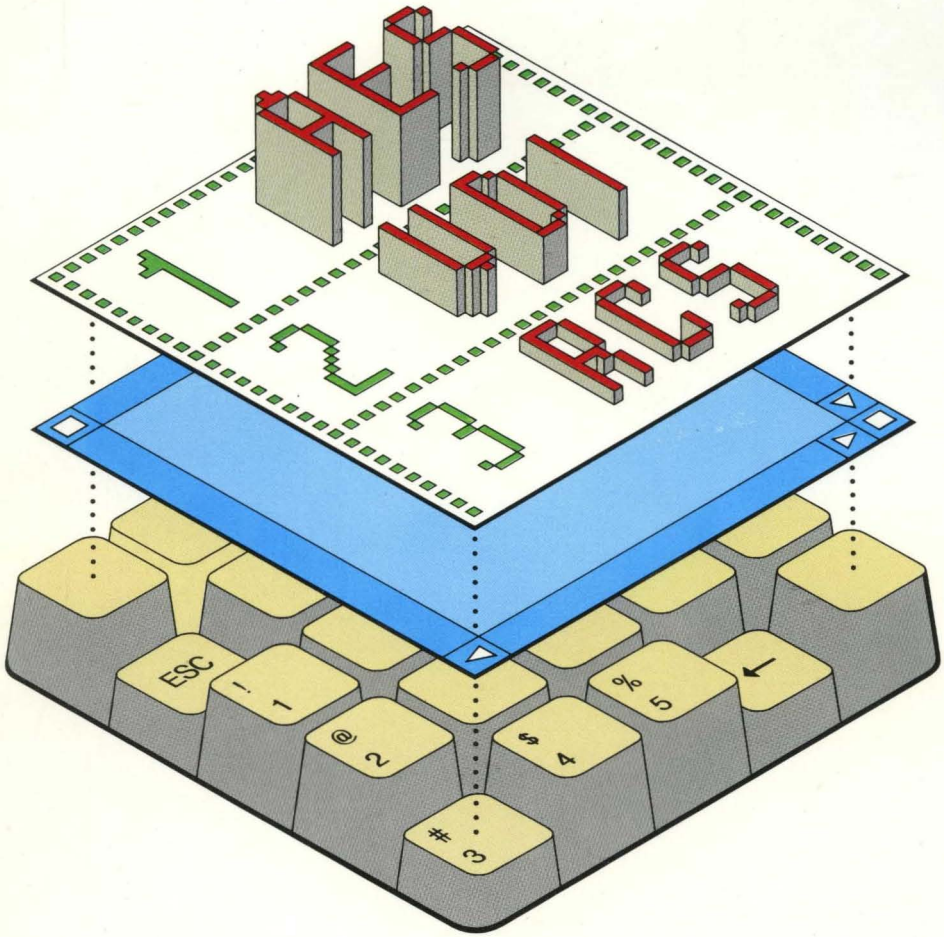
GEM Programmer's Toolkit™



Vol 1
GEM™ AES

GEM™

APPLICATION ENVIRONMENT SERVICES

REFERENCE GUIDE

GEM Version 2.0

8039 - 2024

Second Edition: June 1986

FOREWORD

The GEM™ Application Environment Services is a set of related functions for programming in the Digital Research® GEM graphics environment. An application program uses the Application Environment Services (AES) to perform a variety of tasks, such as managing windows, displaying messages, monitoring mouse movements, and drawing objects on the screen.

Associated Manuals

The following manuals are part of the GEM Developer Kit software and documentation package. Because these manuals are interrelated, you should become familiar with them and the software they document.

- GEM Desktop: Description of the GEM Desktop™ user interface.
- Introduction to GEM Programming: An introduction to programming in the GEM graphics environment. This book contains an overview of the GEM Application Environment Services and GEM Virtual Device Interface, with coding examples written in the C programming language.
- GEM Virtual Device Interface Reference Guide: Reference manual for GEM Virtual Device Interface.
- GEM Programmer's Utilities Guide: Description of the GEM Resource Construction Set, Kermit™, and GEM SID™ programs provided in the GEM Developer Kit.

Organization of This Manual

Section 1 is an introduction to the GEM Application Environment Services.

Section 2 is a general overview of many common tasks an application must perform.

Sections 3 through 14 contain the GEM Application Environment Services, with detailed descriptions of each function.

Changes From GEM Version 1.x

GEM version 1.x provided four functions for animating windows: GRAF_GROWBOX and GRAF_SHRINKBOX in the Graphics Library, and the FMD_GROW and FMD_SHRINK options in the Form library.

In GEM version 2.x this functionality is supported through the Extended Graphics Library. Digital Research will not use the associated 1.x opcodes for any other functions, and future versions of GEM will treat these functions as no-ops.

Contents

Tables

Figures

INTRODUCTION TO GEM APPLICATION ENVIRONMENT SERVICES

The GEM Application Environment Services, called the AES, is a set of functions an application program uses to control the interaction between the user and the application. The AES manages icon selection, drop-down menus, dialog boxes, alert messages, and windows.

The AES is grouped into twelve sets of related functions, referred to as libraries.

The code for GEM AES is resident in memory, and it remains in memory until the user exits the AES.

1.1 Conventions

In this manual, you are the application programmer using the GEM Application Environment Services to write your application program. The user is the person running the application.

Mouse form is a general term for the cursor which is controlled by the mouse. The mouse form can be a pointer, cross hair, or some other form, depending on the context of the application.

In the error codes for the functions, n means any positive number, -n means any negative number.

The following abbreviations appear throughout this manual:

GEM	Graphics Environment Manager
AES	Application Environment Services
VDI	Virtual Device Interface, documented in the GEM Virtual Device Interface Reference Guide
RCS	Resource Construction Set, documented in the GEM Programmer's Utilities Guide

1.2 GEM AES Libraries

The Application Environment Services consist of the twelve libraries described in Table 1-1.

Table 1-1. Application Environment Services

Library	Description
Application	Application initialization and communication
Event	Input and event management
Menu	Menu bar display and management
Object	Object tree display and management
Form	Form display and management
Graphics	Rectangle display and management
Scrap	Data interchange between applications
File Selector	Directory display and file selection
Window	Window management
Resource	Resource file and object address management
Shell	Shell information retrieval and management
Extended Graphics	Supplemental rectangle functions

1.3 Function Names

The name of a function identifies the library to which it belongs and gives a general idea of its purpose.

The part of the function name preceding the underbar is a code for the AES library. For example,

appl_... Application Library
evnt_... Event Library
objc_... Object Library
graf_... Graphics Library
wind_... Window Library

1.4 X- and Y-Coordinates

Many AES functions define objects by their X- and Y-coordinates and their width and height. These X- and Y-coordinates always refer to the upper left corner of the object.

Note: The AES interprets and returns all coordinates as raster coordinates. Raster coordinates are described in the Introduction to GEM Programming.

1.5 GEM AES Data Structures

All AES functions use the following three data structures:

- Parameter Block
- Control Array
- Global Array

1.5.1 Parameter Block

The application must pass a LONG POINTER to the Parameter Block for every AES call.

```
params(0) = long address (32 bits) of control array
params(1) = long address (32 bits) of global array
params(2) = long address (32 bits) of int_in array
params(3) = long address (32 bits) of int_out array
params(4) = long address (32 bits) of addr_in array
params(5) = long address (32 bits) of addr_out array
```

1.5.2 Control Array

The application must set up the Control Array for each AES call it makes.

The Control Array contains the following elements:

```
control(0) = op_code (function number)
control(1) = size in WORDS of int_in array (integer input)
control(2) = size in WORDS of int_out array (integer output)
control(3) = size in LONGS of addr_in array (address input)
control(4) = size in LONGS of addr_out array (address output)
```

Each function contains some or all of the following arrays:

int_in	Each parameter in this array is a WORD.
int_out	Each parameter in this array is a WORD.
addr_in	Each parameter in this array is a POINTER.
addr_out	Each parameter in this array is a POINTER.

The 32 bit values in the addr_in and addr_out arrays are in a format that is native to the host CPU. In Intel™ architecture, the first WORD in memory is the offset or LOW WORD. The second WORD is the segment or HIGH WORD. In Motorola™ architecture, the first WORD in memory is the most significant part of the address or HIGH WORD. The second WORD is the LOW WORD.

This book presents 32 bit addresses in the Intel format. For Motorola format, you must reverse the order of the LOW WORD and the HIGH WORD.

1.5.3 Global Array

The Global Array is 30 bytes long, and is initialized when you make the APPL_INIT call. On return from APPL_INIT, the array values are set as shown in Table 1-2.

Table 1-2. Global Array

Element	Description
global(0)	WORD identifying the GEM version number
global(1)	WORD specifying the number of applications this version of GEM supports concurrently
global(2)	WORD containing a unique application identifier that is in effect as long as the application remains in the AES environment.
global(3)	LOW WORD of window 0 (zero) ob_spec
global(4)	HIGH WORD of window 0 (zero) ob_spec
global(5)	LOW WORD of tree base in memory allocated to resource file
global(6)	HIGH WORD of tree base in memory allocated to resource file
global(7)	LOW WORD of base of memory allocated to resource file
global(8)	HIGH WORD of base of memory allocated to resource file
global(9)	Length in bytes of memory allocated to resource file
global(10)	Number of planes of color (1-4)
global(11)	WORD reserved
global(12)	WORD reserved
global(13)	Bit vector of disk drives installed on GEM Desktop application bit 15 represents drive A bit 14 represents drive B bit 13 represents drive C, and so forth bit set means the drive is installed
global(14)	Bit vector indicating whether installed drive is a floppy or hard disk. Bits are assigned to drives as in global(13). Bit set means the drive is a hard disk

End of Section 1

TYPICAL GEM AES CALLING SEQUENCES

This section describes sequences of AES and VDI calls that a GEM application might make. Where needed, the GEM Desktop application serves as an example of a typical GEM application.

These calling sequences do the following:

- Initialize the application
- Determine the system's screen resolution
- Load the application's resource file
- Get the addresses of the application's resources
- Display the application's menu bar
- Display icons on the desktop
- Let the application await user action
- Let the user select from a menu
- Display a dialog as the result of selecting from a menu
- Let the user make a menu selection by pressing keys
- Let the user select an icon
- Let the user interact with windows

2.1 Initializing an Application

There are three steps to this process:

1. Free unneeded memory.

   When an application is first loaded into memory, it should make a call to the operating system to modify the application's memory allocation. By freeing memory from the end of the application to the top of memory, this call makes space available for the application's resource file and any additional fonts that the application might need. Resource files are described in the GEM Programmer's Utilities Guide.

   If the application does not make this call, the operating system returns an error message when the RSRC_LOAD call (described in

Section 2.3) makes its operating system memory allocation request.

2. Initialize data structures and set up the following arrays:

   - GEM AES Parameter Block
   - Control Array
   - Global Array
   - Integer Input (int_in) Array
   - Integer Output (int_out) Array
   - Address Input (addr_in) Array
   - Address Output (addr_out) Array

   The application allocates space for these arrays and establishes bindings in its code so that parameters go to the right arrays. For examples, see the C language bindings in this guide and in the GEM Developer's Kit.

3. Make the APPL_INIT call.

   APPL_INIT, the application's first AES call, sets up application-specific data structures and returns a system-wide application identifier (ap_id). The AES places ap_id in the Global Array so it can identify the application throughout its calling sequence.

2.2 Finding Screen Resolution

All of an application's text or graphic data that is either specific to a device or to a spoken language is contained in the application's resource file. These materials include the following:

- Text
- Icons
- Menus
- Dialogs
- Forms

All resource files have a .RSC filetype.

Applications usually have at least two resource files, one for a low-resolution screen (640x200 pixels) and another for a high-resolution

screen (640x400, 640x350, or 720x350 pixels). See the discussion of aspect ratio and screen resolution under GEM Resource Construction Set in the GEM Programmer's Utilities Guide.

Before it can load the correct resource file, the application must know the system's screen resolution. To get this information:

1. The application calls the GRAF_HANDLE routine. This call returns the GEM VDI handle for the screen.
2. The application makes a GEM VDI Open Virtual Workstation call, which provides the same information as an Open Workstation call. This call returns the system's screen resolution.

For a description of the GEM VDI Open Virtual Workstation call, see the GEM VDI Reference Guide.

2.3 Loading the Resource File

When the application makes its RSRC_LOAD call, the Resource Library checks the size of the resource file and allocates memory to load it. It then reads in the resource file, adjusting internal pointers to reflect the load address.

The RSRC_LOAD call also converts special X, Y, width, and height screen position information that allows the application to address any pixel on the screen. This coordinate system assumes a standard screen of 80 columns and 25 lines and uses a bit offset method of locating screen positions. For example, on a given line, the AES can address two points that are three pixels apart by identifying their positions as (column 46 + 2 pixels) and (column 46 + 5 pixels).

2.4 Getting Resource Addresses

To get the address of any tree contained in its resource file, the application makes a RSRC_GADDR call. The application can make RSRC_GADDR calls any time after the RSRC_LOAD and before the RSRC_FREE call.

The application makes a series of anticipatory RSRC_GADDR calls,

getting and storing the address of each resource it expects to need in the course of the current session.

2.5 Displaying the Menu Bar

The application's menu bar is a typical resource. To display the menu bar, the application makes two calls:

1. The application calls RSRC_GADDR (if it has not already done so), passing in the menu bar's data structure type and its name, as assigned in the Resource Construction Set.

   RSRC_GADDR returns the LONG ADDRESS to the root of the object tree that represents the menu bar.

2. The application calls MENU_BAR, passing in the menu bar's address and a value of 1 in the showit argument, to display the menu bar.

The Menu Library then displays the menu bar across the top of the screen.

2.6 Displaying Icons in the Desktop Window

The desktop window is the menu bar and the area below it. The AES always assigns a window handle of 0 (zero) to the desktop window.

To display icons in the desktop window, the application must first know the size and location of the desktop window's available work area. The work area of the desktop window includes everything below the menu bar.

To get this information the application makes a WIND_GET call, passing in values that:

- Identify the window as the desktop window (w_handle = 0)
- Ask for the window's X, Y, width, and height values (w_field = 4)

WIND_GET returns the X, Y, width, and height values for the work area of the desktop window.

The application calls the vr_trnfm function, documented in the GEM VDI Reference Guide, to transform the ICONBLK data structure to reflect your hardware device.

The application then makes an OBJC_DRAW call to draw the icons in the work area. The values contained in ib_xicon, ib_yicon, ib_wicon, and ib_hicon in each icon's ICONBLK structure determine where the icon appears.

Note: Most applications use the desktop window only as a backdrop to their own application windows.

2.7 Waiting for a User Event

At this point the application has displayed its menu bar and desktop icons, and it is now ready for user interaction.

The Event Library defines five user interaction events:

- Keystroke
- Pressing a mouse button
- Mouse movement
- Message from a GEM AES process
- Passage of a specified period of time

Although an application can wait for one event at a time, most commonly it makes an EVNT_MULTI call to wait for some combination of events.

When one of the awaited events occurs, GEM AES's dispatcher moves the application from the Not-Ready List to the Ready List. When the application reaches the head of the Ready List, it responds to the user event and then returns to the Not-Ready List to await the next event in the EVNT_MULTI sequence.

Note: It is suggested that GEM applications only accept input from a single mouse button. If a mouse has more than one button, a GEM application should look for input from the button on the left. This allows your application to work with the widest variety of input devices, including touch screens and tablets.

2.8 Menu Selection

An application's menu bar is controlled by the GEM AES Screen Manager; the application is not responsible for user interaction with the menu bar.

The following sequence describes what happens when a user selects Desktop Info..., one of the Desk Menu commands in the GEM Desktop application. (Menu selection for any GEM application should follow the same basic sequence.)

1. The GEM Desktop application makes an EVNT_MULTI call that includes a message as one of the awaited events.

2. The user moves the mouse form into the menu bar, touching the Desktop Menu's title.

3. Receiving notification that the mouse has entered the menu bar, the Screen Manager is dispatched to the Ready List. It determines which menu title the mouse form is touching, saves the part of the screen under the menu, and displays the menu. The Screen Manager highlights menu items as the user moves the mouse form through the menu.

   The GEM Desktop application is still on the Not-Ready List at this time.

4. The user clicks the mouse button on the Desktop Info.... command.

5. The Screen Manager notifies the GEM Desktop application of the user's selection by writing a message to the GEM Desktop's message buffer. The mepbuff parameter of the EVNT_MULTI call contains the buffer's address.

   The predefined AES message MN_SELECTED, described in Section 4.6 contains object tree indexes for the selected menu title and item.

6. When the Screen Manager writes the message, the Dispatcher checks the Not-Ready List for the process that was waiting for the message. It finds the GEM Desktop application and moves it over to the Ready List.

7. The EVNT_MULTI call returns a value to the GEM Desktop application. The bit setting indicates that a message has been received.

8. The GEM Desktop application reads and interprets the message from its buffer and displays the Desktop Information dialog. Displaying a dialog is described in Section 2.9.

9. The menu title remains highlighted until the requested action is complete. In the case of the Desktop Information dialog, the menu title is highlighted until after the final FORM_DIAL call.

   When the action is complete, the application makes a MENU_TNORMAL call with a value of 1 in the normalit argument. This call changes the menu title from its highlighted state to its normal state.

2.9 Displaying a Dialog

To display a dialog, the GEM Desktop application makes the following sequence of calls:

1. The application calls RSRC_GADDR to get the address of the dialog's object tree. The application can make this call at any time between its RSRC_LOAD call and the beginning of the dialog display sequence.

2. The application calls FORM_CENTER to establish the location of the dialog on the screen, because dialog trees generated by the Resource Construction Set have an undefined origin (upper left corner).

3. The application calls the FORM_DIAL routine, passing in a dtype value of 0 (zero), an FMD_START call. This call reserves screen space for the dialog.

4. The application calls the OBJC_DRAW routine to draw the dialog, passing the address of the tree, the first object to draw, the drawing depth, and the rectangle allotted for the dialog.

5. The application calls FORM_DO to monitor the user's interaction with the dialog.

6. When one of the dialog's exit conditions is met, the application compares the dialog's initial values, which were set up in the RSRC_LOAD call when the dialog was created, with the values the dialog now contains. The application notes any changes and acts accordingly.

   In many cases the application makes a series of OBJC_CHANGE calls to reset the dialog objects to their initial values.

   For example, after the user exits the Desktop Information dialog, the GEM Desktop application changes the OK exit button's ob_state from SELECTED to NORMAL so that the next time the user selects the Desktop Info... option, the dialog does not appear with its OK button already highlighted.

   In some cases, the user can overwrite text strings that have been set to initial values. The user backspaces over the string and types a new string. After comparing values and acting accordingly, the application might reset the string to its initial value.

   Note, however, that the user might want to save some changes made to dialogs. For example, in the ITEM INFORMATION/RENAME dialog, the user will want to save both the document's name and its Read-Write or Read-Only status.

7. The application calls FORM_DIAL again, this time passing in a dtype value of 3, a FMD_FINISH call.

   This call removes the dialog from the screen and frees the screen space that had been reserved by the dialog. The call also causes the Screen Manager to send a message to the application to redraw the screen.

   The application can redraw the screen with an OBJC_DRAW call or with several GEM VDI calls.

   To be able to respond to such a redraw message at any time, the application should be in an EVNT_MULTI wait.

2.10 Keystroke Menu Selection

The AES allows the user to select some menu items by pressing a specially designated key or combination of keys instead of using a menu. To enable this feature, the application should specify a keyboard event as one of the awaited events in the EVNT_MULTI call.

With the exception of cursor positioning, it is the responsibility of the application to define the keystrokes and what they mean. The AES does not define or reserve any keys for special purposes.

When the user presses one of the menu item selection keys, the application makes a MENU_TNORMAL call with a value of 0 (zero) in the normalit argument, to highlight the menu title. The user does not see the menu, but the highlighted menu title serves as notice that the application is acting on the user's request.

When the requested action has taken place, the application makes a MENU_TNORMAL call with a normalit value of 1, to return the menu title to its normal state.

2.11 Selecting an Icon

To select an icon, the user places the mouse form on an icon and clicks once.

The following sequence describes icon selection:

1. The application sets the bit for a button event (MU_BUTTON) in the EVNT_MULTI call. Input parameters for this call include the awaited mouse button state (up or down) and the number of times the application wants the button to enter that state within the preset time interval.

2. When the user clicks on an icon, the EVNT_MULTI call returns a value with the bit set for a mouse button event. It also returns the mouse form's X and Y coordinates.

3. The application makes an OBJC_FIND call, passing in the mouse form's X- and Y-coordinates from the previous call. The application also passes in the address of the object tree that draws the icons in the application's window.

4. If the OBJC_FIND call reports that the mouse form is over an icon, the application makes an OBJC_CHANGE call to change the icon's ob_state value from NORMAL to SELECTED.

If the user clicks the mouse button and the mouse form is over nothing, the user is trying to deselect the selected icon.

If the user depresses the mouse button and the mouse form is not over an icon, the application assumes that the user intends to select a group of icons by dragging an expanding (or "rubber") rectangle around them. In that event, the application makes the following sequence of calls:

1. The application makes a GRAF_MKSTATE call to see if the button is still down.

2. If so, the application makes a GRAF_RUBBOX call to draw the rubber rectangle that surrounds the icons the user wishes to select.

   The call's input X and Y values are the X and Y values from the EVNT_MULTI call.

   The GRAF_RUBBOX call's output width and height values (pwend and phend) define the size of the rubber rectangle at the time the user released the mouse button.

3. The application looks for icons inside the rectangle.

4. The application makes an OBJC_CHANGE call for each icon, changing its ob_state from NORMAL to SELECTED.

Selecting an icon can change the appearance of menu items. For example, when a folder, document, application, or disk icon is selected, the File Menu item Open should change state from disabled to enabled. The same icon selection can change other menu items from enabled to disabled.

A disabled menu item appears in dimmed characters, which indicates to the user that the item cannot be selected. An enabled item appears in characters of normal brightness.

When the user selects an icon, the application's code determines which menu items need to change state. The application then makes

a MENU_IENABLE (item enable) call for each of these items, passing in a value of 0 (zero) to disable an item or 1 to enable it in the enableit argument. The application should do this for every context change whether or not icons are selected.

2.12 Creating a Window

When a GEM application is running, the AES and the application share responsibility for drawing and managing windows. The AES is responsible for all user interactions with any components present in the application window's border area. These components include the following:

- Title bar
- Move bar
- Information Line
- Size box
- Full box
- Close box
- Arrows, scroll bars, and sliders

The application is responsible for drawing and managing everything that appears inside the application window's work area, which is everything inside the borders.

For more information on windows see Section 11.

To make a window appear on the screen:

1. Create the window to define what components will be present in the window.
2. Open the window to make it appear.

When the application makes a WIND_CREATE call, it passes in a bit vector with a bit set for each border area component the window will have. To request multiple components, the bits are OR-ed together. The application also passes in the size and location of the window's greatest possible size.

The GEM Desktop application and the calculator desk accessory illustrate how the WIND_CREATE call works. (The calculator actually

appears in a window, although from the user's viewpoint, the calculator and its window are the same.)

When the GEM Desktop makes a WIND_CREATE call, the bit is on for (among others) the full box. This means the full box appears in the GEM Desktop window's border area, and the user can make the window appear in its largest possible size. In addition, the WIND_CREATE call defines the window's largest size as the size of the desktop window's work area (all of the screen below the menu bar).

When the calculator desk accessory makes a WIND_CREATE call, the bit is off for the full box. No full box appears in the window's border area, and the user cannot change the size of the window. The WIND_CREATE call defines the window's greatest possible size (its only size, because there is no full box or size box) as nineteen characters wide by thirteen characters high.

If an application does not support a particular window function (like sizing), it should not request the window control point (the size box) that supports this function.

When an application makes a WIND_CREATE call, the AES returns a window handle, a numeric identifier the application uses for all future AES calls relative to this window. Recall that the identifier for the desktop window is always 0 (zero).

2.13 Calculating Window Dimensions

Before issuing the WIND_OPEN call to open the window, the application might need to make a WIND_CALC call to perform the following calculation.

- Using the size and location of the window's outer dimensions (including the border area) as input parameters, WIND_CALC returns the size and location of the application window's work area.
- Using the size and location of the window's work area as input parameters, WIND_CALC returns the size and location of the window's outer dimensions (including the border area).

In either case, WIND_CALC uses the same bit vector that WIND_CREATE used to identify the components of the window's border.

Note: Every application must handle the case when there are no more windows as indicated by an error returned from the WIND_CREATE call.

2.14 Opening a Window

The WIND_OPEN call causes the window to appear on the screen.

In making the call, the application passes in the window handle from WIND_CREATE and the window's initial size and location.

The application determines the initial size and location. The application can be written to remember a window's previous size and location, or the application can specify that a window always open in the same size and location.

When the application makes the WIND_OPEN call, the GEM AES Screen Manager draws the window's border area and then sends a message to the application to draw the window's work area.

2.15 Slider Size and Location

If the work area of the window contains only part of the directory or document (if only a portion of the amount of data is visible in the physical window), the application makes a WIND_SET call to set the size and location of the vertical and/or horizontal sliders. A separate call is required for the size and location of each slider.

The application makes similar WIND_SET calls each time the size and location of the sliders change.

2.16 Sizing a Window

When the user drags the window's size box, the AES is responsible for displaying the rubber box that shows the user a preview of the window's new size.

When the user releases the mouse button, the AES sends the application a message containing the dimensions of the window the user is requesting. The application must decide if the requested size is valid.

If the requested size is valid, the application issues a WIND_SET call to change the size of the window. If the new window is smaller than the current window, the application does not have to redraw the window's work area. If the new window is larger than the current window, the AES sends the application a WM_REDRAW message requesting that it redraw the contents of the window's work area.

If the requested size is not valid, the application must decide how to respond to such a request. It can do any of the following:

- Ignore the request.
- Automatically size the window to the nearest valid size.
- Display a dialog that informs the user the request is not valid.

2.17 Rectangle List

An application is only responsible for redrawing and updating the visible portion of its windows. To keep track of this area, the AES divides the visible portion of each window's work area into the fewest possible number of non-overlapping rectangles. For example, if the entire window is visible, there is only one rectangle, the work area itself.

The AES keeps a list of these rectangles. The application gets the rectangle by making a series of WIND_GET calls, the first with an input value of WF_FIRSTXYWH, and the subsequent calls with values of WF_NEXTXYWH. The application continues making these calls until the returned width and height values for the rectangle are 0 (zero).

Figure 11-2 shows window rectangles.

2.18 Before Updating a Window

Before it updates a window, or under any circumstances draws to the screen, an application must notify the AES that an update is about to take place. It notifies the AES with the WIND_UPDATE function.

The application makes a WIND_UPDATE call with a beg_update value of 1, which indicates the beginning of a window update. This call prevents menus and alerts from appearing during a window update, and also prevents other applications from drawing to the screen.

When the update is complete, the application makes a WIND_UPDATE call with a beg_update value of 0 (zero), which indicates the end of a window update, and allows other applications to draw to the screen.

2.19 Redrawing the Work Area

When it redraws its window's work area, an application makes a WIND_GET call to get the first rectangle in the rectangle list.

The application then looks to see if the first rectangle has any area in common with the "update rectangle," the part of the work area that is to be redrawn. If so, the application redraws that common area. If not, the application makes another WIND_GET call to get the next rectangle in the list. The application compares the next rectangle with the update rectangle and again redraws any area common to both rectangles.

The application continues this sequence of WIND_GET calls, comparisons, and redraws until it has gone through all the rectangles in the rectangle list.

2.20 Making a Window Active

When the user clicks the mouse button, the application needs to find out where the mouse button was pressed. It makes a WIND_FIND call, passing in the mouse's X- and Y-coordinates, which were returned by the EVNT_MULTI call. The WIND_FIND call returns the window handle of the window under the mouse form.

If the window handle identifies an inactive window (including a desk accessory window), the Screen Manager sends a message to the application that owns the window. The Screen Manager uses the predefined message WM_TOPPED, which tells the application that the user has requested that its window be brought to the top.

To bring the window to the top (make it "active"), the application makes a WIND_SET call with input values including the window's handle and the a code indicating that the window is to be brought to the top.

The window will not be topped until the WIND_SET call is made with WF_TOP in the w_field argument.

2.21 Closing and Deleting a Window

When the user closes a window, either by interaction with the window's border area or by choosing a command from a menu, the Screen Manager sends a message to the application to close the window. The application makes a WIND_CLOSE call, passing in the handle of the window to be closed.

When the window is closed, its handle is still allocated to the application. The handle is not available again until the application makes a WIND_DELETE call.

The user cannot detect WIND_CREATE or WIND_DELETE; the user can only detect WIND_OPEN and WIND_CLOSE. In most cases, an application will make the create and open calls one immediately after the other, and it will do the same with the close and delete calls. However, this is optional.

Section 11, Window Library, contains more details on specific windowing techniques as well as descriptions of the individual Window Library calls.

End of Section 2

APPLICATION LIBRARY

An application must first register with the Application Library before it can use the other libraries.

The Application Library controls access to the other GEM AES libraries. Because many applications use these libraries at the same time, each function must know which application is requesting a service.

The following sequence illustrates the Application Library's role:

1. An application is loaded into memory and starts executing.
2. The application reserves the required space for the Global Array, described in Section 1.5.3, and makes a call to tell the AES to initialize the space.
3. The application enters its main body of code and runs until the user requests that it terminate.
4. The application exits the Application Library.
5. The application terminates.

Note: The application should never tamper with the space allocated to it by the AES. This space contains global data structures that are vital to the successful use of all AES subroutines.

3.1 Application Messages

An application can send a message to another application. The APPL_READ and APPL_WRITE functions are responsible for these message pipes. This message format is similar to but not the same as the predefined messages the AES can send to an application. Predefined messages are described in Section 4.6.

The standard format for application-defined messages is as follows: Use message types in the range 1024 to 32,000.

WORD 0	Message type
WORD 1	Sender's ap_id
WORD 2	0xFFFF
WORD 3	Length of message buffer at pointer
WORD 4	Offset of message buffer
WORD 5	Segment of message buffer
WORD 6	Application specific
WORD 7	Application specific

3.2 Application Library Routines

The Application Library provides the following routines:

APPL_INIT	Initializes a session with the Application Library.
APPL_READ	Lets an application read a specified number of bytes from a message pipe.
APPL_WRITE	Lets an application write a specified number of bytes to a message pipe.
APPL_FIND	Finds the application identifier of another application in the system.
APPL_TPLAY	Plays a piece of an AES recording of the user's actions.
APPL_TRECORD	Records a set of the user's interactions with the AES.
APPL_EXIT	Exits a session with the Application Library.

APPL_INIT

Initializes the application and establishes a number of internal AES data structures. You must use APPL_INIT before calling any other AES function.

The APPL_INIT call resets an internal counter, giving the calling program 10 AES calls before a dispatch occurs. Use this feature to acquire contiguous memory either with a memory allocate or RSRC_LOAD call.

Output Arguments

ap_id — A code indicating whether or not the APPL_INIT call was successful.

- 0 or n — APPL_INIT was successful. The AES places this number in the Global Array, and the application uses it with future calls to Application Library functions.
- -1 — APPL_INIT was not successful. The application should make no further Application Library calls.

Sample Call to C Language Binding

```
WORD  appl_init();
WORD  ap_id;

ap_id = appl_init();
```

Parameter Block Binding

Control	Input	Output
control(0) = 10		int_out(0) = ap_id
control(1) = 0		
control(2) = 1		
control(3) = 0		
control(4) = 0		

APPL_READ

Reads a specified number of bytes from a message pipe.

Input Arguments

rwid	The ap_id of the process whose message pipe the application is reading, usually its own.
length	The number of bytes to read from the message pipe.
pbuff	The address of the buffer that holds the bytes the application is reading.

Output Arguments

retval	A coded return message: 0 error n no error

Sample call to C language binding

```
WORD   appl_read();
WORD   retval,rwid,length;
LONG   pbuff;

retval = appl_read(rwid, length, pbuff);
```

Parameter Block Binding

Control	Input	Output
control(0) = 11	int_in(0) = rwid	int_out(0) = retval
control(1) = 2	int_in(1) = length	
control(2) = 1		
control(3) = 1	addr_in(0) = pbuff	
control(4) = 0		

APPL_WRITE

Writes a specified number of bytes to a message pipe.

Input Arguments

rwid	The ap_id of the process to which the application is writing, usually not itself.
length	The number of bytes to write to the message pipe.
pbuff	The address of the buffer holding the bytes to be written.

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   appl_write();
WORD   retval, rwid, length;
LONG   pbuff;

retval = appl_write(rwid, length, pbuff);
```

Parameter Block Binding

Control	Input	Output
control(0) = 12	int_in(0) = rwid	int_out(0) = retval
control(1) = 2	int_in(1) = length	
control(2) = 1		
control(3) = 1	addr_in(0) = pbuff	
control(4) = 0		

APPL_FIND

Finds the ap_id of another application in the system. The calling application must know the other application's ap_id before it can establish communications.

Input Arguments

pname — Address of a null-terminated string containing the filename of the application for which the current application is searching.

The string must be 8 characters long. If the filename has fewer than 8 characters, the rest of the string must be filled out with blank spaces.

Output Arguments

fid — The ap_id of the application for which the current application is searching.

-1 The AES could not find the application.

Sample Call to C Language Binding

```
WORD   appl_find();
WORD   fid;
LONG   pname;

fid = appl_find(pname);
```

Parameter Block Binding

Control	Input	Output
control(0) = 13	addr_in(0) = pname	int_out(0) = fid
control(1) = 0		
control(2) = 1		
control(3) = 1		
control(4) = 0		

APPL_TPLAY

Plays a piece of an AES recording of the user's actions.

Input Arguments

tlength — The number of user actions to play back.

tscale — A sliding scale from 1 to 10,000 determining the speed at which the AES plays back the user's actions. For example:

50	half speed
100	full speed
200	twice speed

tbuffer — The address of the area in memory holding the recording of user events that the AES will play back.

Output Arguments

retval — Always equals 1 (one).

Sample Call to C Language Binding

```
WORD   appl_tplay();
WORD   retval, tlength, tscale;
LONG   tbuffer;

retval = appl_tplay(tbuffer, tlength, tscale);
```

Parameter Block Binding

Control	Input	Output
control(0) = 14	int_in(0) = tlength	int_out(0) = retval
control(1) = 2	int_in(1) = tscale	
control(2) = 1		
control(3) = 1	addr_in(0) = tbuffer	
control(4) = 0		

APPL_TRECORD

Records a set of the user's interactions with the AES.

Entry of a Ctrl-Backslash (\) terminates a recording sequence. Otherwise, the event recording continues until the tlength is reached.

Each user event uses six bytes in memory, divided into a WORD and a LONG value. In Motorola architecture, each user event is eight bytes, divided into two LONG values.

The WORD contains a code for the event that occurred, as defined by the Event Library. In Motorola architecture, this value is a LONG. The codes are:

0	timer event
1	button event
2	mouse event
3	keyboard event

The LONG value's meaning depends on the type of event that was recorded.

Timer event	The number of elapsed milliseconds.
Button event	The LOW WORD is the button state. 0 button up 1 button down The HIGH WORD is the number of clicks.
Mouse event	The LOW and HIGH WORD are the mouse's X- and Y-coordinates in pixels, respectively.
Keyboard event	The LOW WORD contains the character the user typed. The HIGH WORD contains the keyboard state, which is the state of the keyboard's right-shift, left-shift, Ctrl, and Alt keys, when the user event occurred.

Input Arguments

tlength — The number of user events that the application can store. This number equals the available storage space (in bytes) divided by the 6 bytes used by each event.

tbuffer — The address of an area in memory where the recorded user events will be stored.

Output Arguments

retval — The number of user events the application recorded.

Sample Call to C Language Binding

```
WORD   appl_trecord();
WORD   retval, tlength;
LONG   tbuffer;

retval = appl_trecord(tbuffer, tlength);
```

Parameter Block Binding

Control	Input	Output
control(0) = 15	int_in(0) = tlength	int_out(0) = retval
control(1) = 1		
control(2) = 1	addr_in(0) = tbuffer	
control(3) = 1		
control(4) = 0		

APPL_BVSET

Sets the disk and hard disk configuration information. See Section 1.5.3, Global Array, for more information.

Input Arguments

bvdisk	Bit vector of all disk drives on the system. The most significant bit represents drive A.
bvhard	Bit vector of hard disk drives on the system. The most significant bit represents drive A.

Output Arguments

retval	Undefined

Sample Call to C Language Binding

```
WORD   appl_bvset();
WORD   retval;
UWORD bvdisk, bvhard;

retval = appl_bvset(bvdisk, bvhard);
```

Parameter Block Binding

Control	Input	Output
control(0) = 16	int_in(0) = bvdisk	int_out(0) = retval
control(1) = 2	int_in(1) = bvhard	
control(2) = 1		
control(3) = 0		
control(4) = 0		

APPL_YIELD

Forces a dispatch. This allows all events to be processed, and possibly one other process to be brought into context.

Output Arguments

retval Undefined

Sample Call to C Language Binding

```
WORD   appl_yield();
WORD   retval;

retval = appl_yield();
```

Parameter Block Binding

Control	Input	Output
control(0) = 17		int_out(0) = retval
control(1) = 0		
control(2) = 1		
control(3) = 0		
control(4) = 0		

APPL_EXIT

Lets the Application Library clean up its environment when an application is done making Application Library calls.

An application cannot make calls to the AES after an APPL_EXIT call.

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   appl_exit();
WORD   retval;

retval = appl_exit();
```

Parameter Block Binding

Control	Input	Output
control(0) = 19		int_out(0) = retval
control(1) = 0		
control(2) = 1		
control(3) = 0		
control(4) = 0		

End of Section 3

EVENT LIBRARY

An interactive application must be able to respond quickly to several types of user input, including:

- Typing on a keyboard
- Clicking a mouse button
- Moving a mouse
- Choosing a menu command
- Manipulating a control point on the border of a window
- Doing nothing when something is expected; the lack of input

GEM AES refers to these types of user input as "events" and provides application writers with an Event Library of routines that monitor events. These routines can significantly increase the speed and efficiency of the application.

Note: Do not use GEM VDI input functions in programs that make GEM AES calls. Restrict VDI use to graphic output functions if your application uses the AES.

The Event Library lets an application get input from the keyboard and mouse, from other programs in the system, and from the system itself. In most programming interfaces available for developing desktop-style applications, the programmer must write an application that spins in a tight loop polling the keyboard, mouse, message pipe (described in Section 4.5), and clock. This type of polling can exhaust the system resources.

The Event Library avoids this problem by letting the application tell the operating system what types of events to wait for. The operating system can let other programs run, and it need only activate the application when one or more of the desired events has occurred.

The Event Library lets an application wait for any of the following events:

Keyboard event	The user presses a key.
Button event	The user presses or releases a mouse button.
Mouse event	The user moves the mouse into or out of a specified rectangle.
Message event	An application receives a message from another process.
Timer event	The preset timer amount expires.
Multiple event	Any combination of the other events.

4.1 Waiting for Multiple Events

If an application were to wait only for a single type of event, it would not respond to events of any other type. An application needs to be able to wait for one or more of a specified set of events. The Event Library makes this possible.

When any or all of the events in the set occur, the Event Library notifies the application and returns a value that identifies the events that have occurred. The application uses this value to determine how it should process the events.

After processing the event or events, the application typically specifies another set of events, calls the Event Library, and then awaits notification that one of the new set of events has taken place. While the application is awaiting notification, the system can share CPU time with other processes that are ready to run.

4.2 Keyboard Event

GEM AES recognizes a standard keyboard. The definition of this keyboard is documented in the GEM VDI Reference Guide.

These keyboard events are encoded in a 16-bit form of console input. The state of the Ctrl, Shift, and Alt keys is also available by making the GRAF_MKSTATE function call.

4.3 Mouse Button Event

The AES lets an application wait for a specified mouse button or set of buttons to enter or leave a specified state (down or up).

A mask word performs a logical AND operation on the bits representing the mouse buttons the application wants to ignore. For example, on a three-button mouse, a mask word value of 001 indicates that the application only cares about input from the button on the left. (The left button is represented by the least significant bit.)

State words indicate the state of the mouse buttons, both current and desired. For example, a current state word value of 001 indicates that the user has pressed the left button.

A mouse button event takes place when the following equation is true:

(current_state AND mask) = desired_state

For example, if the user presses the left button and that is what the application is waiting for, a mouse button event takes place. In that case the equation reads:

(001 AND 001) = 001

The application can also wait for the mouse button to enter or leave the desired state a specified number of times in a set interval. The Event Library returns to the application the number of times the mouse button entered or left the desired state in the interval. The number returned is always at least 1 (one) and never more than the number desired by the application.

4.4 Mouse Event

Several kinds of mouse movement can cause an application to change the appearance of the screen, including the following:

- Dragging an icon over the desktop window
- Drawing a rubberband line or rectangle
- Moving the mouse form into a sensitive region

A Mouse Event occurs when the mouse is either inside or outside a pixel-aligned rectangle. For example, using a Mouse Event, an application can change the mouse form from an arrow to a cross hair whenever the mouse is inside a certain area of the screen. The application waits for the Mouse Event that indicates that the mouse is inside a certain rectangle on the screen. When the mouse enters the rectangle, the AES notifies the application. The application can then make a GRAF_MOUSE call to change the mouse form to a cross hair. The application then waits for a mouse event indicating that the mouse is outside the rectangle. As soon as the mouse exits, the AES notifies the application so it can change the mouse form back to an arrow.

The size of this critical rectangle depends on the resolution that is required for the mouse response. For example, dragging objects that can be placed on arbitrary pixel boundaries requires a rectangle that is one pixel high and one pixel wide. However, most applications, including graphics applications that use a grid for aligning elements, do not always require such fine resolution. For example, inverting the items in a menu requires a resolution equal to the size of the menu item in which the mouse form is located.

Systems can achieve significant improvements in overall throughput if the amount of mouse motion significant for each action determines the size of each of the application's mouse event rectangles.

4.5 Message Event

The AES programming environment provides a user interface in which applications can use separate overlapping windows. The windows reside on the physical screen, which is one color with a menu bar running across the top. The window that is on top and has control of the keyboard is called the "active window."

The application that owns the active window provides the AES with the following:

- The set of menus that appears in the menu bar
- The title that the AES places in the title bar of the application's window
- The window control areas, including the close box, full box, and size box, to which the application will respond. Window control areas are described in Section 11.3.

To ensure a consistent user interface and increase programmer productivity, the AES manages all interactions with the user during menu selection and window border manipulation. However, applications need to know the results of these external user interactions. To provide this information, the AES uses Message Events.

To receive notification of external events, applications use a standard message pipe. A Message Event occurs when an application receives a message in its message pipe.

Messages come in a standard format defined by the AES Message Protocol and are placed in an application's message pipe in First-In-First-Out (FIFO) order. Each time an application reads a message in its message pipe, the AES removes the message from the pipe.

4.6 Predefined GEM AES Messages

The AES uses predefined messages to tell an application that a certain event has occurred. Each message type has a maximum length of 16 bytes. All the predefined message types define the first three words in the same way:

WORD 0	A number identifying the message type
WORD 1	The ap_id of the application that sent the message
WORD 2	The length of the message, not counting the predefined 16 bytes. If the value of WORD 2 is not 0 (the message is longer than 16 bytes), the application should call APPL_READ to read the remainder of the message.
WORD 3-7	Varies according to the message

4.6.1 MN_SELECTED

This message tells an application that a the user has selected a menu item.

WORD 0	10
WORD 3	Object index of the menu title selected
WORD 4	Object index of the menu item selected

4.6.2 WM_REDRAW

This message tells an application that the user has taken an action that requires redrawing part of the work area of its window. Values are in raster coordinates.

WORD 0	20
WORD 3	Handle of the window to redraw
WORD 4	X-coordinate of the area to redraw
WORD 5	Y-coordinate of the area to redraw
WORD 6	Width of the area to redraw
WORD 7	Height of the area to redraw

4.6.3 WM_TOPPED

This message tells an application that the user has requested its window or another application's window to be moved to the top (made active).

WORD 0	21
WORD 3	Handle of the window

4.6.4 WM_CLOSED

This message tells an application that the user has requested that its window be closed.

WORD 0	22
WORD 3	Handle of the window.

4.6.5 WM_FULLED

This message tells an application that the user has clicked the mouse button in the window's full box, either to enlarge the window to its fullest possible size or, if the window is already full, to return it to its previous size.

WORD 0	23
WORD 3	Handle of the window

4.6.6 WM_ARROWED

This message tells an application that the user has clicked in the arrows or scroll bars in the window's border area. Holding down the mouse button causes continuous WM_ARROWED messages until the button is released.

WORD 0	24
WORD 3	The handle of the window
WORD 4	The action requested by the user, as follows:

0 page up
1 page down
2 row up
3 row down
4 page left
5 page right
6 column left
7 column right

The user invokes the page actions by clicking on the scroll bars. The user invokes the row and column actions by clicking on the arrows. Section 11.3, Components of the Border Area, describes scrolling.

4.6.7 WM_HSLID

This message tells an application the new position the user has requested for the horizontal slider.

WORD 0	25
WORD 3	The handle of the application's window
WORD 4	A number from 1 to 1000, indicating the requested slider position:

1	leftmost position
1000	rightmost position

4.6.8 WM_VSLID

This message tells an application the new position that the user has requested for the vertical slider.

WORD 0	26
WORD 3	The handle of the application's window.
WORD 4	A number from 1 to 1000, indicating the requested slider position:

1	top position
1000	bottom position

4.6.9 WM_SIZED

This message gives an application its window's new coordinates when the user requests a change in the window's size. The coordinates include the window's title bar, information line (if any), and borders.

WORD 0	27
WORD 3	Handle of the window
WORD 4	Requested X-coordinate (should remain the same as the window's current X-coordinate)
WORD 5	Requested Y-coordinate (should remain the same as the window's current Y-coordinate)
WORD 6	Requested width
WORD 7	Requested height

4.6.10 WM_MOVED

This message gives an application its window's new coordinates when the user requests a change in the window's position. The coordinates include the window's title bar, information line (if any), and borders.

WORD 0	28
WORD 3	Handle of the window
WORD 4	Requested X-coordinate
WORD 5	Requested Y-coordinate
WORD 6	Requested width (should remain the same as the window's current width)
WORD 7	Requested height (should remain the same as the window's current height)

4.6.11 WM_UNTOPPED

This message tells an application that its window is about to be untopped (made inactive). This allows the application to save any state or screen data before any portion of its window is occluded.

WORD 0	30
WORD 3	Handle of the window

4.6.12 AC_OPEN

The AES sends this message to a desk accessory when the user selects it from the Desktop Menu.

WORD 0	40
WORD 3	The desk accessory menu item identifier returned by the MENU_REGISTER call.

4.6.13 AC_CLOSE

The AES sends this message to a desk accessory when the following set of conditions exists:

- The current application has just terminated.
- The screen is about to be cleared.
- Window Library data structures are about to be reinitialized.

The desk accessory should zero any window handle it currently owns.

WORD 0	41
WORD 3	The desk accessory menu item identifier returned by the MENU_REGISTER call.

4.7 Application Dependent Messages

The print spooler prints files in the background and is available to the user through the CALCLOCK accessory. To print a file, an application sends a print message to the accessory specifying the file name. Use the APPL_FIND call (specify CALCLOCK in the pname parameter) to get the spooler ap_id for the APPL_WRITE call.

Use the message with type 100 to specify the file to spool. The spooler returns the message with type 101 to acknowledge receipt of the request.

WORD 0	100
WORD 1	Sender's ap_id
WORD 2	0xFFFF
WORD 3	Length in bytes of file specification
WORD 4	Offset of file specification
WORD 5	Segment of file specification
WORD 6	Number of copies to print
WORD 7	1 = Delete file when done 0 = Do not delete file when done

The spooler returns the message with type 101 to acknowledge receipt of the spool request from message type 100.

WORD 0	101
WORD 1	Print spooler ap_id
WORDS 2-7	0

4.8 Timer Event

An application sometimes needs to wait a certain amount of time before proceeding. For example, the application might be displaying a message that must remain on the screen for a maximum of three seconds. To gauge the time, the application can poll the system clock or do a large number of difficult, hardware-specific calculations. However, both of these methods are inefficient in a multitasking system in which processes can make good use of each other's delay time.

By using Timer Events, the AES provides a more efficient method. A Timer Event occurs when a programmer-specified number of milliseconds has passed since the Timer Event was started.

4.9 Event Library Routines

The Event Library contains the following routines:

EVNT_KEYBD	Waits for a keyboard event.
EVNT_BUTTON	Waits for a mouse button event.
EVNT_MOUSE	Waits for a mouse event.
EVNT_MESAG	Waits for a message event.
EVNT_TIMER	Waits for a timer event.
EVNT_MULTI	Waits for multiple events.
EVNT_DCLICK	Sets and gets the speed required for double-clicking.

EVNT_KEYBD

Notifies the AES that the application is waiting for any kind of keyboard input.

Output Arguments

retval	The standard keyboard scan code as defined in the GEM VDI Reference Guide.

Sample Call to C Language Binding

```
UWORD evnt_keybd();
WORD  retval;

retval = evnt_keybd();
```

Parameter Block Binding

Control	Input	Output
control(0) = 20		int_out(0) = retval
control(1) = 0		
control(2) = 1		
control(3) = 0		
control(4) = 0		

EVNT_BUTTON

Notifies the AES that the application is waiting for a particular mouse button state.

Input Arguments

clicks — The number of times the application is waiting for the mouse button to enter or leave a particular state (the state argument) within a preset time.

Use the high order bit to designate whether you are entering or leaving a state. The AES interprets the remaining 7 bits as the number of times. Bit values for the high bit are assigned as follows:

0 waiting to enter the specified state
1 waiting to leave the specified state

mask — Mouse buttons for which the application is waiting. The AES can theoretically support 16 mouse buttons. In mask, state, and pmb, the following bits represent the buttons:

0x0001 button on left
0x0002 second button from left
0x0004 third button from left, etc.

The button specification can specify multiple buttons. For example, enter 0x003 to specify the leftmost and second from left buttons. The event returns when the button state occurs on either button. Note that you cannot specify multibutton states.

state — Button state for which the application is waiting. These parameters use the following bit settings:

0 button up
1 button down

Output Arguments

retval — Number of times the button actually entered the desired state within the preset time. This number is never less than 1 or greater than the number contained in clicks.

pmx — X-coordinate of the mouse pointer when the user event occurred.

pmy — Y-coordinate of the mouse pointer when the user event occurred.

pmb — Mouse button state when the user event occurred. The following bits represent the buttons:

0x0001 button on left
0x0002 second button from left
0x0004 third button from left, etc.

pks — State of the keyboard's right-Shift, left-Shift, Ctrl, and Alt keys when the user event occurred.

The following bits represent the keys:

0x0001 right-Shift
0x0002 left-Shift
0x0004 Ctrl
0x0008 Alt

This parameter uses the following bit settings:

0 key up
1 key down

Sample Call to C Language Binding

```
WORD   evnt_button();
WORD   retval, clicks, pmx, pmy, pmb, pks;
UWORD mask, state;

retval = evnt_button(clicks, mask, state, &pmx, &pmy, &pmb, &pks);
```

Parameter Block Binding

Control	Input	Output
control(0) = 21	int_in(0) = clicks	int_out(0) = retval
control(1) = 3	int_in(1) = mask	int_out(1) = pmx
control(2) = 5	int_in(2) = state	int_out(2) = pmy
control(3) = 0		int_out(3) = pmb
control(4) = 0		int_out(4) = pks

EVNT_MOUSE

Notifies the AES that the application is waiting for the mouse to enter or leave a specified rectangle.

Input Arguments

flags	Flags for the call. 0x0000 return on entry 0x0001 return on exit
x	X-coordinate of the mouse rectangle in pixel-based screen coordinates.
y	Y-coordinate of the mouse rectangle in pixel-based screen coordinates.
width	Width of the mouse rectangle in pixel-based screen coordinates.
height	Height of the mouse rectangle in pixel-based screen coordinates.

Output Arguments

retval	Reserved; value always equals 1 (one).
pmx	X-coordinate of the mouse pointer when the user event occurred.
pmy	Y-coordinate of the mouse pointer when the user event occurred.
pmb	Mouse button state when the user event occurred. The following bits represent the buttons: 0x0001 button on left 0x0002 second button from left 0x0004 third button from left, etc.

This parameter uses the following bit settings:

0 button up
1 button down

pks State of the keyboard's right-Shift, left-Shift, Ctrl, and Alt keys when the user event occurred.

The following bits represent the keys:

0x0001 right-Shift
0x0002 left-Shift
0x0004 Ctrl
0x0008 Alt

This parameter uses the following bit settings:

0 key up
1 key down

Sample Call to C Language Binding

```
WORD  evnt_mouse();
WORD  retval, flags, x, y, width, height, pmx, pmy, pmb, pks;

retval=evnt_mouse(flags,x,y,width,height,&pmx,&pmy,&pmb,&pks);
```

Parameter Block Binding

Control	Input	Output
control(0) = 22	int_in(0) = flags	int_out(0) = retval
control(1) = 5	int_in(1) = x	int_out(1) = pmx
control(2) = 5	int_in(2) = y	int_out(2) = pmy
control(3) = 0	int_in(3) = width	int_out(3) = pmb
control(4) = 0	int_in(4) = height	int_out(4) = pks

EVNT_MESAG

Notifies the AES that the application is waiting for a standard 16-byte message in the message pipe.

Using message pipes to communicate between processes in the system is very flexible and makes possible many different types of messages in the 16-byte message buffer. For these messages to be meaningful to the receiving application, a well-defined set of message protocols must exist. The AES provides several predefined messages, described in Section 4.6.

Input Arguments

pbuff — Address of the buffer where the message will be placed. Its size must be 16 bytes.

Output Arguments

retval — Reserved; value always equals 1 (one).

Sample Call to C Language Binding

```
WORD   evnt_mesag();
WORD   retval;
LONG   pbuff;

retval = evnt_mesag(pbuff);
```

Parameter Block Binding

Control	Input	Output
control(0) = 23	addr_in(0) = pbuff	int_out(0) = retval
control(1) = 0		
control(2) = 1		
control(3) = 1		
control(4) = 0		

EVNT_TIMER

Notifies the AES that the application is waiting for a specified amount of time to pass. The WORD order in int_in(0) and int_in(1) is correct for both Intel and Motorola architecture.

Input Arguments

locnt	LOW WORD of a LONG value.
hicnt	HIGH WORD of a LONG value.
	Combined, locnt and hicnt are the length of the time interval in milliseconds.

Output Arguments

retval	Reserved; value always equals 1 (one).

Sample Call to C Language Binding

```
WORD  evnt_timer();
WORD  retval;
UWORD locnt, hicnt;

retval = evnt_timer(locnt, hicnt);
```

Parameter Block Binding

Control	Input	Output
control(0) = 24	int_in(0) = locnt	int_out(0) = retval
control(1) = 2	int_in(1) = hicnt	
control(2) = 1		
control(3) = 0		
control(4) = 0		

EVNT_MULTI

Notifies the AES that the application is waiting for one or more events at the same time.

Input Arguments

flags — Type of event for which the application is waiting. This call uses the following bit settings:

0x0001 MU_KEYBD
0x0002 MU_BUTTON
0x0004 MU_M1
0x0008 MU_M2
0x0010 MU_MESAG
0x0020 MU_TIMER

bclk — The number of times the application is waiting for the mouse button to enter or leave a particular state within a preset time.

Use the high order bit to designate whether you are entering or leaving a state. The AES interprets the remaining 7 bits as the number of times. Bit values for the high bit are assigned as follows:

0 waiting to enter the specified state
1 waiting to leave the specified state

bmsk — Mouse buttons for which the application is waiting. The AES can theoretically support 16 mouse buttons. In bmsk, bst, and pmb, the following bits represent the buttons:

0x0001 button on left
0x0002 second button from left
0x0004 third button from left, etc.

The button specification can specify multiple buttons. For example, enter 0x0003 to specify the leftmost and second from left buttons. The event returns when

the button state occurs on either button. Note that you cannot specify multibutton states.

bst — Button state for which the application is waiting. These parameters use the following bit settings:

0 button up
1 button down

m1flags — Flags for the call:

0x0000 return on entry
0x0001 return on exit

m1x — X-coordinate of the mouse rectangle

m1y — Y-coordinate of the mouse rectangle

m1w — Width of the mouse rectangle in pixels

m1h — Height of the mouse rectangle in pixels

m2flags — Flags for the call:

0x0000 return on entry
0x0001 return on exit

m2x — X-coordinate of the mouse rectangle

m2y — Y-coordinate of the mouse rectangle

m2w — Width of the mouse rectangle in pixels

m2h — Height of the mouse rectangle in pixels

tlc — LOW WORD of a LONG value that, combined with thc, equals the length of the time interval in milliseconds

thc — HIGH WORD of a LONG value that, combined with tlc, equals the length of the time interval in milliseconds

mepbuff — Address of the buffer where the message will be placed. Its size must be 16 bytes.

Output Arguments

retval — Event(s) in the flags argument that actually occurred

This call uses the following bit settings:

0x0001 MU_KEYBD
0x0002 MU_BUTTON
0x0004 MU_M1
0x0008 MU_M2
0x0010 MU_MESAG
0x0020 MU_TIMER

pmx — X-coordinate of the mouse pointer when the user event occurred

pmy — Y-coordinate of the mouse pointer when the user event occurred

pmb — Mouse button state when the user event occurred The following bits represent the buttons:

0x0001 button on left
0x0002 second button from left
0x0004 third button from left, etc.

This parameter uses the following bit settings:

0 button up
1 button down

pks — State of the keyboard's right-Shift, left-Shift, Ctrl, and Alt keys when the user event occurred

The following bits represent the keys:

0x0001 right-Shift
0x0002 left-Shift
0x0004 Ctrl
0x0008 Alt

This parameter uses the following bit settings:

0 key up
1 key down

pkr Keyboard code as defined by the GEM VDI Reference Guide

pbr Number of mouse clicks that occurred

Sample Call to C Language Binding

```
WORD   evnt_multi();
WORD   retval;
UWORD  flags, bclk, bmsk, bst, m1flags, m1x, m1y, m1w, m1h,
       m2flags, m2x, m2y, m2w, m2h, tlc, thc, pmx, pmy, pmb, pks,
       pkr, pbr;
LONG   mepbuff;

retval = evnt_multi(flags, bclk, bmsk, bst, m1flags, m1x, m1y, m1w,
       m1h, m2flags, m2x, m2y, m2w, m2h, mepbuff, tlc, thc,
       &pmx, &pmy, &pmb, &pks, &pkr, &pbr);
```

Parameter Block Binding

Control	Input	Output
control(0) = 25	int_in(0) = flags	int_out(0) = retval
control(1) = 16	int_in(1) = bclk	int_out(1) = pmx
control(2) = 7	int_in(2) = bmsk	int_out(2) = pmy
control(3) = 1	int_in(3) = bst	int_out(3) = pmb
control(4) = 0	int_in(4) = m1flags	int_out(4) = pks
	int_in(5) = m1x	int_out(5) = pkr
	int_in(6) = m1y	int_out(6) = pbr
	int_in(7) = m1w	
	int_in(8) = m1h	
	int_in(9) = m2flags	
	int_in(10) = m2x	
	int_in(11) = m2y	
	int_in(12) = m2w	
	int_in(13) = m2h	
	int_in(14) = tlc	
	int_in(15) = thc	
	addr_in(0) = mepbuff	

EVNT_DCLICK

Gets the current setting of the mouse button's double-click speed or sets a new double-click speed for the mouse button.

Input Arguments

rate — New double-click speed the user has selected. This parameter has integer values from 0 (zero) to 4 that correspond to the SLOW-2-3-4-FAST settings of the selection buttons in the GEM Desktop's SET PREFERENCES dialog.

setit — Purpose of the call. If the value of setit is 0, EVNT_DCLICK disregards the rate value in the call.

1 set a new double-click speed
0 get the current double-click speed

Output Arguments

retval — Double-click speed, either newly set (setit = 1) or already existing (setit = 0). This parameter uses the same integer values as rate.

Sample Call to C Language Binding

```
WORD   ev_dclick();
WORD   retval, rate, setit;
retval = evnt_dclick(rate, setit);
```

Parameter Block Binding

Control	Input	Output
control(0) = 26	int_in(0) = rate	int_out(0) = retval
control(1) = 2	int_in(1) = setit	
control(2) = 1		
control(3) = 0		
control(4) = 0		

End of Section 4

MENU LIBRARY

Menus represent groups of options a user can choose within an application. Menus commonly appear as some form of text list.

Each GEM application defines its own menus. Menus are created using the GEM Resource Construction Set, documented in the GEM Programmer's Utilities Guide. When an application is active (controls the keyboard and mouse), the AES displays the titles of its menus in the menu bar at the top of the screen.

To select a menu, the user places the mouse form over the menu title in the menu bar. This causes the menu to drop down. The menu appears in a rectangle below the menu bar and remains visible until the user clicks the mouse button.

The standard menu item is a text string that names the menu command. The text string can also contain a key combination that produces the same result as clicking on the menu item. The user can press the keys instead of displaying the menu and choosing an item. The character appears on the menu to identify this shortcut.

A menu item can contain a non-text object such as a fill pattern or icon, and space for a any type of check mark to the left of the menu item. A check mark indicates that a certain condition is in effect. For example, in a menu of text fonts, a check mark next to the name of a font indicates that user-entered text will appear in that font.

Depending on the current state of the application, menu items can appear in either of two states: enabled (can be chosen) or disabled (cannot be chosen). Menu items are enabled only when choosing them is meaningful to the application. For example, the Desktop File Menu command Open is enabled if the user has selected an icon, but is disabled if the user has not selected an icon.

The Menu Library displays enabled items in standard character brightness; it displays disabled items as dimmed characters.

Responsibility for the user's interaction with menus is shared by the Screen Manager and the Menu Library.

The application uses a Menu Library call to display its menu bar, and it uses Menu Library calls to enable or disable menu items and to display check marks in a menu. Check marks can also exist as defaults from the Resource Construction Set.

The application makes a RSRC_LOAD call to bring menu data into memory, a MENU_BAR call to display the menu, and then waits for a message from its message pipe. If the user touches a menu title with the mouse form, the Screen Manager does the following:

- Highlights the menu title by changing it to reverse video.
- Displays the menu items in a rectangle below the title.

As the user moves the mouse form up and down the menu, the Screen Manager uses reverse video to highlight each enabled item as the mouse form touches it. The item remains highlighted as long as the mouse form is in contact with it.

To choose a menu item, the user clicks the mouse button while the mouse form is over an enabled item. The Screen Manager removes the drop-down portion of the menu from the screen and writes a message to the pipe. The application reads the message and acts accordingly.

A menu remains visible until the user clicks the mouse button.

When the chosen action has been performed, the calling application makes a Menu Library call to change the menu title back to its normal state.

If the user chooses a menu item by using the keyboard shortcut described previously, the application makes a Menu Library call both to highlight the menu title and to return it to its normal state. Section 2.10 describes keyboard menu selection in greater detail.

If the user moves the mouse form outside the menu rectangle, the Screen Manager returns the currently highlighted item (if any) to normal video. If the user moves the mouse form back into the rectangle, the Screen Manager again highlights enabled items as the mouse form touches them.

If the user clicks the mouse button outside the rectangle, the Screen Manager removes the drop-down portion of the menu from the

screen. No item is chosen, and no message is written to the pipe. To redisplay the menu, the user must move the mouse form back to the menu bar and select the menu title.

The Menu Library has two additional special functions:

1. It supports context-sensitive text in menus. An application can change the wording of its menu items depending on the application's current state.
2. Desk accessories use a Menu Library call to make their names appear on the Desk Menu, which is where the user starts them.

The Menu Library offers distinct advantages to both programmer and user:

- The programmer can create menus that meet the unique requirements of individual applications.
- The programmer does not have to be concerned with manipulating the interaction between menu and mouse.
- The programmer can modify menus and/or menu items in an efficient and timely manner.
- The user can expect all AES application menus to be familiar, both in appearance and function.

5.1 Using the Menu Library

The Menu Library is intended to relieve an application of the overhead of handling the interaction between mouse and menu. The Menu Library does the following:

- Displays the appropriate menu bar for each active application
- Enables and disables menu items
- Displays check marks in menus
- Returns a highlighted menu title to its normal state
- Displays context-sensitive menu text
- Displays a desk accessory's name on the Desk Menu

An application need only do the following:

1. Create a menu object tree. (The data for each menu is contained in an object structure, described in Section 6.2. The current state of the application determines whether a check mark appears in the menu and whether an item is enabled.)
2. Add the menu object tree to a resource file.
3. Load the menu object tree into memory, using the Resource Library's RSRC_LOAD call.
4. Call the MENU_BAR routine to have the Menu Library display the menu titles across the top of the screen.

After the application has completed the above steps, the menu titles are visible in the menu bar, and the individual menus are ready for user interaction.

The application's major task is to load the menu resource file. The information in the resource file determines the menu title's location on the menu bar and the location of the menu rectangle below the menu title.

When the user chooses an item, the Screen Manager writes a message to the pipe. Control then returns to the application, which must read the pipe.

The pipe message contains the following:

- A code indicating that it is a menu message
- The object index of the menu title selected
- The object index of the menu item chosen

(If the user does not choose an item, the Screen Manager does not write a message to the pipe.)

After processing the chosen item, the application makes a Menu Library call to return the menu title to normal video and wait for the next message to come through the message pipe.

5.2 Menu Library Routines

The Menu Library contains the following routines:

MENU_BAR	Displays or erases the menu bar.
MENU_ICHECK	Displays or erases a check mark next to a menu item.
MENU_IENABLE	Displays an enabled item in normal brightness and a disabled item in dimmed characters.
MENU_TNORMAL	Displays menu title in normal or reverse video.
MENU_TEXT	Changes the text of a menu item.
MENU_REGISTER	Lets a desk accessory set a text string on the Desk Menu and obtain a desk accessory identifier.
MENU_UNREGISTER	Lets a desk accessory remove its title from the Desk Menu.

MENU_BAR

Displays or erases the application's menu bar.

The application should always call MENU_BAR to erase the menu bar prior to its APPL_EXIT call.

Input Arguments

showit — A code for whether the application displays the menu bar.

- 0 erase the menu bar
- 1 display the menu bar

tree — Address of the object tree that forms this menu.

Output Arguments

retval — A coded return message:

- 0 error
- n no error

Sample Call to C Language Binding

```
WORD   menu_bar();
WORD   retval, showit;
LONG   tree;

retval = menu_bar(tree, showit);
```

Parameter Block Binding

Control	Input	Output
control(0) = 30	int_in(0) = showit	int_out(0) = retval
control(1) = 1	addr_in(0) = tree	
control(2) = 1		
control(3) = 1		
control(4) = 0		

MENU_ICHECK

Displays or erases a check mark next to a menu item.

Input Arguments

itemnum — An object index that uniquely identifies this menu item.

checkit — A code for whether the application displays a check mark next to the menu item identified by itemnum.

- 0 do not display a check mark, or if a check mark is visible, erase it
- 1 display a check mark

tree — The address of the object tree that forms this menu.

Output Arguments

retval — A coded return message:

- 0 error
- n no error

Sample Call to C Language Binding

```
WORD   menu_icheck();
WORD   retval, itemnum, checkit;
LONG   tree;

retval = menu_icheck(tree, itemnum, checkit);
```

Parameter Block Binding

Control	Input	Output
control(0) = 31	int_in(0) = itemnum	int_out(0) = retval
control(1) = 2	int_in(1) = checkit	
control(2) = 1		
control(3) = 1	addr_in(0) = tree	
control(4) = 0		

MENU_IENABLE

Enables or disables a menu item.

Input Arguments

itemnum — An object index that uniquely identifies this menu item. Index can point to a menu title as well as a menu item.

enableit — A code for how the application displays a menu item. The high order bit indicates whether the object designated in itemnum is a menu title (set to 1) or a menu item (set to 0).

- 0 disabled (dimmed characters)
- 1 enabled (normal brightness)

tree — The address of the object tree that forms this menu.

Output Arguments

retval — A coded return message:

- 0 error
- n no error

Sample Call to C Language Binding

```
WORD    menu_ienable();
WORD    retval, itemnum, enableit;
LONG    tree;

retval = menu_ienable(tree, itemnum, enableit);
```

Parameter Block Binding

Control	Input	Output
control(0) = 32	int_in(0) = itemnum	int_out(0) = retval
control(1) = 2	int_in(1) = enableit	
control(2) = 1		
control(3) = 1	addr_in(0) = tree	
control(4) = 0		

MENU_TNORMAL

Displays a menu title in normal or reverse video.

Input Arguments

titlenum — An object index unique to this application that identifies this menu.

normalit — A code for whether the application displays the menu title in normal or reverse video.

0 reverse video
1 normal video

tree — The address of the object tree that forms this menu.

Output Arguments

retval — A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   menu_tnormal();
WORD   retval, titlenum, normalit;
LONG   tree;

retval = menu_tnormal(tree, titlenum, normalit);
```

Parameter Block Binding

Control	Input	Output
control(0) = 33	int_in(0) = titlenum	int_out(0) = retval
control(1) =	int_in(1) = normalit	
control(2) =		
control(3) =	addr_in(0) = tree	
control(4) =		

MENU_TEXT

Changes the text of a menu item.

This routine lets GEM AES support context-sensitive menus. For example, a word processing application that lets the user turn the character insert on and off can have a menu item reading "Insert On" or "Insert Off", depending on the current state of the insert.

You must call MENU_BAR after MENU_TEXT to display the new text.

Input Arguments

inum — An object index that uniquely identifies this menu item.

ptext — The address of the new text string for this menu item. This text string should be no longer than the one it is replacing in the menu object tree structure.

tree — The address of the object tree that forms this menu.

To designate a desk accessory, set the high order word to 0 and put the pid value in the low order word. pid is the desk accessory's ap_id, returned by the MENU_REGISTER function.

Output Arguments

retval — A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   menu_text();
WORD   retval, inum;
LONG   tree, ptext;

retval = menu_text(tree, inum, ptext);
```

Parameter Block Binding

Control	Input	Output
control(0) = 34	int_in(0) = inum	int_out(0) = retval
control(1) = 1		
control(2) = 1	addr_in(0) = tree	
control(3) = 2	addr_in(1) = ptext	
control(4) = 0		

MENU_REGISTER

Places a desk accessory's menu item string on the Desk Menu and returns the accessory's menu item identifier.

Input Arguments

pid — The desk accessory's process identifier. This value is the ap_id returned by the desk accessory's APPL_INIT call.

pstr — The address of the desk accessory's Desk Menu text string.

Output Arguments

retval — The desk accessory's menu item identifier, a value ranging from 0 (zero) to 5.

-1 no room on the Desk Menu for this item

Sample Call to C Language Binding

```
WORD   menu_register();
WORD   retval, pid;
LONG   pstr;

retval = menu_register(pid, pstr);
```

Parameter Block Binding

Control	Input	Output
control(0) = 35	int_in(0)=pid	int_out(0)=retval
control(1) = 1		
control(2) = 1	addr_in(0)=pstr	
control(3) = 1		
control(4) = 0		

MENU_UNREGISTER

Deletes the desk accessory's title from the Desk Menu. This function is only available to desk accessories.

Input Arguments

mid — The desk accessory's menu item identifier. Normally, this is the same as the MENU_REGISTER int_out(0) argument. Enter −1 to delete the menu item of the currently running process.

Output Arguments

retval — A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   menu_unregister();
WORD   retval, mid;

retval = menu_unregister(mid);
```

Parameter Block Binding

Control	Input	Output
control(0) = 36	int_in(0)=mid	int_out(0)=retval
control(1) = 1		
control(2) = 1		
control(3) = 0		
control(4) = 0		

End of Section 5

OBJECT LIBRARY

An object is a collection of data describing something that appears on the screen. For example, GEM AES objects include boxes, characters, and icons. The AES defines several standard objects, and the Object Library provides routines to handle them.

The GEM Resource Construction Set, (GEM RCS), is a GEM application you use to create the objects you manipulate with the Object Library. The Resource Construction Set is described in the GEM Programmer's Utilities Guide.

An application uses the Object Library to set up and manipulate a tree structure of objects. Figure 6-1 shows a typical object tree.

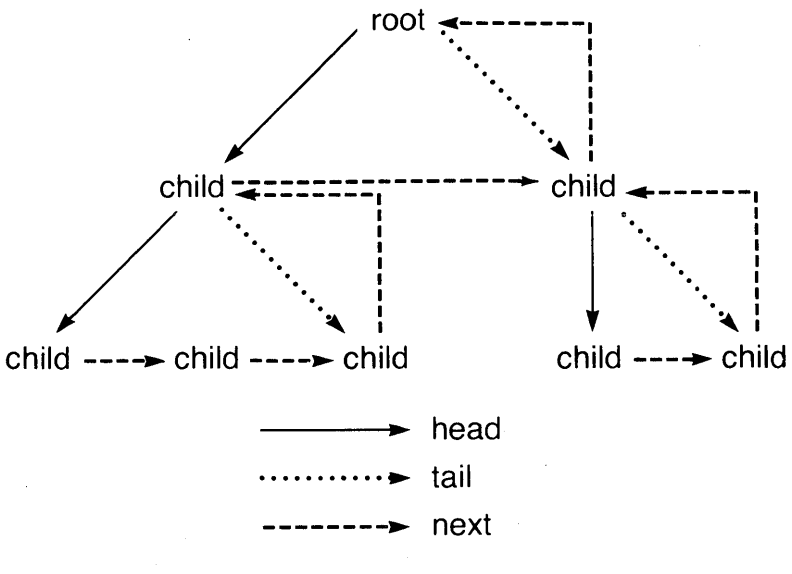


Figure 6-1. Object Tree

An object tree is an array of objects that are contiguous in memory. Starting with a root object, the tree consists of linked lists in which each child points to its next sibling and to its children, if either exists. The last child at each level points back to its parent. The links are actually indices of the objects, relative to the root of the Object Tree.

Figure 6-2 illustrates how the Object Library works. It shows a simple on-screen display: a box containing two boxes, one with text inside.

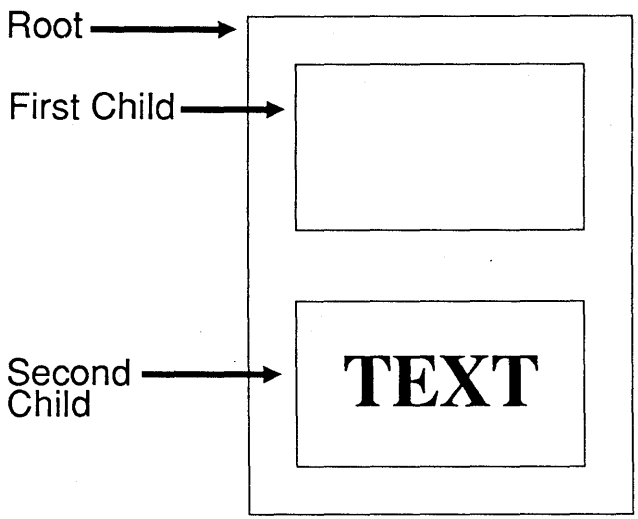


Figure 6-2. On-screen Display

The X-coordinate, Y-coordinate, width, and height values are always interpreted by the object funtions as raster coordinates. Do not use Normalized Device Coordinates (NDC) to specify these values.

The object tree that defines the display in Figure 6-2 contains three objects:

1. Root: The outer box. Its object type is G_BOX. Data for this object includes:

   - X- and Y-coordinates of the upper left corner, relative to the screen
   - Width and height
   - Thickness of the border
   - Foreground and background colors

2. First child: The empty inner box. Its object type is also G_BOX. Data for this object includes:

   - X- and Y-coordinates of the upper left corner, relative to the parent
   - Width and height
   - Thickness of the border
   - Foreground and background colors

3. Second child: The box with TEXT. Its object type is G_BOXTEXT. Data for this object includes:

   - X- and Y-coordinates of the upper left corner, relative to the parent
   - Width and height
   - Thickness of the border
   - Foreground and background colors
   - Text

To create an object tree an application can either build it in the Resource Constuction Set, or create it in the application's code. If you create the object tree in the code, you must make a separate call to the OBJC_ADD routine for each of the root's children.

Using the tree structure created by the OBJC_ADD calls and the data contained in the objects themselves, the Object Library draws the on-screen image with the OBJC_DRAW call.

An application can also load one or more complete object trees with the RSRC_LOAD call. In that case, all parent-child relationships have already been established.

Note: The parent object (in Figure 6-1 the root is the parent) always occupies screen space greater than or equal to that occupied by its children. In other words, the parent must visually contain its children. This is known as the "Visual Hierarchy Rule".

6.1 Object Library Data Structures

The Object Library contains the following data structures:

- OBJECT structure
- TEDINFO structure
- ICONBLK structure
- BITBLK structure
- APPLBLK structure
- PARMBLK structure

If an element of one of these data structures has a value of -1, it is either a nil index or a nil pointer.

The following sections describe these data structures.

6.2 OBJECT Structure

The OBJECT structure contains values that describe:

- The object
- Its relationship to the other objects in the tree
- Its location relative to its parent or, (in the case of the root object) the screen

There is an OBJECT structure for each object in a tree. Figure 6-3 shows the elements in the OBJECT structure. Table 6-1 describes each element.

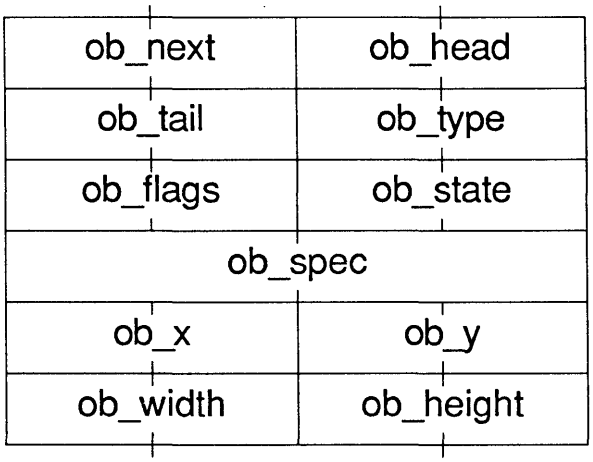


Figure 6-3. OBJECT Structure

Table 6-1. OBJECT Structure Elements

Element	Description
ob_next	WORD containing the index of the object's next sibling in the object tree array
ob_head	WORD containing the index of the first child: the head of the list of the object's children in the object tree array
ob_tail	WORD containing the index of the last child: the tail of the list of the object's children in the object tree array
ob_type	WORD containing the object type (defined in Section 6-2). The AES ignores the high byte of this WORD.
ob_flags	WORD containing the object flags (defined in Section 6.3.3)
ob_state	WORD containing the object state (defined in Section 6.3.4)
ob_spec	LONG value that depends on the value of ob_type. ob_spec can be a POINTER or any combination of WORD and/or BYTE values that add up to 32 bits.
ob_x	WORD containing the X-coordinate of the object relative to its parent or (for the root object) the screen
ob_y	WORD containing the Y-coordinate of the object relative to its parent or (for the root object) the screen
ob_width	WORD containing the width of the object in pixels
ob_height	WORD containing the height of the object in pixels

6.3 Predefined Values

The Object Library routines use predefined values for four elements in the OBJECT structure. The predefined values and their associated elements in the OBJECT structure are as follows:

- Object Types: ob_type
- Object Flags: ob_flags
- Object States: ob_state
- Object Colors: ob_spec

The following sections show the values and definitions associated with these categories.

6.3.1 Object Types

Object types are stored in the ob_type section of the OBJECT structure. All object types are graphic or bitmap object types. The value of ob_type directly affects the value of ob_spec. Table 6-2 shows the relationship between ob_type and ob_spec. The following define statements show the object type and its associated value:

```
#define G_BOX        20
#define G_TEXT       21
#define G_BOXTEXT    22
#define G_IMAGE      23
#define G_PROGDEF    24
#define G_IBOX       25
#define G_BUTTON     26
#define G_BOXCHAR    27
#define G_STRING     28
#define G_FTEXT      29
#define G_FBOXTEXT   30
#define G_ICON       31
#define G_TITLE      32
```

For object types G_BOX, G_IBOX, and G_BOXCHAR, the LONG value of ob_spec is broken into a LOW WORD and a HIGH WORD, as shown in Figure 6-4.

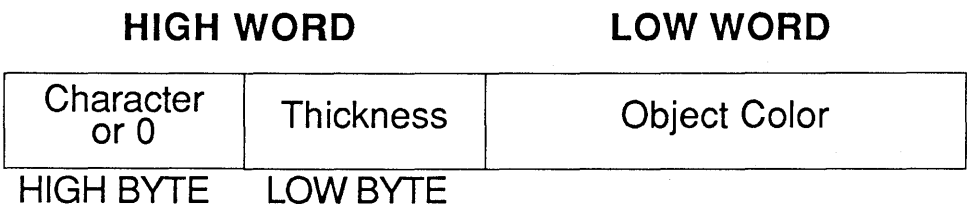


Figure 6-4. ob_spec for G_BOX, G_IBOX, and G_BOXCHAR

- The LOW WORD is the object color, as defined in Section 6.3.2.
- The HIGH WORD is broken into two bytes.
- For object types G_BOX and G_IBOX, the HIGH BYTE of the HIGH WORD equals 0 (zero).
- For G_BOXCHAR, the HIGH BYTE of the HIGH WORD is a character.
- For all three object types, the LOW BYTE of the HIGH WORD is the thickness of the object's border. This byte can have the following values:

00	No thickness
1 to 128	Inside thickness: inward from the object's edge
-1 to -127	Outside thickness: outward from the object's edge

Table 6-2. Object Types and ob-spec Values

ob_type	Description and ob_spec Value
G_BOX	A graphic box; its ob-spec value contains the object's color WORD and thickness of the border.
G_TEXT	Graphic text; POINTER to TEDINFO structure
G_BOXTEXT	Graphic box with graphic text; POINTER to TEDINFO structure
G_IMAGE	Graphic bit-image; POINTER to BITBLK structure
G_PROGDEF	Programmer-defined object; POINTER to APPLBLK structure.
G_IBOX	"Invisible" graphic box; ob_spec contains the object's color WORD and thickness. It has no fill pattern and no internal color. If its border has no thickness, it is invisible. If its border has thickness, it is an outline.
G_BUTTON	Graphic text object centered in a box; POINTER to a null-terminated text string
G_BOXCHAR	Graphic box containing a single text character; ob_spec contains the character, color WORD and thickness
G_STRING	Graphic text object; POINTER to a null-terminated text string
G_FTEXT	Formatted graphic text; POINTER to TEDINFO structure
G_FBOXTEXT	Graphic box containing formatted graphic text; POINTER to TEDINFO structure
G_ICON	Object that describes an icon; POINTER to ICONBLK structure
G_TITLE	Graphic text string used in menu titles; POINTER to a null-terminated text string.

6.3.2 Object Colors

Object colors are stored in the LOW WORD of the ob_spec element in the OBJECT structure and in the te_color element of the TEDINFO structure. A D preceding the name of the color (for example, DGREEN) indicates a dark shade of the color. The following define statements show the value associated with each object color:

```
#define WHITE      0
#define BLACK      1
#define RED        2
#define GREEN      3
#define BLUE       4
#define CYAN       5
#define YELLOW     6
#define MAGENTA    7
#define LGREY      8
#define DGREY      9
#define DRED       10
#define DGREEN     11
#define DBLUE      12
#define DCYAN      13
#define DYELLOW    14
#define DMAGENTA   15
```

Table 6-3 describes the components of the object color WORD.

Figure 6-5 shows the components of the object color WORD.

Table 6-3. Object Color WORD

Bits	Description
15 thru 12	Border color, with values from 0 to 15
11 thru 8	Text color, with values from 0 to 15
7	Writing mode. (Transparent and replace mode are defined in the GEM VDI Reference Guide).
	0 transparent mode 1 replace mode
6 thru 4	Fill pattern, with values from 0 to 7
	0 hollow fill 7 solid fill 1 - 6 dither patterns of increasing darkness
3 thru 0	Inside color, with values from 0 to 15

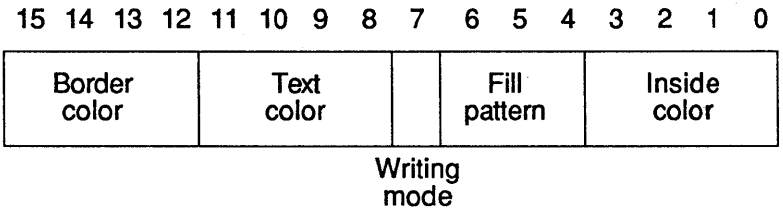


Figure 6-5. Object Color WORD

6.3.3 Object Flags

Object flags are stored as a bit vector in the ob_flags element of the OBJECT structure. Each bit in the ob_flags WORD is significant. Undefined bits should be set to zero.

The following define statements show the object flags and their associated values:

```
#define NONE          0x0000
#define SELECTABLE    0x0001
#define DEFAULT       0x0002
#define EXIT          0x0004
#define EDITABLE      0x0008
#define RBUTTON       0x0010
#define LASTOB        0x0020
#define TOUCHEXIT     0x0040
#define HIDETREE      0x0080
#define INDIRECT      0x0100
```

Table 6-4. Object Flags

Flag	Indication
SELECTABLE	The user can select the object.
DEFAULT	The Form Library examines the object if the user enters a Return/Enter. No more than one object in a form can be flagged DEFAULT. This object is usually an exit button, which lets the user enter a Return/Enter to exit the form without using the mouse.
EXIT	The Form Library returns control to the caller after the exit condition is satisfied (when the user selects the object by clicking on it).
EDITABLE	An object is editable by the user in some way.
RBUTTON	An object called a radio button. Radio buttons appear in groups of two or more, only one of which may be selected at a given time. When the user selects a button, the currently selected button is automatically de-selected. All radio buttons in a group must have the same parent.
LASTOB	An object is the last object in the object tree.
TOUCHEXIT	The Form Library returns control to the caller after the exit condition is satisfied (when the user presses the mouse button while the pointer is over the object).
HIDETREE	Makes a subtree invisible. When the application makes an OBJC_DRAW or OBJC_FIND call, the Object Library does not draw or find the object or any of its children.
INDIRECT	The value in ob_spec is a pointer to the actual value of ob_spec.

6.3.4 Object States

Object states determine how the OBJC_DRAW routine draws objects. Object states are stored as a bit vector in the ob_state element of the OBJECT structure.

The following define statements show the object states and their associated values:

```
#define NORMAL      0x0000
#define SELECTED    0x0001
#define CROSSED     0x0002
#define CHECKED     0x0004
#define DISABLED    0x0008
#define OUTLINED    0x0010
#define SHADOWED    0x0020
#define DRAW3D      0x0040
#define WHITEBAK    0x0080
```

Table 6-5. Object States

State	Indication
NORMAL	The object is drawn in normal foreground-background colors.
SELECTED	The object is highlighted by being drawn with its foreground and background colors reversed.
CROSSED	An "X" is drawn in the object. The object must be a box.
CHECKED	The object (typically one containing text) is drawn with a check mark.
DISABLED	The object (typically one containing text) is drawn faintly.
OUTLINED	An outline is drawn around a box object. This state is used for dialog boxes.
SHADOWED	The object (usually a box) is drawn with a drop shadow.
DRAW3D	Applies to ICONBLK structure only. When the DRAW3D and SELECTED bits are set, the icon mask is drawn three times: once one pixel up and to the left of the location specified, once at the location specified, and once one pixel down and to the right of the location specified.
WHITEBAK	Applies to ICONBLK structure only. When the WHITEBAK bit is set and the background color is white, the icon mask and rectangle outlining the icon text are not drawn.

6.4 TEDINFO Structure

The TEDINFO structure lets a user edit formatted text. The object types G_TEXT, G_BOXTEXT, G_FTEXT, and G_FBOXTEXT use their ob_spec pointers to point to TEDINFO structures. Figure 6-6 shows the TEDINFO data structure. Table 6-6 describes the elements in the TEDINFO structure.

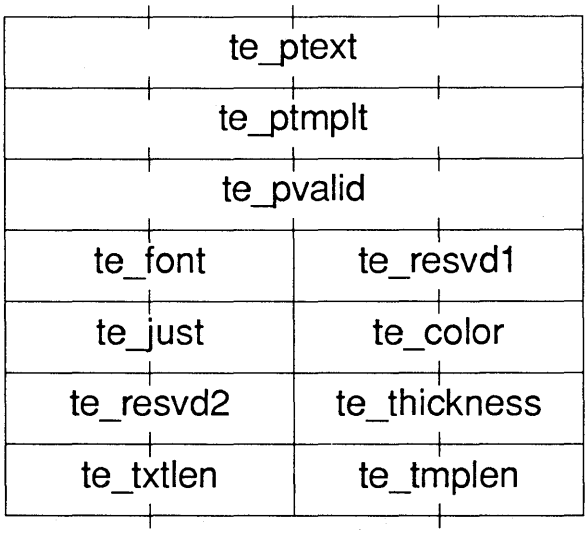


Figure 6-6. TEDINFO Structure

Table 6-6. TEDINFO Structure Elements

Element	Description
te_ptext	POINTER to the actual text string. If the first text character is "@", the field is blank, and the application can use any characters for the remaining character positions in the field. For example, a te_ptext string "@xyzpdq" is seven blank spaces. To set the cursor to the left of the field the application needs to set the first byte of te_ptext to null before displaying the object. For example, LBSET(te_ptext,'\0'); The text string is merged with the template pointed to by te_ptmplt before it is displayed.
te_ptmplt	POINTER to a text string template for any further data entry. The editable portion of the field is represented by underscores.
te_pvalid	POINTER to a text string containing characters that validate any entered text. 9 allow only digits 0 - 9 A allow only uppercase A - Z, plus space a allow upper- and lowercase A - Z, plus space N allow 0 - 9 and uppercase A - Z, plus space n allow 0 - 9 and upper- and lowercase A - Z, plus space F allow all valid filename characters, plus ? * : P allow all valid path name characters, plus \ : ? * p allow all valid path name characters, plus \ : X allow anything

Table 6-6. Cont'd

Element	Description
te_font	WORD identifying the font used to draw the text. 3 system font: used in menus, dialogs, etc. 5 small font: used in icons
te_resvd1	Reserved for future use.
te_just	WORD identifying the type of text justification desired. 0 left-justified 1 right-justified 2 centered
te_color	WORD identifying the color and pattern of box-type objects. See Section 6.3.2, Object Colors.
te_resvd2	Reserved for future use.
te_thickness	WORD containing the thickness in pixels of the border of the text box. This WORD can have the following values: 00 no thickness 1 to 128 inside thickness: inward from the object's edge -1 to -127 outside thickness: outward from the object's edge
te_txtlen	WORD containing the length of the string pointed to by te_ptext.
te_tmplen	WORD containing the length of the string pointed to by te_ptmplt.

The following example illustrates how the TEDINFO structure works:

- te_ptext is a string of raw data for a date. Its value is "061386".
- te_ptmplt, also a string, is a template that shows how to display the data in te_ptext. Its value is "Enter Date: __/__/__".
- te_pvalid is a string of input validation characters. Its value is "999999".
- The editable text facility merges all the above data into one string, "Enter Date: 06/13/86".
- If the user types "1004", the string becomes "Enter Date: 10/04/86".
- If the user presses the Backspace key after typing "1004", the string becomes "Enter Date: 10/0_/86".
- If te_ptext has no data or not enough data to fill out the template, the unfilled parts of the template show underscores. For example, if the user types "01" into an empty date field, it then reads "Enter Date: 01/__/__".

6.5 ICONBLK Structure

The Object Library uses the ICONBLK structure to hold the data that defines icons. The object type G_ICON points with its ob_spec pointer to the ICONBLK structure.

All X, Y, width, and height values for this structure are in pixels. Figure 6-7 shows the ICONBLK structure.

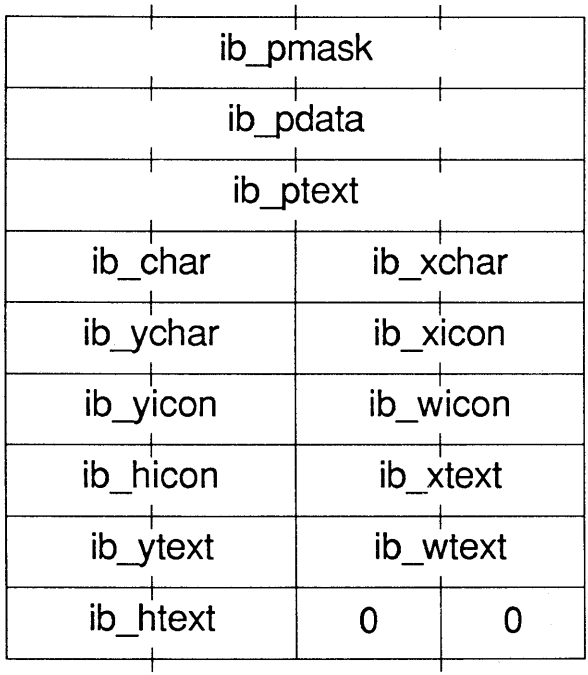


Figure 6-7. ICONBLK Structure

Table 6-7. ICONBLK Structure Elements

Element	Description
ib_pmask	POINTER to an array of WORDS representing the mask bit-image of the icon.
ib_pdata	POINTER to an array of WORDS representing the data bit-image of the icon.
ib_ptext	POINTER to the icon's text.
ib_char	WORD containing a character to be drawn in the icon (for example, the letter "A" on a floppy disk icon). The AES interprets the WORD as follows: bits 15 - 12 foreground color bits 11 - 8 background color bits 7 - 0 ASCII character code The character is displayed in the small system font.
ib_xchar	WORD containing the X-coordinate of ib_char, relative to the ib_xicon value.
ib_ychar	WORD containing the Y-coordinate of ib_char, relative to the ib_yicon value.
ib_xicon	WORD containing the X-coordinate of the icon, relative to the ob_x value in the Object Structure.
ib_yicon	WORD containing the Y-coordinate of the icon, relative to the ob_y value in the Object Structure.
ib_wicon	WORD containing the width of the icon in pixels. This value must be divisible by 16.
ib_hicon	WORD containing the height of the icon in pixels.

Table 6-7. Cont'd

Element	Description
ib_xtext	WORD containing the X-coordinate of the icon's text, relative to the ob_x value in the OBJECT Structure.
ib_ytext	WORD containing the Y-coordinate of the icon's text, relative to the ob_y value in the OBJECT Structure.
ib_wtext	WORD containing the width of a rectangle in which the icon's text will be centered.
ib_htext	WORD containing the height of the icon's text in pixels.

6.6 BITBLK Structure

The object type G_IMAGE uses the BITBLK structure to draw bit images such as cursor forms or icons. Figure 6-8 shows the BITBLK structure.

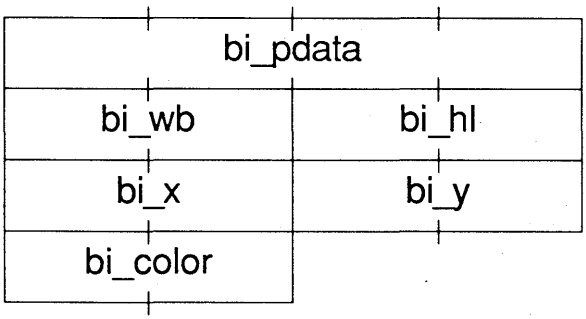


Figure 6-8. BITBLK Structure

Table 6-8. BITBLK Structure Elements

Element	Description
bi_pdata	POINTER to an array of WORDS containing the bit image.
bi_wb	WORD containing the width of the bi_pdata array in bytes. Because the bi_pdata array is made of WORDS, this value must be an even number.
bi_hl	WORD containing the height of the bit block in scan lines (pixels).
bi_x	WORD containing the source X in bit form, relative to the bi_pdata array.
bi_y	WORD containing the source Y in bit form, relative to the bi_pdata array.
bi_color	WORD containing the color the AES uses when displaying the bit-image. See Section 6.3.2 for the color values. The AES uses the color specified in bits 0 thru 3 as the foreground color of the bit image. Set the rest of the bits to zero. Set this word to -1 to blit the image in opaque mode rather than transparent mode.

6.7 APPLBLK Structure

The Object Library uses the APPLBLK Structure to locate and call an application-defined routine that will draw and/or change an object. The object type G_PROGDEF points with its ob_spec pointer to the APPLBLK structure.

You cannot make calls to the AES from application-defined routines. All display graphics must be executed using the VDI.

Note that the call must return the PARMBLK structure's pb_currstate value in a register.

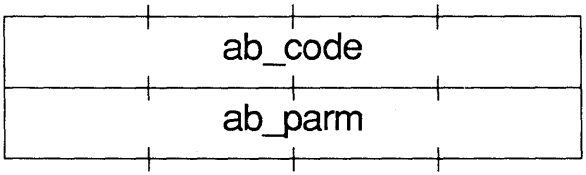


Figure 6-9. APPBLK Structure

Table 6-9. APPLBLK Structure Elements

Element	Description
ab_code	POINTER to the routine for drawing and/or changing the object
ab_parm	LONG value (optionally provided by the application) passed when the Object Library calls the application's object drawing/changing routine

6.8 PARMBLK Structure

The Object Library uses the PARMBLK structure to store information relevant to the application's drawing or changing an object.

When it calls the application's object drawing/changing routine (pointed to by ab_code), the Object Library provides a pointer to a PARMBLK.

The pointer to this structure is passed to the application-defined routine on the stack.

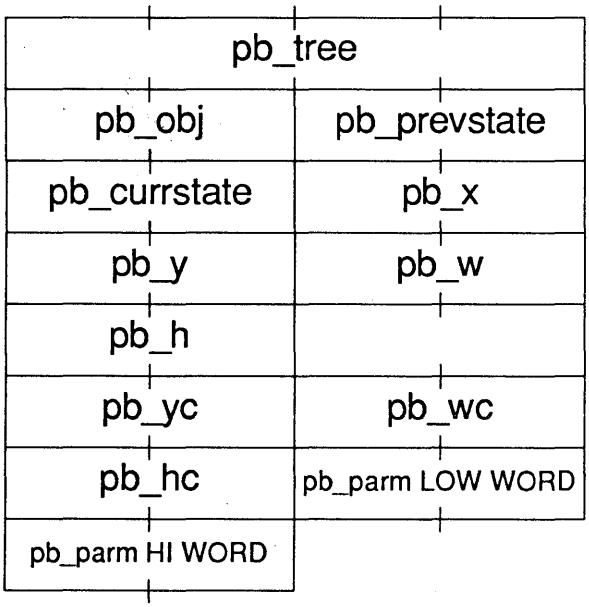


Figure 6-10. PARMBLK Structure

Table 6-10. PARMBLK Structure Elements

Element	Description
pb_tree	POINTER to the object tree that contains the application-defined object.
pb_obj	WORD containing the object index of the application-defined object.
pb_prevstate	WORD containing the old state of an object to be changed.
pb_currstate	WORD containing the changed (new) state of an object. If pb_prevstate and pb_currstate are the same, the application is drawing the object, not changing it.
pb_x	WORD containing the X-coordinate of a rectangle defining the location of the object on the physical screen.
pb_y	WORD containing the Y-coordinate of a rectangle defining the location of the object on the physical screen.
pb_w	WORD containing the width (in pixels) of a rectangle defining the size of the object on the physical screen.
pb_h	WORD containing the height (in pixels) of a rectangle defining the size of the object on the physical screen.
pb_xc	WORD containing the X-coordinate of the current clip rectangle on the physical screen.
pb_yc	WORD containing the Y-coordinate of the current clip rectangle on the physical screen.
pb_wc	WORD containing the width (in pixels) of the current clip rectangle on the physical screen.

Table 6-10. Cont'd

Element	Description
pb_hc	WORD containing the height (in pixels) of the current clip rectangle on the physical screen.
pb_parm	LONG value; identical to ab_parm in the APPLBLK structure. The Object Library passes this value to the application when it is time for the application to draw or change the object.

6.9 Object Library Routines

The Object Library uses the following routines:

OBJC_ADD	Adds an object to an object tree.
OBJC_DELETE	Deletes an object from an object tree.
OBJC_DRAW	Draws an object or object tree.
OBJC_FIND	Determines if the mouse is over an object.
OBJC_OFFSET	Computes an object's location relative to the screen.
OBJC_ORDER	Changes the order of an object within its tree.
OBJC_EDIT	Lets a user edit text in an object.
OBJC_CHANGE	Changes an object's state.

Note: A tree is an array of objects. In the Object Library routine descriptions, references to an object refer to the array index of the object in the tree.

OBJC_ADD

Adds an object to an object tree. The object to be added must be in the same array as the other objects in the tree. In creating an object tree, the application makes separate OBJC_ADD calls to establish the relationship of each child to its parent. For example, if the tree contains one parent with three children and another parent with two children, the tree requires a total of five OBJC_ADD calls. (This assumes that space for the five children has been set aside first in the Resource Construction Set.)

Input Arguments

parent	Object to whose list of children the child will be added
child	Object to add to parent's list of children
tree	Address of the object tree containing parent and child

Output Arguments

retval — A coded return message:

0 error
n the object was successfully added

Sample Call to C Language Binding

```
WORD   objc_add();
WORD   retval, parent, child;
LONG   tree;
retval = objc_add(tree, parent, child);
```

Parameter Block Binding

Control	Input	Output
control(0) = 40	int_in(0) = parent	int_out(0) = retval
control(1) = 2	int_in(1) = child	
control(2) = 1		
control(3) = 1	addr_in(0) = tree	
control(4) = 0		

OBJC_DELETE

Deletes an object from an object tree by unlinking it from its parent object. The object to be deleted must be in the same array as the other objects in the tree.

Input Arguments

delob	Object to be deleted
tree	Address of the object tree that contains the object

Output Arguments

retval A coded return message:

0 error
n the object was successfully deleted

Sample Call to C Language Binding

```
WORD   objc_delete();
WORD   retval, delob;
LONG   tree

retval = objc_delete(tree, delob);
```

Parameter Block Binding

Control	Input	Output
control(0) = 41	int_in(0) = delob	int_out(0) = retval
control(1) = 1		
control(2) = 1	addr_in(0) = tree	
control(3) = 1		
control(4) = 0		

OBJC_DRAW

Draws any object or objects in an object tree.

Each OBJC_DRAW call defines a new clip rectangle, and the Object Library draws only the parts of the object contained within the clip rectangle for that call. The clip rectangle coordinates must be described in raster coordinates. The clip rectangle is defined in the GEM VDI Reference Guide.

Input Arguments

drawob	Starting object on the tree indexed by tree
depth	Number of levels in the object tree to draw, starting from drawob 0 starting object only 1 first level children of starting object 2 second level children of starting object (etc.)
xc	X-coordinate of the clip rectangle
yc	Y-coordinate of the clip rectangle
wc	Width (in pixels) of the clip rectangle
hc	Height (in pixels) of the clip rectangle
tree	Address of the object tree that contains the object

Output Arguments

retval	A coded return message: 0 error n no error

Sample Call to C Language Binding

```
WORD   objc_draw();
WORD   retval, drawob, depth, xc, yc, wc, hc;
LONG   tree;

retval = objc_draw(tree, drawob, depth, xc, yc, wc, hc);
```

Parameter Block Binding

Control	Input	Output
control(0) = 42	int_in(0) = drawob	int_out(0) = retval
control(1) = 6	int_in(1) = depth	
control(2) = 1	int_in(2) = xc	
control(3) = 1	int_in(3) = yc	
control(4) = 0	int_in(4) = wc	
	int_in(5) = hc	
	addr_in(0) = tree	

OBJC_FIND

Finds an object under the mouse form.

The application supplies the X- and Y-coordinates of the mouse's position and a parameter that tells OBJC_FIND how far down the tree to search.

Input Arguments

startob	The object at which to start the search
depth	Number of levels in the object tree to search, starting from startob 0 starting object only 1 first-level children of starting object 2 second-level children of starting object (etc.)
mx	X-coordinate of the mouse's location
my	Y-coordinate of the mouse's location
tree	Address of the object tree containing the object identified by startob

Output Arguments

retval	The found object's number in the tree, ranging from 0 (zero) to n -1 = no object found

Sample Call to C Language Binding

```
WORD   objc_find();
WORD   retval, startob, depth, mx, my;
LONG   tree;

retval = objc_find(tree, startob, depth, mx, my);
```

Parameter Block Binding

Control	Input	Output
control(0) = 43	int_in(0) = startob	int_out(0) = retval
control(1) = 4	int_in(1) = depth	
control(2) = 1	int_in(2) = mx	
control(3) = 1	int_in(3) = my	
control(4) = 0		
	addr_in(0) = tree	

OBJC_OFFSET

Computes an object's X- and Y-coordinates relative to the screen.

Input Arguments

obj	The object whose location is being computed
tree	Address of the object tree containing the object identified by the obj argument

Output Arguments

retval	A coded return message: 0 error n no error
poffx	X-coordinate of obj relative to the screen
poffy	Y-coordinate of obj relative to the screen

Sample Call to C Language Binding

```
WORD   objc_offset();
WORD   retval, obj, poffx, poffy;
LONG   tree;


retval = objc_offset(tree, obj, &poffx, &poffy);
```

Parameter Block Binding

Control	Input	Output
control(0) = 44	int_in(0) = obj	int_out(0) = retval
control(1) = 1		int_out(1) = poffx
control(2) = 3	addr_in(0) = tree	int_out(2) = poffy
control(3) = 1		
control(4) = 0		

OBJC_ORDER

Moves an object to a new position in its parent's list of children. (For example, third child moves to second child's place).

Input Arguments

mov_obj — The object to be moved to a new position

newpos — The new position in which to put the object

- 0 on the bottom
- 1 one from the bottom
- 2 two from the bottom [etc.]
- -1 on top

Output Arguments

retval — A coded return message:

- 0 error
- n no error

tree — The address of the object tree that contains the object identified by mov_obj

Sample Call to C Language Binding

```
WORD   objc_order();
WORD   retval, mov_obj, newpos;
LONG   tree;

retval = objc_order(tree, mov_obj, newpos);
```

Parameter Block Binding

Control	Input	Output
control(0) = 45	int_in(0) = mov_obj	int_out(0) = retval
control(1) = 2	int_in(1) = newpos	
control(2) = 1		
control(3) = 1	addr_in(0) = tree	
control(4) = 0		

OBJC_EDIT

Lets the user edit text in an object.

The object must be of the type G_TEXT or G_BOXTEXT.

Input Arguments

obj — The object containing the text to be edited

inchar — The character input by the user

idx — The index of the next character position in the raw text string

kind — The OBJC_EDIT function to perform:

0 ED_START: reserved for future use

1 ED_INIT: combine values in te_ptext and te_ptmplt into a formatted string; turn on text cursor

2 ED_CHAR: validate typed characters against te_pvalid, update te_ptext, and display string

3 ED_END: turn off text cursor

See Section 6.4, TEDINFO Structure.

tree — The address of the object tree containing the object with the text to be edited

Output Arguments

retval — A coded return message:

0 error
n no error

idx — The updated index of the next character position in the raw text string

Sample Call to C Language Binding

```
WORD   objc_edit();
WORD   retval, obj, inchar, idx, kind;
LONG   tree

retval = objc_edit(tree, obj, inchar, &idx, kind);
```

Parameter Block Binding

Control	Input	Output
control(0) = 46	int_in(0) = obj	int_out(0) = retval
control(1) = 4	int_in(1) = inchar	int_out(1) = idx
control(2) = 2	int_in(2) = idx	
control(3) = 1	int_in(3) = kind	
control(4) = 0		
	addr_in(0) = tree	

OBJC_CHANGE

Changes an object's ob_state value. Refer to Section 6.3.4, Object States.

Each OBJC_CHANGE call defines a new clip rectangle, and the Object Library changes only the parts of the object contained within the clip rectangle for that call.

Input Arguments

drawob	The object to be changed
depth	Reserved; the value must be zero
xc	X-coordinate of the clip rectangle
yc	Y-coordinate of the clip rectangle
wc	Width (in pixels) of the clip rectangle
hc	Height (in pixels) of the clip rectangle
newstate	The ob_state value of the object
redraw	A code for whether to redraw the object 0 do not redraw the object 1 redraw the object
tree	Address of the object tree containing the object

Output Arguments

retval	A coded return message: 0 error n no error

Sample Call to C Language Binding

```
WORD   objc_change();
WORD   retval, drawob, depth, xc, yc, wc, hc, newstate, redraw;
LONG   tree;

retval=objc_change(tree,drawob,depth,xc,yc,wc,hc,newstate,redraw);
```

Parameter Block Binding

Control	Input	Output
control(0) = 47	int_in(0) = drawob	int_out(0) = retval
control(1) = 8	int_in(1) = ob_cresve	
control(2) = 1	int_in(2) = xc	
control(3) = 1	int_in(3) = yc	
control(4) = 0	int_in(4) = wc	
	int_in(5) = hc	
	int_in(6) = newstate	
	int_in(7) = redraw	
	addr_in(0) = tree	

End of Section 6

FORM LIBRARY

A form is a means of giving or gathering information. A form can appear on a piece of paper or on a computer screen. It can be a set of questions, often a list, to which a user responds by checking off boxes or writing text.

The Form Library displays forms almost exactly as they would appear on paper. For example, an application can display a form and then use the responses it receives to update an information database.

There are two ways to process forms:

- Use the FORM_DO function to process a form automatically. The AES collects the user's responses and stores them in the form's object tree.
- Use the FORM_KEYBD and FORM_BUTTON functions to maintain local control of form processing, using the Object Library to move from object to object.

The Form Library offers several advantages, including the following:

- The Form Library, and not the application, is responsible for the user's interaction with the form.
- The application is idle until the user has completed the form. When the user satisfies the form's exit condition, the application regains control and acts on the user's responses.
- The Form Library greatly simplifies the programmer's task by providing a consistent framework for interaction between the application and the user.

7.1 A Model Form

A typical form is the product survey illustrated in Figure 7-1. This kind of form contains several questions, to which the user responds by putting an "X" in the appropriate boxes or by writing a response.

```
1. Age (check one only):
        10-29          []
        30-49          []
        50-69          []
        over 70        []

2. Yearly Income (check one only):
        less than $10,000       []
        $10,000 - $29,999       []
        $30,000 - $49,999       []
        $50,000 - $69,999       []
        over $70,000            []

3. Activities You Enjoy (check all that apply):
        Water Skiing            []
        Hang Gliding            []
        Backpacking             []
        Bicycling               []
        SCUBA Diving            []
        Horseback Riding        []

4. Sporting Goods Store: _____________________________

                Address: _____________________________
```

Figure 7-1. Product Survey Form

The questions in this form require three different kinds of answers:

1. A check in a single box (questions 1 and 2)
2. A check in one or more boxes, or no check at all (question 3)
3. A written answer (question 4)

7.2 Responding to a Form

Forms in the AES are essentially the same as printed forms. For example, the AES uses the same three types of response that appear on the product survey. However, the AES uses the following terminology:

1. Radio buttons: the "check-one-only" type of response. If a user selects one button, all other buttons are automatically de-selected. Radio buttons are a "one of many" structure.

2. Check boxes: the "check-all-that-apply" type of response. The user can select one or more of the options, or none at all. Despite the name, a check mark does not necessarily appear in the box when the user selects it. Check boxes are an "any of many" structure.

3. Editable text: for all responses requiring text entry.

The AES needs one piece of information that does not appear explicitly on the product survey: notification that the user has completed the form. To provide this information, you designate at least one box as an exit button and set the object flag to EXIT. When the user selects that box, the Form Library completes its tasks and then passes control back to the application.

Many AES forms have two exit buttons, one labeled "OK" and the other labeled "Cancel". These buttons have the following functions:

OK	Tells the Form Library that the form is complete. The application can then act on the user's responses.
Cancel	Tells the Form Library to ignore any responses and to return control to the application. The environment remains the same as it was before the AES displayed the form.

A form's exit buttons do not have to be labeled "OK" and "Cancel". For example, a form can have an "OK" button alone, or it can have buttons labeled "Excellent", "Very Good", "Good", "Fair", "Poor". The labels on the exit button or buttons depend entirely on the application.

7.3 Dialog Boxes

A dialog, which is a special kind of form, provides a consistent framework for interaction between an application and a user when either of the following conditions exists:

- The application needs more information before it can carry out a command.
- An error or some other condition occurs that requires that the user be notified immediately.

Dialog boxes appear in the center of the screen. Each box contains text and one or more exit buttons.

Dialogs are stored on disk as resources, which lets an application programmer alter their content (for example, rewrite a message from English to German) without having to make changes to the application using them or to the Form Library itself.

7.4 Editable Text Fields

Many dialogs have editable text fields. The user can edit these fields with the following keys described in Table 7-1. The effect is continuous if the key is held down.

For up- and down-arrow, Tab, and Backtab, the cursor goes to the beginning of the field only if the field is empty. Otherwise, the cursor goes to the first open character position following the field's data string.

Table 7-1. Editing Keys

Key	Description
Backspace	Deletes the character to the left of the cursor. The cursor and any following text move one character space to the left.
Delete	Deletes the character following the cursor. The cursor does not move.
Escape	Clears all characters from the field.
Left and right-arrow	Moves the text cursor left and right within the field.
Down-arrow and Tab	Moves the cursor to the next field.
Up-arrow and Backtab	Moves the cursor to the previous field. To Backtab, the user presses the Shift and Tab keys together.
Return/Enter	Ends editing and terminates the dialog.

The Return/Enter key works this way only if one object in the form has been flagged as a DEFAULT object (see Section 6.3.3, Object Flags). If no object has been flagged as a DEFAULT object, the Form Library ignores Return/Enter keystrokes.

For example, in the Alert in Figure 7-2, the GEM Desktop application has flagged the "Cancel" button as the DEFAULT object. (The DEFAULT object is identified by its heavy border.) If the user presses the Return/Enter key intending to format a disk, the keystroke instead cancels the format request. The DEFAULT flag acts as an extra safety device, forcing the user to click the mouse pointer on "OK" in order to format the disk.

The user can also move through a field by typing a delimiter character that appears to the right of the cursor. (This delimiter character must be a character that is not allowed by the validation string te_pvalid, described in Section 6.4, TEDINFO Structure).

For example, the user might be entering a filename in the following field:

```
_ _ _ _ _ _ _ _ . _ _ _
```

The validation string for this field is FFFFFFFFFFF (all valid operating system filename characters, plus ?, *, and :). The period is not a valid character. If the user types:

```
f i l e .
```

the Form Library looks for a period in the field to the right of the cursor's position. Finding one, it moves the cursor one position past the period, filling all spaces between the text and the period with blanks. The user now sees the following field:

```
f i l e        .|_ _ _
```

Similarly, the user can type:

```
9 / 3 0 / 8 6
```

into the following date field:

```
_ _ / _ _ / _ _
```

and get the following field as a result:

```
9  / 3 0 / 8 6
```

7.5 Alerts

An alert is a special kind of dialog box that notifies the user a condition has arisen that requires immediate attention and, usually, action by the user. Alerts are the AES's and AES-based applications' means of handling error conditions.

Figure 7-2 shows a sample alert.

Figure 7-2. Sample GEM AES Alert

The text of the alert is shown below. The string uses square brackets to separate the components.

[3][Formatting will ERASE all|information on the disk in drive|A:. Click on OK only if you don't|mind losing this information.[Cancel|OK]

The components of the string are:

[Icon Number] [Message Text] [Exit Buttons]

Icon Number — A single character that identifies an icon (if any) that appears at the left side of the alert.

0 no icon
1 NOTE icon
2 WAIT icon
3 STOP icon

Message Text — A maximum of 200 ASCII characters for text. The alert can have five text lines, with no more than 40 characters on each line. In the string, the lines are separated by the logical OR symbol (|).

Exit Buttons — One, two, or three exit buttons, each containing no more than 20 text characters. In the string, the exit button text is separated by the logical OR symbol (|).

7.5.1 Error Boxes

An error box is a special kind of alert that reports operating system errors like "File Not Found." Operating system error codes are defined in your operating system technical manual.

Unlike the text string for other alerts, the text string for an error box is generated by the Form Library instead of the application.

7.6 Displaying a Form

To display a form, the application takes the following steps:

1. The application calls OBJC_DRAW, passing an index to the object tree for the form.
2. The Object Library displays the form in the application's window.
3. The application makes a call to the FORM_DO function, and the Form Library monitors the user's interaction with the form.
4. When the user selects an exit button, the Form Library returns control to the application. In general, the application identifies the object(s) that cause the Form Library to relinquish control.
5. After regaining control, the application must look at the form, determine if any changes took place, and decide on appropriate action.

7.7 Displaying a Dialog

In contrast to an application-defined form, which appears inside a window, a dialog sits on top of windows and desk accessories and does not have to be within a window's boundaries.

To display a dialog, the application takes the following steps:

1. It calls the RSRC_GADDR routine to get the address of the object tree that draws the dialog. This call is described in Section 12.
2. It calls the FORM_CENTER routine to center the dialog on the screen. (This call is optional.)

3. It calls the FORM_DIAL routine to reserve the part of the physical screen in which the dialog will appear. Nothing else can occupy the reserved part of the screen.
4. It calls the OBJC_DRAW routine to display the dialog. This call is described in Section 6.
5. It calls the FORM_DO routine to let the user interact with the dialog.
6. When the user satisfies the dialog's exit condition, by clicking on an exit button or pressing the Enter key, the application calls the FORM_DIAL routine to free the reserved screen space and to redraw the screen.

7.8 Displaying an Alert

To display an alert, the application calls the FORM_ALERT routine. The FORM_ALERT routine contains the following internal steps:

1. It constructs the object tree of the alert, based on the input string whose address is contained in FORM_ALERT.
2. It saves to the menu/alert buffer the screen space that will be taken over by the alert.
3. It calls the OBJC_DRAW routine to display the alert.
4. It calls the FORM_DO routine to let the user respond to the alert, and reports the user's exit button selection to the application.
5. After the user selects an exit button, it redraws the screen from the menu/alert buffer.

To display an error box, the application calls the FORM_ERROR routine. The input parameter for FORM_ERROR is an operating system error code, and its output parameter is a code that tells the application to retry or abandon the requested action. Otherwise, FORM_ERROR uses the same internal sequence as FORM_ALERT.

7.9 Form Library Routines

The Form Library contains the following routines:

FORM_DO	Causes the Form Library to monitor the user's interaction with a form.
FORM_DIAL	Reserves or frees the portion of the screen used for dialog boxes.
FORM_ALERT	Displays an alert box.
FORM_ERROR	Displays an error box.
FORM_CENTER	Centers a dialog box on the screen.
FORM_BUTTON	Processes mouse button input.
FORM_KEYBD	Processes keyboard input.

FORM_DO

Causes the Form Library to monitor the user's interaction with a form.

Input Arguments

start — The number of an object (which must be an editable text field) that the application wants active when the form is displayed. The application can pass in a value of 0 (zero) if the form does not contain editable text fields.

form — The address of the object tree that draws this form

Output Arguments

retval — The number of the object that caused the exit from the user's interaction with the form

Sample Call to C Language Binding

```
WORD   form_do();
WORD   retval, start;
LONG   form;

retval= form_do(form, start);
```

Parameter Block Binding

Control	Input	Output
control(0) = 50	int_in(0) = start	int_out(0) = retval
control(1) = 1		
control(2) = 1	addr_in(0) = form	
control(3) = 1		
control(4) = 0		

FORM_DIAL

Depending on the value in dtype, FORM_DIAL does one of the following:

- Reserves a portion of the physical screen for a dialog box
- Frees the reserved portion of the screen and redraws the screen

Input Arguments

dtype — FORM_DIAL action invoked by the current call:

- 0 FMD_START: Reserves screen space for the dialog box
- 3 FMD_FINISH: Frees screen space reserved by FMD_START; causes application to redraw screen

ix — Reserved. The application should set the following parameters to zero: ix, iy, iw, ih

iy — Reserved

iw — Reserved

ih — Reserved

x — X-coordinate of the box

y — Y-coordinate of the box

w — Width (in pixels) of the box

h — Height (in pixels) of the box

Output Arguments

retval — A coded return message:

- 0 error
- n no error

Sample Call to C Language Binding

```
WORD  form_dial();
WORD  dtype, ix, iy, iw, ih, x, y, w, h;

retval = form_dial(dtype, ix, iy, iw, ih, x, y, w, h);
```

Parameter Block Binding

Control	Input	Output
control(0) = 51	int_in(0) = dtype	int_out(0) = retval
control(1) = 9	int_in(1) = ix	
control(2) = 1	int_in(2) = iy	
control(3) = 0	int_in(3) = iw	
control(4) = 0	int_in(4) = ih	
	int_in(5) = x	
	int_in(6) = y	
	int_in(7) = w	
	int_in(8) = h	

FORM_ALERT

Displays an alert. Section 7.6 describes the complete sequence of calls internal to FORM_ALERT.

Input Arguments

defbut — The form's DEFAULT exit button (see Section 7.4).

- 0 no DEFAULT exit button
- 1 first exit button
- 2 second exit button
- 3 third exit button

string — The address of the string containing the alert

Output Arguments

retval — Number identifying the exit button selected by the user

- 1 first exit button in string
- 2 second exit button in string
- 3 third exit button in string

Sample Call to C Language Binding

```
WORD   form_alert();
WORD   retval, defbut;
LONG   astring;

retval = form_alert(defbut, astring);
```

Parameter Block Binding

Control	Input	Output
control(0) = 52	int_in(0) = defbut	int_out(0) = retval
control(1) = 1		
control(2) = 1	addr_in(0) = astring	
control(3) = 1		
control(4) = 0		

FORM_ERROR

Displays an error box.

Input Arguments

errnum The operating system error code

Output Arguments

retval A code that identifies the user's exit button selection:

1 first exit button in string
2 second exit button in string
3 third exit button in string

Sample Call to C Language Binding

```
WORD  form_error();
WORD  retval, errnum;

retval = form_error(errnum);
```

Parameter Block Binding

Control	Input	Output
control(0) = 53 control(1) = 1 control(2) = 1 control(3) = 0 control(4) = 0	int_in(0) = errnum	int_out(0) = retval

FORM_CENTER

Computes the location of the center of the screen for a given dialog, and writes these values into pcx, pcy, pcw, and pch.

Input Arguments

tree	Address of the object tree that describes the dialog

Output Arguments

retval	Reserved; value equals 1 (one)
pcx	X-coordinate of the centered object tree containing the dialog box
pcy	Y-coordinate of the centered object tree containing the dialog box
pcw	Width (in pixels) of the centered object tree containing the dialog box
pch	Height (in pixels) of the centered object tree containing the dialog box

Sample Call to C Language Binding

```
WORD   form_center();
WORD   retval, pcx, pcy, pcw, pch;
LONG   tree;

retval = form_center(tree, &pcx, &pcy, &pcw, &pch);
```

Parameter Block Binding

Control	Input	Output
control(0) = 54	addr_in(0) = tree	int_out(0) = retval
control(1) = 0		int_out(1) = pcx
control(2) = 5		int_out(2) = pcy
control(3) = 1		int_out(3) = pcw
control(4) = 0		int_out(4) = pch

FORM_KEYBD

Filters keyboard input for Tab, Backtab, Up, Down and Return/Enter characters while in a form, and selects the next object accordingly. The selected object is returned, with a code indicating whether or not the input was the Return/Enter character.

Input Arguments

obj — Current object receiving the keystroke (does not need to be first object in form tree)

thechar — Input character from keyboard

nxt_obj — Reserved; set to 0

form — Address of the object tree that draws this form

Output Arguments

retval — A coded return message:

- 0 The input character (thechar) was the Return/Enter key and the tree contains an object with its default bit set. On screen, the default box is shown selected.
- 1 The input character (thechar) was any key but a Return/Enter.

nxt_obj — Index of the next object selected by the input character

pchar — A coded return message:

- 0 The input character (thechar) was a Tab, Backtab, Up, Down, or Return/Enter.
- c (Any other character). The character is returned here.

Sample Call to C Language Binding

```
WORD    form_keybd();
WORD    retval, obj, nxt_obj, thechar, pnxt_obj, pchar;
LONG    form;

retval = form_keybd(form, obj, nxt_obj, thechar, &pnxt_obj, &pchar);
```

Parameter Block Binding

Control	Input	Output
control(0) = 55	int_in(0) = obj	int_out(0) = retval
control(1) = 3	int_in(1) = thechar	int_out(1) = pnxt_obj
control(2) = 3	int_in(2) = nxt_obj	int_out(2) = pchar
control(3) = 1		
control(4) = 0	addr_in(0) = form	

FORM_BUTTON

Processes input from a mouse button on a form, starting at the object specified. The routine returns when the user enters the specified number of clicks. The screen shows the next object selected. The selected object is returned with a value indicating whether the user selected an exit object.

Input Arguments

obj	Current object receiving the button click (does not need to be the first object in the form tree)
clks	Number of mouse button clicks
form	Address of the object tree that draws this form

Output Arguments

pnxt_obj	Index of next object in tree. This value is set to 0 (zero) if this object is hidden, disabled, or not editable. The high order bit of this value is set if clks is set to 2 and this object is a touchexit.
retval	A coded return message: 1 indicates object selected is not an exit object 0 indicates the next object selected has the exit flag bit or the touchexit flag bit set

Sample Call to C Language Binding

```
WORD   form_button();
WORD   retval, obj, clks, pnxt_obj;
LONG   form;

retval = form_button(form, obj, clks, &pnxt_obj);
```

Parameter Block Binding

Control	Input	Output
control(0) = 56	int_in(0) = obj	int_out(0) = retval
control(1) = 2	int_in(1) = clks	int_out(1) = pnxt_obj
control(2) = 2		
control(3) = 1	addr_in(0) = form	
control(4) = 0		

End of Section 7

GRAPHICS LIBRARY

In certain circumstances, a graphics application might need to manipulate a rectangular outline (a box drawn with lines) on the screen. The Graphics Library provides a set of routines for these manipulations. As a result, each application can make calls to a single library within GEM AES, without having to carry the routines in its own code.

Graphics Library functions are based on GEM VDI functions that are fully described in the GEM Virtual Device Interface Reference Guide. The AES runs on top of GEM VDI, and a graphics application should use GEM VDI for its graphics output. However, all graphics input is made directly through the AES.

Graphics Library routines also return the GEM VDI handle of the currently open screen workstation, change the mouse form to one of a predefined set or to a form defined by the application, and get information on the mouse and keyboard.

The boxes manipulated by the Graphics Library can be used for a variety of purposes. In the GEM Desktop application, for example, the GRAF_RUBBOX routine draws the box that appears when a user drags the mouse to select a group of icons.

All functions that take mouse input require use of the leftmost mouse button.

8.1 Graphics Library Routines

The Graphics Library contains the following routines:

GRAF_RUBBOX	Draws a "rubber" box that expands and contracts from a fixed point as the mouse moves.
GRAF_DRAGBOX	Moves a box, keeping the mouse pointer in the same position in the box.
GRAF_MBOX	Draws a moving box.
GRAF_WATCHBOX	Watches a box to see if the mouse pointer (with button down) is inside.
GRAF_SLIDEBOX	Keeps a sliding box inside its parent box.
GRAF_HANDLE	Returns a GEM VDI handle for the opened screen workstation that the AES libraries use.
GRAF_MOUSE	Lets an application change the mouse form to one of a predefined set or to an application-defined form.
GRAF_MKSTATE	Returns the current mouse location, mouse button state, and keyboard state.

GRAF_RUBBOX

Draws a "rubber box." The position of the box's upper left corner is fixed, but by dragging the lower right corner with the mouse pointer, the user can make the box larger or smaller. The call returns the rubber box's new size when the user releases the mouse button.

Input Arguments

xorigin	X-coordinate of the box
yorigin	Y-coordinate of the box
wmin	Box's smallest possible width in pixels
hmin	Box's smallest possible height in pixels

Output Arguments

retval	A coded return message: 0 error n no error
pwend	Width of the box when the user released the mouse button
phend	Height of the box when the user released the mouse button

Sample Call to C Language Binding

```
WORD   graf_rubbox();
WORD   retval, xorigin, yorigin, wmin, hmin, pwend, phend;

retval = graf_rubbox(xorigin, yorigin, wmin, hmin, &pwend, &phend);
```

Parameter Block Binding

Control	Input	Output
control(0) = 70	int_in(0) = xorigin	int_out(0) = retval
control(1) = 4	int_in(1) = yorigin	int_out(1) = pwend
control(2) = 3	int_in(2) = wmin	int_out(2) = phend
control(3) = 0	int_in(3) = hmin	
control(4) = 0		

GRAF_DRAGBOX

Lets a user drag a box within an application-defined boundary rectangle.

When the user presses the mouse button to begin dragging, the AES starts tracking the mouse's position. As the user drags, this call keeps the mouse pointer in a fixed position relative to the box's upper left corner.

Input Arguments

w	Width in pixels of the box being dragged
h	Height in pixels of the box being dragged
sx	Box's starting X-coordinate
sy	Box's starting Y-coordinate
xc	X-coordinate of the boundary rectangle
yc	Y-coordinate of the boundary rectangle
wc	Width in pixels of the boundary rectangle
hc	Height in pixels of the boundary rectangle

Output Arguments

retval	A coded return message: 0 error n no error
pdx	Box's X-coordinate when the user released the mouse button
pdy	Box's Y-coordinate when the user released the mouse button

Sample Call to C Language Binding

```
WORD   graf_dragbox();
WORD   retval, w, h, sx, sy, xc, yc, wc, hc, pdx, pdy;

retval = graf_dragbox(w, h, sx, sy, xc, wc, hc, &pdx, &pdy);
```

Parameter Block Binding

Control	Input	Output
control(0) = 71	int_in(0) = w	int_out(0) = retval
control(1) = 8	int_in(1) = h	int_out(1) = pdx
control(2) = 3	int_in(2) = sx	int_out(2) = pdy
control(3) = 0	int_in(3) = sy	
control(4) = 0	int_in(4) = xc	
	int_in(5) = yc	
	int_in(6) = wc	
	int_in(7) = hc	

GRAF_MBOX

Draws a box moving from one position to another. The box's size does not change.

Input Arguments

w	Box's width in pixels
h	Box's height in pixels
srcx	Box's X-coordinate, in its initial position
srcy	Box's Y-coordinate, in its initial position
dstx	Box's X-coordinate, in its final position
dsty	Box's Y-coordinate, in its final position

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   graf_mbox();
WORD   retval, w, h, srcx, srcy, dstx, dsty;

retval = graf_mbox(w, h, srcx, srcy, dstx, dsty);
```

Parameter Block Binding

Control	Input	Output
control(0) = 72	int_in(0) = w	int_out(0) = retval
control(1) = 6	int_in(1) = h	
control(2) = 1	int_in(2) = srcx	
control(3) = 0	int_in(3) = srcy	
control(4) = 0	int_in(4) = dstx	
	int_in(5) = dsty	

GRAF_WATCHBOX

Tracks the mouse pointer in and out of a predefined box.

The box's object state changes as the mouse pointer enters and leaves the box. The application makes this call only when the mouse button is being held down, and the routine returns a value only when the user releases the mouse button.

The box is contained in an object tree. The input variables for instate and outstate are defined in Section 6.3.4.

Input Arguments

obj — Index of the object in the tree

instate — Box's state when the mouse pointer (with button down) is inside it

0x0000	NORMAL
0x0001	SELECTED
0x0002	CROSSED
0x0004	CHECKED
0x0008	DISABLED
0x0010	OUTLINED
0x0020	SHADOWED

outstate — Box's state when the mouse pointer (with button down) is outside it

0x0000	NORMAL
0x0001	SELECTED
0x0002	CROSSED
0x0004	CHECKED
0x0008	DISABLED
0x0010	OUTLINED
0x0020	SHADOWED

tree — Address of the object tree containing the box

Output Arguments

retval — Mouse pointer's position when the button was released

0 outside the box
1 inside the box

Sample Call to C Language Binding

```
WORD   graf_watchbox();
WORD   retval, obj;
UWORD  instate, outstate;
LONG   tree;

retval = graf_watchbox(tree, obj, instate, outstate);
```

Parameter Block Binding

Control	Input	Output
control(0) = 75	int_in(0) = [reserved]	int_out(0) = retval
control(1) = 4	int_in(1) = obj	
control(2) = 1	int_in(2) = instate	
control(3) = 1	int_in(3) = outstate	
control(4) = 0		
	addr_in(0) = tree	

GRAF_SLIDEBOX

Tracks a sliding box inside a parent box.

Mouse movement causes the sliding box to move, and the parent box defines the sliding box's range of motion.

An application makes this call only when the mouse button is being held down, and the routine returns a value only when the user releases the mouse button.

Both boxes (slider and parent) are contained in an object tree. The return value is a number that indicates the slider's relative position inside the parent box.

Input Arguments

parent — Index of the parent in the tree.

obj — Index of the object (the slider) in the tree.

isvert — A code for the direction of the slider's movement.

0 horizontal
1 vertical

tree — Address of the object tree containing slider and parent.

Output Arguments

retval — Position of the center of the slider relative to its parent. The value can range from 0 to 1000.

if isvert = 0 0 = left
1000 = right
if isvert = 1 0 = top
1000 = bottom

Sample Call to C Language Binding

```
WORD   graf_slidebox();
WORD   retval, parent, obj, isvert;
LONG   tree;

retval = graf_slidebox(tree, parent, obj, isvert);
```

Parameter Block Binding

Control	Input	Output
control(0) = 76	int_in(0) = parent	int_out(0) = retval
control(1) = 3	int_in(1) = obj	
control(2) = 1	int_in(2) = isvert	
control(3) = 1		
control(4) = 0		

GRAF_HANDLE

Gets the GEM VDI handle for the currently open screen workstation.

GEM AES and GEM applications share this handle whenever they make GEM VDI calls.

Output Arguments

retval	GEM VDI handle
pwchar	Width (in pixels) of a character cell in the system font used in menus and dialogs
phchar	Height (in pixels) of a character cell in the system font used in menus and dialogs
pwbox	Width (in pixels) of a square box large enough to hold a system font character
phbox	Height (in pixels) of a square box large enough to hold a system font character

Sample Call to C Language Binding

```
WORD  graf_handle();
WORD  retval, pwchar, phchar, pwbox, phbox;

retval = graf_handle(&pwchar, &phchar, &pwbox, &phbox);
```

Parameter Block Binding

Control	Input	Output
control(0) = 77		int_out(0) = retval
control(1) = 0		int_out(1) = pwchar
control(2) = 5		int_out(2) = phchar
control(3) = 0		int_out(3) = pwbox
control(4) = 0		int_out(4) = phbox

GRAF_MOUSE

Changes the mouse form to one of a predefined set or to an application-defined form.

Note: The application selects or defines the mouse form that appears in the work area of its topmost (active) window. Outside the work area of the active window, the mouse form must always be an arrow or an hourglass.

If it uses a mouse form other than an arrow, an application must make a GRAF_MOUSE call each time the mouse form exits or enters the active window's work area.

The GRAF_MOUSE call upon exit converts the mouse form to an arrow. The GRAF_MOUSE call upon entry converts the mouse form back to the application's mouse form.

The application uses an EVNT_MULTI call, specifying a mouse rectangle equal to the active window's work area, to detect mouse form exit and entry.

Input Arguments

m_number A code identifying a predefined mouse form

0 arrow
1 text cursor (vertical bar)
2 hourglass
3 hand with pointing finger
4 flat hand, extended fingers
5 thin cross hair
6 thick cross hair
7 outline cross hair
255 mouse form stored in m_addr
256 hide mouse form
257 show mouse form

m_addr Address of a 37-word buffer that fits the input arguments specified in the vsc_form function defined in Section 7, Input Functions, in the GEM VDI Reference Guide.

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   graf_mouse();
WORD   retval, m_number;
LONG   m_addr;

retval = graf_mouse(m_number, m_addr);
```

Parameter Block Binding

Control	Input	Output
control(0) = 78	int_in(0) = m_number	int_out(0) = retval
control(1) = 1		
control(2) = 1	addr_in(0) = m_addr	
control(3) = 1		
control(4) = 0		

GRAF_MKSTATE

Returns the current mouse location, mouse button state, and keyboard state.

Output Arguments

retval Undefined

pmx X-coordinate of the mouse's current location

pmy Y-coordinate of the mouse's current location

pmstate Current mouse button state

The following bits represent the buttons:

0x0001 button on left
0x0002 second button from left
0x0004 third button from left, etc.

This parameter uses the following bit settings:

0 button up
1 button down

pkstate The current state of the keyboard's right-Shift, left-Shift, Ctrl, and Alt keys. The following bits represent the keys:

0x0001 right-Shift
0x0002 left-Shift
0x0004 Ctrl
0x0008 Alt

This parameter uses the following bit settings:

0 key up
1 key down

Sample Call to C Language Binding

```
WORD   graf_mkstate();
WORD   retval, pmx, pmy, pmstate, pkstate;

retval = graf_mkstate(&pmx, &pmy, &pmstate, &pkstate);
```

Parameter Block Binding

Control	Input	Output
control(0) = 79		int_out(0) = retval
control(1) = 0		int_out(1) = pmx
control(2) = 5		int_out(2) = pmy
control(3) = 0		int_out(3) = pmstate
control(4) = 0		int_out(4) = pkstate

End of Section 8

SCRAP LIBRARY

GEM AES's Scrap Library provides a set of routines that increase the usefulness of different applications by managing data interchange between the applications.

A standard data interchange format offers several advantages, including the following:

- Users can assemble an integrated set of independently developed applications.
- An application can take advantage of functions and output provided by other applications.

The Scrap Library's user interface lets the user cut or copy from one application's data space and paste into another's. The temporary holding place for these scraps of data is a clipboard, which is the implied destination for all cuts and the implied source for all pastes.

The user can place data on the clipboard in two ways:

- By cutting, the user deletes the data from the source application's data space.
- By copying, the user leaves the original piece of data in the source application's data space.

The clipboard is only one level deep; each new cut or copy overwrites the current contents of the clipboard.

A paste is, in effect, a copy from the clipboard to the target application. The data remains on the clipboard, allowing the user to paste the same piece of data repeatedly.

The AES's Scrap Library supports the following interactions:

- Writing the name of a scrap directory to the clipboard
- Reading the name of a scrap directory from the clipboard
- Managing the use of the disk as an extended scrap area

The AES stores scrap on the disk. The filename for scrap data is always SCRAP. The data's filetype identifies what kind of data it is. For different applications to be integrated, the AES must define standard data types in which scrap may be stored.

The AES supports the following standard data types, listed with their associated filename extensions:

CSV	Comma-separated values
TXT	ASCII text strings
DIF	Spreadsheet data
GEM	Metafile; GEM VDI graphic images
IMG	File created with GEM Paint™; GEM VDI bit images
DCA	IBM™ document content architecture
USR	OEM-defined

To be accessible to a variety of applications, a piece of scrap might appear on the clipboard in several data types.

All AES programs should at least be able to read and write ASCII data. Image files and Metafiles are described in the GEM VDI Reference Guide.

In addition to these standard data types, programmers can add their own application-specific data types.

An application can find or establish the full path (for example, A:\SCRAPDIR\SCRAP.*) by using the SCRAP_READ and SCRAP_WRITE routines.

Use the resident operating system's file system calls to create, read, and write files in the scrap directory.

9.1 Scrap Library Routines

The Scrap Library contains the following routines:

SCRP_READ Reads the scrap directory path and reports the types of files.

SCRP_WRITE Sets the current scrap directory path and validates its existence.

SCRP_CLEAR Deletes all files with the SCRAP filename in the current scrap directory.

SCRP_READ

Returns the scrap directory path and reports the types of files.

Input Arguments

pscrap — The address of the buffer into which the current scrap directory's path specification will be written.

Output Arguments

retval — A bit vector indicating the types of files present in the directory as follows:

Bit 0	SCRAP.CSV
Bit 1	SCRAP.TXT
Bit 2	SCRAP.GEM
Bit 3	SCRAP.IMG
Bit 4	SCRAP.DCA
Bit 5	SCRAP.USR

Bits 6 through 15 are reserved for future use and set to zero.

A coded return value:

-1 There is no scrap directory.
0 There are no files in the scrap directory.

Sample Call to C Language Binding

```
WORD   scrp_read();
WORD   retval;
LONG   pscrap;

retval = scrp_read(pscrap);
```

Parameter Block Binding

Control	Input	Output
control(0) = 80		int_out(0) = retval
control(1) = 0		
control(2) = 1	addr_in(0) = pscrap	
control(3) = 1		
control(4) = 0		

SCRP_WRITE

Sets the current scrap directory path and validates the existence of the scrap directory.

Note that this routine does not create the scrap directory. You create the scrap directory and write the various SCRAP files using the resident operating system's file system calls.

Input Arguments

pscrap — The address of the buffer from which the new scrap directory will be copied to the clipboard

Output Arguments

retval — A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   scrp_write();
WORD   retval;
LONG   pscrap;

retval = scrp_write(pscrap);
```

Parameter Block Binding

Control	Input	Output
control(0) = 81		int_out(0) = retval
control(1) = 0		
control(2) = 1	addr_in(0) = pscrap	
control(3) = 1		
control(4) = 0		

SCRP_CLEAR

Deletes all files with the SCRAP file name in the current scrap directory. This routine requires you to have previously set the current scrap directory with scrp_write.

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   scrp_clear();
WORD   retval;

retval = scrp_clear();
```

Parameter Block Binding

Control	Input	Output
control(0) = 82		int_out(0) = retval
control(1) = 0		
control(2) = 1		
control(3) = 0		
control(4) = 0		

End of Section 9

FILE SELECTOR LIBRARY

Many applications require that the user provide a filename to create a new file, recall an existing file, or use a file as input for a function like PRINT. Programmers can design applications that elicit the filename from the user in a variety of ways, three of which are described below:

- The application does not display a directory of existing filenames. To act on an existing file, the user must remember its filename. To create a file, the user must provide a filename that does not already exist in the directory.
- The application displays a directory, and the user types a new or existing filename.
- The application displays a directory. To act on an existing file, the user selects the filename directly from the directory. To create a file, the user types a filename.

The last method, the easiest for the user, is the method used by GEM AES's File Selector Library, which provides a consistent user interface for filename entry and selection.

When an application requests the File Selector Library to prompt the user for a filename, a special File Selector dialog appears on the screen. (The GEM application manuals refer to this dialog as the Item Selector.) See Figure 10-1.

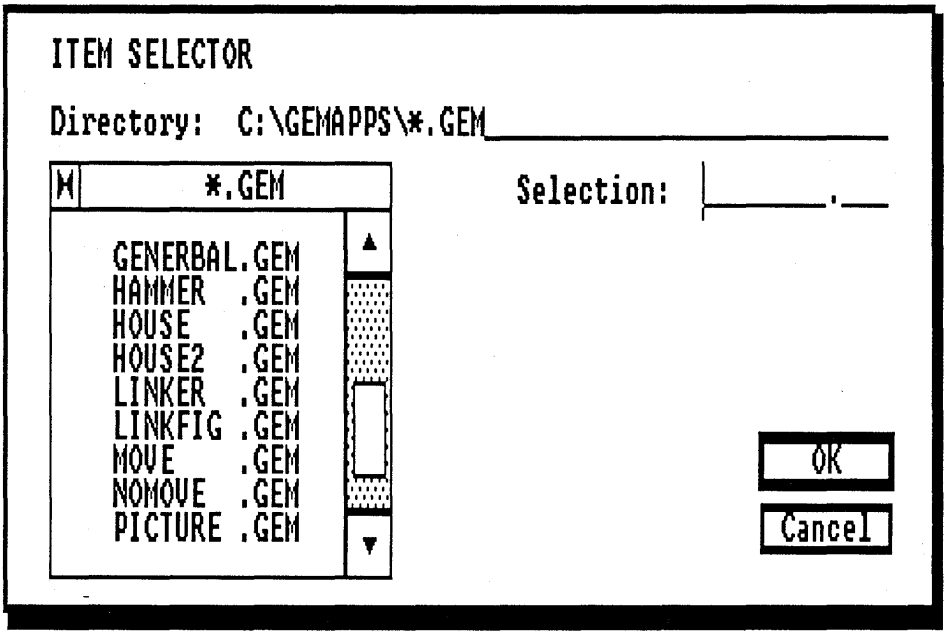


Figure 10-1. File Selector Dialog

The File Selector dialog displays the name of the current directory (including drive identifier), a selection field, and a list of filenames contained in the directory. The scrolling area to the right of the directory list contains up- and down-arrows, a scroll bar, and a slider.

The next section describes how the user interacts with the File Selector.

10.1 Using the File Selector

The File Selector Library provides the programmer with a consistent method of prompting the user for a filename. After the user selects a menu command that requires a filename as input, the following events typically occur:

1. The application calls the File Selector Library to display the File Selector dialog box.
2. Before selecting or entering a filename, the user can do the following:

   - Scroll through the list of files in the directory

     The File Selector dialog box's scrolling area is functionally the same as the vertical scrolling area of a window. For details of the user's interaction with the scrolling area, see the descriptions of the up-arrow, down-arrow, and the vertical scroll bar and slider in your GEM Desktop guide.

   - Change the directory being displayed

     To do so, the user clicks on "Directory:" and then types in a new drive identifier, directory path name, and file specification containing a wildcard in the filename or filetype, as in the following example:

     C:\RECEIPTS*.BAS

     After changing the directory specification, the user clicks the mouse pointer anywhere inside the window containing the list of filenames. The AES updates the window, displaying a list of filenames that fit the new specification.

3. The user selects a filename from the directory in the File Selector dialog or enters a new filename.

   To select an existing file, the user places the mouse pointer over the filename and clicks. If the filename is not currently visible in the list, the user can place the mouse pointer over "Selection:", click, and then type the filename.

To create a new file, the user places the pointer over "Selection:", clicks, and then types the filename.

4. The user selects OK or Cancel.
5. The File Selector Library returns the following information to the application:
   - The filename that was selected or entered
   - The current directory and wildcard specification
   - The way in which the user exited the File Selector dialog (OK or Cancel)
6. If the user selected OK, the application continues on with the filename that was selected or entered.

10.2 File Selector Library Routine

The File Selector Library uses the following routine:

FSEL_INPUT	Displays the File Selector dialog box and lets the user select a filename.

FSEL_INPUT

Displays the File Selector dialog and monitors the user's interaction with it.

The File Selector Library returns the results of this interaction between the user and the dialog to the application.

The FSEL_INPUT function takes 2K bytes of memory for temporary storage of the directory contents. This buffer is released when the user exits the routine.

Input Arguments

pipath — Address of the buffer that holds the initial directory specification displayed in the File Selector dialog. This buffer will also hold the directory specification that was in the File Selector dialog when the user selected OK or Cancel.

The pointer must point to a buffer with a path specification; you cannot have a nul pointer or nul path specification. The pipath buffer can contain comma separated, wildcard specifications. For example, the following command selects for display in the File Selector dialog all files in the PICTURES directory on drive C: with the GEM and IMG extensions.

```
C:\PICTURES\*.IMG,*.GEM
```

pisel — Address of the buffer that holds the initial Selection displayed in the File Selector dialog. This buffer also holds the selection that was in the File Selector dialog when the user selected OK or Cancel.

Output Arguments

retval — A coded return message:

- 0 no memory is available
- n no error

pbutton — A code identifying the exit button selected by the user.

- 0 Cancel
- 1 OK

Sample Call to C Language Binding

```
WORD   fsel_input();
WORD   retval, pbutton;
LONG   pipath, pisel;

retval = fsel_input(pipath, pisel, &pbutton);
```

Parameter Block Binding

Control	Input	Output
control(0) = 90	addr_in(0) = pipath	int_out(0) = retval
control(1) = 0	addr_in(1) = pisel	int_out(1) = pbutton
control(2) = 2		
control(3) = 2		
control(4) = 0		

End of Section 10

WINDOW LIBRARY

A window is an area with clearly defined boundaries. GEM AES provides two kinds of windows:

- Desktop window (window 0, w_handle = 0)
- Application windows

11.1 Desktop Window

The AES opens the workstation and creates window 0, the desktop window. The desktop window consists of the entire screen area, including the menu bar. The area below the menu bar is the total area available to an application. This is called the "desktop window's work area."

While the user is in the AES environment, the desktop window is always present on the screen. It is the backdrop for the application's windows. You cannot delete the desktop window.

The AES supports a maximum of eight (8) windows at a time. Each window is identified by a unique window handle, assigned by the AES when the window is created. The desktop window is always identified by a window handle of zero.

A GEM application can use all eight available windows, although this could result in no windows being available for desk accessories. An application can avoid this problem in the manner adopted by the GEM Desktop application, which sets a limit of two windows for itself, leaving the remaining six windows available for desk accessories.

Note: The GEM Desktop program is a typical application and it uses application windows. Do not confuse the GEM Desktop application program with window 0, the desktop window.

11.2 Application Windows

Although GEM applications can output to the desktop window, they usually create and maintain one or more application windows whenever they need to display data. The AES supports overlapping windows to allow an application to display two pieces of data at the same time. For example, a word processor that lets a user work simultaneously on two files can show each file in a separate window.

Application windows do the following:

- Let the user view help information and application data at the same time.
- Let the user cut and paste data between applications.
- Make it possible to display status information from several different background activities, for example, compiling, sorting, and transferring data.

An application window contains two parts:

- Border area: Title bar, information line, and window control areas
- Work area: Space inside border area

Figure 11-1 shows the application window's work area and the components of the border area. The GEM Desktop manual describes how the end user interacts with the window control areas in an application program.

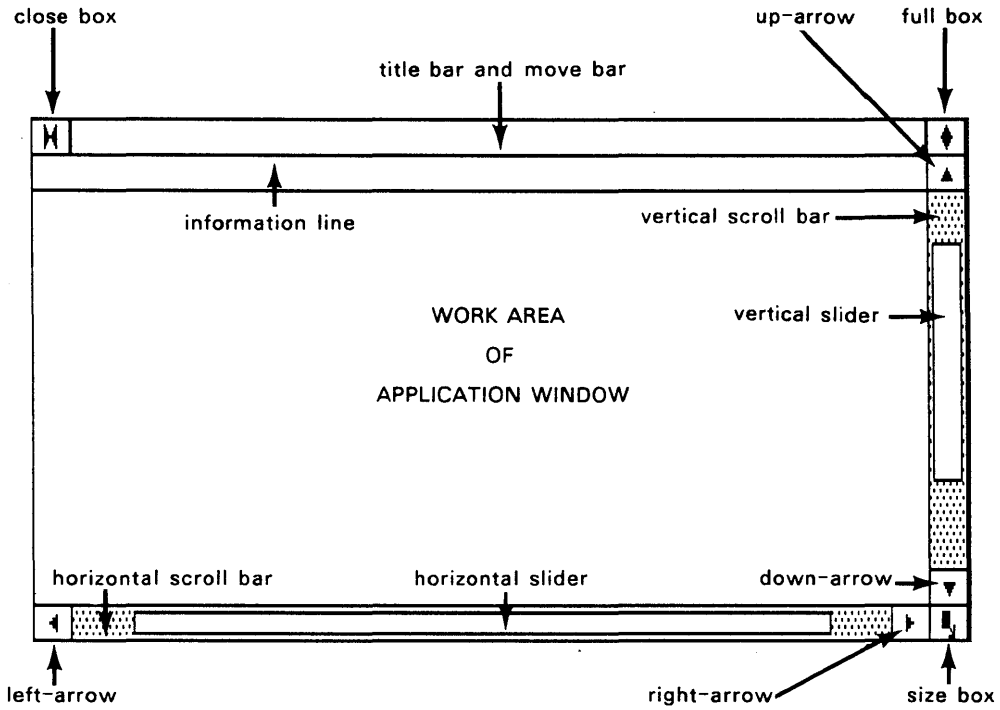


Figure 11-1. Parts of Application Window

The application can use and must manage the work area, which has the following characteristics:

- It is the area available for the application to use.
- All user actions inside the work area are managed by the application. What appears in the work area as a result of user interaction is defined, displayed, and controlled by the application.

The border area has the following characteristics:

- The border area can have several different components. An application determines which components will appear in the border area but does not control them.
- The AES's Screen Manager displays the contents of each window's border and manages all user interactions with the border area.

11.3 Components of the Border Area

Table 11-1 lists the border components and, where applicable, their corresponding predefined messages.

The maximum character length for the title bar and information line is 80 characters; however, the actual number of characters displayed depends on the size of the window, the screen resolution, and the font.

Except for the title bar and information line, the border elements are called "window control areas," because user interaction with any of them causes some change to take place, either in the work area or to the window as a whole. The application determines which window control areas appear in the window.

When a user clicks in a window control area, the AES sends the corresponding predefined message to the application. Predefined messages are described in Section 4.6.

Table 11-1. Border Components

Component	Description and Predefined Message
Title bar	One character high bar across the top of the window containing a maximum of 80 text characters that name the window. The application provides this text in the wf_name field of the WIND_SET function.
Information line	One character high area below the title bar, containing a maximum of 80 text characters of programmer-defined information. The application provides this text in the wf_info field of the WIND_SET function.
Close box	Box at the left end of the title bar. When the user clicks the mouse on the close box, the AES sends a WM_CLOSED message to tell the application that the user wants the window closed.
Full box	Diamond at the right end of the title bar that toggles between the greatest possible size and current size of window. When the user clicks on the full box, the AES sends a WM_FULLED message to the application.
Move bar	Occupies the same space as the title bar, when present. When the user presses the mouse button while the mouse form is on the move bar, the AES displays an XORed outline of the window. The user can drag this outline around the desktop as long as the button is held down. When the user releases the button, the outline disappears, and the AES sends a WM_MOVED message to tell the application the user wants the window moved to the location indicated by the outline's last position.

Table 11-1. Cont'd

Component	Description and Predefined Message
Size box	Box at lower right corner of window. When the user holds down the mouse button on the size box, the AES displays an XORed outline of the window. The upper left corner of the outline remains in a fixed position. The lower right corner can be dragged around as long as the user continues to press the mouse button. When the user releases the button, the AES sends a WM_SIZED message to tell the application the user wants the window's size changed to match the size of the outline when the button was released.
Vertical scroll bar and slider	The vertical scroll bar is located between the up- and down-arrow. The vertical slider moves up and down in the scroll bar. To move quickly to the top or bottom of the window's data or to any point between, the user can drag the slider inside the scroll bar. The size of the slider indicates how much of the data is visible in the window. For example, if the slider is half the size of the scroll bar, half the window's data is visible. When the user clicks above or below the slider, the AES sends a WM_ARROWED message indicating the user wants to move up or down one "page", as defined by the application. When the user drags the slider, the AES sends a WM_VSLID message indicating the new relative position of the slider so that the application can adjust the view area accordingly.

Table 11-1. Cont'd

Component	Description and Predefined Message
Horizontal scroll bar and slider	
	The horizontal scroll bar is located between the left- and right-arrow, and works like the vertical scroll bar and slider. The AES sends the WM_ARROWED and WM_HSLID messages to the application.
Up-arrow	Arrow above vertical scroll bar. When the user clicks on the up-, down-, left-, or right-arrow, the AES sends the application a WM_ARROWED message indicating that the user wants to move the file or directory in the appropriate direction by one row or column, as defined by the application.
Down-arrow	Arrow below the vertical scroll bar. The AES sends the application a WM_ARROWED message.
Left-arrow	Arrow to left of horizontal scroll bar. The AES sends the application a WM_ARROWED message.
Right-arrow	Arrow to right of horizontal scroll bar. The AES sends the application a WM_ARROWED message.

11.4 Division of Labor

The AES and the application divide responsibility for window management. The AES handles all user-mouse interactions that occur in the border areas, including the following:

- Clicking on the close box or full box
- Pressing the mouse button in the move bar to drag the window's outline
- Pressing the mouse button in the size box to produce a larger or smaller window outline
- Manipulating the scroll bars, sliders, and arrows

The AES sends a message to the application that created the window telling it the outcome of these interactions. When it receives one of these messages from the AES, the application has two choices:

- Make a Window Library call that causes the requested change to occur.
- Ignore the message.

This division of labor between the AES and the application has the following advantages:

- The application is not responsible for user interactions outside its window's work area.
- The application determines when and if user-requested window changes take place.
- Because it chooses which window control areas appear in its window's border area, the application also controls the kinds of window changes a user can request.

11.5 Window Management Calls

An application usually follows some variation of the following steps to fulfill its window management responsibilities:

1. It calls WIND_GET with a value of WF_XYWH for window 0 (the desktop window). This call returns the desktop window's X- and Y-coordinates, plus its width and height in pixels. These values identify the part of the screen below the menu bar that is available to the application.
2. It calls WIND_CALC with the width and height values from the previous call, plus a code identifying the border components it is requesting. WIND_CALC returns the size of the work area for this window in its greatest possible size.
3. It determines the size of the work area it wants. This size must be less than or equal to the size returned by the previous call.

4. It calls WIND_CALC with the desired work area size and the parameters describing the window's border. WIND_CALC returns the size of the window including the border area. This size is used in the WIND_CREATE, WIND_OPEN, and WIND_SET calls.

5. It calls WIND_CREATE with the size returned by WIND_CALC and the parameters describing the window's border. The size given to WIND_CREATE determines the window's maximum possible size. The AES returns a window handle (numeric identifier) for use with the other window calls.

6. It calls WIND_OPEN with the window's initial size and location on the desktop. The window appears after this call has been made.

7. It uses the window to display information to the user.

   The application uses an EVNT_MULTI call to wait for messages from the AES regarding user requests to close, full, size, scroll, or top (activate) the window. To support the overlapping window environment, the AES can also send a message requesting that part of the window be redrawn. The redraw procedure is described in Section 11.7.

8. It makes a WIND_CLOSE call when the application no longer wants the window visible on the screen. The window disappears, but it is still allocated to the application and can be reopened.

9. It makes a WIND_DELETE call when the application no longer needs the window at all. This call frees the window handle for use by another application. The application should always close a window before deleting it.

Managing multiple windows is an extension to the procedure described above. When an application gets a message requesting a window change, it uses the handle of the affected window in its Window Library calls.

11.6 Support of Overlapping Windows

Application windows can overlap like sheets of paper. The topmost window is called the active window.

When the user clicks the mouse button outside the active window's border area, the AES looks at what was under the tip of the pointer and acts as follows:

- If the pointer was over the desktop window, the AES does nothing.
- If the pointer was over another window, the AES sends a WM_TOPPED message to the application owning the window that the user wants active. The message informs the application that the user wants that window brought to the top (activated). The application should respond to this message with an AES call to bring the window to the top.

There are two instances when part of a window might not be visible:

- When one window overlaps another
- When the active window has been positioned so that part of it is off the physical screen

When an application sends output to the work area of its window, it draws that output only to that part of the work area that is visible to the user. This selective drawing is called "clipping."

The AES uses a list of rectangular regions to keep track of the portion of the physical screen belonging to each window. Each window has its own list. This list contains the least number of non-overlapping rectangles that define the visible area of the window. For example, if the window is fully visible, the list contains one rectangle, which is equal to the size of the visible window.

If the window is not totally visible, that is, if it has another window on top of it, the AES breaks up the window into two, three, four, or more rectangles, depending on the location of the top or active window.

Figure 11-2 has two windows: the shaded one and the white one. It shows how the number of rectangles is affected by the location of the top window.

AES breaks down underlying window (white rectangles) into two, three, or four rectangles depending on the location of the overlying window (shaded area).

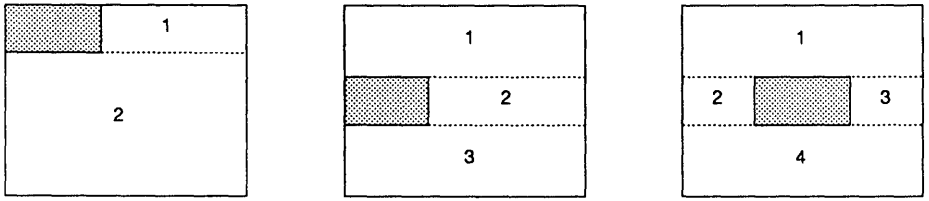


Figure 11-2. Window Rectangles

The application obtains each rectangle on the list by making a series of WIND_GET calls. The application must restrict its output to the rectangle returned by WIND_GET. WIND_GET returns a height and width of 0 when there are no more rectangles in the list.

11.7 Redrawing and Updating

To use the windowing system most efficiently, an application should be able to respond quickly to redraw requests from the AES.

There are three reasons why an application might need to update a window's work area:

- To display new application-generated information to the user
- To respond to a message reporting a user request to scroll the contents of the window
- To respond to a request from the AES to redraw a portion of the window

In each case, some portion of the work area has to be updated. This "update rectangle" can range in size from a one-pixel square to the

entire work area. In the first two cases above, the application defines the update rectangle. In the third case, the AES's WM_REDRAW message contains the X- and Y-coordinates of the update rectangle, as well as its width and height.

Knowing the size and location of the update rectangle, the application follows these steps:

1. It calls WIND_UPDATE with a value of 1, which indicates the beginning of an update. This call freezes the rectangle lists of all the windows on the screen.

2. It calls WIND_GET with a value of WF_FIRSTXYWH, which asks for the location and size of the first rectangle in the window's rectangle list. If the width and height values of this rectangle are not zero, it performs steps 3 through 6. If the values are zero, it goes to step 6.

3. It calculates a "result rectangle," which is the intersection (if any) of the rectangle obtained from the window's rectangle list in the WIND_GET call and the update rectangle defined in the WM_REDRAW message.

   If the result rectangle has width and height, that is, there is an intersection, the application draws the portion of the window defined by the result rectangle. To simplify the process of clipping the window contents to fit the rectangle (which will probably be required), GEM VDI provides a "set clip rectangle" call.

   If the result rectangle has zero width and/or height, the application doesn't draw anything. It continues to the next step.

4. It calls WIND_GET with a value of WF_NEXTXYWH, which asks for the next rectangle from the window's rectangle list.

5. If the width and height values of this rectangle are not zero, it performs steps 3 through 6. If the values are zero, it goes to step 8.

6. It calls WIND_UPDATE with a value of 0, which indicates the end of an update. This call allows the resumption of changes to the rectangle lists of all the windows on the screen.

11.8 Window Library Routines

The Window Library provides the following routines:

WIND_CREATE	Allocates the application's full-size window and returns a handle.
WIND_OPEN	Opens the created window to a specified size.
WIND_CLOSE	Closes an open window.
WIND_DELETE	De-allocates the application's window and handle.
WIND_GET	Gets information on a particular window.
WIND_SET	Sets new values for the fields that determine how a window is displayed.
WIND_FIND	Determines which window is under the mouse's X,Y position.
WIND_UPDATE	Notifies the AES that the application is about to update or has finished updating a window, or that the application is about to take or relinquish control of all mouse functions.
WIND_CALC	Calculates the X- and Y-coordinates and the width and height of a window's work area or border area.

WIND_CREATE

Allocates the application's full-size window and returns the window's handle (numeric identifier). This routine establishes the window's greatest possible size; the WIND_OPEN routine determines the window's actual size when opened.

The window border is taken from the space specified. The border is 2 pixels high.

Input Arguments

kind — The individual components present in the window. The following bits represent the components:

0x0001	NAME	(title bar with name)
0x0002	CLOSE	(close box)
0x0004	FULL	(full box)
0x0008	MOVE	(move bar)
0x0010	INFO	(information line)
0x0020	SIZE	(size box)
0x0040	UPARROW	(up-arrow)
0x0080	DNARROW	(down-arrow)
0x0100	VSLIDE	(vertical slider)
0x0200	LFARROW	(left-arrow)
0x0400	RTARROW	(right-arrow)
0x0800	HSLIDE	(horizontal slider)
0x1000	HOTCLOSEBOX	(hot close box)

This call uses the following bit settings for each component:

0 does not have the component
1 has the component

wx — X-coordinate of the full-size window

wy — Y-coordinate of the full-size window

ww — Width (in pixels) of the full-size window

wh — Height (in pixels) of the full-size window

Output Arguments

retval — The handle (numeric identifier) that will identify this window in future calls.

0 or n	window handle
-n	the AES has no more windows available.

Sample Call to C Language Binding

```
WORD  wind_create();
WORD  retval, wx, wy, ww, wh;
UWORD kind;

retval = wind_create(kind, wx, wy, ww, wh);
```

Parameter Block Binding

Control	Input	Output
control(0) = 100	int_in(0) = kind	int_out(0) = retval
control(1) = 5	int_in(1) = wx	
control(2) = 1	int_in(2) = wy	
control(3) = 0	int_in(3) = ww	
control(4) = 0	int_in(4) = wh	

WIND_OPEN

Opens (displays) a window in its initial size (not necessarily its full size) and location.

Input Arguments

handle	Handle of the window to be opened
wx	X-coordinate of the window in its initial size
wy	Y-coordinate of the window in its initial size
ww	Width (in pixels) of the window in its initial size
wh	Height (in pixels) of the window in its initial size

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   wind_open();
WORD   retval, wx, wy, ww, wh;

retval = wind_open(handle, wx, wy, ww, wh);
```

Parameter Block Binding

Control	Input	Output
control(0) = 101	int_in(0) = handle	int_out(0) = retval
control(1) = 5	int_in(1) = wx	
control(2) = 1	int_in(2) = wy	
control(3) = 0	int_in(3) = ww	
control(4) = 0	int_in(4) = wh	

WIND_CLOSE

Closes an opened window.

Although closed, the window and its handle remain allocated. The application can reopen the window by again calling the WIND_OPEN routine.

Input Arguments

handle — Handle of the window to be closed

Output Arguments

retval — A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD    wind_close();
WORD    retval, handle;

retval = wind_close(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 102	int_in(0) = handle	int_out(0) = retval
control(1) = 1		
control(2) = 1		
control(3) = 0		
control(4) = 0		

WIND_DELETE

Frees the space occupied by the window and its handle.

To open the window again, the application must first recreate it by calling the WIND_CREATE routine and then the WIND_OPEN routine.

Input Arguments

handle — Handle of the window to be deleted

Output Arguments

retval — A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   wind_delete();
WORD   retval, handle;

retval = wind_delete(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 103	int_in(0) = handle	int_out(0) = retval
control(1) = 1		
control(2) = 1		
control(3) = 0		
control(4) = 0		

WIND_GET

Depending on the information requested by the call, returns one of the following:

- X- and Y-coordinates and width and height values for various aspects of the current and previous windows
- Slider location and size
- Handle of the active window
- X- and Y-coordinates, width, and height of the rectangles in the window's rectangle list

Input Arguments

w_handle	Handle of the window about which the application wants information
w_field	A numerical value identifying the field about which the application wants information. The value of w_field determines which of pw1, pw2, pw3, and pw4 are returned. See Table 11-2.

Output Arguments

retval	A coded return message: 0 error n no error
pw1	The value returned depends on the field named in w_field (see above).
pw2	The value returned depends on the field named in w_field (see above).
pw3	The value returned depends on the field named in w_field (see above).
pw4	The value returned depends on the field named in w_field (see above).

Table 11-2. Values and Returned Parameters of w_field

No.	Field and Returned Parameters
1	WF_RESVD1 - Reserved
4	WF_WXYWH - Coordinates of the window's work area: pw1 (X-coordinate) pw2 (Y-coordinate) pw3 (width) pw4 (height)
5	WF_CXYWH - Coordinates of the entire current window, including borders, title bar, information line and the area for the drop shadow. pw1 (X-coordinate) pw2 (Y-coordinate) pw3 (width) pw4 (height)
6	WF_PXYWH - Coordinates of the previous window, including borders, title bar, and information line: pw1 (X-coordinate) pw2 (Y-coordinate) pw3 (width) pw4 (height)
7	WF_FXYWH - Coordinates of the window at its fullest possible size, including borders, title bar, and information line: pw1 (X-coordinate) pw2 (Y-coordinate) pw3 (width) pw4 (height)

Table 11-2. Cont'd

No.	Field and Returned Parameters
8	WF_HSLIDE - A number between 1 and 1000, giving the relative position of the horizontal slider returned in pw1: 1 = leftmost position 1000 = rightmost position
9	WF_VSLIDE - A number between 1 and 1000, giving the relative position of the vertical slider, returned in pw1: 1 = top position 1000 = bottom position
10	WF_TOP - Window handle of the window that is on top (active), returned in pw1:
11	WF_FIRSTXYWH - Coordinates of the first rectangle in the window's rectangle list: pw1 (X-coordinate) pw2 (Y-coordinate) pw3 (width) pw4 (height)
12	WF_NEXTXYWH - Coordinates of the next rectangle in the window's rectangle list: pw1 (X-coordinate) pw2 (Y-coordinate) pw3 (width) pw4 (height)
13	WF_RESVD2 - Reserved
15	WF_HSLSIZE - Size of the horizontal slider returned in pw1: -1 = default minimum size (a square box) 1-1000 = the slider's relative size compared to the horizontal scroll bar

Table 11-2. Cont'd

No.	Field and Returned Parameters
16	WF_VSLSIZE - Size of the vertical slider returned in pw1: -1 = default minimum size (a square box) 1 - 1000 = the slider's relative size compared to the vertical scroll bar
17	WF_SCREEN - Address and length of the internal menu/alert buffers: pw1 (low WORD of address) pw2 (high WORD of address) pw3 (low WORD of length) pw4 (high WORD of length)

Sample Call to C Language Binding

```
WORD   wind_get();
WORD   retval, w_handle, w_field, pw1, pw2, pw3, pw4;

retval = wind_get(w_handle, w_field, &pw1, &pw2, &pw3, &pw4);
```

Parameter Block Binding

Control	Input	Output
control(0) = 104	int_in(0) = w_handle	int_out(0) = retval
control(1) = 2	int_in(1) = w_field	addr_in(1) = pw1
control(2) = 5		addr_in(2) = pw2
control(3) = 0		addr_in(3) = pw3
control(4) = 0		addr_in(4) = pw4

WIND_SET

Changes the value in one of several fields that determine how a window is displayed.

Input Arguments

w_handle	Handle of the window whose fields the application wishes to change.
w_field	Numerical value identifying the field the application wishes to change.
w1	Value depends on the field named in w_field (see below).
w2	Value depends on the field named in w_field (see below).
w3	Value depends on the field named in w_field (see below).
w4	Value depends on the field named in w_field (see below).

Output Arguments

retval A coded return message:

0 error
n no error

Table 11-3. Values and Input Parameters of w_field

No.	Field and Input Parameters
2	WF_NAME - Address of the title bar string; input: w1 (low word of address) w2 (high word of address)
3	WF_INFO - Address of the information line string; input: w1 (low word of address) w2 (high word of address)
5	WF_CXYWH - Coordinates of the entire current window, including borders, title bar, information line and the area for the drop shadow. Input: w1 (X-coordinate) w2 (Y-coordinate) w3 (width) w4 (height)
8	WF_HSLIDE - A number between 1 and 1000, giving the relative position of the horizontal slider input in w1: 1 = leftmost position 1000 = rightmost position
9	WF_VSLIDE - A number between 1 and 1000, giving the relative position of the vertical slider, input in w1: 1 = top position 1000 = bottom position
10	WF_TOP - Window handle of the window that is on top (active), input in w1:
14	WF_NEWDESK - Address of a new default desktop window object tree for the AES to draw; input: w1 (high word of object tree) w2 (low word of object tree) w3 (starting object to draw in tree)

Table 11-3. Cont'd

No.	Field and Input Parameters
15	WF_HSLSIZE - Size of the horizontal slider input in w1: -1 = default minimum size (a square box) 1-1000 = the slider's relative size compared to the horizontal scroll bar
16	WF_VSLSIZE - Size of the vertical slider input in w1: -1 = default minimum size (a square box) 1 - 1000 = the slider's relative size compared to the vertical scroll bar
18	WF_TATTRB - Window attribute bit vector, input in w1; bit 0 = 0 - handle designates top window bit 0 = 1 - handle designates an underlying window bits 1 thru 15 - set to 0 (zero)
19	WF_SIZTOP - Moves the window specified in w_handle to the top; the order of the underlying windows is not changed. w1 (X-coordinate of window's upper lefthand corner) w2 (Y-coordinate of window's upper lefthand corner) w3 (width) w4 (height)

Sample Call to C Language Binding

```
WORD  wind_set();
WORD  retval, w_handle, w_field, w1, w2, w3, w4;

retval = wind_set(w_handle, w_field, w1, w2, w3, w4);
```

Parameter Block Binding

Control	Input	Output
control(0) = 105	int_in(0) = w_handle	int_out(0) = retval
control(1) = 6	int_in(1) = w_field	
control(2) = 1	int_in(2) = w1	
control(3) = 0	int_in(3) = w2	
control(4) = 0	int_in(4) = w3	
	int_in(5) = w4	

WIND_FIND

Finds which window is under the mouse's X,Y position.

Input Arguments

mx	X-coordinate of the mouse's position
my	Y-coordinate of the mouse's position

Output Arguments

retval	Handle of the window under the mouse's X,Y position

Sample Call to C Language Binding

```
WORD   wind_find();
WORD   retval, mx, my;

retval = wind_find(mx, my);
```

Parameter Block Binding

Control	Input	Output
control(0) = 106	int_in(0) = mx	int_out(0) = retval
control(1) = 2	int_in(1) = my	
control(2) = 1		
control(3) = 0		
control(4) = 0		

WIND_UPDATE

Does one of the following:

- Notifies the AES that the application is about to begin updating a window or has finished updating a window. During the update, the AES does not allow changes to take place in the portion of the screen belonging to the window.
- Notifies the AES that the application is taking control of all mouse functions, regardless of the mouse's location on the screen, or is returning to normal mouse function. When the application has control of all mouse functions, the Screen Manager no longer responds to mouse input, and menus and window control points are not active.

Input Arguments

beg_update A code for the call's function:

0 END_UPDATE
1 BEG_UPDATE
2 END_MCTRL
3 BEG_MCTRL

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   wind_update();
WORD   retval, beg_update;

retval = wind_update(beg_update);
```

Parameter Block Binding

Control	Input	Output
control(0) = 107	int_in(0) = beg_update	int_out(0) = retval
control(1) = 1		
control(2) = 1		
control(3) = 0		
control(4) = 0		

WIND_CALC

Calculates the X- and Y-coordinates and the width and height of a window's border area or work area.

In calculating the border area's parameters, this routine uses the corresponding parameters of the work area as input values. In calculating the work area's parameters, this routine uses the corresponding parameters of the border area as input values.

Input Arguments

wctype — The type of calculation to perform. The value of wctype affects all of the input and output arguments in WIND_CALC, except kind.

0 return border area X, Y, width, and height
1 return work area X, Y, width, and height

kind — The individual components present in the window The following bits represent the components:

0x0001	NAME	(title bar with name)
0x0002	CLOSE	(close box)
0x0004	FULL	(full box)
0x0008	MOVE	(move box)
0x0010	INFO	(information line)
0x0020	SIZE	(size box)
0x0040	UPARROW	(up-arrow)
0x0080	DNARROW	(down-arrow)
0x0100	VSLIDE	(vertical slider)
0x0200	LFARROW	(left-arrow)
0x0400	RTARROW	(right-arrow)
0x0800	HSLIDE	(horizontal slider)

This call uses the following bit settings for each component:

0 does not have the component
1 has the component

x	Input X-coordinate of the work area or border area
y	Input Y-coordinate of the work area or border area
w	Input width value of the work area or border area
h	Input height value of the work area or border area

Output Arguments

retval	A coded return message: 0 error n no error
px	Output X-coordinate of the work area or border area
py	Output Y-coordinate of the work area or border area
pw	Output width value of the work area or border area
ph	Output height value of the work area or border area

Sample Call to C Language Binding

```
WORD   wind_calc();
WORD   retval, wctype, x, y, w, h, px, py, pw, ph;
UWORD kind;

retval = wind_calc(wctype, kind, x, y, w, h, &px, &py, &pw, &ph);
```

Parameter Block Binding

Control	Input	Output
control(0) = 108	int_in(0) = wctype	int_out(0) = retval
control(1) = 6	int_in(1) = kind	int_out(1) = px
control(2) = 5	int_in(2) = x	int_out(2) = py
control(3) = 0	int_in(3) = y	int_out(3) = pw
control(4) = 0	int_in(4) = w	int_out(4) = ph
	int_in(5) = h	

End of Section 11

RESOURCE LIBRARY

The Resource Library is the interface between an application and its resources in the resource file. Resources, the collection of data used by the application, include trees, objects, strings, icons, and images.

A resource file isolates hardware-specific and language-specific data from an application's code, providing the following advantages:

- Machine-code portability: To port the application across different environments, you need only change the resource file data.
- Customization of appearance: A non-programmer can change the application's menu structure, dialog layout, and error message text. In most cases, a programmer need not be involved.
- Internationalization of text messages: To change text messages from one language to another, you need only change the text in the resource file.
- Device-independent raster graphics: Because they are stored as resources, the AES's icons and other bit-mapped images can be tailored to the resolution characteristics of various displays.

In all these instances the application's code is unchanged.

Most applications have a single resource file that contains all their resources. The AES follows the convention that all resource files have the filetype .RSC.

You create a resource file using the GEM Resource Construction Set (GEM RCS) described in the GEM Programmer's Utilities Guide.

12.1 Using the Resource Library

When an application calls RSRC_LOAD with the name of a resource file, the Resource Library does the following:

1. It reads the header of the file and finds its total size in bytes.

2. Using the operating system's call to allocate memory, it allocates enough memory space to hold the resource file.
3. It opens the resource file, reads it into the allocated memory space, and closes the file.
4. It makes the following updates to the file:

   - Makes the file device-specific to the screen's resolution.
   - Links all the OBJECT, TEDINFO, ICONBLK, and BITBLK pointers.
   - Builds the array of tree pointers.
   - Stores the address of the tree in the application's Global Array.

The application can now make calls to any routines that require a tree index, including Object Library routines, FORM_DO, and MENU_BAR.

To get or set any pointer in the OBJECT, TEDINFO, ICONBLK, or BITBLK structures, the application calls RSRC_GADDR and RSRC_SADDR.

When the application is finished with the data from the resource file, it calls RSRC_FREE to release the allocated memory and zero the address of the tree array in the Global Array.

12.2 Resource Library Routines

The Resource Library contains the following routines:

RSRC_LOAD	Loads an entire resource file into memory.
RSRC_FREE	Frees the memory allocated during RSRC_LOAD.
RSRC_GADDR	Gets the address of a data structure in memory.
RSRC_SADDR	Stores an index to a data structure.
RSRC_OBFIX	Converts an object's X- and Y-coordinates, width, and height from character coordinates to pixel coordinates.

RSRC_LOAD

Allocates memory and loads a resource file into memory.

Input Arguments

rsname Address of the ASCII string of the resource filename

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD  rsrc_load();
WORD  retval;
LONG  rsname;

retval = rsrc_load(rsname);
```

Parameter Block Binding

Control	Input	Output
control(0) = 110 control(1) = 0 control(2) = 1 control(3) = 1 control(4) = 0	addr_in(0) = rsname	int_out(0) = retval

RSRC_FREE

Frees the memory space allocated in RSRC_LOAD.

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   rsrc_free();
WORD   retval

retval = rsrc_free();
```

Parameter Block Binding

Control	Input	Output
control(0) = 111		int_out(0) = retval
control(1) = 0		
control(2) = 1		
control(3) = 0		
control(4) = 0		

RSRC_GADDR

Gets the address of a data structure in memory.

Input Arguments

rstype — Type of data structure

- 0 tree
- 1 OBJECT
- 2 TEDINFO
- 3 ICONBLK
- 4 BITBLK
- 5 string
- 6 imagedata
- 7 obspec
- 8 te_ptext
- 9 te_ptmplt
- 10 te_pvalid
- 11 ib_pmask
- 12 ib_pdata
- 13 ib_ptext
- 14 bi_pdata
- 15 ad_frstr - Address of a POINTER to a free string
- 16 ad_frimg - Address of a POINTER to a free image

rsid — Index of the data structure

Output Arguments

retval — A coded return message:

- 0 error
- n no error

paddr — Address of the data structure specified by rstype and rsid

Sample Call to C Language Binding

```
WORD   rsrc_gaddr();
WORD   retval, rstype, rsid;
LONG   paddr;

retval = rsrc_gaddr(rstype, rsid, &paddr);
```

Parameter Block Binding

Control	Input	Output
control(0) = 112	int_in(0) = rstype	int_out(0) = retval
control(1) = 2	int_in(1) = rsid	
control(2) = 1		addr_out(0) = paddr
control(3) = 0		
control(4) = 1		

RSRC_SADDR

Stores the address of a data structure in memory.

Input Arguments

rstype — Type of data structure

15 ad_frstr - Address of a POINTER to a free string
16 ad_frimg - Address of a POINTER to a free image

rsid — Location in the data structure where the lngval value will be stored

lngval — Address of the data structure

Output Arguments

retval — A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   rsrc_saddr();
WORD   retval, rstype, rsid;
LONG   lngval;

retval = rsrc_saddr(rstype, rsid, lngval);
```

Parameter Block Binding

Control	Input	Output
control(0) = 113	int_in(0) = rstype	int_out(0) = retval
control(1) = 2	int_in(1) = rsid	
control(2) = 1		
control(3) = 1	addr_in(0) = lngval	
control(4) = 0		

RSRC_OBFIX

Converts an object's location and size from character coordinates to pixel coordinates.

Character coordinates are defined as the object's X, Y, width, and height values, where each WORD has the integral character position in the Least Significant Byte and the positive or negative pixel offset in the Most Significant Byte.

Input Arguments

obj	Index of the object to be converted.
tree	Address of the tree that contains the object.

Output Arguments

retval	Reserved; value always equals 1 (one).

Sample Call to C Language Binding

```
WORD   rsrc_obfix();
WORD   retval, obj;
LONG   tree;

retval = rsrc_obfix(tree, obj);
```

Parameter Block Binding

Control	Input	Output
control(0) = 114	int_in(0) = obj	int_out(0) = retval
control(1) = 1		
control(2) = 1	addr_in(0) = tree	
control(3) = 1		
control(4) = 0		

End of Section 12

SHELL LIBRARY

The Shell library runs on top of the limited multitasking kernel. The Shell invokes GEM and non-GEM applications.

The following examples illustrate how the Shell starts GEM applications.

- When the user enters the command "GEM", the AES is loaded into memory, and the primary application is the GEM Desktop. The user can start other applications, GEM or non-GEM, from the GEM Desktop. After quitting the GEM Desktop, the user returns to the operating system environment.
- When the user enters the command "GEM DRAW", the AES is loaded into memory, and the primary application is GEM Draw™. After quitting GEM Draw, the user returns to the operating system environment.
- When the user enters the command "GEM DRAW /D", the AES is loaded into memory, and the primary application is again GEM Draw. However, the "/D" part of the command tail causes the user to return to the GEM Desktop after quitting GEM Draw.

When the user invokes an application from the GEM Desktop, the GEM Desktop passes to the Shell the following information about the application:

- Whether it is graphic or character-based
- Whether it is a GEM or non-GEM application
- The name of the directory containing the application

(Most GEM applications are graphic, and most non-GEM applications are character-based, but the GEM Desktop needs to pass the information as separate parameters of the SHEL_WRITE call for those cases where this correlation does not hold.)

The GEM Desktop then terminates.

When any application terminates, control returns to the process that invoked it, in this case, the Shell.

The Shell determines if it was instructed to start a new application. If it was not, it starts the GEM Desktop again.

If the Shell was instructed to start a character-based application, it converts the screen to character mode and makes a GEM VDI Close Workstation call.

If the Shell was instructed to start a GEM application, no GEM VDI call or conversion is required.

To go from a character-based application to a GEM application or to the GEM Desktop, the Shell must make a GEM VDI Open Workstation call and convert the screen to graphics mode.

13.1 Using the Shell Library

The Shell Library performs the following three functions:

- Lets an application keep track of the command and command tail that invoked it
- Lets applications invoke other applications directly, without first returning to the GEM Desktop application

  It allows a user to request an application (for example, an output application) from within a running application. The user can, if the application supports this practice, string together several applications in this manner.

  The GEM Desktop is an example of an application that uses the SHEL_WRITE function to invoke other applications.
- Lets an application change the default application. The default application is the program the user returns to when he or she quits the current application. For all applications invoked from the GEM Desktop, the GEM Desktop is the default application.

The GEM Desktop is normally the default application referred to in the SHEL_RDEF and SHEL_WDEF functions. It remains the default application until changed by SHEL_WDEF. When the user quits an application invoked from the GEM Desktop, he or she returns to the GEM Desktop, not the native operating system. The user also returns

to the GEM Desktop when he or she quits an application invoked from another application. (See the SHEL_WRITE function.)

The SHEL_READ and SHEL_WRITE functions use two separate buffers that contain the following:

- The command with which the Shell Library invoked the current application or will invoke the next application
- The command tail with which the Shell Library invoked the current application or will invoke the next application

To learn the name of the command and command tail that invoked it, an application calls the SHEL_READ routine, passing in the following:

- Pointers to the addresses of the application's buffers that will hold the command information. The Shell Library copies the data from its own buffer to the application's buffers.
- A pointer to the application's home directory. The home directory is where the system looks if it does not find the application in the current directory.

To invoke an application, the current application (or the GEM Desktop) follows these steps:

1. It calls the SHEL_WRITE routine and passes in the address of the command, command tail, and home directory for the next application to run. It also indicates whether the application to run is graphic or character-based, and whether it is a GEM or non-GEM application.
2. When the current application terminates, the Shell Library starts the application that was requested next.

To exit the AES, an application makes a SHEL_WRITE call, passing a value of 0 (zero) in the doex argument.

13.2 Shell Library Routines

The Shell Library contains the following routines:

SHEL_READ	Lets an application determine how it was invoked.
SHEL_WRITE	Exits AES or tells which application to run next.
SHEL_FIND	Locates a filename by following the operating system search path.
SHEL_ENVRN	Searches the operating system environment for a parameter and returns the address of its value.
SHEL_RDEF	Returns the default application.
SHEL_WDEF	Writes the default application.

SHEL_READ

Lets an application identify the command that invoked it.

The command and tail buffers must each be 128 bytes. Be sure to initialize the buffers to this length.

Input Arguments

pcmd	Address of a buffer that will hold the command that invoked the application
ptail	Address of a buffer that will hold the command tail invoked with the command

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD   shel_read();
WORD   retval;
LONG   pcmd, ptail;

retval = shel_read(pcmd, ptail);
```

Parameter Block Binding

Control	Input	Output
control(0) = 120	addr_in(0) = pcmd	int_out(0) = retval
control(1) = 0	addr_in(1) = ptail	
control(2) = 1		
control(3) = 2		
control(4) = 0		

SHEL_WRITE

Tells the AES whether to run another application and, if so, which application to run. Note that the default application is normally DESKTOP.APP.

The command and tail buffers must each be 128 bytes.

Input Arguments

doex — A coded instruction to exit the system or run another application when the user exits the current application:

- 0 exit AES and return to the operating system prompt
- 1 run another application

isgr — A code for whether the next application is a graphic application

- 0 not graphic application
- 1 graphic application

isover — A code indicating whether the next program runs above, in the same memory space as, or instead of the default application. The codes are assigned as follows:

- 0 Run the next program above the default application. GEM loads and runs the program specified in the pcmd parameter immediately. Control returns to the program making the SHEL_WRITE call after the program specified in pcmd exits. When isover is 0 (zero), GEM ignores the values for doex and isgr.
- 1 Run the next program in the same place in memory as the program making the SHEL_WRITE call. GEM loads and runs the program specified in pcmd when the program making the

SHEL_WRITE call exits to the operating system. GEM uses the values for isgr and isover.

2 Run the next program in place of the current application and VDI. (Use this option if the program needs full memory.) GEM loads and runs the program specified in pcmd when the program making the SHEL_WRITE call exits. GEM uses the values for isgr and isover.

pcmd Address of the new command file to execute

ptail Address of the command tail for the next program

Output Arguments

retval A coded return message:

0 error
n no error

Sample Call to C Language Binding

```
WORD    shel_write();
WORD    retval, does, isgr, isover;
LONG    pcmd, ptail;

retval = shel_write(doex, isgr, isover, pcmd, ptail);
```

Parameter Block Binding

Control	Input	Output
control(0) = 121	int_in(0) = doex	int_out(0) = retval
control(1) = 3	int_in(1) = isgr	
control(2) = 1	int_in(2) = isover	
control(3) = 2		
control(4) = 0	addr_in(0) = pcmd	
	addr_in(1) = ptail	

SHEL_FIND

Searches for a filename in the current directory and in each directory in the search path; when it finds the filename, it returns its full operating system file specification.

The AES adds the following directories to the user's search path for SHEL_FIND calls:

ROOT DIRECTORY
GEMDESK
GEMSYS
GEMAPPS

Input Arguments

ppath — Address of a buffer with distinct input and output functions:

- input — Holds the filename the application is searching for
- output — Holds the full file specification of the filename's location in the search path. The buffer must be long enough to hold the full operating system file specification (80 character minimum).

Output Arguments

retval — A coded return message:

- 0 error
- n no error

Sample Call to C Language Binding

```
WORD   shel_find();
WORD   retval;
LONG   ppath;

retval = shel_find(ppath);
```

Parameter Block Binding

Control	Input	Output
control(0) = 124	addr_in(0) = ppath	int_out(0) = retval
control(1) = 0		
control(2) = 1		
control(3) = 1		
control(4) = 0		

SHEL_ENVRN

Lets an application search the operating system environment for a predefined paramter string, and returns the address of the byte immediately following the string in a LONG value.

This byte contains the value of the parameter, and its address is stored in a LONG value pointed at by the ppath argument.

Input Arguments

ppath	Address of a LONG value in which this routine will store the address of the byte immediately following the parameter string
psrch	Parameter string for which the application is searching

Output Arguments

retval	Reserved; value equals one (1)

Sample Call to C Language Binding

```
WORD   shel_envrn();
WORD   retval;
LONG   ppath, psrch;

retval = shel_envrn(ppath, psrch);
```

Parameter Block Binding

Control	Input	Output
control(0) = 125	addr_in(0) = ppath	int_out(0) = retval
control(1) = 0	addr_in(1) = psrch	
control(2) = 1		
control(3) = 2		
control(4) = 0		

SHEL_RDEF

Returns the path and command of the default application (the program that runs when the application terminates).

Input Arguments

lpcmd	Address of a buffer into which shel_rdef puts the command name. The command buffer must be 32 bytes.
lpdir	Address of a buffer into which shel_rdef puts the path. The path buffer must be 82 bytes.

Output Arguments

retval	Undefined

Sample Call to C Language Binding

```
WORD    shel_rdef();
WORD    retval;
LONG    lpcmd, lpdir;

retval = shel_rdef(lpcmd, lpdir);
```

Parameter Block Binding

Control	Input	Output
control(0) = 126	addr_in(0) = lpcmd	int_out(1) = retval
control(1) = 0	addr_in(1) = lpdir	
control(2) = 1		
control(3) = 2		
control(4) = 0		

SHEL_WDEF

Change the default application to be run when the current application terminates.

Input Arguments

lpcmd	Address of a buffer with the command name to invoke. The command buffer must be 32 bytes.
lpdir	Address of a buffer with the command's path specification. The path buffer must be 82 bytes.

Output Arguments

retval	Undefined

Sample Call to C Language Binding

```
WORD   shel_wdef();
WORD   retval;
LONG   lpcmd, lpdir;

retval = shel_wdef(lpcmd,lpdir);
```

Parameter Block Binding

Control	Input	Output
control(0) = 127	addr_in(0) = lpcmd	int_out(0) = retval
control(1) = 0	addr_in(1) = lpdir	
control(2) = 0		
control(3) = 2		
control(4) = 0		

End of Section 13

EXTENDED GRAPHICS LIBRARY

The routines in the Extended Graphics Library allow you to create the visual effect of a growing or shrinking box.

14.1 Extended Graphics Library Routines

The Extended Graphics Library contains the following routines:

XGRF_STEPCALC	Calculates increments for expanding or contracting box outline.
XGRF_2BOX	Draws a series of boxes.

XGRF_STEPCALC

Calculates the x and y increments necessary to expand or contract a box on the screen.

Input Arguments

orgw	Initial width of the box in pixels
orgh	Initial height of the box in pixels
xc	Final x coordinate of box
yc	Final y coordinate of box
w	Final width of box in pixels
h	Final height of box in pixels

Output Arguments

retval	A coded return message: 0 error n no error
pcx	Centered x coordinate at the end of the action
pcy	Centered y coordinate at the end of the action
pcnt	Number of steps
pxstep	Amount of each x step
pystep	Amount of each y step

Sample Call to C Language Binding

```
WORD   xgrf_stepcalc();
WORD   retval, orgw, orgh, xc, yc, w, h, pcx, pcy, pcnt, pxstep, pystep;

retval = xgrf_stepcalc(orgw,orgh,xc,yc,w,h,&pcx,&pcy,&pcnt,&pxtep,&pystep);
```

Parameter Block Binding

Control	Input	Output
control(0) = 130	int_in(0) = orgw	int_out(0) = retval
control(1) = 6	int_in(1) = orgh	int_out(1) = pcx
control(2) = 6	int_in(2) = xc	int_out(2) = pcy
control(3) = 0	int_in(3) = yc	int_out(3) = pcnt
control(4) = 0	int_in(4) = w	int_out(4) = pxstep
	int_in(5) = h	int_out(5) = pystep

XGRF_2BOX

Draws a series of xor'd boxes.

Input Arguments

cnt	Number of boxes to draw
xstep	Size of each x step
ystep	Size of each y step
doubled	Value indicating whether increments are doubled, defined as follows: 1 double number of steps 0 use number of steps specified
corners	Value indicating whether just corners or whole box is drawn, defined as follows: 1 draw corners only 0 draw whole box
xc	Initial x coordinate of box
yc	Initial y coordinate of box
w	Initial width of box in pixels
h	Initial height of box in pixels

Output Arguments

retval	A coded return message: 0 error n no error

Sample Call to C Language Binding

```
WORD   xgrf_2box();
WORD   retval, xc, yc, w, h, corners, cnt, xstep, ystep, doubled;

retval = xgrf_2box(xc,yc,w,h,corners,cnt,xstep,ystep,doubled);
```

Parameter Block Binding

Control	Input	Output
control(0) = 131	int_in(0) = cnt	int_out(0) = retval
control(1) = 9	int_in(1) = xstep	
control(2) = 1	int_in(2) = ystep	
control(3) = 0	int_in(3) = doubled	
control(4) = 0	int_in(4) = corners	
	int_in(5) = xc	
	int_in(6) = yc	
	int_in(7) = w	
	int_in(8) = h	

End of Section 14

AES FUNCTIONS IN OPCODE ORDER

Table A-1. AES Functions in Opcode Order

Opcode Dec	Opcode Hex	Function and Binding
10	A	appl_init()
11	B	appl_read(rwid, length, pbuff)
12	C	appl_write(rwid, length, pbuff)
13	D	appl_find(pname)
14	E	appl_tplay(tbuffer, tlength, tscale)
15	F	appl_trecord(tbuffer, tlength)
16	10	appl_bvset(bvdisk, bvhard)
17	11	appl_yield()
19	13	appl_exit()
20	14	evnt_keybd()
21	15	evnt_button(clicks, mask, state, &pmx, &pmy, &pmb, &pks)
22	16	evnt_mouse(flags, x, y, width, height, &pmx, &pmy, &pmb, &pks)
23	17	evnt_mesag(pbuff)
24	18	evnt_timer(locnt, hicnt)
25	19	evnt_multi(flags, bclk, bmsk, bst, m1flags, m1x, m1y, m1w, m1h, m2flags, m2x, m2y, m2w, m2h, mepbuff, tlc, thc &pmx, &pmy, &pmb, &pks, &pkr, &pbr)
26	1A	evnt_dclick(rate, setit)
30	1E	menu_bar(tree, showit)
31	1F	menu_icheck(tree, itemnum, checkit)
32	20	menu_ienable(tree, itemnum, enableit)
33	21	menu_tnormal(tree, titlenum, normalit)
34	22	menu_text(tree, inum, ptext)
35	23	menu_register(pid, pstr)
36	24	menu_unregister(mid)
40	28	objc_add(tree, parent, child)
41	29	objc_delete(tree, delob)
42	2A	objc_draw(tree, drawob, depth, xc, yc, wc, hc)
43	2B	objc_find(tree, startob, depth, mx, my)

Table A-1 (continued)

Opcode Dec	Hex	Function and Binding
44	2C	objc_offset(tree, obj, &poffx, &poffy)
45	2D	objc_order(tree, mov_obj, newpos)
46	2E	objc_edit(tree, obj, inchar, &idx, kind)
47	2F	objc_change(tree, drawob, depth, xc, yc, wc, hc, newstate, redraw)
50	32	form_do(form, start)
51	33	form_dial(dtype, ix, iy, iw, ih, x, y, w, h)
52	34	form_alert(defbut, astring)
53	35	form_error(errnum)
54	36	form_center(tree, &pcx, &pcy, &pcw, &pch)
55	37	form_keybd(form, obj, nxt_obj, thechar, &pnxt_obj, &pchar)
56	38	form_button(form, obj, clks, &pnxt_obj)
70	46	graf_rubbox(xorigin, yorigin, wmin, hmin, &pwend, &phend)
71	47	graf_dragbox(w, h, sx, sy, xc, yc, wc, hc, &pdx, &pdy)
72	48	graf_mbox(w, h, srcx, srcy, dstx, dsty)
75	4B	graf_watchbox(tree, obj, instate, outstate)
76	4C	graf_slidebox(tree, parent, obj, isvert)
77	4D	graf_handle(&pwchar, &phchar, &pwbox, &phbox)
78	4E	graf_mouse(m_number, m_addr)
79	4F	graf_mkstate(&pmx, &pmy, &pmstate, &pkstate)
80	50	scrp_read(pscrap)
81	51	scrp_write(pscrap)
82	52	scrp_clear()
90	5A	fsel_input(pipath, pisel, &pbutton)
100	64	wind_create(kind, wx, wy, ww, wh)
101	65	wind_open(handle, wx, wy, ww, wh)
102	66	wind_close(handle)
103	67	wind_delete(handle)
104	68	wind_get(w_handle, w_field, &pw1, &pw2, &pw3, &pw4)
105	69	wind_set(w_handle, w_field, w1, w2, w3, w4)
106	6A	wind_find(mx, my)
107	6B	wind_update(beg_update)
108	6C	wind_calc(wctype, kind, x, y, w, h, &px, &py, &pw, &ph)

Table A-1 (continued)

Opcode Dec	Opcode Hex	Function and Binding
110	6E	rsrc_load(rsname)
111	6F	rsrc_free()
112	70	rsrc_gaddr(rstype, rsid, &paddr)
113	71	rsrc_saddr(rstype, rsid, lngval)
114	72	rsrc_obfix(tree, obj)
120	78	shel_read(pcmd, ptail)
121	79	shel_write(doex, isgr, isover, pcmd, ptail)
124	7C	shel_find(ppath)
125	7D	shel_envrn(ppath, psrch)
126	7E	shel_rdef(lpcmd, lpdir)
127	7F	shel_wdef(lpcmd, lpdir)
130	82	xgrf_stepcalc(orgw,orgh,xc,yc,w,h,&pcx,&pcy,&pcnt,&pxstep,&pystep)
131	83	xgrf_2box(xc,yc,w,h,corners,cnt,xstep,ystep,doubled)

A

B

C

D

E

H

P

T

U

V

W

X

Y