SetOrigin(WindList[EVTWINDOW].Wholewin.left,
                        WindList[EVTWINDOW].Wholewin.top);
                ClipRect(&(WindList[EVTWINDOW].Wholewin));

                if (WindList[EVTWINDOW].Parts & W_SIZE)
                        DrawGrowIcon(MyWindow);

                DrawControls(MyWindow);

                SetOrigin(WindList[EVTWINDOW].Workwin.left,
                        WindList[EVTWINDOW].Workwin.top);
                ClipRect(&(WindList[EVTWINDOW].Workwin));

        end
        break;

case inGrow:
        if (EVTWINDOW == Active_win)
        begin
                SetRect(&brect,40,40,1000,1000);

                SetOrigin(WindList[EVTWINDOW].Wholewin.left,
                        WindList[EVTWINDOW].Wholewin.top);
                ClipRect(&(WindList[EVTWINDOW].Wholewin));

                amtmove = GrowWindow(MyWindow,event.where,
                                                &brect);
                hval = LoWord(amtmove);
                vval = HiWord(amtmove);
                copyrect(WindList[EVTWINDOW].Workwin,&brect);
                SizeWindow(MyWindow,hval,vval,FALSE);
```

```
copyrect((*MyWindow).portRect,&(WindList[EVTWINDOW].Wholewin));

copyrect((*MyWindow).portRect,&(WindList[EVTWINDOW].Workwin));
                        copyrect((*MyWindow).portRect,&arect);
                        arect.right -= 16;
                        arect.bottom -= 16;
                        ClipRect(&(WindList[EVTWINDOW].Wholewin));

                        if (((WindList[EVTWINDOW].Parts & W_SIZE) > 0) &&
                                (amtmove != 0))
                        begin
                                DrawGrowIcon(MyWindow);
                                WindList[EVTWINDOW].Workwin.bottom -= 16;
                                WindList[EVTWINDOW].Workwin.right -= 16;
                        end
                        ClipRect(&arect);

                        if (((WindList[EVTWINDOW].Parts & W_HSCROLL) > 0)
                                && (amtmove != 0))
                        begin
                        SizeControl(WindList[EVTWINDOW].Hscrhandle,
                                        hval - 15,16);

                        MoveControl(WindList[EVTWINDOW].Hscrhandle,
                                WindList[EVTWINDOW].Wholewin.left,
                                WindList[EVTWINDOW].Wholewin.bottom - 16);

                        if (!((WindList[EVTWINDOW].Parts & W_SIZE) > 0))
                                WindList[EVTWINDOW].Workwin.bottom -= 16;
                        end

                        if (((WindList[EVTWINDOW].Parts & W_VSCROLL) > 0)
                                && (amtmove != 0))
                        begin
                        SizeControl(WindList[EVTWINDOW].Vscrhandle,
                                                16, vval - 15);
                        MoveControl(
                                WindList[EVTWINDOW].Vscrhandle, hval - 16,0);

                        if (!((WindList[EVTWINDOW].Parts & W_SIZE) > 0))
                                WindList[EVTWINDOW].Workwin.right -= 16;

                        end

                        ClipRect(&(WindList[EVTWINDOW].Wholewin));
                        DrawControls(MyWindow);

                        if (amtmove != 0)
                                OffsetRect(&(WindList[EVTWINDOW].Workwin),
                                                brect.left,brect.top);
                        else
                                copyrect(brect,
```

```
                                        &(WindList[EVTWINDOW].Workwin));

                                SetOrigin(WindList[EVTWINDOW].Workwin.left,
                                        WindList[EVTWINDOW].Workwin.top);
                                ClipRect(&(WindList[EVTWINDOW].Workwin));
                                InvalRect(&(WindList[EVTWINDOW].Workwin));
                                ValidRect(&brect);

                        end  /* if */
                        break;

                case inGoAway:
                        if (TrackGoAway(MyWindow,event.where))
                        begin
                                EVTTYPE = CLOSEWIN;
                                evtstop = TRUE;
                        end

                        break;

                default: break;

                end
        end
end
```

```
/*-----------------------------------------------------------------*/
/*                          ASWIN.C                                */
/*-----------------------------------------------------------------*/

#include        "Windecl.h"
#include        "aswini.c"


/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/

State   sys_end()

begin
   ExitToShell();
end

/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/

State   sys_init()

begin
        InitGraf(&thePort);
        InitFonts();
        FlushEvents(everyEvent, 0);
        InitWindows();
        InitMenus();
        TEInit();
        InitDialogs(&sys_end);
        InitCursor();
        wind_init();
end

/*-------------------------------------------------------------------------*/
/* activate_win: Function which causes the specified window to become      */
/*   the active window.  It causes any window (but the desktop with a      */
/*   id number of 0) to be moved to the top and a new backround will       */
/*   be drawn in, however, the contents will not be automatically          */
/*   redrawn.                                                              */
/*-------------------------------------------------------------------------*/

State   activate_win(Id)

        Int     Id;

begin

        if((Id != Active_win) && (Id != 25))
        begin
                        /* if control bars present remove them from the window      */
                        /* being deactivated                                        */

                if (Active_win != DESK_WIN)
```

```
begin
        SetPort(WindList[Active_win].Winhandle);
        SetOrigin(WindList[Active_win].Wholewin.left,
                WindList[Active_win].Wholewin.top);
        ClipRect(&(WindList[Active_win].Wholewin));

        if ((WindList[Active_win].Parts & W_HSCROLL) ==
                W_HSCROLL)
                HideControl(WindList[Active_win].Hscrhandle);

        if ((WindList[Active_win].Parts & W_VSCROLL) ==
                W_VSCROLL)
                HideControl(WindList[Active_win].Vscrhandle);

end
/* Draw the grow box and scroll bars in the window being activated */

SelectWindow(WindList[Id].Winhandle);
Last_active = Active_win;
Active_win = Id;
SetPort(WindList[Id].Winhandle);

SetOrigin(WindList[Active_win].Wholewin.left,
                WindList[Active_win].Wholewin.top);
ClipRect(&(WindList[Active_win].Wholewin));

if ((WindList[Active_win].Parts & W_SIZE) == W_SIZE)
        DrawGrowIcon(WindList[Active_win].Winhandle);

if ((WindList[Active_win].Parts & W_HSCROLL) == W_HSCROLL)
        ShowControl(WindList[Active_win].Hscrhandle);

if ((WindList[Active_win].Parts & W_VSCROLL) == W_VSCROLL)
        ShowControl(WindList[Active_win].Vscrhandle);

SetOrigin(WindList[Active_win].Workwin.left,
                WindList[Active_win].Workwin.top);
ClipRect(&(WindList[Active_win].Workwin));


                /* erase the grow box in the newly inactive window */
if (Last_active != DESK_WIN)
begin
        SetPort(WindList[Last_active].Winhandle);

        if ((WindList[Last_active].Parts & W_SIZE) == W_SIZE)
                DrawGrowIcon(WindList[Last_active].Winhandle);

        SetOrigin(WindList[Last_active].Workwin.left,
                        WindList[Last_active].Workwin.top);
        ClipRect(&(WindList[Last_active].Workwin));

        SetPort(WindList[Id].Winhandle);
```

```
            end

        end
end

/*-------------------------------------------------------------------------------*/
/* show_window: Function which draws an invisible but previously defined  */
/*    window onto the screen.  This window becomes the active window.     */
/*-------------------------------------------------------------------------------*/

State   show_window(Id)

        Int     Id;

begin

        WindowPtr    tempptr;

        if (Id != DESK_WIN)
        begin
            if (!WindList[Id].Wdefrec.visible)
            begin
                ShowWindow(WindList[Id].Winhandle);
                activate_win(Id);
            end
        end
end

/*-------------------------------------------------------------------------*/
/* hide_window: Function which removes the specified window     */
/*    from the screen without deallocating it.                  */
/*-------------------------------------------------------------------------*/

State   hide_window(Id)

        Int     Id;

begin

        WindowPtr    tempptr;
        Window_id    newactid;

        if ((Id != DESK_WIN) && (WindList[Id].Wdefrec.visible))
        begin
            HideWindow(WindList[Id].Winhandle);

            if ((Id == Active_win) && any_visible())
            begin
                tempptr = FrontWindow();
                newactid = GetWRefCon(tempptr);
                activate_win(newactid);
            end

        end
```

```
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

Window_id   set_new_window(InitRect, Partspec, Title, is_Visible)

            Rect            *InitRect;
            Bit16           Partspec;
            Char            *Title;
            Boolean                 is_Visible;

begin

            Bool        IfWName;
            Bool        IfWClose;
            Bool        IfWSize;
            Bool        IfWScrollH;
            Bool        IfWScrollV;
            Bool        IfShrunk;
            Int             oldRef;
            Int             procID;
            WindowPtr   myWindow;
            Char        *Name,temp[255];
            Rect        vScrollRect,hScrollRect,tempWdef;
static  Int             refCon = 0; /* Reference constant for new window   */


            IfWName    = Partspec & W_NAME;
            IfWClose   = Partspec & W_CLOSE;
            IfWSize    = Partspec & W_SIZE;
            IfWScrollH = Partspec & W_HSCROLL;
            IfWScrollV = Partspec & W_VSCROLL;



            if (!get_next_rec(&refCon))
                    return(INVAL_WIN);

            WindList[refCon].Parts = Partspec;
            SetPt(&(WindList[refCon].Txtpen),0,20);
            copyrect(*InitRect, &tempWdef);
            tempWdef.top += 20;
            OffsetRect(&tempWdef,0,20);

            if (IfWSize)
                    procID = documentProc;
            else begin
                    if (IfWName || IfWClose)
                            procID = noGrowDocProc;
                    else
                            procID = plainDBox;
            end
```

```
if (IfWName) begin
        strcpy(temp,Title);
        CtoPstr(temp);
        Name = temp;
end else
        Name = "\p";
myWindow = NewWindow(&(WindList[refCon].Wdefrec), &tempWdef,
                        Name, false, procID, NIL, IfWClose, refCon);
Available_win[refCon] = false;
WindList[refCon].Winhandle = myWindow;
SetPort(myWindow);

copyrect((*myWindow).portRect, &(WindList[refCon].Wholewin));
copyrect((*myWindow).portRect, &(WindList[refCon].Workwin));
if (IfWSize) begin
        IfShrunk = true;
        WindList[refCon].Workwin.bottom -= 17;
        WindList[refCon].Workwin.right -= 17;
end else IfShrunk = false;

if (IfWScrollH) begin
        copyrect(WindList[refCon].Wholewin, &hScrollRect);
        hScrollRect.top = hScrollRect.bottom-16;
        hScrollRect.right -= 15;
        WindList[refCon].Hscrhandle =
                NewControl(myWindow, &hScrollRect, "\p", false,
                MINSCR, MINSCR, MAXSCR,scrollBarProc,refCon);
        if (!IfShrunk)
                WindList[refCon].Workwin.bottom -= 16;
end else
WindList[refCon].Hscrhandle = 0;

if (IfWScrollV) begin
        copyrect(WindList[refCon].Wholewin, &vScrollRect);
        vScrollRect.left = vScrollRect.right-16;
        vScrollRect.bottom -= 15;
        WindList[refCon].Vscrhandle =
                NewControl(myWindow, &vScrollRect,"\p", false,
                MINSCR,MINSCR, MAXSCR,scrollBarProc,refCon);
        if (!IfShrunk)
                WindList[refCon].Workwin.right -= 16;
end else WindList[refCon].Vscrhandle = 0;

ClipRect(&(WindList[refCon].Workwin));

WindList[refCon].wincol = LTBLACK;
WindList[refCon].winpat = SOLID;
WindList[refCon].winmode = REPLACE;
TextMode(srcBic);
TextFont(monaco);

if (is_Visible)
        show_window(refCon);
else
```

```
                SetPort(WindList[Active_win].Winhandle);
        return(refCon);

end

/*-------------------------------------------------------------------------*/
/* set_pattern: Function which sets the pattern to be used to draw     */
/*    and fill in shapes in the active window.                         */
/*-------------------------------------------------------------------------*/

   State
set_pattern(newpattern)

        Pattern_id      newpattern;

begin

        if (((WindList[Active_win].wincol == DKWHITE) ||
                (WindList[Active_win].wincol == LTWHITE)) &&
                        (WindList[Active_win].winmode == REPLACE))
                PenPat(black);

        else
        begin
                switch (newpattern)
                begin

                        case HEAVYHATCH:
                                PenPat(dkGray);
                                break;

                        case HATCH:
                                PenPat(gray);
                                break;

                        case LTHATCH:
                                PenPat(ltGray);
                                break;

                        case EMPTY:
                                PenPat(white);
                                break;

                        default:
                                PenPat(black);
                                break;
                end
        end

        WindList[Active_win].winpat = newpattern;

end

/*-------------------------------------------------------------------------*/
```

```
/* set_xfer_mode: function which will set the mode for drawing into        */
/*    the active window.                                                   */
/*-------------------------------------------------------------------------*/

   State
set_xfer_mode(newmode)

        Mode_id                 newmode;

begin
        if ((WindList[Active_win].wincol == DKWHITE) ||
                (WindList[Active_win].wincol == LTWHITE))
        begin

                switch (newmode)
                begin
                        case    TRANSPAR:
                                PenMode(patBic);
                                TextMode(srcBic);
                                break;

                        case XOR:
                                PenMode(patXor);
                                TextMode(srcXor);
                                break;

                        case REVTRANS:
                                PenMode(notPatBic);
                                TextMode(srcBic);
                                break;

                        default:
                                PenMode(notPatCopy);
                                TextMode(srcBic);
                                PenPat(black);
                                break;
                end
        end

        else
        begin

                switch (newmode)
                begin
                        case    TRANSPAR:
                                PenMode(patOr);
                                TextMode(srcOr);
                                break;

                        case XOR:
                                PenMode(patXor);
                                TextMode(srcXor);
                                break;
```

```
                        case REVTRANS:
                                PenMode(notPatOr);
                                TextMode(srcOr);
                                break;

                        default:
                                PenMode(patCopy);
                                TextMode(srcOr);
                                break;
                end

                set_pattern(WindList[Active_win].winpat);
        end

        WindList[Active_win].winmode = newmode;
end


/*--------------------------------------------------------------------*/
/* set_color: Function which sets the global color for drawing. */
/*--------------------------------------------------------------------*/

   State
set_color(newcolor)

        Int     newcolor;

begin

        Int     theColor;

        switch(newcolor)
        begin
                case LTBLACK:
                case DKBLACK:
                        theColor = blackColor;
                        break;
                case LTWHITE:
                case DKWHITE:
                        theColor = whiteColor;
                        break;
                case LTRED:
                case DKRED:
                        theColor = redColor;
                        break;
                case LTGREEN:
                case DKGREEN:
                        theColor = greenColor;
                        break;
                case LTBLUE:
                case DKBLUE:
                        theColor = blueColor;
                        break;
                case LTCYAN:
```

```
                case DKCYAN:
                        theColor = cyanColor;
                        break;
                case LTYELLOW:
                case DKYELLOW:
                        theColor = yellowColor;
                        break;
                case LTMAGENTA:
                case DKMAGENTA:
                        theColor = magentaColor;
                        break;
                default:break;
        end
        ForeColor(theColor);
        WindList[Active_win].wincol = newcolor;
end

/*-------------------------------------------------------------------------*/
/* get_active: Function which returns the Id of the active window.    */
/*-------------------------------------------------------------------------*/

Window_id     get_active()

begin
        return(Active_win);
end

/*-------------------------------------------------------------------------*/
/* drawline: Function which draws a line in the currently active window.*/
/*   Input coordinates are relative to the top left hand corner of the     */
/*   active window.                                                        */
/*-------------------------------------------------------------------------*/

    State
drawline(St_pt,End_pt)

        Point   *St_pt,*End_pt;

begin
        MoveTo((*St_pt).h,(*St_pt).v);
        LineTo((*End_pt).h,(*End_pt).v);
end

/*-------------------------------------------------------------------------*/
/* drawrect: Function to draw the outline of a rectangle in the active     */
/*    window.  The coordinates of the input rectangle are asumed to be     */
/*    relative to the top left corner of the active window's work area.    */
/*-------------------------------------------------------------------------*/

    State
drawrect(In_rect)

        Rect            *In_rect;
```

```
begin

        FrameRect(In_rect);

end


/*-------------------------------------------------------------------------------*/
/* drawellipse: Function which draws an ellipse within the area of the           */
/*    active window specified by the input rectangle.  The coordinates           */
/*    of the input rectangle are assumed to be relative to the top left          */
/*    corner of the work area of the active window.                              */
/*-------------------------------------------------------------------------------*/

    State
drawellipse(In_rect)

        Rect            *In_rect;

begin

        FrameOval(In_rect);

end

/*-------------------------------------------------------------------------*/
/*  drawarc: Function which draws an elliptical arc between the two        */
/*    input angles (begang and endang) specified and within the            */
/*    rectangular area of the active window specified.  The input          */
/*    rectangle is assumed to be relative to the top left corner of the    */
/*    work area of the active window.                                      */
/*-------------------------------------------------------------------------*/

    State
drawarc(R,begang,endang)

        Rect            *R;
        Int             begang,endang;

begin
        begang = (begang/10);
        endang = (endang/10);

        FrameArc(R,begang,endang);
end

/*-------------------------------------------------------------------------------*/
/* drawrndrect: Function which draws the outline of a  rounded rectangle         */
/*   within the specified rectangular area of the active window.                 */
/*-------------------------------------------------------------------------------*/

    State
drawrndrct(In_rect)
```

```
        Rect            *In_rect;

begin
        Int             width,height;

        width = RRWIDTH;
        height = RRHEIGHT;

        FrameRoundRect(In_rect,width,height);
end

/*-------------------------------------------------------------------*/
/* fillrect: Function which draws a pattern within the specified  */
/*   rectangular area of the active window.                          */
/*-------------------------------------------------------------------*/

   State
fillrect(In_rect)

        Rect            *In_rect;

begin

        PaintRect(In_rect);

end

/*-----------------------------------------------------------------------*/
/* fillellipse: Function which fills an ellipse within the area of the   */
/*   active window specified by the input rectangle. The coordinates     */
/*   of the input rectangle are assumed to be relative to the top left   */
/*   corner of the work area of the active window.                       */
/*-----------------------------------------------------------------------*/

   State
fillellipse(In_rect)

        Rect            *In_rect;

begin

        PaintOval(In_rect);

end

/*-----------------------------------------------------------------------------*/
/*  fillarc: Function which fills an elliptical arc between the two            */
/*   input angles (begang and endang) specified and within the                 */
/*   rectangular area of the active window specified. The input                */
/*   rectangle is assumed to be relative to the top left corner of the         */
/*   work area of the active window. Angles are reversed in the GEM            */
/*   function call to force correspondence to Mac.                             */
/*-----------------------------------------------------------------------------*/
```

```
    State
fillarc(R,begang,endang)

        Rect            *R;
        Int             begang,endang;

begin
        begang = (begang/10);
        endang = (endang/10);

        PaintArc(R,begang,endang);
end

/*-------------------------------------------------------------------------------*/
/* fillrndrect: Function which fills the outline of a  rounded rectangle        */
/*   within the specified rectangular area of the active window.                 */
/*-------------------------------------------------------------------------------*/

    State
fillrndrct(In_rect)

        Rect            *In_rect;

begin
        Int             width,height;

        width = RRWIDTH;
        height = RRHEIGHT;

        PaintRoundRect(In_rect,width,height);
end

/*-------------------------------------------------------------------------*/
/* get_color: Function which returns the drawing color of the active  */
/*       window.                                                           */
/*-------------------------------------------------------------------------*/

        Color_id
get_color()

begin
        return(WindList[Active_win].wincol);
end

/*-------------------------------------------------------------------------------*/
/* get_pattern: Function which returns the drawing pattern of the active */
/*       window.                                                                 */
/*-------------------------------------------------------------------------------*/

        Pattern_id
get_pattern()

begin
        return(WindList[Active_win].winpat);
```

```
end

/*-----------------------------------------------------------------------*/
/* get_xfer_mode: Function which returns the drawing transfer  */
/*        mode of the currently active window.                 */
/*-----------------------------------------------------------------------*/

        Mode_id
get_xfer_mode()

begin
        return(WindList[Active_win].winmode);
end

/*-----------------------------------------------------------------------*/
/*      txtpen: Function which sets the location where          */
/*      the next call to drawchar or drawstring will place      */
/*      that string  in the active  window.                     */
/*-----------------------------------------------------------------------*/


        State
txtpen(inpt)

        Point   *inpt;

begin
        copypt(*inpt,&(WindList[Active_win].Txtpen));
end



/*-----------------------------------------------------------------------*/
/* set_txtpen: Function which returns the location where       */
/*      the next call to drawchar or drawstring will place      */
/*      that string  in the active window.                      */
/*-----------------------------------------------------------------------*/
        State
set_txtpen(pen)

        Point   *pen;

begin
        copypt(WindList[Active_win].Txtpen,pen);
end


/*-------------------------------------------------------------------------------*/
/* drawstring: Function which draws the input string into the active           */
/*      window.  Note that at present, the MacIntosh Monaco font is            */
/*      used (see the initialization in set_new_window) and the string         */
/*      drawing transfer modes are limited to transparent and xor for          */
/*      the time being.                                                        */
/*-------------------------------------------------------------------------------*/
```

```
        State
drawstring(strptr)

        Char    strptr[];

begin

        Char            *newstrptr;
        Char            tempstr[250];
        Int             length;

        length = strlen(strptr);
        strcpy(tempstr,strptr);

        *newstrptr = CtoPstr(tempstr);

        MoveTo(WindList[Active_win].Txtpen.h,WindList[Active_win].Txtpen.v);
        DrawString(newstrptr);

        GetPen(&(WindList[Active_win].Txtpen));

end


/*-------------------------------------------------------------------------------*/
/* drawchar: Function which draws the input character into the active            */
/*       window.  Note that at present, the MacIntosh Monaco font is             */
/*       used (see the initialization in set_new_window) and the string          */
/*       drawing transfer modes are limited to transparent and xor for           */
/*       the time being.                                                         */
/*-------------------------------------------------------------------------------*/

drawchar(inchr)

        Char    inchr;

begin

        MoveTo(WindList[Active_win].Txtpen.h,WindList[Active_win].Txtpen.v);
        DrawChar(inchr);

        GetPen(&(WindList[Active_win].Txtpen));

end


/*-------------------------------------------------------------------------------*/
/* get_wchar: Function which returns the width of the characters being           */
/*       drawn onto the screen.  This function assumes that a MacIntosh          */
/*       fixed width font (such as Monaco) is used for the interface.            */
/*-------------------------------------------------------------------------------*/

        Int
```

```
get_wchar()

begin

        FontInfo        info;
        Int             height;

        GetFontInfo(&info);
        return(info.widMax);

end


/*-------------------------------------------------------------------*/
/* get_hchar: Function which returns the width of the          */
/*       characters being drawn onto the screen.               */
/*-------------------------------------------------------------------*/

        Int
get_hchar()

begin

        FontInfo        info;
        Int             height;

        GetFontInfo(&info);
        height = info.ascent + info.descent + info.leading;
        return(height);

end

/*-------------------------------------------------------------------------*/
/* close_window: Function which permanently closes the specified  */
/*       window  anddeallocates its window record.                 */
/*-------------------------------------------------------------------------*/

        State
close_window(Id)

        Window_id    Id;

begin

        Int      Recnum;

                        /* determine if the window id refers to */
                        /* a declared window                    */

        Available_win[Id] = true;
                        /* if so, dispose of it                 */

        hide_window(Id);
        CloseWindow(WindList[Id].Winhandle);      /*user w record storage*/
```

```
end

/*-------------------------------------------------------------------------------*/
/* update_win: Function which sets the system into the update window     */
/*  mode.  In this mode, drawing will be limited to the visible region   */
/*  of the window to be updated (as identified by the ID number input)   */
/*  to the function.  When given an rectangular area to update, the      */
/*  update region will be replaced by this rectangle.  Input of an       */
/*  empty rectangle signifies that the update is in responce to a        */
/*  system generated update event.  The programmer should not change     */
/*  the rectangle provided with the update event (by the event manager)  */
/*  but pass it on unmodified to this function.                          */
/*-------------------------------------------------------------------------------*/

Bool    update_win(ID,Up_rct,Dr_rct)

        Int             ID;
        Rect            *Up_rct,*Dr_rct;

begin
        WindowPtr    tempport;
        GetPort(&tempport);
        SetPort(WindList[ID].Winhandle);

                        /* If the input rectangle is not empty indicating    */
                        /* that the user is not responding to a system       */
                        /* update event, make the input rectangle the        */
                        /* update region.
                        */

        if (!EmptyRect(Up_rct))
        begin
                ValidRect(&(WindList[ID].Workwin));
                InvalRect(Up_rct);
        end

        if((!EmptyRgn(WindList[ID].Wdefrec.updateRgn)) && (!Update_in_prog))
        begin
                copyrect(WindList[ID].Workwin,Dr_rct);

                SetOrigin(WindList[ID].Wholewin.left,
                                WindList[ID].Wholewin.top);
                ClipRect(&(WindList[ID].Wholewin));

                DrawControls(WindList[ID].Winhandle);

                SetOrigin(WindList[ID].Workwin.left,
                                WindList[ID].Workwin.top);
                ClipRect(&(WindList[ID].Workwin));

                Update_in_prog = TRUE;
                Last_active = Active_win;
```

```
                Active_win = ID;
                BeginUpdate(WindList[ID].Winhandle);
                EraseRgn((*WindList[ID].Winhandle).visRgn);
                return(TRUE);
        end

        else
        begin
                SetPort(tempport);
                Update_in_prog = FALSE;
                SetRect(Dr_rct,0,0,0,0);
                return(FALSE);
        end
end

/*-------------------------------------------------------------------------*/
/* next_update: A dummy function in the MacIntosh implementation */
/*       which always returns FALSE.                                       */
/*-------------------------------------------------------------------------*/

Bool    next_update(Up_rct,Dr_rct)

        Rect    *Up_rct,*Dr_rct;

begin
        SetRect(Dr_rct,0,0,0,0);
        return(FALSE);
end

/*-------------------------------------------------------------------------*/
/* end_update: procedure to end the update mode and restore the      */
/*   clip area to match the active (topmost) window.                       */
/*-------------------------------------------------------------------------*/

State   end_update()

begin
        if(Update_in_prog)
        begin
                EndUpdate(WindList[Active_win].Winhandle);
                Active_win = Last_active;
                SetPort(WindList[Active_win].Winhandle);
                Update_in_prog = FALSE;
        end
end

/*-----------------------------------------------------------------------------*/
/* hscroll: Function which scrolls the content area of the active window       */
/*     by the number of "pixels" specified by num.  If the num is              */
/*     positive, the region will  move to the left, and to the right if        */
/*     negative.                                                               */
/*-----------------------------------------------------------------------------*/

State   hscroll(num,Up_rect)
```

```
        Int             num;
        Rect            *Up_rect;

begin

        RgnHandle               Temprgn;

        SetRect(Up_rect,0,0,0,0);
        if(num != 0)
        begin

                Temprgn = NewRgn();
                ScrollRect(&(WindList[Active_win].Workwin),-num,0,Temprgn);
                OffsetRect(&(WindList[Active_win].Workwin),num,0);
                copyrect(WindList[Active_win].Workwin,Up_rect);

                if(num > 0)
                        (*Up_rect).left = (*Up_rect).right - num;
                else
                        (*Up_rect).right = (*Up_rect).left - num;

                SetOrigin(WindList[Active_win].Workwin.left,
                                WindList[Active_win].Workwin.top);
                ClipRect(&(WindList[Active_win].Workwin));
                DisposeRgn(Temprgn);
        end
end

/*-------------------------------------------------------------------------------*/
/* vscroll: Function which scrolls the content area of the active window       */
/*     by the number of "pixels" specified by num.  If the num is              */
/*     positive, the region will  move up,and down if negative.                */
/*-------------------------------------------------------------------------------*/

State   vscroll(num,Up_rect)

        Int             num;
        Rect    *Up_rect;

begin

        RgnHandle               Temprgn;

        SetRect(Up_rect,0,0,0,0);
        if(num != 0)
        begin

                Temprgn = NewRgn();
                ScrollRect(&(WindList[Active_win].Workwin),0,-num,Temprgn);
                OffsetRect(&(WindList[Active_win].Workwin),0,num);
                copyrect(WindList[Active_win].Workwin,Up_rect);

                if(num > 0)
```

```
                (*Up_rect).top = (*Up_rect).bottom - num;
        else
                (*Up_rect).bottom = (*Up_rect).top - num;

        SetOrigin(WindList[Active_win].Workwin.left,
                WindList[Active_win].Workwin.top);
        ClipRect(&(WindList[Active_win].Workwin));
        DisposeRgn(Temprgn);
    end
end

/*-------------------------------------------------------------------------------*/
/* get_hscroll: Function which returns the horizontal scroll bar value.         */
/*-------------------------------------------------------------------------------*/

Int             get_hscroll()

begin
    if ((WindList[Active_win].Parts & W_HSCROLL) > 0)
        return(GetCtlValue(WindList[Active_win].Hscrhandle));
    else
        return(-1);
end

/*-------------------------------------------------------------------------*/
/* get_vscroll: Function which returns the vertical scroll bar value.     */
/*-------------------------------------------------------------------------*/

Int             get_vscroll()

begin
    if ((WindList[Active_win].Parts & W_VSCROLL) > 0)
        return(GetCtlValue(WindList[Active_win].Vscrhandle));
    else
        return(-1);
end

/*-------------------------------------------------------------------------*/
/* set_hscroll: Function which sets the value of the horizontal            */
/*    scroll bar of the active window to the input val.                    */
/*-------------------------------------------------------------------------*/

State   set_hscroll(val)

    Int     val;

begin
    if (val < MINSCR)
        val = MINSCR;
    else if (val > MAXSCR)
        val = MAXSCR;

    if (WindList[Active_win].Parts & W_HSCROLL)
    begin
```

```
                SetOrigin(0,0);
                ClipRect(&(WindList[Active_win].Wholewin));
                SetCtlValue(WindList[Active_win].Hscrhandle,val);
                SetOrigin(WindList[Active_win].Workwin.left,
                        WindList[Active_win].Workwin.top);
                ClipRect(&(WindList[Active_win].Workwin));
        end
end

/*-------------------------------------------------------------------------*/
/* set_vscroll: Function which sets the value of the vertical scroll     */
/*    bar to the input val.                                                */
/*-------------------------------------------------------------------------*/

State   set_vscroll(val)

        Int     val;

begin
        if (val < MINSCR)
                val = MINSCR;
        else if (val > MAXSCR)
                val = MAXSCR;

        if (WindList[Active_win].Parts & W_VSCROLL)
        begin
                SetOrigin(0,0);
                ClipRect(&(WindList[Active_win].Wholewin));
                SetCtlValue(WindList[Active_win].Vscrhandle,val);
                SetOrigin(WindList[Active_win].Workwin.left,
                WindList[Active_win].Workwin.top);
                ClipRect(&(WindList[Active_win].Workwin));
        end
end
```

```
/*-------------------------------------------------------------------*/
/*                          ASWINI.C                                 */
/*-------------------------------------------------------------------*/

Window_id    Active_win;
Winrec       WindList[MAXNUMREC];
Window_id    Last_active;          /* index of previous active window   */
Bool         Update_in_prog;       /* is update occuring                */
                                   /* array of available record indices */
Bool         Available_win[MAXNUMWIN];

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

State   wind_init()

begin

        Int                 i;
        WindowPtr    Wmgr;

        GetPort(&Wmgr);

        WindList[DESK_WIN].Winhandle = Wmgr;
        WindList[DESK_WIN].Parts = 0;
        SetPt(&(WindList[DESK_WIN].Txtpen),0,0);

        Available_win[0] = false;
        for (i=1; i <= MAXNUMWIN; i++)
                Available_win[i] = true;

        Active_win = DESK_WIN;
        Last_active = DESK_WIN;
        Update_in_prog = false;

        WindList[Active_win].wincol = LTBLACK;
        WindList[Active_win].winpat = SOLID;
        WindList[Active_win].winmode = REPLACE;
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

Bool    get_next_rec(ref)

        Int     *ref;

begin
        Int     i;

        i = 1;
        while ((i <= MAXNUMWIN)&&(!Available_win[i]))
                i++;
        if (i > MAXNUMWIN)
```

```
                return(false);
        else begin
                *ref = i;
                return(true);
        end
end

/*-------------------------------------------------------------------------*/
/* any_visible: Function which returns TRUE if any user defined     */
/*      window is visible on the screen.                                   */
/*-------------------------------------------------------------------------*/

Bool    any_visible()

begin
        Int     I;

        for (I = 0; I < MAXNUMWIN; I++)
        begin
                if (!Available_win[I])
                begin
                        if (WindList[I].Wdefrec.visible)
                                return(TRUE);
                end

        end

        return(FALSE);
end
```

```
/*-------------------------------------------------------------------*/
/*                          WINDECL.H                                */
/*-------------------------------------------------------------------*/

#include        "Asbind1.h"

typedef         struct  Winrec                  /* Window record structure (abs spec)     */
begin
        WindowRecord            Wdefrec;        /* Mac window record structure            */
        WindowPtr       Winhandle;              /* Mac window pointer(window Graf port)*/
        Rect            Wholewin;               /* Rectangle for work area + scroll bars  */
                                                /* top left corner always at (0,0) local  */
        Rect            Workwin;                /* Rectangle for work area - scroll bars  */
                                                /* top left corner in sync with scrolled  */
                                                /* picture                                */
        Bit16           Parts;                  /* spec for parts included in window      */
        ControlHandle           Hscrhandle;     /* handle for horizontal scroll bar       */
        ControlHandle           Vscrhandle;     /* handle for vertical scroll bar         */
        Point           Txtpen;                 /* location to draw next text             */
        Mode_id                 winmode;        /* drawing transfer mode for window       */
        Color_id        wincol;                 /* drawing color for window               */
        Pattern_id      winpat;                 /* drawing pattern for window             */
end Winrec;
```

```
/*-------------------------------------------------------------------*/
/*                          ASMENU.C                                 */
/*-------------------------------------------------------------------*/

#include "asbind1.h"

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
State init_menu(filename,barid)

        Char    *filename;
        Int     barid;

begin
        Handle     barhand;
        MenuHandle  deskhand;

        CtoPstr(filename);
        OpenResFile(filename);
        barhand = GetNewMBar(barid);
        if (barhand != 0)
        begin
                deskhand = GetMenu(DESKMENU);
                AddResMenu(deskhand, 'DRVR');
                SetMenuBar(barhand);
                DrawMenuBar();
        end
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

State   item_enable(menunum,itemnum)

        Int             menunum,itemnum;

begin
        MenuHandle   temphand;

        temphand = GetMHandle(menunum);
        EnableItem(temphand,itemnum);
end


/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

State   item_disable(menunum,itemnum)

        Int             menunum,itemnum;

begin
        MenuHandle   temphand;
```

```
        temphand = GetMHandle(menunum);
        DisableItem(temphand,itemnum);
end


/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/

State   item_mark(menunum,itemnum,mark)

        Int     menunum,itemnum;
        Bool    mark;

begin
        Int                     i;
        MenuHandle   temphand;

        if (itemnum==0) begin
                temphand = GetMHandle(menunum);
                for (i= 1;i<=CountMItems(temphand);i++)
                        CheckItem(temphand,i,mark);
        end else begin
                temphand = GetMHandle(menunum);
                CheckItem(temphand,itemnum,mark);
        end
end

/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/

State   menu_hilight(menunum,hilight)

        Int             menunum;
        Bool    hilight;

begin
        if (hilight)
                HiliteMenu(menunum);
        else
                HiliteMenu(0);
end
```

```
/*-------------------------------------------------*/
/*                  ASBIND1.H                      */
/*-------------------------------------------------*/

#include "Quickdraw.h"
#include "WindowMgr.h"
#include "ControlMgr.h"
#include "MenuMgr.h"
#include "EventMgr.h"
#include "FontMgr.h"

#define         begin           {
#define         end             }
#define         NIL             0

typedef         int             Bool;
typedef         int             Int;
typedef         char            Char;
typedef         long            Long;
typedef         unsigned int    Bit16;

#define         State           void
#define         Void            void

typedef         int             Pattern_id;
typedef         int             Mode_id;
typedef         int             Color_id;
typedef         int             Window_id;

#define         W_NAME              0X0009
#define         W_CLOSE             0X0002
#define         W_SIZE              0x0020
#define         W_HSCROLL           0x0E00
#define         W_VSCROLL           0x01C0

#define         INVAL_WIN           -1
#define         DESK_WIN            0
#define         MAXNUMWIN           7
#define         MAXNUMREC           8

#define         SOLID               1
#define         HEAVYHATCH          2
#define         HATCH               3
#define         LTHATCH             4
#define         EMPTY               5

#define         LTWHITE             0
#define         LTBLACK             1
#define         LTRED               2
#define         LTGREEN             3
#define         LTBLUE              4
#define         LTCYAN              5
#define         LTYELLOW            6
#define         LTMAGENTA           7
```

```
#define         DKWHITE         8
#define         DKBLACK         9
#define         DKRED           10
#define         DKGREEN         11
#define         DKBLUE          12
#define         DKCYAN          13
#define         DKYELLOW        14
#define         DKMAGENTA       15

#define         REPLACE         1
#define         TRANSPAR        2
#define         XOR             3
#define         REVTRANS        4

#define         FALSE           0x0000
#define         TRUE            0x0001

#define         POINTER(x)      (int)(x)
#define         ASMAIN()        main()

typedef         struct  Evtmsg
begin
        int             type;
        Window_id       winid;
        Rect            evrec;
        Point           evpoint;
        int             scrpart;
        int             scrposn;
        int             scrmoved;
        char            keystroke;
        int             mod;
        int             mtitle;
        int             mitem;
end             Evtmsg;

#define         EVTTYPE         Message.type
#define         EVTWINDOW       Message.winid
#define         EVTRECT         Message.evrec
#define         EVPOINT         Message.evpoint
#define         EVTSCRPART      Message.scrpart
#define         EVTSCRPOSN      Message.scrposn
#define         EVTSCRMOVE      Message.scrmoved
#define         EVTKEY          Message.keystroke
#define         EVTMOD          Message.mod
#define         EVTMTITLE       Message.mtitle
#define         EVTMITEM        Message.mitem


#define         REDRAW          0
#define         TOPPED          1
#define         CLOSEWIN        2
#define         SCROLLBAR       3
#define         MOUSEDOWN       4
```

```
#define     KEYBOARD        5
#define     MOUSEUP         6
#define     MENUHIT         7

#define     V_PAGEUP        0
#define     V_PAGEDOWN      1
#define     V_ROWUP         2
#define     V_ROWDOWN       3
#define     H_PAGEUP        4
#define     H_PAGEDOWN      5
#define     H_ROWUP         6
#define     H_ROWDOWN       7
#define     V_THUMB         8
#define     H_THUMB         9

#define     MINSCR          0
#define     MAXSCR          1000

typedef     int             Menu_id;

#define     DESKMENU        32767

#define     RRHEIGHT        15
#define     RRWIDTH         15

#define     NUL_CHR         '\0'
#define     CARR_RET        0x0D
#define     BACK_SP         0x08
#define     BLANK           0x20
```

```
/*-----------------------------------------------------------------*/
/*                      ASBIND.H (for Demo.c use)                  */
/*-----------------------------------------------------------------*/

#define         begin   {
#define         end     }

typedef struct  Point
begin
        int     v,h;
end
Point;

typedef struct  Rect
begin
        Point   topLeft;
        Point   botRight;
end
Rect;

typedef         int     Bool;

#define         Void    /**/
#define         State   /**/

typedef         int             Int;
typedef          long           Long;
typedef         char            Char;
typedef         unsigned int    Bit16;

typedef         int     Pattern_id;
typedef         int     Mode_id;
typedef         int     Color_id;
typedef         int     Window_id;
typedef         int     Menu_id;

#define         W_NAME          0x0009
#define         W_CLOSE         0X0002
#define         W_SIZE          0x0020
#define         W_HSCROLL       0x0E00
#define         W_VSCROLL       0X01C0

#define         INVAL_WIN       -1
#define         DESK_WIN        0
#define         MAXNUMWIN       7

#define         SOLID           1
#define         HEAVYHATCH      2
#define         HATCH           3
#define         LTHATCH         4
#define         EMPTY           5

#define         LTWHITE         0
```

```
#define     LTBLACK         1
#define     LTRED           2
#define     LTGREEN         3
#define     LTBLUE          4
#define     LTCYAN          5
#define     LTYELLOW        6
#define     LTMAGENTA       7
#define     DKWHITE         8
#define     DKBLACK         9
#define     DKRED           10
#define     DKGREEN         11
#define     DKBLUE          12
#define     DKCYAN          13
#define     DKYELLOW        14
#define     DKMAGENTA       15

#define     REPLACE         1
#define     TRANSPAR        2
#define     XOR             3
#define     REVTRANS        4

#include    "portab.h"
#define     ASMAIN()        GEMAIN()

typedef     struct  Evtmsg
begin
        int     type;
        int     winid;
        Rect    evrec;
        Point   evpoint;
        int     scrpart;
        int     scrposn;
        int     scrmoved;
        char    keystroke;
        int     mod;
        int     mtitle;
        int     mitem;
end     Evtmsg;

extern  Evtmsg      Message;

#define     EVTTYPE         Message.type
#define     EVTWINDOW       Message.winid
#define     EVTRECT         Message.evrec
#define     EVPOINT         Message.evpoint
#define     EVTSCRPART      Message.scrpart
#define     EVTSCRPOSN      Message.scrposn
#define     EVTSCRMOVE      Message.scrmoved
#define     EVTKEY          Message.keystroke
#define     EVTMOD          Message.mod
#define     EVTMTITLE       Message.mtitle
#define     EVTMITEM        Message.mitem
```

```
#define     REDRAW      0
#define     TOPPED      1
#define     CLOSEWIN    2
#define     SCROLLBAR   3
#define     MOUSEDOWN   4
#define     KEYBOARD    5
#define     MOUSEUP     6
#define     MENUHIT     7

#define     V_PAGEUP    0
#define     V_PAGEDOWN  1
#define     V_ROWUP     2
#define     V_ROWDOWN   3
#define     H_PAGEUP    4
#define     H_PAGEDOWN  5
#define     H_ROWUP     6
#define     H_ROWDOWN   7
#define     V_THUMB     8
#define     H_THUMB     9

#define     MINSCR      0
#define     MAXSCR      1000

#define     NUL_CHR     '\0'
#define     CARR_RET    0x0D
#define     BACK_SP     0x08
#define     BLANK       0x20
```

```
/*-------------------------------------------------------------------*/
/*                    DEMO.H (for Demo.c use)                        */
/*-------------------------------------------------------------------*/

#define INVALID     -1

#define TEST5BAR    0       /* TREE */

#define MNDRAW      4       /* OBJECT in TREE #0 */
#define ITOUTLN     20      /* OBJECT in TREE #0 */
#define ITFILL      21      /* OBJECT in TREE #0 */
#define ITRECT      23      /* OBJECT in TREE #0 */
#define ITARC90     27      /* OBJECT in TREE #0 */
#define ITARC180    26      /* OBJECT in TREE #0 */
#define ITARC270    25      /* OBJECT in TREE #0 */
#define ITRNDRCT    28      /* OBJECT in TREE #0 */
#define ITSHAPE     30      /* OBJECT in TREE #0 */
#define ITLINE      31      /* OBJECT in TREE #0 */

#define MNMODE      5       /* OBJECT in TREE #0 */
#define ITREPLCE    33      /* OBJECT in TREE #0 */
#define ITTRANS     34      /* OBJECT in TREE #0 */
#define ITXOR       35      /* OBJECT in TREE #0 */
#define ITREVTR     36      /* OBJECT in TREE #0 */

#define MNCOLOR     6       /* OBJECT in TREE #0 */
#define ITDARK      38      /* OBJECT in TREE #0 */
#define ITLIGHT     39      /* OBJECT in TREE #0 */
#define ITBLACK     41      /* OBJECT in TREE #0 */
#define ITWHITE     42      /* OBJECT in TREE #0 */
#define ITRED       43      /* OBJECT in TREE #0 */
#define ITGREEN     44      /* OBJECT in TREE #0 */
#define ITBLUE      45      /* OBJECT in TREE #0 */
#define ITCYAN      46      /* OBJECT in TREE #0 */
#define ITYELLOW    47      /* OBJECT in TREE #0 */
#define ITMAGENT    48      /* OBJECT in TREE #0 */

#define MNPATTRN    7       /* OBJECT in TREE #0 */
#define ITSOLID     50      /* OBJECT in TREE #0 */
#define ITHVYHT     51      /* OBJECT in TREE #0 */
#define ITHATCH     52      /* OBJECT in TREE #0 */
#define ITLTHAT     53      /* OBJECT in TREE #0 */
#define ITEMPTY     54      /* OBJECT in TREE #0 */
#define ITELLIP     24      /* OBJECT in TREE #0 */

#define DESKMENU    3       /* OBJECT in TREE #0 */

#define MNWIN       8       /* OBJECT in TREE #0 */
#define ITWIN1      56      /* OBJECT in TREE #0 */
#define ITWIN2      57      /* OBJECT in TREE #0 */
#define ITWIN3      58      /* OBJECT in TREE #0 */
#define ITWIN4      59      /* OBJECT in TREE #0 */
#define ITWIN5      60      /* OBJECT in TREE #0 */
```

```
#define ITWIN6      61      /* OBJECT in TREE #0 */
#define ITWIN7      62      /* OBJECT in TREE #0 */
```

```
/*-------------------------------------------------------------------*/
/*                          ASPRIM.C                                 */
/*-------------------------------------------------------------------*/

#include        "asbind.h"
#include        "asprimi.c"

/*-------------------------------------------------------------------*/
/* Set_point: given two integers which represent the x and y   */
/*       coordinates (the horizontal and vertical positions of    */
/*       the point respectively), the function returns a point.   */
/*-------------------------------------------------------------------*/
        State
set_point(x,y,pt)
        Int     x,y;
        Point   *pt;

begin
        pt -> h = x;
        pt -> v = y;
end


/*-------------------------------------------------------------------*/
/* get_x_coord: Function which returns the horizontal          */
/*    coordinate of the input point pt.                        */
/*-------------------------------------------------------------------*/
        Int
get_x_coord(pt)
        Point   *pt;

begin
        return (pt -> h);
end


/*-------------------------------------------------------------------*/
/* get_y_coord: Function which returns the vertical            */
/*    coordinate of the input point pt.                        */
/*-------------------------------------------------------------------*/
        Int
get_y_coord(pt)
        Point   *pt;

begin
        return(pt -> v);
end


/*-------------------------------------------------------------------------------*/
/* set_rect: Function which, given two points, determines the smallest      */
/*    rectangle that those points could define and sets the top left        */
/*    and bottom right points of the output rectangle r to correspond       */
```

```
/*    to that rectangle.                                                    */
/*--------------------------------------------------------------------------*/
        State
set_rect(p1,p2,r)
        Point   *p1;
        Point   *p2;
        Rect    *r;

begin
                                /* case 1 p2 is to the right and below p1 */
        if (rt_below(p2,p1))
                assign_rect((p1 -> h),(p1 -> v),(p2 -> h),(p2 -> v),r);

                                /* case 2 p1 is to the right and below p2 */
        else if (rt_below(p1,p2))
                assign_rect((p2 -> h),(p2 -> v), (p1 -> h),(p1 -> v),r);

                                /* case 3 p1 is to the right and above p2 */
        else if (rt_above(p1,p2))
                assign_rect((p2 -> h),(p1 -> v),(p1 -> h),(p2 -> v),r);

                                /* case 4 p2 is to the right and above p1 */
        else if (rt_above(p2,p1))
                assign_rect((p1 -> h),(p2 -> v),(p2 -> h),(p1 -> v),r);

end


/*--------------------------------------------------------------------------*/
/* set_topLeft: Function which returns the top left point of the input      */
/*    rectangle r as p.                                                     */
/*--------------------------------------------------------------------------*/
        State
set_topLeft(r,p)
        Rect    *r;
        Point   *p;

begin
        (p -> h) = (r -> topLeft).h;
        (p -> v) = (r -> topLeft).v;
end


/*--------------------------------------------------------------------------*/
/* get_botRight: Function which returns the bottom right point of the       */
/*    input rectangle r as p.                                               */
/*--------------------------------------------------------------------------*/
        State
set_botRight(r,p)
        Rect    *r;
        Point   *p;

begin
```

```
        (p -> h) = (r -> botRight).h;
        (p -> h) = (r -> botRight).v;
end


/*-------------------------------------------------------------------------------*/
/* pt_in_rect: Function which determines if the input point p is within          */
/*    or on the border of the input rectangle r.                                 */
/*-------------------------------------------------------------------------------*/
        Bool
pt_in_rect(p,r)
        Point   *p;
        Rect    *r;

begin
        if ((rt_below(p,&(r -> topLeft))) && (lf_above(p,&(r -> botRight))))
                return(TRUE);
        else
                return(FALSE);
end


/*-------------------------------------------------------------------------------*/
/* set_insect_rect: Function which determines the rectangle                      */
/*    which is formed by the intersection of the input rectangles r1             */
/*    and r2.  The resulting rectangle is returned in rint.  If the              */
/*    intersection is empty, the rectangle returned in rint will be              */
/*       defined by a top left and bottom right point of (0,0).                  */
/*-------------------------------------------------------------------------------*/
        State
set_insect_rect(r1,r2,rint)
        Rect    *r1;
        Rect    *r2;
        Rect    *rint;

begin
        if (insect_rect(r1,r2))
        begin
                if ((r1 -> topLeft).h >= (r2 -> topLeft).h)
                        (rint -> topLeft).h = (r1 -> topLeft).h;
                else
                        (rint -> topLeft).h = (r2 -> topLeft).h;

                if ((r1 -> topLeft).v >= (r2 -> topLeft).v)
                        (rint -> topLeft).v = (r1 -> topLeft).v;
                else
                        (rint -> topLeft).v = (r2 -> topLeft).v;

                if ((r1 -> botRight).h <= (r2 -> botRight).h)
                        (rint -> botRight).h = (r1 -> botRight).h;
                else
                        (rint -> botRight).h = (r2 -> botRight).h;
```

```
                if ((r1 -> botRight).v <= (r2 -> botRight).v)
                        (rint -> botRight).v = (r1 -> botRight).v;
                else
                        (rint -> botRight).v = (r2 -> botRight).v;
        end

        else
                assign_rect(0,0,0,0,rint);

end

/*-------------------------------------------------------------------------------*/
/* insect_rect: Function which determines whether the two input                  */
/*    rectangles r1 and r2 intersect.                                            */
/*-------------------------------------------------------------------------------*/
        Bool
insect_rect(r1,r2)
        Rect    *r1;
        Rect    *r2;

begin

        if (((r1 -> topLeft).h > (r2 -> botRight).h) ||
                ((r2 -> topLeft).h > (r1 -> botRight).h))
                return(FALSE);

        else if (((r1 -> topLeft).v > (r2 -> botRight).v) ||
                ((r2 -> topLeft).v > (r1 -> botRight).v))
                return(FALSE);

        else
                return(TRUE);

end


/*-------------------------------------------------------------------------------*/
/* equalpt: Function which determines if the two input points are the            */
/*    same point.                                                                */
/*-------------------------------------------------------------------------------*/
        Bool
equalpt(p1,p2)
        Point   *p1;
        Point   *p2;

begin
        if (((p1 -> h) == (p2 -> h)) && ((p1 -> v) == (p2 -> v)))
                return(TRUE);
        else
                return(FALSE);
end
```

```
/*-------------------------------------------------------------------------------*/
/* equalrect: Function which determines if the two input rectangles are          */
/*    same rectangle.                                                            */
/*-------------------------------------------------------------------------------*/
        Bool
equalrect(r1,r2)
        Rect    *r1;
        Rect    *r2;

begin
        if ((equalpt(&(r1 -> topLeft),&(r2 -> topLeft))) &&
                (equalpt(&(r1 -> botRight),&(r2 -> botRight))))

                return(TRUE);
        else
                return(FALSE);
end


/*-------------------------------------------------------------------------------*/
/* copypt: Function which copies the source point into the destination           */
/*    point.                                                                     */
/*-------------------------------------------------------------------------------*/
        State
copypt(source,dest)

        Point   *source,*dest;

begin
        (*dest).h = (*source).h;
        (*dest).v = (*source).v;
end

/*-------------------------------------------------------------------------------*/
/* copyrect: Function which copies the source rectangle into the                 */
/*    destination rectangle.                                                     */
/*-------------------------------------------------------------------------------*/
        State
copyrect(source,dest)

        Rect    *source,*dest;

begin
        copypt(&((*source).topLeft),&((*dest).topLeft));
        copypt(&((*source).botRight),&((*dest).botRight));
end
```

```
/*-----------------------------------------------------------------*/
/*                        ASPRIMI.C                                */
/*-----------------------------------------------------------------*/


/*-------------------------------------------------------------------------------*/
/* rt_below: Function which determines whether the point p1 is to the            */
/*    right of and below the point p2.  Note: the larger the h, the              */
/*    farther right the point is and the larger the v the farther                */
/*    below the point is.                                                        */
/*-------------------------------------------------------------------------------*/
        Bool
rt_below(p1,p2)
        Point   *p1;
        Point   *p2;

begin
        if (((p1 -> h) >= (p2 -> h)) && ((p1 -> v) >= (p2 -> v)))
                return(TRUE);
        else
                return(FALSE);
end


/*-------------------------------------------------------------------------------*/
/* rt_above: Function which determines whether the point p1 is to the            */
/*    right and above point p2.                                                  */
/*-------------------------------------------------------------------------------*/
        Bool
rt_above(p1,p2)
        Point   *p1;
        Point   *p2;

begin
        if (((p1 -> h) >= (p2 -> h)) && ((p1 -> v) <= (p2 -> v)))
                return(TRUE);
        else
                return(FALSE);
end


/*-------------------------------------------------------------------------------*/
/* lf_above: Function to determine if point p1 is to the left and above          */
/*    point p2.                                                                  */
/*-------------------------------------------------------------------------------*/
        Bool
lf_above(p1,p2)
        Point   *p1;
        Point   *p2;

begin
        if (((p1 -> h) <= (p2 -> h)) && ((p1 -> v) <= (p2 -> v)))
                return(TRUE);
```

```
        else
                return(FALSE);
end


/*-------------------------------------------------------------------------------*/
/* lf_below: Function to determine if point p1 is to the left and below          */
/*    point p2.                                                                  */
/*-------------------------------------------------------------------------------*/
        Bool
lf_below(p1,p2)
        Point   *p1;
        Point   *p2;

begin
        if (((p1 -> h) <= (p2 -> h)) && ((p1 -> v) >= (p2 -> v)))
                return(TRUE);
        else
                return(FALSE);
end

/*-------------------------------------------------------------------------------*/
/* assign_rect: Function to assign the values of the top left point and          */
/*    bottom right point of the rectangle r.  Warning: the top left              */
/*    point as determined by xtop and ytop MUST be to the left and               */
/*    above the bottom right point as specified by xbot and ybot.                */
/*    This function is provided as a short form rectangle builder for            */
/*    the implementer only.                                                      */
/*-------------------------------------------------------------------------------*/
        State
assign_rect(xtop,ytop,xbot,ybot,r)
        Int     xtop,ytop,xbot,ybot;
        Rect    *r;

begin
        (r -> topLeft).h = xtop;
        (r -> topLeft).v = ytop;
        (r -> botRight).h = xbot;
        (r -> botRight).v = ybot;
end
```

```
/*------------------------------------------------------------------*/
/*                          ASEVT.C                                 */
/*------------------------------------------------------------------*/

        State
get_event()

begin
        Bool    Stop;
        Int     outarr[4];
        Int     buffer[8];
        Int     tempx;
        Int     tempy;
        Int     mouseX;
        Int     mouseY;
        Int     buttonstate;
        Int     modifiers;
        Int     keybdreturn;
        Int     numstroke;

        U_int   evvector;


        Stop = FALSE;
        while(!Stop)
        begin

                                        /* look for a GEM keyboard,button    */
                                        /* or message event                  */

                evvector = evnt_multi
        (MU_KEYBD | MU_BUTTON | MU_MESAG,  /* keyboard,button,message
events*/
        1,                              /* single button stroke              */
        0x0001,                         /* leftmost button                   */
        button_flag,                    /* look for mouse down or up         */
        0x0000,                         /* return on exit                    */
        0,                              /* empty rect spec                   */
        0,
        0,
        0,
        0x0000,                         /* return on exit                    */
        0,                              /* empty rect spec                   */
        0,
        0,
        0,
        ADDR(buffer),                           /* address of message buffer */
        17,                             /* 17/1000 sec delay for             */
        00,                             /* timer event (60 th sec)           */
        &mouseX,                        /* X mouse position                  */
        &mouseY,                        /* Y mouse position                  */
        &buttonstate,                   /* button state                      */
        &modifiers,                     /* keyboard modifiers                */
```

```
&keybdreturn,                    /* unmodified key code          */
&numstroke);                     /* number of button strokes     */

                                 /* GEM message event handler    */

        wind_update(1);
        if ((evvector & MU_MESAG) == MU_MESAG)
        begin
                switch (buffer[0])
                begin

                                 /* Menu hit event               */

                case MN_SELECTED:
                begin

                        EVTTYPE = MENUHIT;
                        EVTMTITLE = buffer[3];
                        EVTMITEM = buffer[4];

                                /* insure only one title hilited*/
                        if (mhilighted > 0)
                                menu_tnormal(baraddr,mhilighted,TRUE);

                        mhilighted = EVTMTITLE;
                        Stop = TRUE;
                        break;
                end;

                                 /* Redraw event        -- give program  */
                                 /* rectangle to redraw                  */

                case WM_REDRAW:
                begin

                        windowID(buffer[3],&EVTWINDOW);
                        do_rev_map(&(Winlist[EVTWINDOW].Coordmap),
                                &buffer[4],&buffer[5]);
                        buffer[6] += buffer[4] - 1;
                        buffer[7] += buffer[5] - 1;

                        EVTTYPE = REDRAW;
                        assign_rect(buffer[4],buffer[5],buffer[6],
                                buffer[7],&EVTRECT);
                        Stop = TRUE;
                        break;
                end;

                                 /* Topped event                 */

                case WM_TOPPED:
                begin
                        windowID(buffer[3],&EVTWINDOW);
```

```
        do_rev_map(&(Winlist[EVTWINDOW].Coordmap),
                &mouseX,&mouseY);
        set_point(mouseX,mouseY,&EVPOINT);
        EVTTYPE = TOPPED;
        EVTMOD = modifiers;
        Stop = TRUE;
        break;
end;

                /* Close box event                          */

case WM_CLOSED:
begin
        windowID(buffer[3],&EVTWINDOW);
        EVTTYPE = CLOSEWIN;
        Stop = TRUE;
        break;
end;

                /* Scroll bar event where one               */
                /* of the up or down arrows or              */
                /* page up or down areas was                */
                /* selected.                                */

case WM_ARROWED:
begin
        windowID(buffer[3],&EVTWINDOW);
        EVTTYPE = SCROLLBAR;
        EVTSCRPART = buffer[4];
        Stop = TRUE;
        break;
end;

                /* Scroll bar event where the               */
                /* user selected the slide (or              */
                /* thumb) for the horizontal                */
                /* scroll bar.                              */

case WM_HSLID:
begin
        windowID(buffer[3],&EVTWINDOW);
        EVTTYPE = SCROLLBAR;
        EVTSCRPART = H_THUMB;
        EVTSCRPOSN = buffer[4];
        EVTSCRMOVE = buffer[4] -
                        Winlist[EVTWINDOW].H_value;
        Stop = TRUE;
        break;
end;

                /* Scroll bar event where the               */
                /* user selected the slide (or              */
                /* thumb) for the vertical                  */
```

```
                    /* scroll bar.                          */

case WM_VSLID:
begin
        windowID(buffer[3],&EVTWINDOW);
        EVTTYPE = SCROLLBAR;
        EVTSCRPART = V_THUMB;
        EVTSCRPOSN = buffer[4];
        EVTSCRMOVE = buffer[4] -
                        Winlist[EVTWINDOW].V_value;
        Stop = TRUE;
        break;
end;

                /* Change the size of the window        */
                /* if the user has dragged the          */
                /* grow box.                            */

case WM_SIZED:
begin
        windowID(buffer[3],&EVTWINDOW);
        wind_set(buffer[3],WF_CXYWH,buffer[4],
                buffer[5],buffer[6],buffer[7]);

        Winlist[EVTWINDOW].defX = buffer[4];
        Winlist[EVTWINDOW].defY = buffer[5];
        Winlist[EVTWINDOW].defW = buffer[6];
        Winlist[EVTWINDOW].defH = buffer[7];

        wind_get(buffer[3],WF_WXYWH,&buffer[4],
                &buffer[5],&buffer[6],&buffer[7]);

        outarr[0] = buffer[4];
        outarr[1] = buffer[5];
        outarr[2] = buffer[4] + buffer[6] - 1;
        outarr[3] = buffer[5] + buffer[7] - 1;

        vs_clip(Device,1,outarr);

        break;
end;

                /* Move the window if the user          */
                /* has dragged the title bar.           */

case WM_MOVED:
begin
        windowID(buffer[3],&EVTWINDOW);
        wind_set(buffer[3],WF_CXYWH,buffer[4],
                buffer[5],buffer[6],buffer[7]);

        Winlist[EVTWINDOW].defX = buffer[4];
        Winlist[EVTWINDOW].defY = buffer[5];
```

```
                Winlist[EVTWINDOW].defW = buffer[6];
                Winlist[EVTWINDOW].defH = buffer[7];

                wind_get(buffer[3],WF_WXYWH,&buffer[4],
                        &buffer[5],&buffer[6],&buffer[7]);

                outarr[0] = buffer[4];
                outarr[1] = buffer[5];
                outarr[2] = buffer[4] + buffer[6] - 1;
                outarr[3] = buffer[5] + buffer[7] - 1;

                vs_clip(Device,1,outarr);

                get_origin(EVTWINDOW,&tempx,&tempy);
                set_map(&(Winlist[EVTWINDOW].Coordmap),tempx,
                        tempy,outarr[0],outarr[1]);

                break;
        end;

        default: break;

        end
end

                        /* Case for mouse down and  */
                        /* mouse up events          */

else if ((evvector & MU_BUTTON) == MU_BUTTON)
begin
        if (button_flag == LOOKMDOWN)
        begin
                EVTTYPE = MOUSEDOWN;
                button_flag = LOOKMUP;
        end

        else
        begin
                EVTTYPE = MOUSEUP;
                button_flag = LOOKMDOWN;
        end

        EVTMOD = modifiers;
        tempx = wind_find(mouseX,mouseY);
        windowID(tempx,&EVTWINDOW);
        do_rev_map(&(Winlist[EVTWINDOW].Coordmap),&mouseX,
                        &mouseY);

        Stop = TRUE;
        set_point(mouseX,mouseY,&(EVPOINT));

end
```

```
                                        /* Case for keyboard event     */

                else if ((evvector & MU_KEYBD) == MU_KEYBD)
                begin
                        EVTTYPE = KEYBOARD;
                        EVTKEY = ((Char)(keybdreturn & 0x007F));
                        EVTMOD = modifiers;
                        Stop = TRUE;
                        evvector = evvector ^ MU_KEYBD;
                end
                wind_update(0);
        end
end


/*-------------------------------------------------------------------------------*/
/* get_mouse: Function which reports the current location of the cursor          */
/*        in the local coordinates of the window specified by Id.                */
/*-------------------------------------------------------------------------------*/
        State
get_mouse(Id,pt)

        Int     Id;
        Point   *pt;

begin
        Int     x,y,button,mod;

        graf_mkstate(&x,&y,&button,&mod);

        do_rev_map(&(Winlist[Id].Coordmap),&x,&y);
        set_point(x,y,pt);
end


/*-------------------------------------------------------------------------------*/
/* mouse_up: Function which reports of the mouse button is up or not.           */
/*        Use of this function will cause the event manager to look for          */
/*        the opposite mouse button state returned by this function.  This       */
/*        is analogus to the Mac WaitMouseUp function which unqueues a           */
/*        mouse up event if detected.                                            */
/*-------------------------------------------------------------------------------*/

        Bool
mouse_up()

begin
        Int     x,y,button,mod;

        graf_mkstate(&x,&y,&button,&mod);
        button = button & 0x0001;

        if (!button)
```

```
            button_flag = LOOKMDOWN;
        else
            button_flag = LOOKMUP;

        return(!button);
end
```

```
/*-------------------------------------------------------------------*/
/*                        ASEVTI.C                                   */
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------------------*/
/* get_origin: Hidden function which returns the x and y coordinates of          */
/*       the top left corner of the work area (in local coordinates).            */
/*-------------------------------------------------------------------------------*/
        State
get_origin(Id,x,y)

        Window_id       Id;
        Int             *x,*y;

begin
        (*x) = Winlist[Id].Coordmap.Xorigin;
        (*y) = Winlist[Id].Coordmap.Yorigin;
end


/*-------------------------------------------------------------------------------*/
/* windowId: Hidden function which matches the input GEM handle to an            */
/*       abstract window id and returns it in the Id parameter.  The             */
/*       return indicates whether or not a successful match was made.            */
/*-------------------------------------------------------------------------------*/

        Bool
windowID(handle,Id)

        Int             handle;
        Window_id       *Id;

begin

        Int     I;

        (*Id) = 10;
        I = 0;

        if (handle == 0)
        begin
                (*Id) = 0;
                return(TRUE);
        end

        while (I <= 8)
        begin

                if (Alloc_win[I] != 0)
                begin
                        if (Winlist[Alloc_win[I]].Winhandle == handle)
                                (*Id) = Alloc_win[I];
                end
```

```
            I++;
        end

        if ((*Id) == 10)
            return(FALSE);
        else
            return(TRUE);
end
```

```
/*------------------------------------------------------------------*/
/*                          ASWIN.C                                 */
/*------------------------------------------------------------------*/


#include "ASBIND1.H"
#include "machine.h"
#include "obdefs.h"
#include "treeaddr.h"
#include "gembind.h"
#include "vdibind.h"

#include        "aswini.c"
#include        "asmenu.c"

/*------------------------------------------------------------------------------*/
/* set_xfer_mode: function which will set the global mode for drawing           */
/*    onto the screen.                                                          */
/*------------------------------------------------------------------------------*/
        State
set_xfer_mode(newmode)

        Mode_id         newmode;

begin
        if((newmode < REPLACE) || (newmode > REVTRANS))
                newmode = REPLACE;

        vswr_mode(Device,newmode);
        Winlist[Active_win].winmode = newmode;
end


/*------------------------------------------------------------------------------*/
/* set_pattern: Function which sets the pattern to be used to draw              */
/*    and fill in shapes.                                                       */
/*------------------------------------------------------------------------------*/
        State
set_pattern(newpattern)

        Pattern_id      newpattern;

begin
        switch (newpattern)
        begin

                case HEAVYHATCH:
                begin
                        vsl_type(Device,2);
                        vsf_interior(Device,2);
                        vsf_style(Device,7);
                        Winlist[Active_win].winpat = newpattern;
                        break;
```

```
                end;

                case HATCH:
                begin
                        vsl_type(Device,7);
                        vsl_udsty(Device,0xE38E);
                        vsf_interior(Device,2);
                        vsf_style(Device,5);
                        Winlist[Active_win].winpat = newpattern;
                        break;
                end;

                case LTHATCH:
                begin
                        vsl_type(Device,3);
                        vsf_interior(Device,2);
                        vsf_style(Device,2);
                        Winlist[Active_win].winpat = newpattern;
                        break;
                end;

                case EMPTY:
                begin
                        vsl_type(Device,7);
                        vsl_udsty(Device,0x0000);
                        vsf_interior(Device,0);
                        Winlist[Active_win].winpat = newpattern;
                        break;
                end;

                default:
                begin
                        vsl_type(Device,1);
                        vsf_interior(Device,1);
                        Winlist[Active_win].winpat = SOLID;
                        break;
                end;
        end
end


/*-------------------------------------------------------------------------------*/
/* set_color: Function which sets the global color for drawing.                  */
/*-------------------------------------------------------------------------------*/
        State
set_color(newcolor)
        Int     newcolor;
begin
        if ((newcolor < LTWHITE) || (newcolor > DKMAGENTA))
                newcolor = LTBLACK;

        vsl_color(Device,newcolor);
        vsf_color(Device,newcolor);
```

```
        vst_color(Device,newcolor);
        Winlist[Active_win].wincol = newcolor;
end


/*-------------------------------------------------------------------------------*/
/* sys_init: Function to initialize the Gem system to run the Abstract          */
/*     Specification Interface                                                   */
/*-------------------------------------------------------------------------------*/
        State
sys_init()

begin
        Int     I;
        Int     outarr[4];

        outarr[0] = 50;
        outarr[1] = 50;
        outarr[2] = 200;
        outarr[3] = 200;

        ap_id = appl_init();

        if (ap_id < 0)
        begin
                for(I = 0; I < -1; I++) ;
        end

        for (I = 0; I < 10; I++)
                work_in[I] = 1;

        work_in[10] = 2;

        gem_Device = graf_handle(&hwchar,&hhchar,&hwbox,&hhbox);
        Device = gem_Device;

        v_opnvwk(work_in,&Device,work_out);
        vsf_perimeter(Device,0);

        scrn_form.mp = 0x0L;
        graf_mouse(0,MOUSEADDR);

        wind_init();

        set_xfer_mode(REPLACE);
        set_pattern(SOLID);
        set_color(LTBLACK);

end

/*-------------------------------------------------------------------------------*/
/* sys_end: Function which returns all allocated resources to the GEM           */
/*    system on the end of the program.                                          */
```

```
/*-------------------------------------------------------------------------------*/
        State
sys_end()

begin
        Int     I;

        for(I = 0; I < MAXNUMWIN; I++)
        begin
                if (Alloc_win[I] != 0)
                begin
                        if (Winlist[Alloc_win[I]].Visible)
                                wind_close(Winlist[Alloc_win[I]].Winhandle);

                        wind_delete(Winlist[Alloc_win[I]].Winhandle);
                end
        end

        v_clsvwk(Device);
        appl_exit();
end


/*-------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------------*/

        Window_id
set_new_window(InitRect,Partspec,Title,is_Visible)
        Rect            *InitRect;
        unsigned int    Partspec;
        Char            *Title;
        Bool            is_Visible;

begin
        Bool            NoErrorFlag;  /* no error encountered              */
        Window_id       Recnum;       /* number of window record alloc     */
        Int             temphand;     /* temporary window handle           */
        Long            tempaddr;     /* temporary address                 */
        Int             haddr;        /* high address of title             */
        Int             laddr;        /* low address of title              */
        Int             outarr[4];    /* input array to GEM VDI            */


                                      /* get rid of unnecessary specs      */
        Partspec = Partspec & 0xFFEB;

        NoErrorFlag = get_next_rec(&Recnum);

                                      /* if able to allocate window        */
        if (!NoErrorFlag)
                return(INVAL_WIN);

        else
```

```
begin
                        /* GEM defination of window        */

    Winlist[Recnum].Winhandle = wind_create(Partspec,
                                0,0,700,700);
    temphand = Winlist[Recnum].Winhandle;

    if (temphand < DESK_WIN)
    begin
        dalloc_win(Recnum);
        return(INVAL_WIN);
    end

                        /* Set optional window features */
                        /* Set horizontal scroll bar value */

    if ((Partspec & W_HSCROLL) > 0)
    begin
        wind_set(temphand,WF_HSLSIZE,-1,0,0,0);
        wind_set(temphand,WF_HSLIDE,0,0,0,0);
        Winlist[Recnum].H_value = 0;
    end

                        /* Set vertical scroll bar value */

    if ((Partspec & W_VSCROLL) > 0)
    begin
        wind_set(temphand,WF_VSLSIZE,-1,0,0,0);
        wind_set(temphand,WF_VSLIDE,0,0,0,0);
        Winlist[Recnum].V_value = 0;
    end

                        /* Set Title                       */

    if ((Partspec & W_NAME) > 0)
    begin
        haddr = (Int) LHIWD(ADDR(Title));
        laddr = (Int) LLOWD(ADDR(Title));
        wind_set(temphand,WF_NAME,laddr,haddr,0,0);
    end

                        /* map defination rectangle to  */
                        /* desktop coordinates             */

    get_gem_rect(InitRect,&(outarr[0]),&(outarr[1]),&(outarr[2]),
        &(outarr[3]));
    do_map(&(Winlist[DESK_WIN].Coordmap),&(outarr[0]),
        &(outarr[1]));
    set_point(20,20,&(Winlist[Recnum].txtpen));

    Winlist[Recnum].defX = outarr[0];
    Winlist[Recnum].defY = outarr[1];
    Winlist[Recnum].defW = outarr[2];
```

```
        Winlist[Recnum].defH = outarr[3];

                                /* draw visible windows to screen*/
                                /* and make active               */


        if (is_Visible == TRUE)
        begin
                NoErrorFlag = wind_open(temphand,outarr[0],outarr[1],
                        outarr[2],outarr[3]);

                wind_get(temphand,WF_WXYWH,&outarr[0],&outarr[1],
                        &outarr[2],&outarr[3]);

                                /* set clip area to window      */
                                /* content region and whiten    */

                outarr[2] += (outarr[0] - 1);
                outarr[3] += (outarr[1] - 1);

                vs_clip(Device,1,outarr);
                whiterec(outarr);
                Active_win = Recnum;


                set_map(&(Winlist[Recnum].Coordmap),0,0,
                        outarr[0],outarr[1]);

                                /* set GEM VDI global drawing       */
                                /* parameters and record in         */
                                /* window record                    */

                set_color(LTBLACK);
                set_xfer_mode(REPLACE);
                set_pattern(SOLID);
        end

                                /* set the window's drawing  */
                                /* parameters                */

        else
        begin
                Winlist[Active_win].wincol = LTBLACK;
                Winlist[Active_win].winpat = SOLID;
                Winlist[Active_win].winmode = REPLACE;
        end

        Winlist[Recnum].Visible = is_Visible;

        return(Recnum);
    end
end
```

```
/*-------------------------------------------------------------------------------*/
/* close_window: Function to close and permanently deallocate the                */
/*       specified window.                                                       */
/*-------------------------------------------------------------------------------*/

        State
close_window(Id)

        Window_id     Id;

begin

        Int       Recnum;

        for (Recnum = 0;
                ((Recnum < MAXNUMWIN) && (Alloc_win[Recnum] != Id));
                  Recnum++);

        if (Recnum >= MAXNUMWIN)
                return;

        hide_window(Id);
        wind_delete(Winlist[Id].Winhandle);
        dalloc_win(Recnum);

end


/*-------------------------------------------------------------------------------*/
/* update_win: Function which sets the system into the update window             */
/*   mode.  In this mode, drawing will be limited to the visible region          */
/*   of the window to be updated (as identified by the ID number input)          */
/*   to the function.  When given an rectangular area to update, the             */
/*   function will return the intersection between that area and one of          */
/*   the rectangles which define the visible area of the window to be            */
/*   updated.                                                                    */
/*-------------------------------------------------------------------------------*/

        Bool
update_win(ID,Up_rct,Dr_rct)

        Window_id     ID;
     Rect       *Up_rct,*Dr_rct;

begin
        Int       Firstx;               /* top left x of first vis rect          */
        Int       Firsty;               /* top left y of first vis rect          */
        Int       Firstw;                     /* width of first visible rect     */
        Int       Firsth;               /* height of first visible rect          */
        Int       outarr[4];            /* GEM VDI input array                   */

                                        /* get first visible rectangle           */
```

```
        wind_get(Winlist[ID].Winhandle,WF_FIRSTXYWH,&Firstx,&Firsty,&Firstw,
            &Firsth);

        if ((Firstw > 0) && (Firsth > 0))
        begin

                                        /* calculate intersection of        */
                                        /* visible rectangle and rect to    */
                                        /* be updated                       */

                do_rev_map(&(Winlist[ID].Coordmap),&Firstx,&Firsty);
                Firstw += Firstx - 1;
                Firsth += Firsty - 1;

                Assign_rect(Firstx,Firsty,Firstw,Firsth,Dr_rct);

                set_insect_rect(Up_rct,Dr_rct,Dr_rct);

                                        /* set clip area to intersection    */
                                        /* rectangle and whiten             */

                get_gem_rect(Dr_rct,&outarr[0],&outarr[1],&outarr[2],
                        &outarr[3]);
                do_map(&(Winlist[ID].Coordmap),&outarr[0],&outarr[1]);
                outarr[2] += (outarr[0] - 1);
                outarr[3] += (outarr[1] - 1);

                                        /* remember which is top window     */

                Last_active = Active_win;
                Active_win = ID;
                activedraw();

                vs_clip(Device,1,outarr);
                whiterec(outarr);

                                        /* set GEM update mode              */

                wind_update(1);
                Update_in_prog = TRUE;
                return(TRUE);
        end

        else
                return(FALSE);
end


/*-------------------------------------------------------------------------------*/
/* next_update:  Function which returns the intersection of the desired          */
/*   update area (Up_rct) and the next rectangle in the gem rectangle            */
/*   list which defines the visible area of a window (output is Dr_rct).         */
```

```
/*  A function return of false indicates no more rectangles are left in      */
/*  the gem visible rectangle list.                                          */
/*---------------------------------------------------------------------------*/

        Bool
next_update(Up_rct,Dr_rct)

        Rect    *Up_rct,*Dr_rct;

begin
        Int     Nextx;                  /* top left x of next vis rect      */
        Int     Nexty;                  /* top left y of next vis rect      */
        Int     Nextw;                  /* width of next visible rect       */
        Int     Nexth;                  /* height of next visible rect      */
        Int     outarr[4];              /* GEM VDI input array              */

        if (Update_in_prog)
        begin

                                        /* get next visible rectangle     */

                wind_get(Winlist[Active_win].Winhandle,WF_NEXTXYWH,&Nextx,
                 &Nexty,&Nextw,&Nexth);

                if ((Nextw > 0) && (Nexth > 0))
                begin

                                        /* calculate intersection of        */
                                        /* visible rectangle and rect to    */
                                        /* be updated                       */

                        do_rev_map(&(Winlist[Active_win].Coordmap),
                                &Nextx,&Nexty);
                        Nextw += Nextx - 1;
                        Nexth += Nexty - 1;

                        Assign_rect(Nextx,Nexty,Nextw,Nexth,Dr_rct);
                        set_insect_rect(Up_rct,Dr_rct,Dr_rct);

                                        /* set clip area to intersection*/
                                        /* rectangle and whiten             */

                        get_gem_rect(Dr_rct,&outarr[0],&outarr[1],
                                        &outarr[2],&outarr[3]);
                        do_map(&(Winlist[Active_win].Coordmap),
                                &outarr[0],&outarr[1]);
                        outarr[2] += (outarr[0] - 1);
                        outarr[3] += (outarr[1] - 1);
                        vs_clip(Device,1,outarr);
                        whiterec(outarr);

                        return(TRUE);
                end
```

```
                else
                        return(FALSE);
        end

        else
                return(FALSE);

end

/*-------------------------------------------------------------------------------*/
/* end_update: procedure to end the update mode and restore the clip            */
/*   area to match the active (topmost) window.                                 */
/*-------------------------------------------------------------------------------*/
        State
end_update()

begin
        Int     outarr[4];

        if (Update_in_prog)
        begin
                Active_win = Last_active;
                wind_get(Winlist[Active_win].Winhandle,
            WF_WXYWH,&outarr[0],&outarr[1],&outarr[2],&outarr[3]);

                outarr[2] += (outarr[0] - 1);
                outarr[3] += (outarr[1] - 1);
                vs_clip(Device,1,outarr);
                activedraw();
                wind_update(0);
                Update_in_prog = FALSE;
        end
end


/*-------------------------------------------------------------------------------*/
/* Note for all drawing routines:  mouse is hidden during all drawing           */
/*    routines to prevent unwanted interaction between the drawing              */
/*    being done and the mouse buffer which is used to save and restore         */
/*    the backround behind the mouse.                                           */
/*-------------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------------*/
/* drawline: Function which draws a line in the currently active window.        */
/*   Input coordinates are relative to the top left hand corner of the          */
/*   active window.                                                             */
/*-------------------------------------------------------------------------------*/
        State
drawline(St_pt,End_pt)

        Point   *St_pt,*End_pt;
```

```
begin
        Int     outarr[4];

        if (!equalpt(St_pt,End_pt))
        begin

                outarr[0] = (St_pt -> h);
                outarr[1] = (St_pt -> v);
                outarr[2] = (End_pt -> h);
                outarr[3] = (End_pt -> v);

                do_map(&(Winlist[Active_win].Coordmap),&outarr[0],&outarr[1]);
                do_map(&(Winlist[Active_win].Coordmap),&outarr[2],&outarr[3]);

                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_pline(Device,2,outarr);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end
end


/*-------------------------------------------------------------------------------*/
/* drawrect: Function to draw the outline of a rectangle in the active           */
/*    window.  The coordinates of the input rectangle are asumed to be           */
/*    relative to the top left corner of the active window's work area.          */
/*-------------------------------------------------------------------------------*/
        State
drawrect(In_rect)

        Rect    *In_rect;

begin
        Int     outarr[10];

        if (!equalpt(&((*In_rect).topLeft),&((*In_rect).botRight)))
        begin

                outarr[0] = (*In_rect).topLeft.h;
                outarr[1] = (*In_rect).topLeft.v;
                outarr[4] = (*In_rect).botRight.h - 1;
                outarr[5] = (*In_rect).botRight.v - 1;

                do_map(&(Winlist[Active_win].Coordmap),&outarr[0],&outarr[1]);
                do_map(&(Winlist[Active_win].Coordmap),&outarr[4],&outarr[5]);

                outarr[2] = outarr[4];
                outarr[3] = outarr[1];
                outarr[6] = outarr[0];
                outarr[7] = outarr[5];
                outarr[8] = outarr[0];
                outarr[9] = outarr[1];
```

```
                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_pline(Device,5,outarr);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end
end


/*-------------------------------------------------------------------------------*/
/* drawellipse: Function which draws an ellipse within the area of the          */
/*    active window specified by the input rectangle.  The coordinates          */
/*    of the input rectangle are assumed to be relative to the top left         */
/*    corner of the work area of the active window.                             */
/*-------------------------------------------------------------------------------*/
        State
drawellipse(In_rect)

        Rect    *In_rect;

begin
        Int     x_ctr,y_ctr,x_rad,y_rad;
        Int     tempp,tempxfer;

        if (!equalpt(&((*In_rect).topLeft),&((*In_rect).botRight)))
        begin

                polar_coord(In_rect,&x_ctr,&y_ctr,&x_rad,&y_rad);
                do_map(&(Winlist[Active_win].Coordmap),&x_ctr,&y_ctr);

                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_ellarc(Device,x_ctr,y_ctr,x_rad,y_rad,0,3600);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end
end


/*-------------------------------------------------------------------------------*/
/*  drawarc: Function which draws an elliptical arc between the two             */
/*    input angles (begang and endang) specified and within the                 */
/*    rectangular area of the active window specified.  The input               */
/*    rectangle is assumed to be relative to the top left corner of the         */
/*    work area of the active window.  Angles are reversed to force             */
/*    correspondence with Mac.                                                  */
/*-------------------------------------------------------------------------------*/
        State
drawarc(R,begang,endang)

        Rect    *R;
        Int     begang,endang;

begin
        Int     x_ctr,y_ctr,x_rad,y_rad;
```

```
        if (!equalpt(&((*R).topLeft),&((*R).botRight)))
        begin

                polar_coord(R,&x_ctr,&y_ctr,&x_rad,&y_rad);
                do_map(&(Winlist[Active_win].Coordmap),&x_ctr,&y_ctr);

                map_angle(&begang);
                map_angle(&endang);
                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_ellarc(Device,x_ctr,y_ctr,x_rad,y_rad,endang,begang);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end

end


/*-------------------------------------------------------------------------------*/
/* drawrndrect: Function which draws the outline of a  rounded rectangle     */
/*   within the specified rectangular area of the active window.             */
/*-------------------------------------------------------------------------------*/
        State
drawrndrct(In_rect)

        Rect    *In_rect;

begin
        Int     outarr[4];

        if (!equalpt(&((*In_rect).topLeft),&((*In_rect).botRight)))
        begin

                outarr[0] = (*In_rect).topLeft.h;
                outarr[1] = (*In_rect).topLeft.v;
                outarr[2] = (*In_rect).botRight.h - 1;
                outarr[3] = (*In_rect).botRight.v - 1;


                do_map(&(Winlist[Active_win].Coordmap),&outarr[0],&outarr[1]);
                do_map(&(Winlist[Active_win].Coordmap),&outarr[2],&outarr[3]);

                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_rbox(Device,outarr);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end

end


/*-------------------------------------------------------------------------------*/
/* fillrect: Function which draws a pattern within the specified             */
/*    rectangular area of the active window.                                 */
/*-------------------------------------------------------------------------------*/
        State
```

```
fillrect(In_rect)

        Rect    *In_rect;

begin
        Int     outarr[4];

        if (!equalpt(&((*In_rect).topLeft),&((*In_rect).botRight)))
        begin

                outarr[0] = (*In_rect).topLeft.h;
                outarr[1] = (*In_rect).topLeft.v;
                outarr[2] = (*In_rect).botRight.h - 1;
                outarr[3] = (*In_rect).botRight.v - 1;


                do_map(&(Winlist[Active_win].Coordmap),&outarr[0],&outarr[1]);
                do_map(&(Winlist[Active_win].Coordmap),&outarr[2],&outarr[3]);

                graf_mouse(HIDEMOUSE,MOUSEADDR);
                vr_recfl(Device,outarr);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end

end


/*-------------------------------------------------------------------------------*/
/* fillrndrect: Function which fills the outline of a  rounded rectangle         */
/*   within the specified rectangular area of the active window.                 */
/*-------------------------------------------------------------------------------*/
        State
fillrndrct(In_rect)

        Rect    *In_rect;

begin
        Int     outarr[4];

        if (!equalpt(&((*In_rect).topLeft),&((*In_rect).botRight)))
        begin

                outarr[0] = (In_rect -> topLeft).h;
                outarr[1] = (In_rect -> topLeft).v;
                outarr[2] = (In_rect -> botRight).h - 1;
                outarr[3] = (In_rect -> botRight).v - 1;


                do_map(&(Winlist[Active_win].Coordmap),&outarr[0],&outarr[1]);
                do_map(&(Winlist[Active_win].Coordmap),&outarr[2],&outarr[3]);

                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_rfbox(Device,outarr);
```

```
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end

end


/*-------------------------------------------------------------------------------*/
/* fillellipse: Function which fills an ellipse within the area of the           */
/*   active window specified by the input rectangle. The coordinates             */
/*   of the input rectangle are assumed to be relative to the top left           */
/*   corner of the work area of the active window.                               */
/*-------------------------------------------------------------------------------*/
        State
fillellipse(In_rect)

        Rect    *In_rect;

begin
        Int     x_ctr,y_ctr,x_rad,y_rad;

        if (!equalpt(&((*In_rect).topLeft),&((*In_rect).botRight)))
        begin

                polar_coord(In_rect,&x_ctr,&y_ctr,&x_rad,&y_rad);
                do_map(&(Winlist[Active_win].Coordmap),&x_ctr,&y_ctr);

                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_ellipse(Device,x_ctr,y_ctr,x_rad,y_rad);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end

end


/*-------------------------------------------------------------------------------*/
/*  fillarc: Function which fills an elliptical arc between the two              */
/*   input angles (begang and endang) specified and within the                   */
/*   rectangular area of the active window specified.  The input                 */
/*   rectangle is assumed to be relative to the top left corner of the           */
/*   work area of the active window.  Angles are reversed in the GEM             */
/*   function call to force correspondence to Mac.                               */
/*-------------------------------------------------------------------------------*/
        State
fillarc(R,begang,endang)

        Rect    *R;
        Int     begang,endang;

begin
        Int     x_ctr,y_ctr,x_rad,y_rad;

        if (!equalpt(&((*R).topLeft),&((*R).botRight)))
        begin
```

```
                polar_coord(R,&x_ctr,&y_ctr,&x_rad,&y_rad);
                do_map(&(Winlist[Active_win].Coordmap),&x_ctr,&y_ctr);

                map_angle(&begang);
                map_angle(&endang);
                graf_mouse(HIDEMOUSE,MOUSEADDR);
                v_ellpie(Device,x_ctr,y_ctr,x_rad,y_rad,endang,begang);
                graf_mouse(SHOWMOUSE,MOUSEADDR);
        end

end

/*-------------------------------------------------------------------------------*/
/* activate_win: Function which causes the specified window to become     */
/*    the active window.  It causes any window (but the desktop with a    */
/*    id number of 0) to be moved to the top and a new backround will     */
/*    be drawn in, however, the contents will not be automatically        */
/*    redrawn.                                                            */
/*-------------------------------------------------------------------------------*/
        State
activate_win(ID)
        Window_id    ID;

begin
        Int     outarr[4];              /* input to GEM VDI          */


        if (!(ID == Active_win))
        begin

                if ((ID >= DESK_WIN))
                begin

                                        /* if not the desktop, bring    */
                                        /* specified window to top      */

                        if((ID >= 1) && (ID <= MAXNUMREC))
                        begin
                                graf_mouse(HIDEMOUSE,0X0L);
                                wind_set(Winlist[ID].Winhandle,WF_TOP,0,0,0,0);
                        end

                                        /* set clip area to content area        */

                        Active_win = ID;
                        wind_get(Winlist[ID].Winhandle,WF_WXYWH,&outarr[0],
                                &outarr[1],&outarr[2],&outarr[3]);

                        outarr[2] += (outarr[0] - 1);
                        outarr[3] += (outarr[1] - 1);

                        vs_clip(Device,1,outarr);
```

```
                        if ((ID >= 1) && (ID <= MAXNUMREC))
                                graf_mouse(SHOWMOUSE,0X0L);

                        activedraw();

                end
        end
end


/*-------------------------------------------------------------------------------*/
/* hscroll: Function which scrolls the content area of the active window        */
/*     by the number of "pixels" specified by num.  If the num is               */
/*     positive, the region will  move to the left, and to the right if         */
/*     negative.                                                                */
/*-------------------------------------------------------------------------------*/
        State
hscroll(num,Up_rect)

        Int     num;
        Rect    *Up_rect;

begin
        Int     X;                      /* top left x of content area     */
        Int     Y;                      /* top left y of content area     */
        Int     W;                      /* width of content area          */
        Int     H;                      /* height of content area         */
        Int     outarr[8];              /* output to GEM VDI bit copy fcn */
        Int     whtarr[4];              /* GEM VDI rectangle to whiten    */

        if (!(num == 0))
        begin

                                        /* set to scroll the content area */
                                        /* left and whiten the vacated    */
                                        /* rectangle                      */

                if (num > 0)
                begin
                        wind_get(Winlist[Active_win].Winhandle,
                                WF_WXYWH,&X,&Y,&W,&H);
                        outarr[0] = X + num;
                        outarr[1] = Y;
                        outarr[2] = X + W - 1;
                        outarr[3] = Y + H - 1;
                        outarr[4] = X;
                        outarr[5] = Y;
                        outarr[6] = X + W - 1 - num;
                        outarr[7] = Y + H - 1;
                        whtarr[0] = outarr[6];
                        whtarr[1] = outarr[1];
                        whtarr[2] = outarr[2];
```

```
                whtarr[3] = outarr[3];
        end
                                /* set to scroll the content area       */
                                /* right and whiten the vacated         */
                                /* rectangle                    */


        if (num < 0)
        begin
                wind_get(Winlist[Active_win].Winhandle,
                        WF_WXYWH,&X,&Y,&W,&H);
                outarr[0] = X;
                outarr[1] = Y;
                outarr[2] = X + W - 1 + num;
                outarr[3] = Y + H - 1;
                outarr[4] = X - num;
                outarr[5] = Y;
                outarr[6] = X + W - 1;
                outarr[7] = Y + H - 1;
                whtarr[0] = outarr[0];
                whtarr[1] = outarr[1];
                whtarr[2] = outarr[4];
                whtarr[3] = outarr[3];
        end

                                /* bit copy to scroll           */

        graf_mouse(HIDEMOUSE,0X0L);
        vro_cpyfm(Device,3,outarr,&scrn_form,&scrn_form);
        graf_mouse(SHOWMOUSE,0X0L);
        translate_origin(Active_win,num,0);
        whiterec(whtarr);

    end

    else
        for(X = 0; X < 4; X++)
            whtarr[X] = 0;

                                /* assign the rect to be updated        */
                                /* in window local coord        */

    do_rev_map(&(Winlist[Active_win].Coordmap),&whtarr[0],&whtarr[1]);
    do_rev_map(&(Winlist[Active_win].Coordmap),&whtarr[2],&whtarr[3]);

    assign_rect(whtarr[0],whtarr[1],whtarr[2],whtarr[3],Up_rect);
end


/*-------------------------------------------------------------------------------*/
/* vscroll: Function which scrolls the content area of the active window        */
/*     by the number of "pixels" specified by num.  If the num is               */
/*     positive, the region will  move up,and down if negative.                 */
```

```
/*-------------------------------------------------------------------------------*/
        State
vscroll(num,Up_rect)

        Int     num;
        Rect    *Up_rect;

begin
        Int     X;                      /* top left x of content area     */
        Int     Y;                      /* top left y of content area     */
        Int     W;                      /* width of content area          */
        Int     H;                      /* height of content area         */
        Int     outarr[8];              /* output to GEM VDI bit copy fcn */
        Int     whtarr[4];              /* GEM VDI rectangle to whiten    */

        if (!(num == 0))
        begin

                                        /* set to scroll the content area */
                                        /* up and whiten the vacated      */
                                        /* rectangle                      */

                if (num > 0)
                begin
                        wind_get(Winlist[Active_win].Winhandle,
                                WF_WXYWH,&X,&Y,&W,&H);
                        outarr[0] = X;
                        outarr[1] = Y + num;
                        outarr[2] = X + W - 1;
                        outarr[3] = Y + H - 1;
                        outarr[4] = X;
                        outarr[5] = Y;
                        outarr[6] = X + W - 1;
                        outarr[7] = Y + H - 1 - num;
                        whtarr[0] = outarr[0];
                        whtarr[1] = outarr[7];
                        whtarr[2] = outarr[2];
                        whtarr[3] = outarr[3];
                end

                                        /* set to scroll the content area */
                                        /* down and whiten the vacated    */
                                        /* rectangle                      */

                if (num < 0)
                begin
                        wind_get(Winlist[Active_win].Winhandle,
                                WF_WXYWH,&X,&Y,&W,&H);
                        outarr[0] = X;
                        outarr[1] = Y;
                        outarr[2] = X + W - 1;
                        outarr[3] = Y + H - 1 + num;
                        outarr[4] = X;
```

```
                outarr[5] = Y - num;
                outarr[6] = X + W - 1;
                outarr[7] = Y + H - 1;
                whtarr[0] = outarr[0];
                whtarr[1] = outarr[1];
                whtarr[2] = outarr[2];
                whtarr[3] = outarr[5];
        end

                                /* bit copy to scroll          */

        graf_mouse(HIDEMOUSE,0X0L);
        vro_cpyfm(Device,3,outarr,&scrn_form,&scrn_form);
        graf_mouse(SHOWMOUSE,0X0L);
        translate_origin(Active_win,0,num);
        whiterec(whtarr);

    end

    else
        for(X = 0; X < 4; X++)
            whtarr[X] = 0;

                                /* assign the rect to be updated    */
                                /* in window local coord    */

    do_rev_map(&(Winlist[Active_win].Coordmap),&whtarr[0],&whtarr[1]);
    do_rev_map(&(Winlist[Active_win].Coordmap),&whtarr[2],&whtarr[3]);

    assign_rect(whtarr[0],whtarr[1],whtarr[2],
                whtarr[3],Up_rect);
end

/*-------------------------------------------------------------------------------*/
/* set_hscroll: Function which sets the value of the horizontal scroll      */
/*    bar of the active window to the input val.                            */
/*-------------------------------------------------------------------------------*/
    State
set_hscroll(val)

    Int     val;

begin

    if (val < 0)
        val = 0;

    if (val > 1000)
        val = 1000;

    wind_set(Winlist[Active_win].Winhandle,WF_HSLIDE,val,0,0,0);
    Winlist[Active_win].H_value = val;
```

```
end

/*-------------------------------------------------------------------------------*/
/* set_vscroll: Function which sets the value of the vertical scroll bar         */
/*    to the input val.                                                          */
/*-------------------------------------------------------------------------------*/
        State
set_vscroll(val)

        Int     val;

begin

        if (val < 0)
                val = 0;

        if (val > 1000)
                val = 1000;

        wind_set(Winlist[Active_win].Winhandle,WF_VSLIDE,val,0,0,0);
        Winlist[Active_win].V_value = val;

end


/*-------------------------------------------------------------------------------*/
/* get_hscroll: Function which returns the horizontal scroll bar value.          */
/*-------------------------------------------------------------------------------*/
        Int
get_hscroll()

begin
        return(Winlist[Active_win].H_value);
end


/*-------------------------------------------------------------------------------*/
/* get_vscroll: Function which returns the vertical scroll bar value.            */
/*-------------------------------------------------------------------------------*/
        Int
get_vscroll(val)

begin
        return(Winlist[Active_win].V_value);
end

/*-------------------------------------------------------------------------------*/
/* hide_window: Function which removes the specified window from the    */
/*    screen without deallocating it.                                            */
/*-------------------------------------------------------------------------------*/
        State
hide_window(Id)
```

```
        Window_id     Id;

begin

        Int       temphandle;

        if (Winlist[Id].Visible && (Id != DESK_WIN))
        begin
                wind_close(Winlist[Id].Winhandle);
                Winlist[Id].Visible = FALSE;

                if (Id == Active_win)
                begin
                        wind_get(0,WF_TOP,&temphandle,0,0,0);
                        windowID(temphandle,&Active_win);
                        activate_win(Active_win);
                end
        end
end

/*-------------------------------------------------------------------------------*/
/* show_window: Function which draws an invisible but previously defined*/
/*     window onto the screen.  This window becomes the active window.   */
/*-------------------------------------------------------------------------------*/
        State
show_window(Id)

        Window_id     Id;

begin

        Int       outarr[4];

        if ((!Winlist[Id].Visible) && (Id != DESK_WIN))
        begin
                wind_open(Winlist[Id].Winhandle,Winlist[Id].defX,
                        Winlist[Id].defY,Winlist[Id].defW,Winlist[Id].defH);
                Winlist[Id].Visible = TRUE;
                activate_win(Id);
        end
end

/*-------------------------------------------------------------------------------*/
/* get_active: Function which returns the identifier of the active               */
/*     window.                                                                   */
/*-------------------------------------------------------------------------------*/

        Window_id
get_active()

begin
        return(Active_win);
end
```

```
/*-------------------------------------------------------------------------------*/
/* get_color: Function which returns the identifier of the drawing color        */
/*-------------------------------------------------------------------------------*/
        Color_id
get_color()

begin
        return(Winlist[Active_win].wincol);
end


/*-------------------------------------------------------------------------------*/
/* get_mode: Function which returns the identifier of the drawing trans-        */
/*       fer mode.                                                               */
/*-------------------------------------------------------------------------------*/
        Mode_id
get_xfer_mode()

begin
        return(Winlist[Active_win].winmode);
end


/*-------------------------------------------------------------------------------*/
/* get_pattern: Function which returns the identifier of the drawing            */
/*       pattern.                                                                */
/*-------------------------------------------------------------------------------*/
        Pattern_id
get_pattern()

begin
        return(Winlist[Active_win].winpat);
end


/*-------------------------------------------------------------------------------*/
/* txtpen: Function which sets the location of the next character to            */
/*       be drawn in the active window (location of text pen in window          */
/*       local coordinates).                                                     */
/*-------------------------------------------------------------------------------*/
        State
txtpen(inpt)

        Point   *inpt;

begin
        copypt(inpt,&(Winlist[Active_win].txtpen));
end


/*-------------------------------------------------------------------------------*/
/* set_txtpen: Function which returns the location of the text pen for          */
```

```
/*      the currently active window (in window local coordinates).          */
/*--------------------------------------------------------------------------*/
        State
set_txtpen(pen)

        Point   *pen;

begin
        copypt(&(Winlist[Active_win].txtpen),pen);
end


/*--------------------------------------------------------------------------*/
/* drawstring: Function which draws a string into the active window at      */
/*      the current location of its text pen.                               */
/*--------------------------------------------------------------------------*/

        State
drawstring(strptr)

        Char    *strptr;

begin
        Int     x,y;
        Int     extent[8];

        x = Winlist[Active_win].txtpen.h;
        y = Winlist[Active_win].txtpen.v;

        do_map(&(Winlist[Active_win].Coordmap),&x,&y);

        graf_mouse(HIDEMOUSE,MOUSEADDR);
        v_gtext(Device,x,y,strptr);
        graf_mouse(SHOWMOUSE,MOUSEADDR);

        vqt_extent(Device,strptr,extent);

        Winlist[Active_win].txtpen.h += extent[2];


end


/*--------------------------------------------------------------------------*/
/* drawchar: Function which draws a character at the current location of    */
/*      the active window's text pen.                                       */
/*--------------------------------------------------------------------------*/

        State
drawchar(inchr)

        Char    inchr;
```

```
begin
        Char    outstr[2];
        Int     x,y;
        Int     extent[8];

        if (Winlist[Active_win].winmode != XOR)
                        vswr_mode(Device,TRANSPAR);

        outstr[0] = inchr;
        outstr[1] = NUL_CHR;

        x = Winlist[Active_win].txtpen.h;
        y = Winlist[Active_win].txtpen.v;


        do_map(&(Winlist[Active_win].Coordmap),&x,&y);

        graf_mouse(HIDEMOUSE,MOUSEADDR);
        v_gtext(Device,x,y,outstr);
        graf_mouse(SHOWMOUSE,MOUSEADDR);
        vqt_extent(Device,outstr,extent);

        Winlist[Active_win].txtpen.h += extent[2];

        if (Winlist[Active_win].winmode != XOR)
                        vswr_mode(Device,Winlist[Active_win].winmode);

end


/*-------------------------------------------------------------------------------*/
/* get_wchar: Function which returns the current character width.                */
/*-------------------------------------------------------------------------------*/
        Int
get_wchar()

begin
        return(hwchar);
end


/*-------------------------------------------------------------------------------*/
/* get_hchar: Function which returns the current character height.               */
/*-------------------------------------------------------------------------------*/
        Int
get_hchar()

begin
        return(hhchar);
end
```

```
/*-----------------------------------------------------------------*/
/*                          ASWINI.C                               */
/*-----------------------------------------------------------------*/

/* Module global data declarations -- These variables are required to      */
/*   be global to allow linkage with the GEM driver modules.               */

Int     contrl[12];
Int     intin[128];
Int     ptsin[128];
Int     intout[128];
Int     ptsout[128];

/* Local data declarations of data structures to be hidden from the        */
/*   user.                                                                 */


static  Int     hwchar;             /* width of a character              */
static  Int     hhchar;             /* height of a character             */
static  Int     hwbox;              /* width of a character box          */
static  Int     hhbox;              /* height of a character box         */
static  Int     work_in[11];        /* GEM open v workstation input      */
static  Int     work_out[57];       /* GEM open v workstation output     */
static  Int     ap_id;              /* GEM appilication id               */

static  Int     Device;             /* handle for GEM virtual screen     */
static  Int     gem_Device;         /* handle for GEM screen             */

typedef         struct  Map         /* type defination of global to      */
begin                               /* window local coordinate map       */
        Int     Xorigin;            /* horiz window origin               */
        Int     Yorigin;            /* vert window origin                */
        Int     Xreal;              /* horiz real screen coord           */
        Int     Yreal;              /* vert real screen coord            */
end
Map;

typedef         struct  Winrec      /* window record structure           */
begin
        Int             Winhandle;  /* GEM window handle                 */
        Map             Coordmap;   /* global to local map               */
        Int             H_value;    /* current horiz scroll value        */
        Int             V_value;    /* current vert scroll value         */
        Bool            Visible;    /* is window visible on screen       */
        Int             defX;       /* global x of entire window         */
        Int             defY;       /* global y of entire window         */
        Int             defW;       /* width of entire window            */
        Int             defH;       /* height of entire window           */
        Point           txtpen;     /* location to draw next txt         */
        Mode_id         winmode;    /* window drawing mode               */
        Pattern_id      winpat;     /* window drawing pattern            */
        Color_id        wincol;     /* window color                      */
end
```

```
Winrec;

                                        /* records for windows + desk          */
static   Winrec              Winlist[MAXNUMREC];

                                        /* array of available record indeces */
static   Window_id    Available_win[MAXNUMWIN];

                                        /* array of allocated record indexes */
static   Window_id    Alloc_win[MAXNUMWIN];

static   Window_id    Active_win;    /* index of active window              */
static   Window_id    Last_active;   /* index of previous active window     */
static   Bool         Update_in_prog;/* is update occuring                  */

static   MFDB         scrn_form;     /* GEM bit block str for screen        */
static   U_int        button_flag;   /* flag to determine whether to        */
                                     /* look for mouse up or down           */

static   Long         baraddr;       /* address of the GEM menu bar         */
static   Int          mhilighted;    /* object index of hilighted menu      */

         Evtmsg       Message;       /* event message for user              */

#include "asevti.c"
#include "asevt.c"

/*-------------------------------------------------------------------------------*/
/* init_alloc_str: Function to initialize the structures (Available_win          */
/*    and Active_win) used to keep track of window records available to          */
/*    be allocated and already allocated.                                        */
/*-------------------------------------------------------------------------------*/
         State
init_alloc_str()
begin
         Int      I;
         for (I = 0; I < MAXNUMWIN; I++)
         begin
                  Available_win[I] = I + 1;
                  Alloc_win[I] = 0;
         end
end


/*-------------------------------------------------------------------------------*/
/* wind_init: Function to initialize the record for the desktop window           */
/*   and set it to be the initial active window.                                 */
/*-------------------------------------------------------------------------------*/
         State
wind_init()

begin
         Bool    NoErrorFlag;
```

```
        Int     X,Y,W,H,outarr[4];
        Point   tmppoint;

        Winlist[DESK_WIN].Winhandle = DESK_WIN;

        NoErrorFlag = wind_get(DESK_WIN,WF_WXYWH,&X,&Y,&W,&H);

                        /* set desktop coordinate map */
        Winlist[DESK_WIN].Coordmap.Xorigin = 0;
        Winlist[DESK_WIN].Coordmap.Yorigin = 0;
        Winlist[DESK_WIN].Coordmap.Xreal = X;
        Winlist[DESK_WIN].Coordmap.Yreal = Y;

        Winlist[DESK_WIN].Visible = TRUE;

                        /* set defination coordinates and clip rectangle */

        Winlist[DESK_WIN].defX = X;
        Winlist[DESK_WIN].defY = Y;
        Winlist[DESK_WIN].defW = W;
        Winlist[DESK_WIN].defH = H;
        outarr[0] = X;
        outarr[1] = Y;
        outarr[2] = W + X - 1;
        outarr[3] = H + Y - 1;

        vs_clip(Device,1,outarr);

        set_point(0,0,&(Winlist[DESK_WIN].txtpen));

        init_alloc_str();
        Active_win = 0;
        Last_active = 0;
        Update_in_prog = FALSE;
        button_flag = LOOKMDOWN;
        mhilighted = 0;
end


/*-------------------------------------------------------------------------------*/
/* activedraw: Function to set the global drawing parameters of the GEM     */
/*       VDI to those of the drawing window.                                     */
/*-------------------------------------------------------------------------------*/

        State
activedraw()

begin
        set_pattern(Winlist[Active_win].winpat);
        set_color(Winlist[Active_win].wincol);
        set_xfer_mode(Winlist[Active_win].winmode);
end
```

```
/*-------------------------------------------------------------------------------*/
/* get_gem_rect: Hidden function to give the x and y coordinates of the      */
/*    top left corner of an 'abstract' rectangle along with its width        */
/*    and height.                                                            */
/*-------------------------------------------------------------------------------*/
        State
get_gem_rect(R,X,Y,W,H)
        Rect    *R;
        Int     *X,*Y,*W,*H;

begin
        (*X) = (R -> topLeft).h;
        (*Y) = (R -> topLeft).v;
        (*W) = (R -> botRight).h - (R -> topLeft).h + 1;
        (*H) = (R -> botRight).v - (R -> topLeft).v + 1;
end

/*-------------------------------------------------------------------------------*/
/* do_map: Function to map window local coordinates (x and y                 */
/*    coordinates) to global screen coordinates which Gem VDI will           */
/*    recognize.                                                             */
/*-------------------------------------------------------------------------------*/
        State
do_map(Cmap,X,Y)
        Map     *Cmap;
        Int     *X,*Y;

begin
        (*X) += (Cmap -> Xreal) - (Cmap -> Xorigin);
        (*Y) += (Cmap -> Yreal) - (Cmap -> Yorigin);
end

/*-------------------------------------------------------------------------------*/
/* do_rev_map: Function to map global screen coordinates to window local     */
/*    coordinates as defined by the input coordinate map (Cmap).             */
/*-------------------------------------------------------------------------------*/
        State
do_rev_map(Cmap,X,Y)
        Map     *Cmap;
        Int     *X,*Y;

begin
        (*X) -= (Cmap -> Xreal) - (Cmap -> Xorigin);
        (*Y) -= (Cmap -> Yreal) - (Cmap -> Yorigin);
end

/*-------------------------------------------------------------------------------*/
/* set_map: Function to set the mapping from window local coordinates        */
/*    to screen global coordinates.                                          */
/*-------------------------------------------------------------------------------*/
        State
set_map(Cmap,Orig_x,Orig_y,Real_x,Real_y)
```

```
        Map     *Cmap;
        Int     Orig_x,Orig_y,Real_x,Real_y;

begin
        (Cmap -> Xorigin) = Orig_x;
        (Cmap -> Yorigin) = Orig_y;
        (Cmap -> Xreal) = Real_x;
        (Cmap -> Yreal) = Real_y;
end


/*-------------------------------------------------------------------------------*/
/* get_next_rec: Function which returns a boolean TRUE if a window              */
/*    is available for allocation, FALSE otherwise.  The index to the           */
/*       allocated record is returned as the integer pointed to by RECNUM       */
/*-------------------------------------------------------------------------------*/
Bool
get_next_rec(Recnum)
        Int     *Recnum;
begin
        Int     I,J;

        I = 0;
        J = 0;

        while ((Available_win[I] == 0) && (I < MAXNUMWIN))
                I++;

        while ((Alloc_win[J] != 0) && (J < MAXNUMWIN))
                J++;

        if (I >= MAXNUMWIN)
                return(FALSE);
        else
        begin
                Alloc_win[J] = Available_win[I];
                (*Recnum) = Available_win[I];
                Available_win[I] = 0;
                return(TRUE);
        end
end

/*-------------------------------------------------------------------------------*/
/* dalloc_win: Dealocates an allocated window record                             */
/*-------------------------------------------------------------------------------*/
        State
dalloc_win(Recnum)
        Int     Recnum;
begin
        Int     I,J;

        if ((Recnum > 0) && (Recnum < 9))
        begin
```

```
                I = 0;
                J = 0;

                while ((Alloc_win[J] != Recnum) && (J < MAXNUMWIN))
                        J++;

                while ((Available_win != 0) && (I < MAXNUMWIN))
                        I++;

                if ((J < MAXNUMWIN) && (I < MAXNUMWIN))
                begin
                        Available_win[I] = Alloc_win[J];
                        Alloc_win[J] = 0;
                end
        end
end

/*-------------------------------------------------------------------------------*/
/* whiterec: Paints the rectangle specified by the array of 4 integers           */
/*    pointed to by outarr white.  Array must be in the form: [0]:               */
/*    x of top left point, [1]: y of top left point, [2]: x of bottom            */
/*    right point, [3]: y of bottom right point.  All points must be in          */
/*    global screen coordinates.                                                 */
/*-------------------------------------------------------------------------------*/
        State
whiterec(outarr)

        Int     *outarr;

begin

        Mode_id                 tempxfer;
        Pattern_id      tempp;
        Color_id        tempc;

        graf_mouse(HIDEMOUSE,MOUSEADDR);
        tempxfer = Winlist[Active_win].winmode;
        tempp = Winlist[Active_win].winpat;
        tempc = Winlist[Active_win].wincol;

        set_xfer_mode(REPLACE);
        set_pattern(SOLID);
        set_color(LTWHITE);

        vr_recfl(Device,outarr);

        set_xfer_mode(tempxfer);
        set_pattern(tempp);
        set_color(tempc);
        graf_mouse(SHOWMOUSE,MOUSEADDR);

end
```

```
/*-------------------------------------------------------------------------------*/
/* polar_coord: Function which converts the coordinates of a rectangle           */
/*   input in the form of two opposing corners into a polar coordinate           */
/*   like form returning the center of the rectangle and the x and y             */
/*   radiuses.                                                                   */
/*-------------------------------------------------------------------------------*/
        State
polar_coord(R,x_ctr,y_ctr,x_rad,y_rad)

        Rect    *R;
        Int     *x_ctr,*y_ctr,*x_rad,*y_rad;

begin
        Int     gemx,gemy,gemw,gemh;

        get_gem_rect(R,&gemx,&gemy,&gemw,&gemh);

        (*x_ctr) = gemx + (gemw / 2);
        (*y_ctr) = gemy + (gemh / 2);
        (*x_rad) = gemw / 2;
        (*y_rad) = gemh / 2;
end


/*-------------------------------------------------------------------------------*/
/* map_angle: Function which converts a GEM angle to a Mac angle                 */
/*-------------------------------------------------------------------------------*/
        State
map_angle(angle)

        Int     *angle;

begin
        Int     I;

        if (angle < 0)
                for(I = (*angle); I < 0; I += 3600);
        else
                I = (*angle);

        (*angle) = (900 - I + 3600) % 3600;

end


/*-------------------------------------------------------------------------------*/
/* translate_origin: Function which moves the origin of the global to            */
/*    local map of the specified window by the amount dX and dY.                 */
/*-------------------------------------------------------------------------------*/
        State
translate_origin(Id,dX,dY)
```

```
        Int     Id,dX,dY;

begin
        Winlist[Id].Coordmap.Xorigin += dX;
        Winlist[Id].Coordmap.Yorigin += dY;
end



/*-------------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------------------*/
        State
greenrec(outarr)

        Int     *outarr;

begin

        Mode_id                 tempxfer;
        Pattern_id      tempp;
        Color_id        tempc;

        graf_mouse(HIDEMOUSE,MOUSEADDR);
        tempxfer = Winlist[Active_win].winmode;
        tempp = Winlist[Active_win].winpat;
        tempc = Winlist[Active_win].wincol;

        set_xfer_mode(REPLACE);
        set_pattern(SOLID);
        set_color(LTGREEN);

        vr_recfl(Device,outarr);

        set_xfer_mode(tempxfer);
        set_pattern(tempp);
        set_color(tempc);
        graf_mouse(SHOWMOUSE,MOUSEADDR);

end

/*-------------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------------------*/
        State
bluerec(outarr)

        Int     *outarr;

begin

        Mode_id                 tempxfer;
        Pattern_id      tempp;
        Color_id        tempc;
```

```
        graf_mouse(HIDEMOUSE,MOUSEADDR);
        tempxfer = Winlist[Active_win].winmode;
        tempp = Winlist[Active_win].winpat;
        tempc = Winlist[Active_win].wincol;

        set_xfer_mode(REPLACE);
        set_pattern(SOLID);
        set_color(LTBLUE);

        vr_recfl(Device,outarr);

        set_xfer_mode(tempxfer);
        set_pattern(tempp);
        set_color(tempc);
        graf_mouse(SHOWMOUSE,MOUSEADDR);

end
```

```
/*-------------------------------------------------------------------*/
/*                          ASMENU.C                                 */
/*-------------------------------------------------------------------*/


/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

        State
init_menu(filename,barId)

        char            *filename;
        Menu_id                 barId;

begin
        rsrc_load(ADDR(filename));
        rsrc_gaddr(0,barId,&baraddr);
        menu_bar(baraddr,1);
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
        State
item_enable(menunum,itemnum)

        int     menunum,itemnum;

begin
        menu_ienable(baraddr,itemnum,1);
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

        State
item_disable(menunum,itemnum)

        int     menunum,itemnum;

begin
        menu_ienable(baraddr,itemnum,0);
end

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/

        State
item_mark(menunum,itemnum,mark)

        int     menunum,itemnum;
        Bool    mark;

begin
```

```
        menu_icheck(baraddr,itemnum,mark);
end

/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/

        State
menu_hilight(menunum,hilight)

        int     menunum;
        Bool    hilight;

begin
        if (hilight)
        begin

                if (mhilighted > 0)
                        menu_tnormal(baraddr,mhilighted,TRUE);

                menu_tnormal(baraddr,menunum,FALSE);
                mhilighted = menunum;
        end

        else if (mhilighted > 0)
        begin
                menu_tnormal(baraddr,mhilighted,TRUE);
                mhilighted = 0;
        end

end
```

```
/*-------------------------------------------------------------------*/
/*                        ASBIND1.H                                  */
/*-------------------------------------------------------------------*/


#define         begin           {
#define         end             }

typedef struct  Point
begin
        int     v,h;
end
Point;

typedef struct  Rect
begin
        Point   topLeft;
        Point   botRight;
end
Rect;

typedef         int             Bool;

#define         Void            /**/
#define         State           /**/

typedef         int             Int;
typedef         long            Long;
typedef         char            Char;
typedef         unsigned int    U_int;

typedef         int             Pattern_id;
typedef         int             Mode_id;
typedef         int             Color_id;
typedef         int             Window_id;
typedef         int             Menu_id;

#define         W_NAME          0x0009
#define         W_CLOSE         0X0002
#define         W_SIZE          0x0020
#define         W_HSCROLL       0x0E00
#define         W_VSCROLL       0X01C0

#define         INVAL_WIN       -1
#define         DESK_WIN        0
#define         MAXNUMWIN       7
#define         MAXNUMREC       8

#define         SOLID           1
#define         HEAVYHATCH      2
#define         HATCH           3
#define         LTHATCH         4
#define         EMPTY           5
```

```
#define     LTWHITE         0
#define     LTBLACK         1
#define     LTRED           2
#define     LTGREEN         3
#define     LTBLUE          4
#define     LTCYAN          5
#define     LTYELLOW        6
#define     LTMAGENTA       7
#define     DKWHITE         8
#define     DKBLACK         9
#define     DKRED           10
#define     DKGREEN         11
#define     DKBLUE          12
#define     DKCYAN          13
#define     DKYELLOW        14
#define     DKMAGENTA       15

#define     REPLACE         1
#define     TRANSPAR        2
#define     XOR             3
#define     REVTRANS        4

#include    "portab.h"
#define     ASMAIN()    GEMAIN()

typedef     struct  Evtmsg
begin
        int     type;
        int     winid;
        Rect    evrec;
        Point   evpoint;
        int     scrpart;
        int     scrposn;
        int     scrmoved;
        char    keystroke;
        int     mod;
        int     mtitle;
        int     mitem;
end     Evtmsg;

extern  Evtmsg              Message;

#define     EVTTYPE         Message.type
#define     EVTWINDOW       Message.winid
#define     EVTRECT         Message.evrec
#define     EVPOINT         Message.evpoint
#define     EVTSCRPART      Message.scrpart
#define     EVTSCRPOSN      Message.scrposn
#define     EVTSCRMOVE      Message.scrmoved
#define     EVTKEY          Message.keystroke
#define     EVTMOD          Message.mod
#define     EVTMTITLE       Message.mtitle
```

```
#define     EVTMITEM        Message.mitem

#define     REDRAW          0
#define     TOPPED          1
#define     CLOSEWIN        2
#define     SCROLLBAR       3
#define     MOUSEDOWN       4
#define     KEYBOARD        5
#define     MOUSEUP         6
#define     MENUHIT         7

#define     V_PAGEUP        0
#define     V_PAGEDOWN      1
#define     V_ROWUP         2
#define     V_ROWDOWN       3
#define     H_PAGEUP        4
#define     H_PAGEDOWN      5
#define     H_ROWUP         6
#define     H_ROWDOWN       7
#define     V_THUMB         8
#define     H_THUMB         9

#define     MINSCR          0
#define     MAXSCR          1000

#define     NUL_CHR         '\0'
#define     CARR_RET        0x0D
#define     BACK_SP         0x08
#define     BLANK           0x20

#define     MOUSEADDR       0x0L
#define     HIDEMOUSE       256
#define     SHOWMOUSE       257

#define     LOOKMDOWN       0x0001
#define     LOOKMUP         0x0000
```

LIST OF REFERENCES

1. Balma, P. and Fitler, W., *Programmer's Guide To GEM*, SYBEX Inc., CA., 1986.

2. Apple Computer, Inc., *Inside Macintosh Volumn I-III*, Addison-Wesley Publishing Company, Inc., Fifth Printing, April 1987.

3. Chernicoff, S., *Macintosh Revealed Volumn I-II*, 2nd ed., Hayden Book Company, IN., 1987.

4. Stubbs, D. F. and Webre, N. W., *Data Structure with Abstract Data Types and Pascal*, Brooks/Cole Publishing Company, CA., 1985.

5. Kelley, A. and Pohl, I., *A Book on C*, The Benjamin/cummings Company, Inc., CA., 1984.

Thesis
K716246 Ko-Hsin
c.1 An abstract interactive graphics interface for the IBM/PC and Macintosh.