GEM

Programmer's Guide

Volume 2: AES

COPYRIGHT

DISCLAIMER

TRADEMARKS

```
*****************************************
*  Third   Edition:    January  1989  *
*****************************************
```

This manual was edited on a Mega ST4 computer using Micro Emacs. It was formatted on the same computer using NROFF and printed on the Atari SLM 804 laser printer.

Table of Contents

Table of Contents

(continued)

3 Application Library

Table of Contents

(continued)

Table of Contents

(continued)

Table of Contents

(continued)

Table of Contents

(continued)

Table of Contents

(continued)

Figures

Section 1

Introduction to GEM AES

1.1 Purpose of This Programmer's Guide

This programmer's guide has three major purposes:

o Introduce a programmer to the concepts and structures underlying GEM..Application Environment Services.

o Prepare the programmer to write a GEM application.

o Serve as a reference guide for the programmer writing a GEM application.

1.1.1 Abbreviations of Names

The following abbreviations appear throughout this programmer's guide:

o GEM - Graphics Environment Manager.

o GEM AES - GEM Application Environment Services

o GEM VDI - GEM Virtual Device Interface. The GEM VDI is described in detail in the GEM Programmer's Guide, Volume 1: VDI.

o DOS - GEM DOS

1.2 Before Programming to GEM AES

Before starting to write a GEM application, and even before reading further in this programmer's guide, a programmer should do the following:

o Read the GEM Programmer's Guide, Volume 1: VDI.

Because GEM AES uses GEM VDI function calls extensively, an understanding of GEM VDI is essential to writing a successful GEM application.

o Use the GEM Desktop.

Familiarity with GEM AES's user interface--the GEM Desktop, icons, drop-down menus, and windows--is also essential to writing a successful GEM application.

The "GEM Desktop Reference Guide" (Part 2 of <u>The GEM Desktop</u> manual) describes GEM AES functionality in detail.

1.3 Structure of this Programmer's Guide

This programmer's guide contains three major divisions:

1. This introduction, which includes descriptions of the major components of GEM AES.

2. The description of a sample calling sequence for a typical GEM application.

3. Descriptions of the GEM AES subroutine libraries.

1.3.1 Contents of the Introduction

The remainder of this introduction describes the relationship of GEM AES to operating system files, GEM VDI, and applications, and describes how GEM AES shares space in memory with each of the three.

This section also describes the architecture of GEM AES itself, as well as its components and their relationship to each other.

1.3.2 Sample GEM AES Calling Sequence

Section 2 describes a typical sequence of calls from a GEM application to GEM AES.

The calling sequence does not try to describe a specific application, but instead tries to give a general overview of GEM AES function calls. However, where specific examples are needed, the GEM Desktop application serves as a typical case.

1.3.3 Subroutine Libraries

The remaining sections of this programmer's guide describe GEM AES's subroutine libraries. Each section has the following basic structure:

o an introduction to the library

o a section describing how an application uses the library

o descriptions of the library's data structures (if applicable)

o descriptions of each of the library's routines, including all of the routine's parameters and a sample C language binding

1.3.3.1 Naming Convention

The parameter name for a routine does the following:

o It identifies the parameter.
o It identifies the subroutine library the routine belongs to.
o It identifies the routine to which the parameter belongs.

The part of the parameter name preceding the underbar is a code for the subroutine library to which the routine belongs. The following list gives several partial parameter names and the names of the subroutine libraries to which they belong.

```
ap_...    Application Library
ev_...    Event Library
ob_...    Object Library
gr_...    Graphics Library
wi_...    Window Library
```

The first letter (or, in cases where there would be duplication, two letters) following the underbar identifies the specific routine to which the parameter belongs. The following list gives several partial parameter names from the Graphics Library and the names of the routines to which they belong.

```
gr_r...    GRAF_RUBBERBOX
gr_m...    GRAF_MOVEBOX
gr_s...    GRAF_SHRINKBOX
gr_sl...   GRAF_SLIDEBOX
gr_a...    GRAF_APPLMOUSE
```

Note that a second letter was required to distinguish GRAF_SLIDEBOX from GRAF_SHRINKBOX.

1.4 GEM AES's Position in Memory

GEM AES shares memory space with the following:

o DOS
o GEM VDI
o an application

When a user starts GEM AES, the system has already loaded DOS into memory, followed by GEM VDI. The application takes as much of the remaining memory space as it requires.

The application can be any of the following:

o the GEM Desktop
o a GEM application
o a DOS application

The GEM Desktop application is in fact a GEM application; it uses
no special function calls and does nothing that another GEM ap-
plication cannot do.

The user can start an application (GEM or DOS) from the GEM
Desktop.   When   the current application   terminates, GEM AES
automatically invokes and runs the GEM Desktop unless the user has
already invoked another application. For example, the user can in-
voke the GEM Output application from within another GEM applica-
tion.

Note: The user can also start any GEM application from the DOS
prompt.

An application can use any of three sets of calls, each with a
specific purpose:

o DOS calls are responsible for managing the file system. The
   programmer should be familiar with the material contained in the
   DOS manual.

o GEM VDI calls manage graphics output to the screen or other
   peripheral devices.   The programmer should be familiar with the
   material in the GEM Programmer's Guide, Volume 1: VDI.

o GEM AES calls manage graphics input.   They make possible a
   variety of high-level user interface graphics primitives that are
   used for icons, drop-down menus, dialog boxes, alert messages,
   and windows.

1.5 GEM AES Components

The following GEM AES components occupy space in memory:

o subroutine libraries
o a limited multitasking kernel and dispatcher
o the shell
o a desk accessory buffer
o a menu/alert buffer

1.5.1 Subroutine Libraries

GEM AES's subroutine libraries provide routines for a wide variety
of tasks, including windowing, monitoring the mouse's movement,
displaying system messages and error messages, and drawing objects
on the screen.

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

1.5.2 Limited Multitasking Kernel

The limited multitasking kernel supports the following:

o up to three desk accessory programs (see Section 1.5.2.1) or
   background tasks/processes

o one primary application, GEM or DOS, such as the GEM Desktop ap-
   plication or Wordstar.

o the GEM AES Screen Manager

The function of the limited multitasking kernel is to divide CPU
time between the primary application, background processes, and the
Screen Manager in such a way that the user does not see any
degradation in the performance of the primary application.

1.5.2.1 Desk Accessories

A desk accessory is an application that does not take over the en-
tire display screen. It runs in a specially designed window on top
of the GEM Desktop application or any other GEM application. The
calculator is a typical desk accessory.

The limited multitasking kernel supports as many as six desk acces-
sories in   three   desk accessory programs, called DESK1.ACC,
DESK2.ACC, and DESK3.ACC.   Each desk accessory program can contain
more than one desk accessory.   For example, in the GEM Desktop ap-
plication, the program DESK1.ACC contains both the clock and the
calculator.

The desk accessory programs are loaded into memory using the Over-
lay option of the DOS EXEC function call (documented in the DOS
manual). They remain in memory until the user exits GEM AES.

A desk accessory program is loaded only if, after loading it and
the rest of the GEM software, 128 kilobytes of memory remain avail-
able for the primary application.

If a desk accessory registers with GEM AES (see Section 5.3.6, the
MENU_REGISTER call), it appears in the Desk Menu. The user starts
the desk accessory by choosing it from the menu.

1.5.2.2 Screen Manager

The Screen Manager is a special process that monitors the actions
of the mouse when it is outside the work area of the top (active)
window. The work area is the part of the window exclusive of the
title bar, information line (if any), and border area.

The Screen Manager monitors the user's interaction with the
following:

o the border area of the top window

o the menu bar and drop-down menus

o any other part of the screen, except for the work area of the top
  window

The Screen Manager sends the results of all these user interactions
to the application that is currently running.

When necessary, the Screen Manager is responsible for converting
the mouse form to an arrow (pointer).

1.5.2.3 Dispatcher

The dispatcher is the part of the multitasking kernel that makes
sure that no process dominates CPU time to the exclusion of the
other processes.

The kernel divides CPU time between the primary application, back-
ground processes, and the Screen Manager by assigning each task to
one of two lists: the Ready List and the Not-Ready List.

The Not-Ready List contains all processes that are waiting for one
of the following events:

o a keystroke
o a mouse button press
o mouse movement
o a message
o passage of a time interval

The Ready List contains all processes that are ready to run. A process is ready to run when it is not waiting for one of the events listed above.

The process at the head of the Ready List is the one that is currently running. Any others on the list are waiting to run. The multitasking kernel uses "round-robin scheduling" to run the processes on the Ready List in the order in which they appear on the list.

To prevent a single process from dominating CPU time, the multitasking kernel dispatches at intervals. Dispatching involves two steps:

1. Moving processes from the Not-Ready List to the Ready List, if they now qualify.

2. Moving the currently running process to the end of the Ready List, to give the next process on the list a chance to run.

GEM AES dispatches nonpre-emptively, which means that a process periodically gives up its claim to CPU time and lets the next process run. Dispatching takes place each time the running process makes a GEM AES call.

If a process does not make any other kind of GEM AES call in the course of its normal activity, it should periodically make an EVNT_TIMER call, specifying an interval of 0 (zero) milliseconds. (See the description of the EVNT_TIMER routine, Section 4.3.5.)

At dispatch time, GEM AES's dispatcher first moves any processes that have become ready to run from the Not-Ready List to the end of the Ready List. GEM AES next stops the currently running process and looks to see if any other processes are on the Ready List. If so, the dispatcher moves the current process to the end of the Ready List and starts the next process. If there are no other processes on the Ready List, the current process begins to run again.

This dispatching procedure is repeated with each GEM AES call.

Dispatching also occurs when the running process "blocks," which means the process goes from a ready state to a not-ready state (for example, when it needs to look for mouse movement before continuing).

If no processes are on the Ready List, the multitasking kernel continually checks all processes to see if any can be moved over to the Ready List.

1.5.3   The Shell

The Shell is in fact one of the GEM AES subroutine libraries, but
functionally it runs on top of the limited multitasking kernel.
The Shell invokes GEM AES and DOS applications, causing the
primary application to run.

The following examples illustrate how the Shell starts GEM ap-
plications.   .

o When the user enters the command "GEM", GEM AES is loaded into
  memory, and the primary application is the GEM Desktop. The user
  can start other applications, GEM or DOS, from the GEM Desktop.
  After quitting the GEM Desktop, the user returns to the DOS en-
  vironment.

o When the user enters the command "GEM DRAW", GEM AES is loaded
  into memory, and the primary application is GEM Draw.. After
  quitting GEM Draw, the user returns to the DOS environment.

o When the user enters the command "GEM DRAW /D", GEM 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.

If the user invokes another application from the GEM Desktop, the
GEM Desktop   passes to the Shell the following information about
the new application:

o whether it is graphic or character-based
o whether it is a GEM application or a DOS application
o the name of the directory containing the application

(Most GEM applications are graphic, and most DOS 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, control returns to the Shell.

The Shell determines if it was instructed to start a new applica-
tion. 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.

When this application terminates, the Shell starts the next application, either a user-requested application or (if none was requested) the GEM Desktop.

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.

1.5.4 Desk Accessory Buffer

The desk accessory buffer contains the .ACC files for the GEM AES desk accessories.

As noted above, the multitasking kernel supports as many as three desk accessory programs. The desk accessories remain in memory until the user exits GEM AES.

1.5.5 Menu/Alert Buffer

Drop-down menus and alert boxes (the latter are a special form of dialog box) appear at different times in a GEM application. They always appear layered on top of any windows, icons, or dialogs located in the same area.

When the menu or alert is no longer being displayed, GEM AES redraws the screen from a buffer in which it stores the parts of the screen that were displaced by the menu or alert. The application does not redraw the screen.

This buffer can hold data equal to one-fourth the size of the screen area. Consequently, no single menu or alert can be larger than one-fourth the size of the screen.

GEM AES uses the menu/alert buffer because an application redraw is typically slower than a redraw from the buffer.

1.5.6 X- and Y-Coordinates

Several GEM AES subroutine libraries 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.

End of Section 1

Section 2

Typical GEM AES Calling Routines

2.1 Introduction

This section describes a hypothetical sequence of GEM AES and GEM
VDI calls that might be made by a GEM application. Where needed,
the GEM Desktop application serves as an example of a typical GEM
application.

These calling routines do the following:

o initialize the application

o determine the system's screen resolution

o load the application's resource file

o get the addresses of the application's resources

o display the application's menu bar

o display icons on the desktop

o let the application await user action

o let the user select from a menu

o display a dialog as the result of selecting from a menu

o let the user make a menu selection by pressing a specially desig-
  nated key or combination of keys

o let the user select an icon

o let the user interact with windows

2.2 Initializing an Application

Application initialization consists of three steps:

1. Freeing unneeded memory.

   When an application is first loaded into memory, it should make a
   DOS call 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.

If the application does not make this call, the operating system
will return an error message when the RSRC_LOAD call (described
in Section 2.4) makes its DOS memory allocation request.

2. Initializing internal data structures and setting up GEM AES ar-
   rays.

   The application initializes its internal data structures and sets
   up the following GEM AES 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.
   Examples of these bindings are the sample language bindings in
   this guide and the GEM Programmer's Toolkit..

3. The APPL_INIT call.

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

2.3 Finding Screen Resolution

All of an application's textual or graphic data that is either
device- or language-specific (spoken/written language, not program-
ming language) is contained in the application's resource file.
These materials include the following:

   o text
   o icons
   o menus
   o dialogs
   o forms

All resource files have a .RSC filetype.

Applications usually have at least two resource files, one for a
low-resolution screen (320x200 pixels) and another for a high-
resolution screen (640x400 pixels).

Before it can load the correct resource file, the application needs
to know the system's screen resolution. Two steps are required 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 Programmer's Guide, Volume 1: VDI.

2.4 Loading the Resource File

When the application makes its RSRC_LOAD call, the Resource Library
allocates memory for the resource file, loads it into memory, and
sets the pointer references in the file.

The RSRC_LOAD call also transforms 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, GEM AES
can address two points that are three pixels apart by identifying
their positions as (column 46 + 2 pixels) and (column 46 + 5
pixels).

2.5 Getting Resource Addresses

To get the address of any object contained in its resource file, the
application makes a RSRC_GADDR call.   The application can make
RSRC_GADDR calls at either of the following times:

  o immediately after making the RSRC_LOAD call
  o each time it needs a particular resource

In the first case, 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.

2.6 Displaying the Menu Bar

The application's menu bar is a 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 the index to
   the menu bar.

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

2. The application calls MENU_BAR, passing in the menu bar's address
   and an me_bshow value of 1 (display the menu bar).

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

2.7 Displaying Desktop Icons

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 includes everything but the menu bar and border
area (if any).

To get this information the application makes a WIND_GET call, pas-
sing in values that do the following:

o identify the window as the desktop window (wi_ghandle = 0)

o ask for the window's X, Y, width, and height values (wi_gfield =
  4)

The call returns the work area's X, Y, width, and height values.

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: to their own application windows. The GEM Desktop applica-
tion, however, does display icons (disks and the trash can) in the
desktop window.

2.8 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:

o keystroke
o pressing a mouse button
o mouse movement
o message from a GEM AES process
o passage of a specified period of time

Although an application can wait for one event at a time, most commonly it will make 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: single mouse button. If a mouse has more than one button, a GEM application should look for input from the button on the left. DOS applications can accept input from more than one mouse button.

2.9 **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 into the menu bar, touching the Desk 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 pointer 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 pointer 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 **Desktop Info....**

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

   The message (the predefined GEM AES message MN_SELECTED, described in Section 4.2.5.2) 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 mes-
    sage. It finds the GEM Desktop application and moves it over to
    the Ready List.

7. The EVNT_MULTI call returns an ev_mwhich value to the GEM Desktop
    application. The bit setting in ev_mwhich indicates that a mes-
    sage 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 2.10.

9. The menu title remains highlighted until the requested action is
    complete.   In the case of displaying 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 an me_nnormal value of 1. This call changes the menu
    title from its highlighted state to its normal state.

2.10 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.

     (As noted in Section 2.5, 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 the FORM_DIAL routine, passing in an
     fo_diflag value of 0 (zero), an FMD_START call.

     This call reserves screen space for the dialog.

3.   The application can make a second FORM_DIAL call, this time pas-
     sing in an fo_diflag value of 1, an FMD_GROW call.

     This call draws an expanding rectangle out to the position where
     the dialog's borders will be.

     This call is optional.   Its primary purpose is to produce a
     pleasing visual effect.

4.   The application calls the OBJC_DRAW routine to draw the dialog,
     whose root object is usually a G_BOX object type.

One of the OBJC_DRAW call's input parameters is the address of the dialog's object tree.

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--set up in the RSRC_LOAD call or in the application's code--with the values the dialog now contains. The application notes any changes and acts accordingly.

7. 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 **Desktop Info....,** 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. One example is the DOCUMENT INFORMATION dialog, in which the user will want to save both the document's name and its Read/Write or Read-Only status.

8. The application can call FORM_DIAL again, passing in an fo_diflag value of 2, an FMD_SHRINK call.

   This call draws a shrinking box from the dialog's borders. Like the call that draws the expanding box, this call is optional.

9. The application calls FORM_DIAL the last time, this time passing in an fo_diflag value of 3, an 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.

2.11 Keystroke Menu Selection

GEM AES supports letting the user 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.

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

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

2.12 Selecting an Icon

To select an icon, the user clicks the mouse button once on the
icon.

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 an
   ev_mwhich value with the bit set for a mouse button event.

3. The application makes a GRAF_MKSTATE call to get the mouse poin-
   ter's X- and Y-coordinates.

4. The application makes an OBJC_FIND call, passing in the mouse
   pointer's X- and Y-coordinates from the previous call. The ap-
   plication also passes in the address of the object tree that
   draws the icons on the desktop (if the pointer was in the
   desktop window) or in the application's window.

5. If the OBJC_FIND call reports that the mouse pointer was over an
   icon, the application makes an OBJC_CHANGE call to change the
   icon's ob_state from NORMAL to SELECTED.

If the mouse pointer was not over an icon, the application assumes
that the user intends to select a group of icons by dragging an ex-
panding (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_RUBBERBOX call to draw the
   rubber rectangle that will surround the icons the user wishes to
   select.

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

   The GRAF_RUBBERBOX call's output width and height values
   (gr_rlastwidth and gr_rlastheight) 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 ex-
ample, 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 ap-
pears 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 me_eenable values of 0 (zero)   to disable an item or 1 to enable
it.

2.13 Creating a Window

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

   o title bar
   o move bar
   o size box
   o full box
   o close box
   o arrows, scroll bars, and sliders

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

Making a window appear on the screen actually consists of two steps:
creating the window and opening it. Creating the window defines
what components will be present in the window; opening makes the
created window 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. 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 actu-
ally appears in a window, although from the user's viewpoint, the
calculator and its window are indistinguishable.)

When the GEM Desktop makes a WIND_CREATE call, the bit is on for
(among others) the size box.   This means the size box appears in
the GEM Desktop window's border area, and the user can change the
window's size. In addition, the WIND_CREATE call defines the win-
dow's largest size as the size of the GEM Desktop 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 size box. No size box appears in the window's bor-
der 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 size box) as nineteen characters
wide by thirteen characters high.

When an application makes a WIND_CREATE call, GEM AES returns a win-
dow   handle, a numeric identifier the application uses for all
future GEM AES calls.

2.14 Calculating Work Area or Outer Dimensions

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

o 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 window's work area.

o Using the size and location of the window's work area as input
   parameters, WIND_CALC returns the size and location of the win-
   dow'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.

2.15 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 initial size and location in which the window will open.

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.

2.16 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 virtual 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.

2.17 Sizing a Window

When the user drags the window's size box, GEM 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, GEM 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, GEM 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
it responds to such a request. It can do any of the following: .

o ignore the request

o automatically size the window to the nearest valid size

o display a dialog that informs the user the request is not valid

If an application does not support a particular window function
(like sizing),   it should not in its WIND_CREATE call request the
window control point (like the size box) that supports this func-
tion.

2.18 Rectangle List

An application is only responsible for redrawing and updating the
visible   portion of its windows.   To keep track of the area for
which it is responsible, the application divides the visible por-
tion 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 application keeps a list of these rectangles 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 applica-
tion continues making   these calls until the returned width and
height values for the rectangle are 0 (zero).

2.19 Before Updating a Window

Before it updates a window, an application must notify GEM AES and
any other processes that an update is about to take place.

The application makes a WIND_UPDATE call with a wi_ubegend value of
1, which indicates the beginning of a window update.   This call
freezes the rectangle lists of all windows except the one about to
be updated. The call also prevents menus and alerts from appearing
during window update.

When the update is complete, the application makes a WIND_UPDATE
call with a wi_ubegend value of 0 (zero), which indicates the end
of a window update.   This call frees the frozen rectangle lists,
allowing the other windows to change as required.

2.20 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.

2.21 Making a Window Active

When the user presses the mouse button over a window, the application--if it has made an EVNT_MULTI call that includes a mouse button event--receives a message from the Screen Manager that the button has been pressed.

The application then needs to find out where the 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 pointer's position.

If the window handle is 0 (zero), the mouse pointer is on the GEM Desktop, and the application is not responsible for the mouse button event. The Screen Manager, assuming that the user intends to select desktop icons, draws a rubber box on the GEM Desktop.

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 a code indicating that the window is to be brought to the top.

2.22 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 ap-
plication.   The application does not free the handle until it 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 win-
dowing techniques, as well as descriptions of the individual Window
Library calls.

                         End of Section 2

Section 3

Application Library

3.1 Introduction

GEM AES can extend a single-tasking operating system like GEM DOS to a limited multitasking form that can simultaneously run a single foreground application and several background applications. The foreground application can be one of the following:

o a primary application like a word processing or drawing program
o a desk accessory like a clock/calendar or calculator

Background applications include print spoolers, network communications drivers, and the GEM AES Screen Manager. The GEM AES Screen Manager monitors the activities of the mouse when it is outside the work area of the topmost window. The Screen Manager's responsibilities include the following:

o drop-down menu interaction
o window border manipulation

In the limited multitasking environment, an internal GEM AES dispatcher switches CPU time between the foreground application and the background tasks, while the user switches CPU time between the primary application and the desk accessories.

In limited multitasking systems, an application must first register with the Application Library before it can use the other GEM subroutine libraries. The following sections describe the routines that an application uses to interact with the Application Library.

3.2 Using the Application Library

The Application Library controls access to the other GEM AES sub-
routine   libraries. Because multiple applications use these sub-
routine libraries at the same time, each subroutine must know which
application is requesting a service. The steps in the following se-
quence illustrate 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 the next section) and makes a call to tell GEM 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.

The application should never tamper with the space it allocates for
GEM AES.   The space contains global data structures that are vital
to the successful use of all GEM AES subroutines.

In addition to initialization, Application Library routines are also
responsible   for   message   pipes   (APPL_READ,   APPL_WRITE,   and
APPL_FIND). Section 4 contains a detailed description of message
pipes and other miscellaneous function calls.

3.3 Global Array

All of GEM AES's subroutines use the Global Array as a parameter.
The Global Array is 30 bytes long and contains the following infor-
mation:

```
global(0)      =   ap_version
global(1)      =   ap_count
global(2)      =   ap_id
global(3,4)    =   ap_private
global(5,6)    =   ap_ptree
global(7,8)    =   ap_1resv
global(9,10)   =   ap_2resv
global(11,12)  =   ap_3resv
global(13,14)  =   ap_4resv
```

In the parameter definitions below, the code (G) indicates that GEM
AES supplies the value at the time of the APPL_INIT call. The code
(A) indicates that the application supplies the value prior to the
APPL_INIT call.

o ap_version - (G) A WORD of data identifying the version of GEM AES
   being used.

o ap_count - (G) A WORD of data specifying the maximum number of ap-
   plications this version of GEM AES supports concurrently.

o ap_id - (G) A WORD of data containing a unique application iden-
   tifier that is in effect as long as the application remains in the
   GEM AES application environment.

o ap_private - (A) A LONG data value that is private to the applica-
   tion and can hold any kind of information the application re-
   quires.

o ap_ptree - (A) A LONG address that points to the array of tree ad-
   dresses initialized by the RSRC_LOAD call. This value should in-
   itially be zeroed by the application.

o ap_1resv - (A) A LONG value that is reserved for future use and
   should initially be zeroed by the application.

o ap_2resv - (A) A LONG value that is reserved for future use and
   should initially be zeroed by the application.

o ap_3resv - (A) A LONG value that is reserved for future use and
   should initially be zeroed by the application.

o ap_4resv - (A) A LONG value that is reserved for future use and
   should initially be zeroed by the application.

3.4 Application Library Routines

The Application Library provides the following routines:

o APPL_INIT - initializes a session with the Application Library

o APPL_READ - lets an application read a specified number of bytes
   from a message pipe

o APPL_WRITE - lets an application write a specified number of bytes
   to a message pipe

o APPL_FIND - finds the application identifier of another applica-
   tion in the system

o APPL_TPLAY - plays a piece of a GEM AES recording of the user's
   actions

o APPL_TRECORD - records a set of the user's interactions with GEM
   AES

o APPL_EXIT - exits a session with the Application Library

The following sections describe these routines.

In addition to the Global Array described in Section 3.3, each
Application Library routine has a GEM AES Parameter Block and a
Control Array that contain the following information:

GEM AES Parameter Block
    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

Control Array
    control(0) = op_code
    control(1) = size in WORDS of int_in array
    control(2) = size in WORDS of int_out array
    control(3) = size in LONGS of addr_in array
    control(4) = size in LONGS of addr_out array

In addition, each routine contains some or all of the following ar-
rays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

3.4.1 APPL_INIT

Purpose:

    Initializes the application and establishes a number of internal
GEM AES data structures prior to calls to other Application
Library subroutines.

Parameters:

```
control(0)  =  10
control(1)  =  0
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_out(0)  =  ap_id
```

o ap_id - If APPL_INIT was successful, ap_id is zero or a positive
number. GEM AES places this number in the Global Array, and the
application uses it with future calls to Application Library
routines.

If APPL_INIT was not successful, the value of ap_id is -1. The
application should make no further Application Library calls.

_____

Sample call to C language binding:

ap_id = appl_init();

3.4.2 APPL_READ

Purpose:

   Reads a specified number of bytes from a message pipe.

Parameters:

```
control(0)  =  11
control(1)  =  2
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ap_rid
int_in(1)   =  ap_rlength

int_out(0)  =  ap_rreturn

addr_in(0)  =  ap_rpbuff
```

o ap_rid - the ap_id of the process whose message pipe the applica-
  tion is reading (usually its own)

o ap_rlength - the number of bytes to read from the message pipe

o ap_rreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o ap_rpbuff - address of the buffer that will hold the bytes the ap-
  plication is reading

Sample call to C language binding:

ap_rreturn = appl_read(ap_rid, ap_rlength, ap_rpbuff);

3.4.3 APPL_WRITE

Purpose:

  Writes a specified number of bytes to a message pipe.

Parameters:

```
control(0)  =  12
control(1)  =  2
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ap_wid
int_in(1)   =  ap_wlength

int_out(0)  =  ap_wreturn

addr_in(0)  =  ap_wpbuff
```

o ap_wid - the ap_id of the process to which the application is writing (usually not itself)

o ap_wlength - the number of bytes to write to the message pipe

o ap_wreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o ap_wpbuff - address of the buffer holding the bytes that will be written

---

Sample call to C language binding:

ap_wreturn = appl_write(ap_wid, ap_wlength, ap_wpbuff);

3.4.4 APPL_FIND

Purpose:

Finds the ap_id of another application in the system.

The application must know the ap_id before it can establish com-
munications with the other application.

Parameters:

```
control(0)  =  13
control(1)  =  0
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_out(0)  =  ap_fid

addr_in(0)  =  ap_fpname
```

o ap_fid - The ap_id of the application for which the current ap-
plication is searching.

   -1 - GEM AES could not find the application

o ap_fpname - 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 programmer must fill out the rest of the
string with blank spaces.

Sample call to C language binding:

ap_fid = appl_find(ap_fpname);

3.4.5 APPL_TPLAY

Purpose:

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

Parameters:

```
control(0)  =  14
control(1)  =  2
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ap_tpnum
int_in(1)   =  ap_tpscale

int_out(0)  =  ap_tpreturn

addr_in(0)  =  ap_tpmem
```


o ap_tpnum - the number of user actions to play back

o ap_tpscale - a sliding scale (from 1 to 10,000) determining the
   speed at   which GEM AES plays back the user's actions, for ex-
   ample:

```
 50 = half speed
100 = full speed
200 = twice speed
```

o ap_tpreturn - always equals 1 (one)

o ap_tpmem - the address of the area in memory holding the recording
   of user events that GEM AES will play back

Sample call to C language binding:

ap_tpreturn = appl_tplay(ap_tpmem, ap_tpnum, ap_tpscale);

3.4.6 APPL_TRECORD

Purpose:

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

Each user event uses six bytes in memory, divided into a WORD and a LONG value as follows:

o The WORD contains a code for the event (as defined by the Event Library) that occurred.

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

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

- timer event: The number of milliseconds elapsed.

- 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.

Parameters:

```
control(0)  =  15
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ap_trcount

int_out(0)  =  ap_trreturn

addr_in(0)  =  ap_trmem
```

o ap_trcount - 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.

o ap_trreturn - The number of user events the application recorded.

o ap_trmem - The address of an area in memory where the recorded
  user events will be stored.

───────────────────────────────

Sample call to C language binding:

ap_trreturn = appl_trecord(ap_trmem, ap_trcount);

3.4.7 APPL_EXIT

Purpose:

   Lets the Application Library clean up its environment when an ap-
   plication is done making Application Library calls.

Parameters:

   control(0) = 19
   control(1) = 0
   control(2) = 1
   control(3) = 0
   control(4) = 0

   int_out(0) = ap_xreturn

o ap_xreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

_____

Sample call to C language binding:

ap_xreturn = appl_exit();

                        End of Section 3

Section 4

Event Library

4.1 Introduction

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

o typing on a keyboard

o clicking a mouse button

o moving a mouse

o choosing a menu command

o manipulating a control point on the border of a window

o doing nothing when something is expected (that is, the lack of input)

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

4.2 Using the Event Library

The Event Library lets an application get input from the keyboard and mouse, other programs in the system, and 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 4.2.5), and clock. This type of polling will exhaust the resources of a multitasking system.

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:

o keyboard event - The user presses a key.

o mouse button event - The user presses or releases a mouse button.

o mouse event - The user moves the mouse into or out of a specified rectangle.

o message event - An application receives a message from another process.

o timer event - The preset timer amount expires.

o multiple event - Any combination of the other events.

4.2.1 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.

4.2.2 Keyboard Event

GEM AES recognizes a standard keyboard. The definition of this keyboard is the same as the hardware standard implied in the IBM. PC ROM BIOS documented in the GEM Programmer's Guide, Volume 1: VDI.

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 GEM AES GRAF_MKSTATE function call.

4.2.3 Mouse Button Event

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

A mask word performs a logical AND operation on the bits represen-ting the mouse buttons the application wants to ignore.   For ex-ample, on a three-button mouse, a mask 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 the
desired state   a specified number of times in a set interval. One
occurrence in the interval is referred to as a "click"; two occur-
rences are called a "double-click."

The Event Library returns to the application the number of times the
mouse button entered 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.

4.2.4 Mouse Event

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

o dragging an icon over the desktop
o drawing a rubberband line or rectangle
o 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 ap-
plication can change the mouse form from an arrow to a cross hair
whenever the mouse is inside a certain area of the screen. The ap-
plication waits for the Mouse Event that indicates that the mouse is
inside a certain rectangle on the screen. When the mouse enters the
rectangle, GEM AES notifies the application. The application makes
a GRAF_MOUSE call to change the mouse form to a cross hair, and
then disengages to wait for the mouse to exit the rectangle.

As long as the mouse remains in the rectangle, the application is
inactive. As soon as the mouse exits, GEM AES reactivates the ap-
plication 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 rec-
tangle that is one pixel high and one pixel wide. However, most ap-
plications, 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.

4.2.5 Message Event

The GEM AES programming environment provides a user interface in
which applications use separate overlapping windows. The windows
reside on the physical screen, which looks like a gray desktop 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 GEM AES with
the following:

o a set of menus that appears in the menu bar

o the title that GEM AES places in the top border of the ap-
   plication's window

o the kinds of window control points including the close box, full
   box, and size box to which the application will respond

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

To receive notification of external events, applications use a stan-
dard 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 GEM 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, GEM AES removes the message from the pipe.

4.2.5.1 Predefined GEM AES Messages

GEM AES provides several predefined message types. Each type has a
maximum length of 16 bytes. All the predefined message types define
the first three words in the same way:

o word 0 - A number identifying the message type.

o word 1 - The ap_id of the application that sent the message.

o word 2 - The length of the message, not counting the predefined 16 bytes. If the value of word 2 is not 0 (in other words, if the message is longer than 16 bytes), the application should use an APPL_READ call for the remainder of the message.

4.2.5.2 MN_SELECTED

GEM AES uses this message to notify an application that a user has selected a menu item.

o word 0 = 10
o word 3 = the object index of the menu title selected
o word 4 = the object index of the menu item selected

4.2.5.3 WM_REDRAW

GEM AES uses this message to tell an application that a user has taken an action that requires redrawing part of the work area of its window. The work area is the part of the window exclusive of the title bar, information line (if any), and border area.

o word 0 = 20

o word 3 = the handle of the window to redraw

o word 4 = X-coordinate of the portion of the redraw area to redraw
           (in screen coordinates)

o word 5 = Y-coordinate of the portion of the redraw area to redraw
           (in screen coordinates)

o word 6 = width of the portion of the redraw area to redraw (in
           screen coordinates)

o word 7 = height of the portion of the redraw area to redraw (in
           screen coordinates)

4.2.5.4 WM_TOPPED

GEM AES uses this message to tell the application that the user has requested its window or another application's window be moved to the top (made active).

o word 0 = 21
o word 3 = the handle of the window

4.2.5.5 WM_CLOSED

GEM AES uses this message to tell an application that the user has requested that its window be closed.

o word 0 = 22
o word 3 = the handle of the window

4.2.5.6 **WM_FULLED**

GEM AES uses this message to tell an application that the user has
clicked the   mouse button in the window's full box, either to en-
large the window to its fullest possible size or, if the window is
already full, to return it to its previous size.

o word 0 = 23
o word 3 = the handle of the window

4.2.5.7 **WM_ARROWED**

GEM AES uses this message to tell an application that the user has
clicked in the arrows or scroll bars in the window's border area.

o word 0 = 24
o word 3 = the handle of the window
o 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 row and column actions are invoked when the user clicks on the
arrows. Section 11.2.1, "Components of the Border Area," describes
scrolling in detail.

4.2.5.8 **WM_HSLID**

GEM AES uses this message to tell an application the new position
the user has requested for the horizontal slider.

o word 0 = 25

o word 3 = the handle of the application's window

o word 4 = a number from 0 to 1000, indicating the requested slider
           position

```
0 = leftmost position
1000 = rightmost position
```

4.2.5.9 WM_VSLID

GEM AES uses this message to tell an application the new position
that the user has requested for the vertical slider.

o word 0 = 26

o word 3 = the handle of the application's window

o word 4 = a number from 0 to 1000, indicating the requested slider
          position

```
0 = top position
1000 = bottom position
```


4.2.5.10 WM_SIZED

GEM AES uses this message to give 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.

o word 0 = 27

o word 3 = the handle of the window

o word 4 = the requested X-coordinate (should remain the same as the
          window's current X-coordinate)

o word 5 = the requested Y-coordinate (should remain the same as the
          window's current Y-coordinate)

o word 6 = the requested width

o word 7 = the requested height

4.2.5.11 WM_MOVED

GEM AES uses this message to give an application its window's new
coordinates when the user requests a change in the window's posi-
tion. The coordinates include the window's title bar, information
line (if any), and borders.

o word 0 = 28

o word 3 = the handle of the window

o word 4 = the requested X-coordinate

o word 5 = the requested Y-coordinate

o word 6 = the requested width (should remain the same as the win-
            dow's current width)

o word 7 = the requested height (should remain the same as the win-
           dow's current height)

4.2.5.12 AC_OPEN

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

o word 0 = 40

o word 4 = me_rmenuid: the desk accessory menu item identifier
           returned by the MENU_REGISTER call

4.2.5.13 AC_CLOSE

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

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

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

o word 0 = 41

o word 4 = me_raccmenid: the desk accessory menu item identifier
           returned by the MENU_REGISTER call

4.2.6 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 cal-
culations. However, both of these methods are inefficient in a mul-
titasking system in which processes can make good use of each
other's delay time.

By using Timer Events, GEM 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.

4.3 Event Library Routines

The Event Library provides the following routines:

o EVNT_KEYBD - waits for a keyboard event

o EVNT_BUTTON - waits for a mouse button event

o EVNT_MOUSE - waits for a mouse event

o EVNT_MESAG - waits for a message event

o EVNT_TIMER - waits for a timer event

o EVNT_MULTI - waits for multiple events

o EVNT_DCLICK - sets and gets the speed required for double-clicking

The following sections describe these routines.

Each Event Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

```
GEM AES Parameter Block
    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

Control Array
    control(0)  =  op_code
    control(1)  =  size in WORDS of int_in array
    control(2)  =  size in WORDS of int_out array
    control(3)  =  size in LONGS of addr_in array
    control(4)  =  size in LONGS of addr_out array
```

Global Array
    global(0)   = ap_version
    global(1)   = ap_count
    global(2)   = ap_id
    global(3,4)   = ap_private
    global(5,6)   = ap_ptree
    global(7,8)   = ap_1resv
    global(9,10)   = ap_2resv
    global(11,12) = ap_3resv
    global(13,14) = ap_4resv

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

4.3.1 EVNT_KEYBD

Purpose:

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

Parameters:

```
control(0)  =  20
control(1)  =  0
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_out(0)  =  ev_kreturn
```

o ev_kreturn - the standard keyboard code for the values returned in
  AH and AL, as defined by the IBM PC DOS ROM BIOS

```
          +----------+----------+
          |scan code |ASCII code|
          +----------+----------+
          15        8 7         0
```

_____

Sample call to C language binding:

ev_kreturn = evnt_keybd();

4.3.2 EVNT_BUTTON

Purpose:

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

Parameters:

```
control(0)   =   21
control(1)   =   3
control(2)   =   5
control(3)   =   0
control(4)   =   0

int_in(0)    =   ev_bclicks
int_in(1)    =   ev_bmask
int_in(2)    =   ev_bstate

int_out(0)   =   ev_breturn
int_out(1)   =   ev_bmx
int_out(2)   =   ev_bmy
int_out(3)   =   ev_bbutton
int_out(4)   =   ev_bkstate
```

o ev_bclicks - the number of times the application is waiting for
  the mouse button to enter a particular state (ev_bstate) within a
  preset time

o ev_bmask - the mouse buttons the application is waiting for

  GEM AES can theoretically support 16 mouse buttons.

  In ev_bmask, ev_bstate, and ev_bbutton, the following bits
  represent the buttons:

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

  These parameters use the following bit settings:

  0 - button up
  1 - button down

o ev_bstate - The button state for which the application is waiting.

o ev_breturn - The 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 ev_bclicks.

o ev_bmx - The X-coordinate of the mouse pointer when the user event
occurred.

o ev_bmy - the Y-coordinate of the mouse pointer when the user event
occurred.

o ev_bbutton - The mouse button state when the user event occurred.

o ev_bkstate - The 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:

```
ev_breturn = evnt_button(ev_bclicks, ev_bmask,
                         ev_bstate, &ev_bmx, &ev_bmy,
                         &ev_bbutton, &ev_bkstate);
```

4.3.3 EVNT_MOUSE

Purpose:

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

Parameters:

```
control(0)   =   22
control(1)   =   5
control(2)   =   5
control(3)   =   0
control(4)   =   0

int_in(0)    =   ev_moflags
int_in(1)    =   ev_mox
int_in(2)    =   ev_moy
int_in(3)    =   ev_mowidth
int_in(4)    =   ev_moheight

int_out(0)   =   ev_moresvd
int_out(1)   =   ev_momx
int_out(2)   =   ev_momy
int_out(3)   =   ev_mobutton
int_out(4)   =   ev_mokstate
```

o ev_moflags - flags for the call

   0x0000 - return on entry
   0x0001 - return on exit

o ev_mox   - The X-coordinate of the mouse rectangle in pixel-based
   screen   coordinates.

o ev_moy   - The Y-coordinate of the mouse rectangle in pixel-based
   screen   coordinates.

o ev_mowidth - The width of the mouse rectangle in pixel-based
   screen   coordinates.

o ev_moheight - The height of the mouse rectangle in pixel-based
   screen   coordinates.

o ev_moresvd - RESERVED; value always equals 1 (one).

o ev_momx - The X-coordinate of the mouse pointer when the user
   event occurred.

o ev_momy - The Y-coordinate of the mouse pointer when the user
   event occurred.

o ev_mobutton - The 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

o ev_mokstate - The 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:

```
ev_moresvd = evnt_mouse(ev_moflags, ex_mox, ev_moy,
                        ev_mowidth, ev_moheight,
                        &ev_momx, &ev_momy,
                        &ev_mobutton, &ev_mokstate);
```

4.3.4 EVNT_MESAG

Purpose:

  Notifies GEM 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 mes-
  sages 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.   GEM AES   provides several
  predefined messages, which are described in Section 4.2.5.

Parameters:

  control(0) = 23
  control(1) = 0
  control(2) = 1
  control(3) = 1
  control(4) = 0

  int_out(0) = ev_mgresvd

  addr_in(0) = ev_mgpbuff

o ev_mgresvd - RESERVED; value always equals 1 (one).

o ev_mgpbuff - Address of the buffer where the message will be
   placed. Its size must be 16 bytes.

Sample call to C language binding:

ev_mgresvd = evnt_mesag(ev_mgpbuff);

4.3.5 EVNT_TIMER

Purpose:

  Notifies GEM AES that the application is waiting for a specified
  amount of time to pass.

Parameters:

```
control(0)  =  24
control(1)  =  2
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  ev_tlocount
int_in(1)   =  ev_thicount

int_out(0)  =  ev_tresvd
```

o ev_tlocount - LOW WORD of a LONG value.

o ev_thicount - HIGH WORD of a LONG value.

  Combined, ev_tlocount and ev_thicount are the length of the time
  interval in milliseconds.

o ev_tresvd - RESERVED; value always equals 1 (one).

_____

Sample call to C language binding:

ev_tresvd = evnt_timer(ev_tlocount, ev_thicount);

4.3.6 EVNT_MULTI

Purpose:

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

Parameters:

  control(0)   =   25
  control(1)   =   16
  control(2)   =   7
  control(3)   =   1
  control(4)   =   0

  int_in(0)   =   ev_mflags
  int_in(1)   =   ev_mbclicks
  int_in(2)   =   ev_mbmask
  int_in(3)   =   ev_mbstate
  int_in(4)   =   ev_mm1flags
  int_in(5)   =   ev_mm1x
  int_in(6)   =   ev_mm1y
  int_in(7)   =   ev_mm1width
  int_in(8)   =   ev_mm1height
  int_in(9)   =   ev_mm2flags
  int_in(10)   =   ev_mm2x
  int_in(11)   =   ev_mm2y
  int_in(12)   =   ev_mm2width
  int_in(13)   =   ev_mm2height
  int_in(14)   =   ev_mtlocount
  int_in(15)   =   ev_mthicount

  int_out(0)   =   ev_mwhich
  int_out(1)   =   ev_mmox
  int_out(2)   =   ev_mmoy
  int_out(3)   =   ev_mmobutton
  int_out(4)   =   ev_mmokstate
  int_out(5)   =   ev_mkreturn
  int_out(6)   =   ev_mbreturn

  addr_in(0)   =   ev_mmgpbuff

  Many of the EVNT_MULTI parameters are defined under the other
  Event Library   routines. For example, the parameter ev_mbclicks
  corresponds to the parameter ev_bclicks, defined under the
  EVNT_BUTTON routine.   Unique parameters are defined below.

  The two sets of mouse event parameters, ev_mm1... and ev_mm2...,
  describe two distinct mouse rectangles. These parameters cor-
  respond to the X, Y, width, and height parameters (ev_mox, etc.)
  described under the EVNT_MOUSE routine.

o ev_mflags - the 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

o ev_mwhich - the event(s) in ev_mflags 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

Sample call to C language binding:

```
ev_mwhich = evnt_multi(ev_mflags, ev_mbclicks, ev_mbmask,
                       ev_mbstate, ev_mm1flags,
                       ev_mm1x, ev_mm1y, ev_mm1width,
                       ev_mm1height, ev_mm2flags,
                       ev_mm2x, ev_mm2y, ev_mm2width,
                       ev_mm2height, ev_mmgpbuff,
                       ev_mtlocount, ev_mthicount,
                       &ev_mmox, &ev_mmoy,
                       &ev_mmobutton, &ev_mmokstate,
                       &ev_mkreturn, &ev_mbreturn);
```

4.3.7 EVNT_DCLICK

Purpose:

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

Parameters:

```
control(0)  =  26
control(1)  =  2
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  ev_dnew
int_in(1)   =  ev_dgetset

int_out(0)  =  ev_dspeed
```

o ev_dnew - the new double-click speed the user has selected

   This parameter has integer values from 0 (zero) to 4 that cor-
   respond to the   SLOW-2-3-4-FAST settings of the selection buttons
   in the GEM Desktop's SET PREFERENCES dialog.

o ev_dgetset - the purpose of the call

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

   If the value of ev_dgetset is 0, EVNT_DCLICK disregards the
   ev_dnew value in the call.

o ev_dspeed - the double-click speed, either newly set (ev_dgetset =
   1) or already existing (ev_dgetset = 0)

   This parameter uses the same integer values as ev_dnew.

_____

Sample call to C language binding:

ev_dspeed = evnt_dclick(ev_dnew, ev_dgetset);

<div align="center">End of Section 4</div>

Section 5

Menu Library

5.1 Introduction

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

Each GEM application defines its own menus. When an application is
active (controls the keyboard and mouse), GEM AES displays the tit-
les of its menus in a menu bar at the top of the screen.

To select a menu, the user places the mouse form over the menu's
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.

Figure 5-1 illustrates a typical menu.
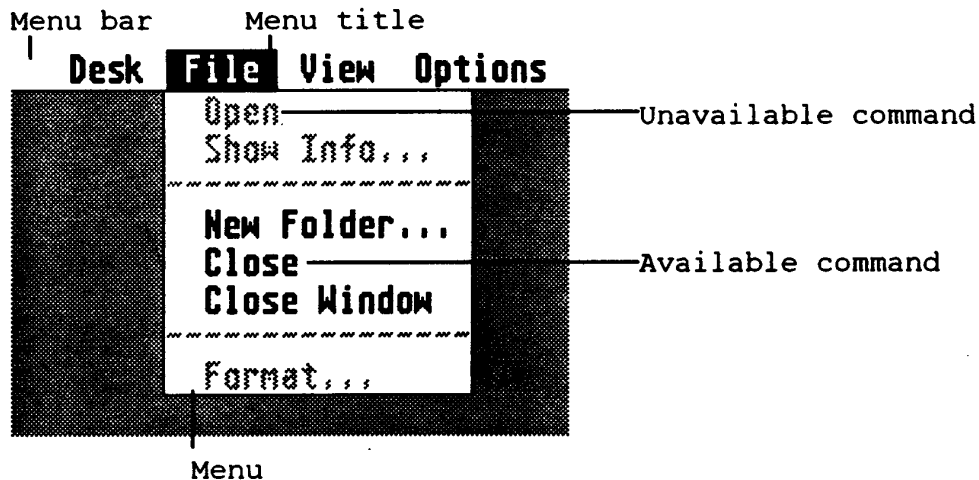
Figure 5-1. Typical Menu

The standard menu item is a text string that names the menu command.
In addition, a menu item can contain either of the following:

o A space for a 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.

o A character identifying a key that produces the same result as
   choosing the menu item.

   The user can press the key instead of displaying the menu and
   choosing an item. The character appears on the menu to identify
   this shortcut.

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 File Menu com-
mand **Open** is enabled if the user has selected an icon and is dis-
abled if the user has not selected an icon.

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

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

o It highlights the menu title in reverse video.

o It displays the menu items in a rectangle that appears 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 accor-
dingly.

When the chosen action has been performed, the calling application
makes a Menu Library call to change the menu title back to its nor-
mal 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 2.11 describes keyboard menu selection in greater
detail.

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

If the user moves the mouse form outside the menu rectangle, the
Screen Manager dehighlights the currently highlighted item (if
any). 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.   First, it
supports   context-sensitive text in menus. An application can
change the wording of its menu items depending on the application's
current state. Second, 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:

o The programmer can create menus that meet the unique requirements
  of individual applications.

o The programmer does not have to be concerned with manipulating the
  interaction between menu and mouse.

o The programmer can modify menus and/or menu items in an efficient
  and timely manner.

o The user can expect all GEM AES application menus to be familiar,
  both in appearance and function.

5.2 Using the Menu Library

The Menu Library is intended to relieve the application of the over-head of handling the interaction between mouse and menu. The Menu Library has the following responsibilities:

o displaying the appropriate menu bar for each active application
o enabling and disabling menu items
o displaying check marks in menus
o returning a highlighted menu title to its normal state
o displaying context-sensitive menu text
o displaying a desk accessory's name on the Desk Menu

The 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 6.3.1. 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 desktop.

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 establish 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:

o a code indicating that it is a menu message
o the object index of the menu title selected
o 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 dehighlight the menu title and waits for the next
message to come through the message pipe.

5.3 Menu Library Routines

The Menu Library uses the following routines:

o MENU_BAR - displays or erases the menu bar

o MENU_ICHECK - displays or erases a check mark next to a menu item

o MENU_IENABLE - displays an enabled item in normal brightness and a
  disabled item in dimmed characters

o MENU_TNORMAL - displays menu title in normal or reverse video

o MENU_TEXT - changes the text of a menu item

o MENU_REGISTER - lets a desk accessory set a text string on the
  Desk Menu and obtain a desk accessory identifier

The following sections describe these routines.

Each Menu Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

GEM AES Parameter Block
      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

Control Array
      control(0) = op_code
      control(1) = size in WORDS of int_in array
      control(2) = size in WORDS of int_out array
      control(3) = size in LONGS of addr_in array
      control(4) = size in LONGS of addr_out array

Global Array
```
    global(0)      =  ap_version
    global(1)      =  ap_count
    global(2)      =  ap_id
    global(3,4)    =  ap_private
    global(5,6)    =  ap_ptree
    global(7,8)    =  ap_1resv
    global(9,10)   =  ap_2resv
    global(11,12)  =  ap_3resv
    global(13,14)  =  ap_4resv
```

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

5.3.1 **MENU_BAR**

Purpose:

Displays or erases the application's menu bar.

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

Parameters:

```
control(0)  =  30
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  me_bshow

int_out(0)  =  me_breturn

addr_in(0)  =  me_btree
```

o me_bshow - a code for whether the application displays the menu bar

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

o me_breturn - a coded return message

0 - an error exists
n (positive integer) - no error exists

o me_btree - the address of the object tree that forms this menu

_____

Sample call to C language binding:

mebreturn = menu_bar(me_btree, me_bshow);

5.3.2 MENU_ICHECK

Purpose:

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

Parameters:

```
control(0)  =  31
control(1)  =  2
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  me_citem
int_in(1)   =  me_ccheck

int_out(0)  =  me_creturn

addr_in(0)  =  me_ctree
```


o me_citem - a number that uniquely identifies this menu item

o me_ccheck - a code for whether the application displays a check
  mark next to the menu item identified by me_citem

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

o me_creturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o me_ctree - the address of the object tree that forms this menu

---

Sample call to C language binding:

me_creturn = menu_icheck(me_ctree, me_citem, me_ccheck);

5.3.3 **MENU_IENABLE**

Purpose:

    Enables or disables a menu item.

    Parameters:

    control(0) = 32
    control(1) = 2
    control(2) = 1
    control(3) = 1
    control(4) = 0

    int_in(0)   = me_eitem
    int_in(1)   = me_eenable

    int_out(0) = me_ereturn

    addr_in(0) = me_etree

    o me_eitem - a number that uniquely identifies this menu item

    o me_eenable - a code for how the application displays a menu item

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

    o me_ereturn - a coded return message

       0 - an error exists
       n (positive integer) - no error exists

    o me_etree - the address of the object tree that forms this menu

    _____

Sample call to C language binding:

me_ereturn = menu_ienable(me_etree, me_eitem, me_eenable);

5.3.4 MENU_TNORMAL

Purpose:

   Displays a menu title in normal or reverse video.

Parameters:

```
control(0)  =  33
control(1)  =  2
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  me_ntitle
int_in(1)   =  me_nnormal

int_out(0)  =  me_nreturn

addr_in(0)  =  me_ntree
```


o me_ntitle - a number unique to this application that identifies
  this menu

o me_nnormal - a code for whether the application displays the menu
  title in normal or reverse video

   0 - reverse video
   1 - normal video

o me_nreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o me_ntree - the address of the object tree that forms this menu

_____

Sample call to C language binding:

```
me_nreturn = menu_tnormal(me_ntree, me_ntitle,
                          me_nnormal);
```

5.3.5 MENU_TEXT

Purpose:

   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.

Parameters:

```
control(0)   =   34
control(1)   =   1
control(2)   =   1
control(3)   =   2
control(4)   =   0

int_in(0)    =   me_titem

int_out(0)   =   me_treturn

addr_in(0)   =   me_ttree
addr_in(1)   =   me_ttext
```

o me_titem - A number that uniquely identifies this menu item.

o me_treturn - A coded return message.

   0 - an error exists
   n (positive integer) - no error exists

o me_ttext - 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.

o me_ttree - The address of the object tree that forms this menu.

Sample call to C language binding:

```
me_treturn = menu_text(me_ttree, me_titem, me_ttext);
```

5.3.6 **MENU_REGISTER**

Purpose:

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

   The Desk Menu can list no more than six desk accessories.

Parameters:

```
control(0)  =  35
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  me_rapid

int_out(0)  =  me_rmenuid

addr_in(0)  =  me_rpstring
```

o me_rapid - the desk accessory's process identifier

   This value is the ap_id returned by the desk accessory's APPL_INIT
   call.

o me_rmenuid - 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

o me_rpstring - the address of the desk accessory's Desk Menu text
   string

   _____

   Sample call to C language binding:

   me_rmenuid = menu_register(me_rapid, me_rpstring);

                  End of Section 5

Section 6

Object Library

6.1 Introduction

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

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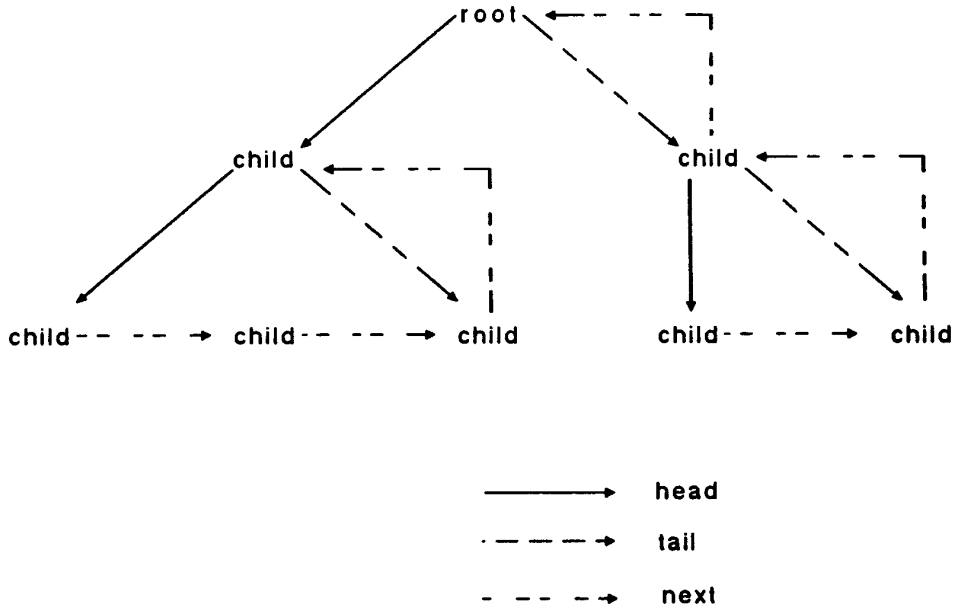
Figure 6-1. Object Tree

An object tree is an array of objects. 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. In addition,
the last child at each level points back to its parent.

6.2 Using the Object Library

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.

```
+-------------------+
|                   |
|   +-------------+ |
|   |             | |
|   |             | |
|   |             | |
|   +-------------+ |
|                   |
|   +-------------+ |
|   |             | |
|   |    TEXT     | |
|   |             | |
|   +-------------+ |
|                   |
+-------------------+
```
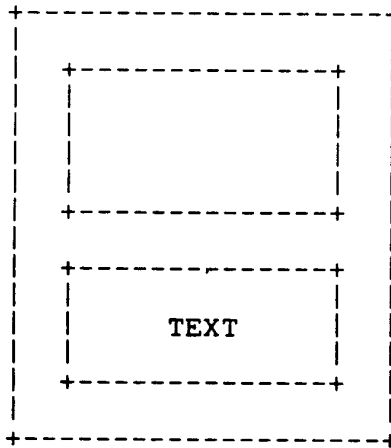
Figure 6-2. On-screen Display

The object tree that defines this display contains three objects:

o The root: the outer box. Its object type is G_BOX. Data for this
  object include the following:

  - X- and Y-coordinates of the upper left corner
  - width and height
  - thickness of the border
  - foreground and background colors

o The first child: the empty inner box. Its object type is also
  G_BOX. Data for this object include the following:

  - X- and Y-coordinates of the upper left corner, relative to the
    parent

  - width and height

  - thickness of the border

  - foreground and background colors

o The second child: the box with TEXT.   Its object type is
   G_BOXTEXT. Data for this object include the following:

   - 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

An application can create an object tree by making separate calls to
the OBJC_ADD routine for each of the root's children. The applica-
tion then calls the OBJC_DRAW routine.   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.

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 that 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 contain its
children.

6.3 Object Library Data Structures

The Object Library uses the following data structures:

o OBJECT structure
o TEDINFO structure
o ICONBLK structure
o BITBLK structure
o USERBLK structure
o 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.

6.3.1 OBJECT Structure

The OBJECT structure contains values that   describe the object, its
relationship to the   other objects in the   tree, and   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.

```
+--------+--------+--------+--------+
|    ob_next      |    ob_head      |
+--------+--------+--------+--------+
|    ob_tail      |    ob_type      |
+--------+--------+--------+--------+
|    ob_flags     |    ob_state     |
+--------+--------+--------+--------+
|              ob_spec             |
+--------+--------+--------+--------+
|     ob_x        |     ob_y        |
+--------+--------+--------+--------+
|    ob_width     |    ob_height    |
+--------+--------+--------+--------+
```

Figure 6-3. OBJECT Structure

o ob_next - a WORD containing the index of the object's next sibling
  in the object tree array

o ob_head - a WORD containing the index of the first child: the head
  of the list of the object's children in the object tree array

o ob_tail - a WORD containing ·the index of the last child: the tail
  of the list of the object's children in the object tree array

o ob_type - a WORD containing the object type (defined in Section
  6.3.7.1). GEM AES ignores the high byte of this WORD.

o ob_flags - a WORD containing the object flags (defined in Section
  6.3.7.2)

o ob_state - a WORD containing the object state (defined in Section
  6.3.7.3)

o ob_spec - a LONG value containing object-specific data.   A
  detailed   description of ob_spec appears at the end of this
  parameter list.

o ob_x - a WORD containing the X-coordinate of the object relative
  to its parent or (for the root object) the screen

o ob_y - a WORD containing the Y-coordinate of the object relative
  to its parent or (for the root object) the screen

o ob_width - a WORD that contains the width of the object in
   pixels

o ob_height - a WORD that contains the height of the object in
   pixels

Depending on the object's type, ob_spec can be a POINTER or any com-
bination of WORD and/or BYTE values that add up to 32 bits. Object
types are described in Section 6.3.7.1.

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. The LOW WORD is
the object color, as defined in Section 6.3.7.4.

The HIGH WORD is broken into two bytes.   For the object type
G_BOXCHAR, the HIGH BYTE of the HIGH WORD is a character. For all
other object types, the HIGH BYTE equals zero.

The LOW BYTE of the HIGH WORD is the thickness of the border of the
object.   This byte can have the following values:

00 = no thickness

1 - 128 (positive values) = inside thickness: inward from the ob-
ject's edge

-1 - (-127) (negative values) = outside thickness: outward from the
object's edge

6.3.2 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.

```
+--------+--------+--------+--------+
|              te_ptext             |
+--------+--------+--------+--------+
|              te_ptmplt            |
+--------+--------+--------+--------+
|              te_pvalid            |
+--------+--------+--------+--------+
|      te_font    |     te_resvd1   |
+--------+--------+--------+--------+
|      te_just    |     te_color    |
+--------+--------+--------+--------+
|     te_resvd2   |   te_thickness  |
+--------+--------+--------+--------+
|     te_txtlen   |    te_tmplen    |
+--------+--------+--------+--------+
```
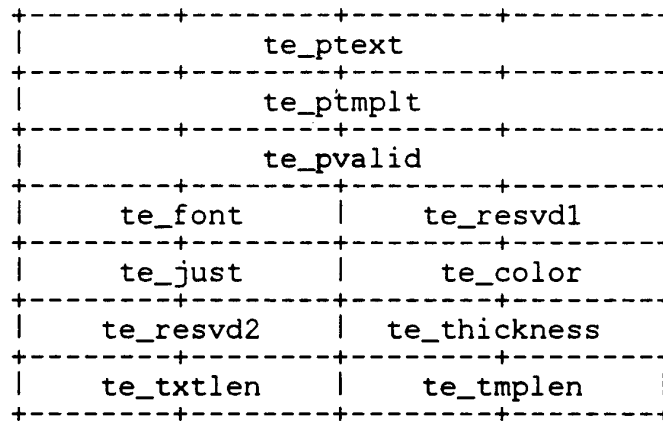
Figure 6-4. TEDINFO Structure

o te_ptext - A POINTER to the actual text.

   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.

o te_ptmplt - A POINTER to a text string template for any further
   data entry.   The editable portion of the field is represented by
   underscores.

o te_pvalid - A 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 DOS filename characters, plus ? * :
   P - allow all valid DOS path name characters, plus \ : ? *
   p - allow all valid DOS path name characters, plus \ :
   X - allow anything

o te_font - A WORD identifying the font used to draw the text.

   3 - system font: used in menus, dialogs, etc.
   5 - small font: used in icons

o te_resvd1 - Reserved for future use.

o te_just - A WORD identifying the type of text justification desired.

  0 - left-justified
  1 - right-justified
  2 - centered

o te_color - A WORD identifying the color and pattern of box-type objects.   See Section 6.3.7.4, "Object Colors."

o te_resvd2 - Reserved for future use.

o te_thickness - A WORD containing the thickness in pixels of the border of the text box.   This WORD can have the   following values:

  00 = no thickness

  1 - 128 (positive values) = inside thickness: inward from the ob-ject's edge

  -1 - (-127) (negative values) = outside thickness: outward from the object's edge

o te_txtlen - A WORD containing the length of the string pointed to by te_ptext.

o te_tmplen - A WORD containing the length of the string pointed to by te_ptmplt.

The following example illustrates how the TEDINFO structure works.

o te_ptext is a string of raw data for a date.   Its value is "061384".

o te_ptmplt, also a string, is a template that shows how to display the data in te_ptext. Its value is "Enter Date: __/__/__".

o te_pvalid is a string of input validation characters.   Its value is "999999".

o The editable text facility merges all the above data into one string, "Enter Date: 06/13/84".

o If the user types "1004", the string becomes "Enter Date: 10/04/84".

o If the user presses the Backspace key after typing "1004", the string becomes "Enter Date: 10/0_/84".

o If te_ptext has no data or not enough data to fill out the temp-
  late, 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/__/__".

6.3.3 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 an ICONBLK structure.

```
+--------+--------+--------+--------+
|                ib_pmask           |
+--------+--------+--------+--------+
|                ib_pdata           |
+--------+--------+--------+--------+
|                ib_ptext           |
+--------+--------+--------+--------+
|     ib_char     |     ib_xchar    |
+--------+--------+--------+--------+
|     ib_ychar    |     ib_xicon    |
+--------+--------+--------+--------+
|     ib_yicon    |     ib_wicon    |
+--------+--------+--------+--------+
|     ib_hicon    |     ib_xtext    |
+--------+--------+--------+--------+
|     ib_ytext    |     ib_wtext    |
+--------+--------+--------+--------+
|     ib_htext    |   0    |   0    |
+--------+--------+--------+--------+
```
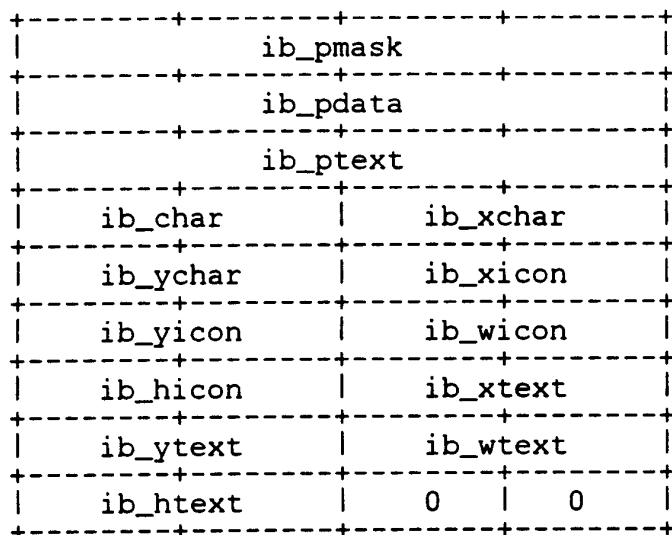
Figure 6-5. ICONBLK Structure

All X, Y, width, and height values for this structure are in pixels.

o ib_pmask - A POINTER to an array of WORDS representing the mask
  bit-image of the icon.

o ib_pdata - A POINTER to an array of WORDS representing the data
  bit-image of the icon.

o ib_ptext - A POINTER to the icon's text.

o ib_char - A WORD containing a character to be drawn in the icon
  (for example, the letter "A" on a floppy disk icon).   The high
  byte contains the foreground color in the high nybble and the
  background color in the low nybble.
  foreground = (ch>>12) & 0x000f   background = (ch>>8) & 0x000f

o ib_xchar - A WORD containing the X-coordinate of ib_char.

o ib_ychar - A WORD containing the Y-coordinate of ib_char.

o ib_xicon - A WORD containing the X-coordinate of the icon.

o ib_yicon - A WORD containing the Y-coordinate of the icon.

o ib_wicon - A WORD containing the width of the icon in pixels.
This value must be divisible by 16.

o ib_hicon - A WORD containing the height of the icon in pixels.

o ib_xtext - A WORD containing the X-coordinate of the icon's text.

o ib_ytext - A WORD containing the Y-coordinate of the icon's text.

o ib_wtext - A WORD containing the width of a rectangle in which the
icon's text will be centered.

o ib_htext - A WORD containing the height of the icon's text in
pixels.

6.3.4 BITBLK Structure

The object type G_IMAGE uses the BITBLK structure to draw bit images
like cursor forms or icons.

```
+--------+--------+--------+--------+
|            bi_pdata               |
+--------+--------+--------+--------+
|    bi_wb        |    bi_hl        |
+--------+--------+--------+--------+
|    bi_x         |    bi_y         |
+--------+--------+--------+--------+
|  bi_color       |                 |
+--------+--------+--------+--------+
```
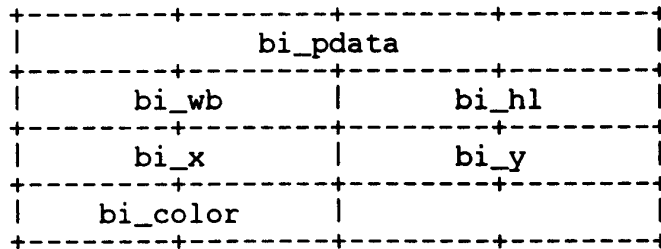
Figure 6-6. BITBLK Structure

o bi_pdata - A POINTER to an array of WORDS containing the bit im-
age.

o bi_wb - A 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.

o bi_hl - A WORD containing the height of the bit block in scan
   lines (pixels).

o bi_x - A WORD containing the source X in bit form, relative to the
   bi_pdata array.

o bi_y - A WORD containing the source Y in bit form, relative to the
   bi_pdata array.

o bi_color - A WORD containing the color GEM AES uses when dis-
   playing the bit-image. See Section 6.3.7.4 for the color values.

6.3.5 USERBLK Structure

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

```
+--------+--------+--------+--------+
|                 ub_code           |
+--------+--------+--------+--------+
|                 ub_parm           |
+--------+--------+--------+--------+
```

Figure 6-7. USERBLK Structure

o ub_code - a POINTER to the routine for drawing and/or changing the
   object

o ub_parm - a LONG value (optionally provided by the application)
   passed as a parameter when the Object Library calls the ap-
   plication's object drawing/changing routine

6.3.6 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.

```
+--------+--------+--------+--------+
|              pb_tree             |
+--------+--------+--------+--------+
|     pb_obj      |   pb_prevstate  |
+--------+--------+--------+--------+
|  pb_currstate   |      pb_x       |
+--------+--------+--------+--------+
|      pb_y       |      pb_w       |
+--------+--------+--------+--------+
|      pb_h       |     pb_xc       |
+--------+--------+--------+--------+
|     pb_yc       |     pb_wc       |
+--------+--------+--------+--------+
|     pb_hc       |pb_parm LOW WORD |
+--------+--------+--------+--------+
| pb_parm HI WORD |
+--------+--------+
```

Figure 6-8. PARMBLK Structure

o pb_tree - A POINTER to the object tree that contains the ap-
   plication-defined object.

o pb_obj - A WORD containing the object index of the application-
   defined object.

o pb_prevstate - A WORD containing the old state of an object to be
   changed.

o pb_currstate - A WORD containing the changed (new) state of an ob-
   ject.

   Note: If pb_prevstate and pb_currstate are the same, the applica-
   tion is drawing the object, not changing it.

o pb_x - A WORD containing the X-coordinate of a rectangle defining
   the location of the object on the physical screen.

o pb_y - A WORD containing the Y-coordinate of a rectangle defining
   the location of the object on' the physical screen.

o pb_w - A WORD containing the width (in pixels) of a rectangle
   defining the size of the object on the physical screen.

o pb_h - A WORD containing the height (in pixels) of a rectangle
   defining the size of the object on the physical screen.

o pb_xc - A WORD containing the X-coordinate of the current clip
   rectangle on the physical screen.

o pb_yc - A WORD containing the Y-coordinate of the current clip
   rectangle on the physical screen.

o pb_wc - A WORD containing the width (in pixels) of the current
    clip rectangle on the physical screen.

o pb_hc - A WORD containing the height (in pixels) of the current
    clip rectangle on the physical screen.

o pb_parm - A LONG value; identical to ab_parm in the USERBLK struc-
    ture.   The   Object Library passes this value to the application
    when it is time for the application to draw or change the object.

6.3.7 Predefined Values

The Object Library routines use the following predefined values:

o object types
o object flags
o object states
o object colors

The following sections define these values.

6.3.7.1 Object Types

```
#define G_BOX        20
#define G_TEXT       21
#define G_BOXTEXT    22
#define G_IMAGE      23
#define G_USERDEF    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
```

Object types are stored in the ob_type section of the OBJECT struc-
ture. All object types are graphic or bitmap object types.

o G_BOX - A graphic box; its ob-spec value contains the object's
    color WORD and thickness.

o G_TEXT - Graphic text; its ob_spec value is a POINTER to a TEDINFO
    structure in which the value of te_ptext points to the actual
    text string as displayed.

o G_BOXTEXT - A graphic box containing graphic text; its ob_spec
  value is a   POINTER to a TEDINFO structure in which the value of
  te_ptext points to the actual text string as displayed.

o G_IMAGE - A graphic bit-image; its ob_spec value is a POINTER to a
  BITBLK structure.

o G_USERDEF - A programmer-defined object; its ob_spec value is a
  POINTER to an USERBLK structure.

o G_IBOX - An "invisible" graphic box; its ob_spec value 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 truly
  invisible. If its border has thickness, it is an outline.

o G_BUTTON - A graphic text object centered in a box; its ob_spec
  value is a POINTER to a null-terminated text string.

o G_BOXCHAR - A graphic box containing a single text character; its
  ob_spec value contains the character, plus the object's color WORD
  and thickness.

o G_STRING - A graphic text object; its ob_spec value is a POINTER
  to a null-terminated text string.

o G_FTEXT - Formatted graphic text; its ob_spec value is a POINTER
  to a TEDINFO structure in which the value of te_ptext points to a
  text string. The text string is merged with the template pointed
  to by te_ptmplt before it is displayed.

o G_FBOXTEXT - A graphic box containing formatted graphic text; its
  ob_spec value is a POINTER to a TEDINFO structure in which the
  value of te_ptext points to a text string. The text string is
  merged with the template pointed to by te_ptmplt before it is dis-
  played.

o G_ICON - An object that describes an icon; its ob_spec value is a
  POINTER to an ICONBLK structure.

o G_TITLE - A graphic text string used in menu titles; its ob_spec
  value is a POINTER to a null-terminated text string.

6.3.7.2 Object Flags

```
#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
```

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

o SELECTABLE - Indicates that the user can select the object.

o DEFAULT - Indicates that the Form Library will examine the object
  if the user enters a carriage return. 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 carriage return to exit the
  form without using the mouse.

o EXIT - Indicates that the Form Library will return control to the
  caller after the exit condition is satisfied. The exit condition
  is satisfied when the user selects the object by clicking on it.

o EDITABLE - Indicates that an object is editable by the user in
  some way.

o 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.

  The "Floppy" and "Hard" buttons in the GEM Desktop application's
  INSTALL DISK DRIVE dialog are examples of radio buttons.

o LASTOB - Indicates that an object is the last object in the object
  tree.

o TOUCHEXIT - Indicates that the Form Library will return control to
  the caller after the exit condition is satisfied. The exit con-
  dition is satisfied when the user presses the mouse button while
  the pointer is over ("touching") the object.

o HIDETREE - Makes a subtree invisible. When the application makes
  an OBJC_DRAW or OBJC_FIND call, the Object Library will not draw
  or find the object or any of its children.

o INDIRECT - Indicates that the value in ob_spec is a pointer to the
  actual value of ob_spec.

6.3.7.3 Object States

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

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

o NORMAL - Indicates that the object is drawn in normal foreground-
  background colors.

o SELECTED - Indicates that the object is highlighted by being drawn
  with its foreground and background colors reversed.

o CROSSED - Indicates that an "X" is drawn in the object. The ob-
  ject must be a box.

o CHECKED - Indicates that the object (typically one containing
  text) is drawn with a check mark.

o DISABLED - Indicates that the object (typically one containing
  text) is drawn faintly.

o OUTLINED - Indicates that an outline appears around a box object.
  This state is used for dialog boxes.

o SHADOWED - Indicates that the object (usually a box) is drawn with
  a drop shadow.

6.3.7.4 Object Colors

```
#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 WHITE     8
#define BLACK     9
#define LRED      10
#define LGREEN    11
#define LBLUE     12
#define LCYAN     13
#define LYELLOW   14
#define LMAGENTA  15
```

Object colors are stored as a WORD in the ob_spec portion of the
OBJECT structure and the te_color portion of the TEDINFO structure.
An L preceding the name of the color (for example, "LGREEN") in-
dicates a light shade of the color.

Figure 6-9 shows the components of the object color WORD.

```
+----+----+-+---+----+
|  a  |  b  |c| d  |  e  |
+----+----+-+---+----+
```

Figure 6-9. Object Color WORD

The high four bits ("a" in Figure 6-9) are the border color, with
values ranging from 0 to 15.

The next four bits ("b") are the text color, with values ranging
from 0 to 15.

The next bit ("c") indicates how text is written.

0 = transparent mode
1 = replace mode

(Transparent and replace mode are defined in the GEM Programmer's
Guide, Volume 1: VDI.)

The next three bits ("d") indicate the object's fill pattern, with
values ranging from 0 to 7.

0 = hollow fill
7 = solid fill
1 - 6 = dither patterns of increasing darkness

The low four bits ("e") are the object's inside color, with values
ranging from 0 to 15.

6.4 Object Library Routines

The Object Library uses the following routines:

o OBJC_ADD - adds an object to an object tree
o OBJC_DELETE - deletes an object from an object tree
o OBJC_DRAW - draws an object or object tree
o OBJC_FIND - determines if the mouse is over an object
o OBJC_OFFSET - computes an object's location relative to the screen
o OBJC_ORDER - changes the order of an object within its tree
o OBJC_EDIT - lets a user edit text in an object
o OBJC_CHANGE - changes an object's state

The following sections describe these routines.

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.

Each Object Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

GEM AES Parameter Block
    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

Control Array
    control(0)   =   op_code
    control(1)   =   size in WORDS of int_in array
    control(2)   =   size in WORDS of int_out array
    control(3)   =   size in LONGS of addr_in array
    control(4)   =   size in LONGS of addr_out array

Global Array
```
    global(0)      =  ap_version
    global(1)      =  ap_count
    global(2)      =  ap_id
    global(3,4)    =  ap_private
    global(5,6)    =  ap_ptree
    global(7,8)    =  ap_1resv
    global(9,10)   =  ap_2resv
    global(11,12)  =  ap_3resv
    global(13,14)  =  ap_4resv
```

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

6.4.1 OBJC_ADD

Purpose:

Adds an object to an object 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.

Parameters:

```
control(0)  =  40
control(1)  =  2
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ob_aparent
int_in(1)   =  ob_achild

int_out(0)  =  ob_areturn

addr_in(0)  =  ob_atree
```

o ob_aparent - the object to whose list of children the child will be added

o ob_achild - the object to add to parent's list of children

o ob_areturn - a coded return message

   0 - an error exists
   n (positive integer) - the object was successfully added

o ob_atree - the address of the object tree containing parent and child

---

Sample call to C language binding:

ob_areturn = objc_add(ob_atree, ob_aparent, ob_achild);

6.4.2 OBJC_DELETE

Purpose:

   Deletes an object from an object tree by unlinking it from its
   parent object.

Parameters:

```
control(0)  =  41
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ob_dlobject

int_out(0)  =  ob_dlreturn

addr_in(0)  =  ob_dltree
```

o ob_dlobject - the object to be deleted

o ob_dlreturn - a coded return message

   0 - an error exists
   n (positive integer) - the object was successfully deleted

o ob_dltree - the address of the object tree that contains the
   object

_____

Sample call to C language binding:

ob_dlreturn = objc_delete(ob_dltree, ob_dlobject);

6.4.3 OBJC_DRAW

Purpose:

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.

This call automatically turns off the mouse while drawing and re-stores its state when finished.

The clip rectangle is defined in the GEM Programmer's Guide, Volume 1: VDI.

Parameters:

```
control(0)  =  42
control(1)  =  6
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ob_drstartob
int_in(1)   =  ob_drdepth
int_in(2)   =  ob_drxclip
int_in(3)   =  ob_dryclip
int_in(4)   =  ob_drwclip
int_in(5)   =  ob_drhclip

int_out(0)  =  ob_drreturn

addr_in(0)  =  ob_drtree
```

o ob_drstartob - the starting object on the tree indexed by ob_drtree

o ob_drdepth - how many levels in the object tree to draw, starting from ob_drstartob

    0 - starting object only
    1 - first level children of starting object
    2 - second level children of starting object [etc.]

o ob_drxclip - the X-coordinate of the clip rectangle

o ob_dryclip - the Y-coordinate of the clip rectangle

o ob_drwclip - the width (in pixels) of the clip rectangle

o ob_drhclip - the height (in pixels) of the clip rectangle

o ob_drreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o ob_drtree - the address of the object tree that contains the ob-
   ject

Sample call to C language binding:

```
ob_drreturn = objc_draw(ob_drtree, ob_drstartob,
                        ob_drdepth, ob_drxclip,
                        ob_dryclip, ob_drwclip,
                        ob_drhclip);
```

6.4.4 OBJC_FIND

Purpose:

Finds an object under the mouse form.

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

Parameters:

```
control(0)  =  43
control(1)  =  4
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ob_fstartob
int_in(1)   =  ob_fdepth
int_in(2)   =  ob_fmx
int_in(3)   =  ob_fmy

int_out(0)  =  ob_fobnum

addr_in(0)  =  ob_ftree
```

o ob_fstartob - the object at which to start the search

o ob_fdepth - how many levels in the object tree to search, starting
  from ob_fstartob

  0 - starting object only
  1 - first-level children of starting object
  2 - second-level children of starting object [etc.]

o ob_fmx - the X-coordinate of the mouse's location

o ob_fmy - the Y-coordinate of the mouse's location

o ob_fobnum - the found object's number in the tree, ranging from 0
  (zero) to n

  -1 - no object found

o ob_ftree - the address of the object tree containing the object
  identified by ob_fstartob

Sample call to C language binding:

```
ob_fobnum = objc_find(ob_ftree, ob_fstartob, ob_fdepth,
                      ob_fmx, ob_fmy);
```

6.4.5 OBJC_OFFSET

Purpose:

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

Parameters:

```
control(0)  =  44
control(1)  =  1
control(2)  =  3
control(3)  =  1
control(4)  =  0

int_in(0)   =  ob_ofobject

int_out(0)  =  ob_ofreturn
int_out(1)  =  ob_ofxoff
int_out(2)  =  ob_ofyoff

addr_in(0)  =  ob_oftree
```

o ob_ofobject - the object whose location is being computed

o ob_ofreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o ob_ofxoff - the X-coordinate of ob_ofobject relative to the screen

o ob_ofyoff - the Y-coordinate of ob_ofobject relative to the screen

o ob_oftree - the address of the object tree containing the object
   identified by ob_ofobject

Sample call to C language binding:

```
ob_ofreturn = objc_offset(ob_oftree, ob_ofobject,
                          &ob_ofxoff, &ob_ofyoff);
```

6.4.6 OBJC_ORDER

Purpose:

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

Parameters:

```
control(0)   =   45
control(1)   =   2
control(2)   =   1
control(3)   =   1
control(4)   =   0

int_in(0)    =   ob_orobject
int_in(1)    =   ob_ornewpos

int_out(0)   =   ob_orreturn

addr_in(0)   =   ob_ortree
```

o ob_orobject - the object to be moved to a new position

o ob_ornewpos - 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

o ob_orreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o ob_ortree - the address of the object tree that contains the ob-
   ject identified by ob_orobject

_____

Sample call to C language binding:

```
ob_orreturn = objc_order(ob_ortree, ob_orobject,
                         ob_ornewpos);
```

6.4.7 OBJC_EDIT

Purpose:

Lets the user edit text in an object.

The object must be of the type G_TEXT or G_BOXTEXT.

Parameters:

```
control(0)  =  46
control(1)  =  4
control(2)  =  2
control(3)  =  1
control(4)  =  0

int_in(0)   =  ob_edobject
int_in(1)   =  ob_edchar
int_in(2)   =  ob_edidx
int_in(3)   =  ob_edkind

int_out(0)  =  ob_edreturn

addr_in(0)  =  ob_edtree
```

o ob_edobject - the object containing the text to be edited

o ob_edchar - the character input by the user

o ob_edidx - the index of the next character position in the raw
   text string

o ob_edkind - 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 for-
       matted 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 6.3.2, "TEDINFO Structure."

o ob_edreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o ob_edtree - the address of the object tree containing the object
  with the text to be edited

_____

Sample call to C language binding:

```
ob_edreturn = objc_edit(ob_edtree, ob_edobject,
                        ob_edchar, ob_edidx,
                        ob_edkind);
```

6.4.8 OBJC_CHANGE

Purpose:

Changes an object's ob_state value. Refer to Section 6.3.7.3, "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.

Parameters:

```
control(0)  =  47
control(1)  =  8
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  ob_cobject
int_in(1)   =  ob_cresvd
int_in(2)   =  ob_cxclip
int_in(3)   =  ob_cyclip
int_in(4)   =  ob_cwclip
int_in(5)   =  ob_chclip
int_in(6)   =  ob_cnewstate
int_in(7)   =  ob_credraw

int_out(0)  =  ob_creturn

addr_in(0)  =  ob_ctree
```

o ob_cobject - The object to be changed.

o ob_cresvd - Reserved; the value must be zero.

o ob_cxclip - The X-coordinate of the clip rectangle.

o ob_cyclip - The Y-coordinate of the clip rectangle.

o ob_cwclip - The width (in pixels) of the clip rectangle.

o ob_chclip - The height (in pixels) of the clip rectangle.

o ob_cnewstate - The ob_state value of the object.

o ob_credraw - A code for whether to redraw the object.

   0 - do not redraw the object
   1 - redraw the object

o ob_creturn - A coded return message.

  0 - an error exists
  n (positive integer) - no error exists

o ob_ctree - The address of the object tree containing the object.

_____

Sample call to C language binding:

ob_creturn = objc_change(ob_ctree, ob_cobject, ob_cresvd,
                         ob_cxclip, ob_cyclip,
                         ob_cwclip, ob_chclip,
                         ob_cnewstate, ob_credraw);

                        End of Section 6

Section 7

Form Library

7.1 Introduction

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

GEM AES's 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. The Form Library collects the user's responses and stores
them in the object tree that describes the form.

In making this high-level interaction between application and user
possible, the Form Library offers several advantages, including the
following:

o The Form Library, and not the application, is responsible for the
  user's interaction with the form.

o 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.

o The Form Library greatly simplifies the programmer's task by
  providing a consistent framework for interaction between the ap-
  plication and the user.

7.1.1 Forms: A Model

A typical form is the product survey illustrated in Figure 7-1. This
kind of form, which might be included with a newly purchased com-
puter, 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. Computer Dealer's Name: _____ |
|                                                           |
|               Address: _____ |
|                                                           |
+-----------------------------------------------------------+
```

Figure 7-1.   Typical Product Survey Form

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

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

7.1.2   GEM AES Forms: The User's View

Forms in GEM AES are essentially the same as printed forms. For ex-
ample, GEM AES uses the same three types of response that appear on
the product survey. However, GEM AES uses its own terminology:

o 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.

o 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.

o editable text - For all responses requiring text entry.

GEM AES needs one piece of information that does not appear ex-
plicitly on the product survey: notification that the user has com-
pleted the form. To provide this information, the programmer desig-
nates at least one box as an exit button and sets the object flag to
EXIT. When the user selects the box, the Form Library completes its
tasks and then passes control back to the application.

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

o OK - Tells the Form Library that the form is complete. The ap-
plication can then act on the user's responses.

o 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 GEM 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 ap-
plication.

7.1.3 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:

o The application needs more information before it can carry out a
command.

o 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 the Form Library itself.

7.1.3.1 Editable Text Fields

Many dialogs have editable text fields. The user can move and edit in these fields with the following keys:

o left- and right-arrow - Move the text cursor left and right within the field.

  If the key is held down, the cursor moves continuously until it reaches the beginning or end of the field.

o down-arrow and Tab - Move the cursor to the next field.

  If the key is held down, the cursor moves continuously from field to field until it reaches the last field.

  Note: 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.

o up-arrow and Backtab - Move the cursor to the previous field.

  To Backtab, the user presses the Tab and Shift keys at the same time.

  If the keys are held down, the cursor moves continuously from field to field until it reaches the first field.

  Note: 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.

o Delete - Deletes the character following the cursor.

  The cursor does not move.

  If held down, the key deletes continuously until all characters following the cursor have been deleted.

o Backspace - Deletes the character to the left of the cursor.

  The cursor and any following text move one character space to the left.

  If held down, the Backspace key moves the cursor and any following text continuously to the left, deleting characters until the cursor reaches the beginning of the field.

o 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 6.3.7.2, "Object Flags"). If no object has been flagged as a DEFAULT object, the

Form Library ignores any Return/Enter keystrokes.

For example, in the dialog in Figure 7-2, the GEM Desktop applica-
tion 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".

o Escape - Clears all characters from the field.

The user can also move through a field by typing a delimiter charac-
ter 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 (see Section 6.3.2, "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 DOS
filename characters, plus ?, *, and :. The period is not a valid
character. If the user types "file.", 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 delimiter with blanks. The user now
sees the following field:

f i l e   . |_ _ _

Similarly, the user can type "9/30/55" into the following date
field:

_ _ / _ _ / _ _

The following field results:

9   / 3 0 / 5 5

7.1.4 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 GEM AES's and GEM AES-based ap-
plications' means of handling error conditions.

Figure 7-2 shows a sample alert.
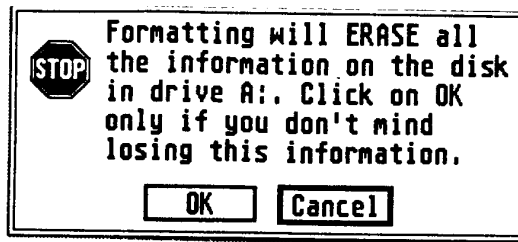
Figure 7-2. Sample GEM AES Alert

The alert is contained in a text string.   In the examples that
follow, the first is the actual alert string. The second describes
the components of the string. 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|Retry]

[<icon#>][<message text>][<exit buttons>]

The components of the string are the following:

o <icon#> - 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

o <message text> - A maximum of 150 8-bit characters for text.

   The alert can have five text lines, with no more than 30 charac-
   ters on each line. In the string, the lines are separated by the
   logical OR symbol [|].

o <exit buttons> - One, two, or three exit buttons, each containing
  no more than 10 text characters. The size of the largest button
  determines the size of all the buttons. So if a button contains
  5 characters, another has 7 and a third 3 then together the but-
  tons contain 21 characters.

   In the string, the exit button text is separated by the logical OR
   symbol.

7.1.4.1 Error Boxes

An error box is a special kind of alert that reports GEMDOS errors
like "File Not Found."

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

7.1.5 GEM AES Forms: The Programmer's View

Structurally, a GEM AES form is an object tree, as described in Sec-
tion 6. In the alert in Figure 7-2, the root of the object tree is
the box that gives the form its structure. The root has three
children-objects, the text and two exit buttons. The three children
have no children-objects of their own.

Figures 7-3 and 7-4 show the OBJECT structure and TEDINFO structure
elements for each part of the form's object tree.

OBJECT structure element	root	text	OK	CANCEL
next	-1	2	3	0
head	1	-1	-1	-1
tail	3	-1	-1	-1
type	G_BOX	G_TEXT	G_BOXTEXT	G_BOXTEXT
flags	NONE	NONE	see below	
state	NORMAL	NORMAL	NORMAL	NORMAL
spec	0x00020007L	0x0L	0x0L	0x0L
X	96	80	368	368
Y	152	16	16	48
width	448	272	64	64
height	96	64	16	16

Figure 7-3. OBJECT Structure Elements

Valid flags for "OK" are the Boolean OR of SELECTABLE and EXIT.

Valid flags for "Cancel" are the Boolean OR of SELECTABLE, EXIT, DEFAULT, and LASTOB.

The X- and Y-coordinates of the root are relative to the physical screen. The X- and Y-coordinates of any children are relative to their parents--in this case, the root.

TEDINFO structure element	root	text	OK	CANCEL
ptext	-	see below	"OK"	"Cancel"
ptmplt	-	NULLPTR	NULLPTR	NULLPTR
pvalid	-	NULLPTR	NULLPTR	NULLPTR
font	-	IBM	IBM	IBM
resvd1	-	0	0	0
just	-	TE_LEFT	TE_CNTR	TE_CNTR
color	-	SYS_FG	SYS_FG	SYS_FG
resvd2	-	0	0	0
thickness	-	0	-2	-2
txtlen	-	0	0	0
tmplen	-	0	0	0

ptext: "Formatting will ERASE all information on the disk in
       drive A:. Click on OK only if you don't mind losing
       this information."

Figure 7-4. TEDINFO Structure Elements

To display the whole form, the Form Library follows the structure of the object tree from root to children, displaying each object in the tree according to the information contained in the object. For a more detailed discussion of displaying object trees, see Section 6.

7.2 Using the Form Library

To display a form, the application calls OBJC_DRAW, passing an index to the object tree for the form. The Object Library displays the form in the application's window. The application then makes a call to the FORM_DO routine, and the Form Library monitors the user's interaction with the form.

When the user selects an exit button, the Form Library returns con- trol to the application. In general, the application identifies the object(s) that cause the Form Library to relinquish control. After regaining control, the application must look at the form, determine if any changes took place, and decide on appropriate action.

By contrast with 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
   12.3.

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 physi-
   cal screen in which the dialog will appear. Nothing else can oc-
   cupy the reserved part of the screen.

4. It calls the FORM_DIAL routine again to draw an expanding box in
   which the dialog will appear. (This call is optional.)

5. It calls the OBJC_DRAW routine to display the dialog. This call
   is described in Section 6.4.

6. It calls the FORM_DO routine to let the user interact with the
   dialog.

7. 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 draw a shrinking box. (This call
   is optional.)

8. It calls the FORM_DIAL routine again, to free the reserved screen
   space and to redraw the screen.

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.

5. After the user selects an exit button, it redraws the screen from
    the menu/alert buffer.

6. It reports the user's exit button selection to the application.

To display an error box, the application calls the FORM_ERROR
routine. The   input parameter for FORM_ERROR is a DOS 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.

7.3 Form Library Routines

The Form Library uses the following routines:

o FORM_DO - Causes the Form Library to monitor the user's inter-
   action with a form.

o FORM_KEYBD - Interact with a form with an already existent
   keyboard event.

o FORM_BUTTON - Interact with a form with an already existent button
   event.

o FORM_DIAL - Reserves or frees the portion of the screen used for
   dialog boxes; draws an expanding or shrinking box.

o FORM_ALERT - Displays an alert box.

o FORM_ERROR - Displays an error box.

o FORM_CENTER - Centers a dialog box on the screen.

Each Form Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

GEM AES Parameter Block
     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

Control Array
     control(0) = op_code
     control(1) = size in WORDS of int_in array
     control(2) = size in WORDS of int_out array
     control(3) = size in LONGS of addr_in array
     control(4) = size in LONGS of addr_out array

Global Array
     global(0)   = ap_version
     global(1)   = ap_count
     global(2)   = ap_id
     global(3,4)   = ap_private
     global(5,6)   = ap_ptree
     global(7,8)   = ap_1resv
     global(9,10) = ap_2resv
     global(11,12) = ap_3resv
     global(13,14) = ap_4resv

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

7.3.1 FORM_DO

Purpose:

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

Parameters:

```
control(0)  =  50
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  fo_dostartob

int_out(0)  =  fo_doreturn

addr_in(0)  =  fo_dotree
```

o fo_dostartob - 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 if the form
  does not contain editable text fields. Passing a value of -1
  makes the text field uneditable.

o fo_doreturn - The number of the object that caused the exit from
  the user's interaction with the form. If the object is a touch
  exit and double clicked, the high order bit will be set.

o fo_dotree - The address of the object tree that draws this
  form.

_____

Sample call to C language binding:

fo_doreturn = form_do(fo_dotree, fo_dostartob);

7.3.2 FORM_DIAL

Purpose:

Depending on the value in fo_diflag, does one of the following:

o reserves a portion of the physical screen for a dialog box

o draws an expanding box in which the dialog will appear

o draws a shrinking box when the user has completed interacting with
   the dialog

o frees the reserved portion of the screen and redraws the screen

Parameters:

```
control(0)   =   51
control(1)   =   9
control(2)   =   1
control(3)   =   0
control(4)   =   0

int_in(0)    =   fo_diflag
int_in(1)    =   fo_dilittlx
int_in(2)    =   fo_dilittly
int_in(3)    =   fo_dilittlw
int_in(4)    =   fo_dilittlh
int_in(5)    =   fo_dibigx
int_in(6)    =   fo_dibigy
int_in(7)    =   fo_dibigw
int_in(8)    =   fo_dibigh

int_out(0)   =   fo_direturn
```

o fo_diflag - The FORM_DIAL action being invoked by the current
   call.

   0 - FMD_START - Reserves screen space for the dialog box.

   1 - FMD_GROW - Draws expanding box from small to large X, Y,
        width, and height (see description of GRAF_GROWBOX routine in
        Chapter 8).

   2 - FMD_SHRINK - Draws shrinking box from large to small X, Y,
        width, and height (see description of GRAF_SHRINKBOX in Chap-
        ter 8).

   3 - FMD_FINISH - Frees screen space reserved by FMD_START; causes
        application to redraw screen.

o fo_dilittlx - The X-coordinate of the box's smallest size.

o fo_dilittly - The Y-coordinate of the box's smallest size.

o fo_dilittlw - The width (in pixels) of the box's smallest size.

o fo_dilittlh - The height (in pixels) of the box's smallest
  size.

o fo_dibigx - The X-coordinate of the box's largest size.

o fo_dibigy - The Y-coordinate of the box's largest size.

o fo_dibigw - The width (in pixels) of the box's largest size.

o fo_dibigh - The height (in pixels) of the box's largest size.

o fo_direturn - A coded return message.

  0 - an error exists
  n (positive integer) - no error exists

_____

Sample call to C language binding:

```
fo_direturn = form_dial(fo_diflag, fo_dix, fo_diy,
                        fo_diw, fo_dih, fo_clipx,
                        fo_clipy, fo_clipw, fo_cliph);
```

7.3.3 FORM_ALERT

Purpose:

  Displays an alert.

  Section 7.2 describes the complete sequence of calls internal to
  FORM_ALERT.

Parameters:

```
control(0)  =  52
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  fo_adefbttn

int_out(0)  =  fo_aexbttn

addr_in(0)  =  fo_astring
```

o fo_adefbttn - the form's DEFAULT exit button (see Section 7.1.3.1)

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

o fo_aexbttn - a number that identifies 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

o fo_astring - the address of the string containing the alert

   * Each alert line must be less than 30 characters.
   * Each button must be less than 10 characters. The largest
     button determines the size of all the buttons.

Sample call to C language binding:

```
fo_aexbttn = form_alert(fo_adefbttn, fo_astring);
```

7.3.4 FORM_ERROR

Purpose:

  Displays an error box.

Parameters:

```
control(0)  =  53
control(1)  =  1
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  fo_enum

int_out(0)  =  fo_eexbttn
```

o fo_enum - the DOS error code

```
2,3,18  This application cannot find the folder you just tried
        to access.
4       This application does not have room to open another
        document. To make room, close any document that you
        do not need.
5       An item with this name already exists in the directory,
        or this item is set to Read-only status.
8,10,11 There is not enough memory for the application you
        just tried to run.
15      The drive you specified does not exist.
```

o fo_eexbttn -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:

fo_eexbttn = form_error(fo_enum);

7.3.5 FORM_CENTER

Purpose:

Centers a dialog box on the screen.

Parameters:

```
control(0)  =  54
control(1)  =  0
control(2)  =  5
control(3)  =  1
control(4)  =  0

int_out(0)  =  fo_cresvd
int_out(1)  =  fo_cx
int_out(2)  =  fo_cy
int_out(3)  =  fo_cw
int_out(4)  =  fo_ch

addr_in(0)  =  fo_ctree
```

o fo_cresvd - [RESERVED]; values equals 1 (one).

o fo_cx - The X-coordinate of the centered object tree containing
   the dialog box.

o fo_cy - The Y-coordinate of the centered object tree containing
   the dialog box.

o fo_cw - The width (in pixels) of the centered object tree con-
   taining the dialog box.

o fo_ch - The height (in pixels) of the centered object tree con-
   taining the dialog box.

o fo_ctree - The address of the object tree that describes the
   dialog.

Sample call to C language binding:

```
fo_cresvd = form_center(fo_ctree, &fo_cx, &fo_cy, &fo_cw,
                        &fo_ch);
```

7.3.6 FORM_KEYBD

Purpose:

> Depending on the input control characters, this function
> returns the next appropriate editable object index.

Parameters:

```
control(0)  =  55
control(1)  =  3
control(2)  =  3
control(3)  =  1
control(4)  =  0

addr_in(0)  =  fo_tree

int_in(0)  =  fo_obj
int_in(1)  =  fo_thechar
int_in(2)  =  fo_nxtobj

int_out(0)  =  fo_keyret
int_out(1)  =  fo_pnxtobj
int_out(2)  =  fo_pchar
```

o fo_tree - The address of the object tree.

o fo_obj - The object index of the editable text field that the ap-
   plication wants to start with.

o fo_nxtobj - Same as fo_obj.

o fo_thechar - The input character.

o fo_pnxtobj - The new selected object index.

o fo_pchar - The input character buffer address.

o fo_keyret - A coded return message.

   0 - a default object is selected.

   n (positive integer) - no error exits.

Sample call to C language binding:

```
fo_keyret = form_keybd(fo_tree, fo_obj, fo_nxtobj, fo_thechar,
                  &fo_pnxtobj, &fo_pchar);
```

7.3.7 FORM_BUTTON

Purpose:

   Handle an already occurred button event inside a dialogue box. It
   includes changing the selected object into reverse field.

Parameters:

```
control(0)  =  56
control(1)  =  2
control(2)  =  2
control(3)  =  1
control(4)  =  0

addr_in(0)  =  fo_tree

int_in(0)  =  fo_obj
int_in(1)  =  fo_clks

int_out(0)  =  fo_butret
int_out(1)  =  fo_pnxtobj
```

o fo_tree - The address of the object tree.

o fo_obj - The index of the object that the mouse is over.

o fo_clks - The number of clicks.

o fo_pnxtobj - The new selected object index.

o fo_butret - A coded return message.

   0 - a default or exit object is selected.

   n (positive integer) - no error exists.

Sample call to C language binding:

```
fo_butret = form_button(fo_tree, fo_obj, fo_clks, &fo_pnxtobj);
```

End of Section 7

Section 8

Graphics Library

8.1 Introduction

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 routines are based on GEM VDI routines that are
fully described in the GEM Programmer's Guide, Volume 1: VDI. GEM
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 GEM AES.

Graphics Library routines also return the GEM VDI handle of the cur-
rent opened screen workstation, change the mouse form to one of a
predefined set or to a form defined by the application, and get in-
formation on the mouse and keyboard.

8.2 Using the Graphics Library

The boxes manipulated by the Graphics Library routines can be used
for a variety of purposes.   In the GEM Desktop application, for ex-
ample, the GRAF_RUBBERBOX routine draws the box that appears when a
user drags with the mouse to select a group of icons.   The
GRAF_GROWBOX routine draws the expanding box that appears when a
user opens an icon, and the GRAF_SHRINKBOX routine draws the box
that appears when the user closes a window.

8.3 Graphics Library Routines

The Graphics Library uses the following routines:

o GRAF_RUBBERBOX - draws a "rubber" box that expands and contracts
  from a fixed point as the mouse moves

o GRAF_DRAGBOX - moves a box, keeping the mouse pointer in the same
  position in the box

o GRAF_MOVEBOX - draws a moving box

o GRAF_GROWBOX - draws an expanding box outline

o GRAF_SHRINKBOX - draws a shrinking box outline

o GRAF_WATCHBOX - watches a box to see if the mouse pointer (with button down) is inside

o GRAF_SLIDEBOX - keeps a sliding box inside its parent box

o GRAF_HANDLE - returns a GEM VDI handle for the opened screen workstation that the GEM AES libraries use

o GRAF_MOUSE - lets an application change the mouse form to one of a predefined set or to an application-defined form

o GRAF_MKSTATE - returns the current mouse location, mouse button state, and keyboard state

The following sections describe these routines.

Each Graphics Library routine has a GEM AES Parameter Block, Control Array, and Global Array that contain the following information:

GEM AES Parameter Block
    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

    control(0) = op_code
    control(1) = size in WORDS of int_in array
    control(2) = size in WORDS of int_out array
    control(3) = size in LONGS of addr_in array
    control(4) = size in LONGS of addr_out array

Global Array
    global(0)   = ap_version
    global(1)   = ap_count
    global(2)   = ap_id
    global(3,4)   = ap_private
    global(5,6)   = ap_pname
    global(7,8)   = ap_1resv
    global(9,10)   = ap_2resv
    global(11,12) = ap_3resv
    global(13,14) = ap_4resv

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter in this array is a POINTER.

8.3.1 GRAF_RUBBERBOX

Purpose:

Draws a "rubber box."

The position of the box's upper left corner is fixed, but by drag-
ging 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.

Parameters:

```
control(0)   =   70
control(1)   =   4
control(2)   =   3
control(3)   =   0
control(4)   =   0

int_in(0)    =   gr_rx
int_in(1)    =   gr_ry
int_in(2)    =   gr_rminwidth
int_in(3)    =   gr_rminheight

int_out(0)   =   gr_rreturn
int_out(1)   =   gr_rlastwidth
int_out(2)   =   gr_rlastheight
```

o gr_rx - the box's X-coordinate

o gr_ry - the box's Y-coordinate

o gr_rminwidth - the box's smallest possible width in pixels

o gr_rminheight - the box's smallest possible height in pixels

o gr_rreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o gr_rlastwidth - the width of the box when the user released the
   mouse button

o gr_rlastheight - the height of the box when the user released the
   mouse button

Sample call to C language binding:

```
gr_rreturn = graf_rubberbox(gr_rx, gr_ry,
                            gr_rminwidth, gr_rminheight,
                            &gr_rlastwidth,
                            &gr_rlastheight);
```

8.3.2 GRAF_DRAGBOX

Purpose:

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

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

Parameters:

```
control(0)   =   71
control(1)   =   8
control(2)   =   3
control(3)   =   0
control(4)   =   0

int_in(0)    =   gr_dwidth
int_in(1)    =   gr_dheight
int_in(2)    =   gr_dstartx
int_in(3)    =   gr_dstarty
int_in(4)    =   gr_dboundx
int_in(5)    =   gr_dboundy
int_in(6)    =   gr_dboundw
int_in(7)    =   gr_dboundh

int_out(0)   =   gr_dreturn
int_out(1)   =   gr_dfinishx
int_out(2)   =   gr_dfinishy
```

o gr_dwidth - the width in pixels of the box being dragged

o gr_dheight - the height in pixels of the box being dragged

o gr_dstartx - the box's starting X-coordinate

o gr_dstarty - the box's starting Y-coordinate

o gr_dboundx - the X-coordinate of the boundary rectangle

o gr_dboundy - the Y-coordinate of the boundary rectangle

o gr_dboundw - the width in pixels of the boundary rectangle

o gr_dboundh - the height in pixels of the boundary rectangle

o gr_dreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o gr_dfinishx - the box's X-coordinate when the user released the
  mouse button

o gr_dfinishy - the box's Y-coordinate when the user released the
  mouse button

_____

Sample call to C language binding:

```
gr_dreturn = graf_dragbox(gr_dwidth, gr_dheight,
                          gr_dstartx, gr_dstarty,
                          gr_dboundx, gr_dboundy,
                          gr_dboundw, gr_dboundh,
                          &gr_dfinishx, &gr_dfinishy);
```

8.3.3 GRAF_MOVEBOX

Purpose:

Draws a box moving from one position to another.

The box's size does not change.

Parameters:

```
control(0)  =  72
control(1)  =  6
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  gr_mwidth
int_in(1)   =  gr_mheight
int_in(2)   =  gr_msourcex
int_in(3)   =  gr_msourcey
int_in(4)   =  gr_mdestx
int_in(5)   =  gr_mdesty

int_out(0)  =  gr_mreturn
```

o gr_mwidth - the box's width in pixels

o gr_mheight - the box's height in pixels

o gr_msourcex - the box's X-coordinate, in its initial position

o gr_msourcey - the box's Y-coordinate, in its initial position

o gr_mdestx - the box's X-coordinate, in its final position

o gr_mdesty - the box's Y-coordinate, in its final position

o gr_mreturn - a coded return message

```
0 - an error exists
n (positive integer) - no error exists
```

Sample call to C language binding:

```
gr_mreturn = graf_movebox(gr_mwidth, gr_mheight,
                          gr_msourcex, gr_msourcey,
                          gr_mdestx, gr_mdesty);
```

8.3.4 GRAF_GROWBOX

Purpose:

 Draws an expanding box outline.

Parameters:

```
control(0)   =   73
control(1)   =   8
control(2)   =   1
control(3)   =   0
control(4)   =   0

int_in(0)    =   gr_gstx
int_in(1)    =   gr_gsty
int_in(2)    =   gr_gstwidth
int_in(3)    =   gr_gstheight
int_in(4)    =   gr_gfinx
int_in(5)    =   gr_gfiny
int_in(6)    =   gr_gfinwidth
int_in(7)    =   gr_gfinheight

int_out(0)   =   gr_greturn
```

o gr_gstx - the box's X-coordinate, in its initial size

o gr_gsty - the box's Y-coordinate, in its initial size

o gr_gstwidth - the box's initial width in pixels

o gr_gstheight - the box's initial height in pixels

o gr_gfinx - the box's X-coordinate, in its final size

o gr_gfiny - the box's Y-coordinate, in its final size

o gr_gfinwidth - the box's final width in pixels

o gr_gfinheight - the box's final height in pixels

o gr_greturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

Sample call to C language binding:

```
gr_greturn = graf_growbox(gr_gstx, gr_gsty, gr_gstwidth,
                          gr_gstheight, gr_gfinx,
                          gr_gfiny, gr_gfinwidth,
                          gr_gfinheight);
```

8.3.5 GRAF_SHRINKBOX

Purpose:

  Draws a shrinking box outline.

Parameters:

```
control(0)  =  74
control(1)  =  8
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  gr_sfinx
int_in(1)   =  gr_sfiny
int_in(2)   =  gr_sfinwidth
int_in(3)   =  gr_sfinheight
int_in(4)   =  gr_sstx
int_in(5)   =  gr_ssty
int_in(6)   =  gr_sstwidth
int_in(7)   =  gr_sstheight

int_out(0)  =  gr_sreturn
```

o gr_sfinx - the box's X-coordinate, in its final size

o gr_sfiny - the box's Y-coordinate, in its final size

o gr_sfinwidth - the box's final width in pixels

o gr_sfinheight - the box's final height in pixels

o gr_sstx - the box's X-coordinate, in its initial size

o gr_ssty - the box's Y-coordinate, in its initial size

o gr_sstwidth - the box's initial width in pixels

o gr_sstheight - the box's initial height in pixels

o gr_sreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

Sample call to C language binding:

```
gr_sreturn = graf_shrinkbox(gr_sfinx, gr_sfiny,
                            gr_sfinwidth,
                            gr_sfinheight, gr_sstx,
                            gr_ssty, gr_sstwidth,
                            gr_sstheight);
```

8.3.6 GRAF_WATCHBOX

Purpose:

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 gr_winstate and gr_woutstate are defined in Section 6.3.7.3.

Parameters:

```
control(0)  =  75
control(1)  =  4
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  [reserved]
int_in(1)   =  gr_wobject
int_in(2)   =  gr_winstate
int_in(3)   =  gr_woutstate

int_out(0)  =  gr_wreturn

addr_in(0)  =  gr_wptree
```

o gr_wobject - the index of the object in the tree

o gr_winstate - the 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
```

o gr_woutstate - the 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
```

o gr_wreturn - the mouse pointer's position when the button was released

    0 - outside the box
    1 - inside the box

o gr_wptree - the address of the object tree containing the box

---

Sample call to C language binding:

```
gr_wreturn = graf_watchbox(gr_wptree, gr_wobject,
                           gr_winstate, gr_woutstate);
```

8.3.7 GRAF_SLIDEBOX

Purpose:

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.

Parameters:

```
control(0)   =   76
control(1)   =   3
control(2)   =   1
control(3)   =   1
control(4)   =   0

int_in(0)    =   gr_slparent
int_in(1)    =   gr_slobject
int_in(2)    =   gr_slvh

int_out(0)   =   gr_slreturn

addr_in(0)   =   gr_slptree
```

o gr_slparent - The index of the parent in the tree.

o gr_slobject - The index of the object (the slider) in the tree.

o gr_slvh - A code for the direction of the slider's movement.

   0 - horizontal
   1 - vertical

o gr_slreturn - The position of the center of the slider relative to its parent. The value can range from 0 to 1000.

   if gr_slvh = 0:   0 = left
                   1000 = right

   if gr_slvh = 1:   0 = top
                   1000 = bottom

o gr_slptree - The address of the object tree containing slider and parent.

Sample call to C language binding:

```
gr_slreturn = graf_slidebox(gr_slptree, gr_slparent,
                            gr_slobject, gr_slvh);
```

8.3.8 GRAF_HANDLE

Purpose:

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

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

Parameters:

```
control(0)  =  77
control(1)  =  0
control(2)  =  5
control(3)  =  0
control(4)  =  0

int_out(0)  =  gr_handle
int_out(1)  =  gr_hwchar
int_out(2)  =  gr_hhchar
int_out(3)  =  gr_hwbox
int_out(4)  =  gr_hhbox
```

o gr_handle - the GEM VDI handle

o gr_hwchar - the width (in pixels) of a character cell in the system font used in menus and dialogs

o gr_hhchar - the height (in pixels) of a character cell in the system font used in menus and dialogs

o gr_hwbox - the width (in pixels) of a square box large enough to hold a system font character

o gr_hhbox - the height (in pixels) of a square box large enough to hold a system font character

---

Sample call to C language binding:

```
gr_handle = graf_handle(&gr_hwchar, &gr_hhchar,
                        &gr_hwbox, &gr_hhbox);
```

8.3.9 GRAF_MOUSE

Purpose:

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

Note: The application selects or defines the mouse form that ap-
pears 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 rec-
tangle equal to the active window's work area, to detect mouse
form exit and entry.

Parameters:

control(0) = 78
control(1) = 1
control(2) = 1
control(3) = 1
control(4) = 0

int_in(0)   = gr_monumber

int_out(0) = gr_moreturn

addr_in(0) = gr_mofaddr

o gr_monumber - 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 gr_mofaddr
    256 - hide mouse form
    257 - show mouse form

o gr_moreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o gr_mofaddr - the address of a 35-word buffer that fits the mouse
  form definition block specified in the GEM Programmer's Guide,
  Volume 1: VDI.

Sample call to C language binding:

gr_moreturn = graf_mouse(gr_monumber, gr_mofaddr);

8.3.10 GRAF_MKSTATE

Purpose:

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

Parameters:

```
control(0)  =  79
control(1)  =  0
control(2)  =  5
control(3)  =  0
control(4)  =  0

int_out(0)  =  gr_mkresvd
int_out(0)  =  gr_mkmx
int_out(0)  =  gr_mkmy
int_out(0)  =  gr_mkmstate
int_out(0)  =  gr_mkkstate
```

o gr_mkresvd - [RESERVED]; value always equals 1 (one).

o gr_mkmx - The X-coordinate of the mouse's current location.

o gr_mkmy - The Y-coordinate of the mouse's current location.

o gr_mkmstate - The 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

o gr_mkkstate - 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:

gr_mkresvd = graf_mkstate(&gr_mkmx, &gr_mkmy,
                          &gr_mkmstate,
                          &gr_mkkstate);

End of Section 8

Section 9

Scrap Library

9.1 Introduction

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

A standard data interchange format offers several advantages, in-
cluding the following:

o Users can assemble an integrated set of independently developed
  applications.

o 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 tem-
porary 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:

o By cutting, the user deletes the data from the source ap-
  plication's data space.

o 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 over-
writes the current contents of the clipboard.

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

9.2 Using the Scrap Library

GEM AES's Scrap Library supports the following interactions:

o writing the name of a scrap directory to the clipboard
o reading the name of a scrap directory from the clipboard
o managing the use of the disk as an extensible scrap area

GEM 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, GEM AES must define standard data types in which scrap may be stored.

GEM AES supports the following standard data types. Each is identified by a filetype in parentheses.

o ASCII text strings (.TXT)
o spreadsheet data (.DIF)
o metafile - GEM VDI-type graphic images (.GEM)
o bit-images - GEM VDI standard forms (.IMG)

All GEM AES programs should at least be able to read and write ASCII data. Metafiles are described in the GEM Programmer's Guide, Volume 1: VDI.

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.

To be accessible to a variety of applications, a piece of scrap might appear on the clipboard in several data types. The application can find and/or manipulate the actual size, contents, and existence of scrap by making the following DOS file system calls:

o search
o create
o open
o read
o write
o close
o delete
o get filesize

9.3 Scrap Library Routines

The Scrap Library uses the following routines:

o SCRP_READ - reads the scrap directory currently stored on the clipboard

o SCRP_WRITE - writes the scrap directory to the clipboard

The following sections describe these routines.

Each Scrap Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

GEM AES Parameter Block
     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

Control Array
     control(0) = op_code
     control(1) = size in WORDS of int_in array
     control(2) = size in WORDS of int_out array
     control(3) = size in LONGS of addr_in array
     control(4) = size in LONGS of addr_out array

Global Array
     global(0)   = ap_version
     global(1)   = ap_count
     global(2)   = ap_id
     global(3,4)   = ap_private
     global(5,6)   = ap_ptree
     global(7,8)   = ap_1resv
     global(9,10)   = ap_2resv
     global(11,12) = ap_3resv
     global(13,14) = ap_4resv

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

9.3.1 SCRP_READ

Purpose:

  Reads the current scrap directory on the clipboard.

Parameters:

  control(0) = 80

  control(1) = 0
  control(2) = 1
  control(3) = 1
  control(4) = 0

  int_out(0) = sc_rreturn

  addr_in(0) = sc_rpscrap

o sc_rreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o sc_rpscrap - the address of the buffer into which the scrap direc-
  tory will be copied

Sample call to C language binding:

sc_rreturn = scrp_read(sc_rpscrap);

9.3.2 SCRP_WRITE

Purpose:

   Writes a new scrap directory to the clipboard, as a result of a
   CUT or COPY command from the user.

Parameters:

   control(0)   = 81
   control(1)   = 0
   control(2)   = 1
   control(3)   = 1
   control(4)   = 0

   int_out(0)   = sc_wreturn

   addr_in(0)   = sc_wpscrap

o sc_wreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o sc_wpscrap - the address of the buffer from which the new scrap
   directory will be copied to the clipboard

_____

Sample call to C language binding:

sc_wreturn = scrp_write(sc_wpscrap);

                         End of Section 9

10.1 Introduction

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:

o 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.

o The application displays a directory, and the user types a new or existing filename.

o 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 dialog box called the File Selector (or "Item Selector") appears on the screen. See Figure 10-1.
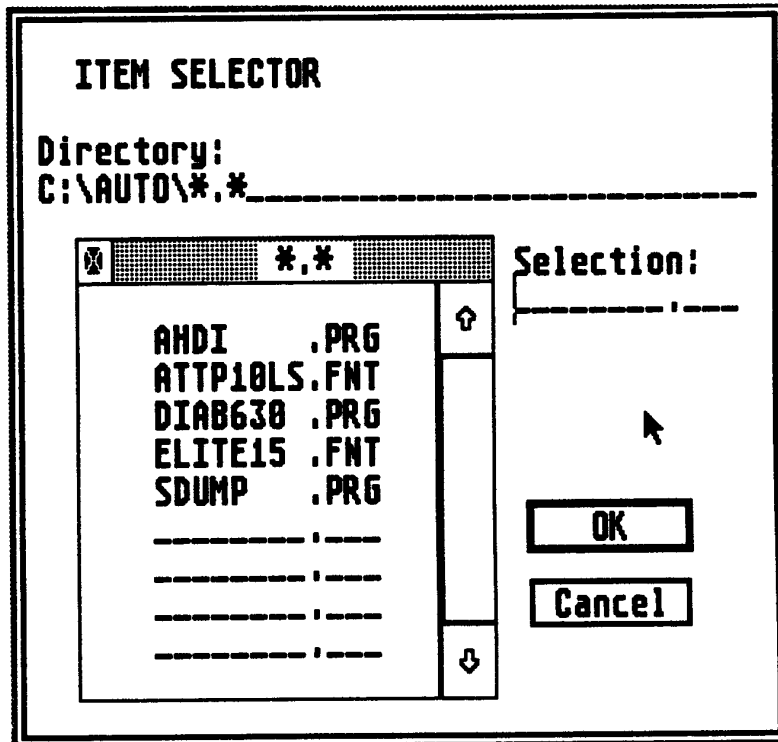
Figure 10-1. File Selector Dialog Box

The File Selector dialog box displays the name of the current direc-
tory (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.

10.2 Using the File Selector Library

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:

    o 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 Section 11.2.1.

o 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 specifica-
  tion 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.   GEM 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 Selec-
   tor dialog box 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 "Selec-
   tion:", 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:

   o the filename that was selected or entered

   o the current directory and wildcard specification

   o the way in which the user exited the File Selector dialog box
     (OK or Cancel)

6. If the user selected OK, the application continues on with the
   filename that was selected or entered.

10.3 File Selector Library Routine

The File Selector Library uses the following routine:

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

This File Selector Library routine has a GEM AES Parameter Block,
Control Array, and Global Array that contain the following informa-
tion:

GEM AES Parameter Block
    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

Control Array
    control(0) = op_code
    control(1) = size in WORDS of int_in array
    control(2) = size in WORDS of int_out array
    control(3) = size in LONGS of addr_in array
    control(4) = size in LONGS of addr_out array

Global Array
    global(0)   = ap_version
    global(1)   = ap_count
    global(2)   = ap_id
    global(3,4)   = ap_private
    global(5,6)   = ap_ptree
    global(7,8)   = ap_1resv
    global(9,10) = ap_2resv
    global(11,12) = ap_3resv
    global(13,14) = ap_4resv

Global Array parameters are described in Section 3.

This routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
   this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
   in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
   in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
   in this array is a POINTER.

10.3.1 FSEL_INPUT

Purpose:

  Displays the File Selector dialog box and monitors the user's in-
  teraction with it.

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

Parameters:

```
control(0)  =  90
control(1)  =  0
control(2)  =  2
control(3)  =  2
control(4)  =  0

int_out(0)  =  fs_ireturn
int_out(1)  =  fs_iexbutton

addr_in(0)  =  fs_iinpath
addr_in(1)  =  fs_iinsel
```

o fs_ireturn - A coded return message.

  0 - an error exists
  n (positive integer) - no error exists

o fs_iexbutton - A code identifying the exit button selected by the
  user.

  0 - Cancel
  1 - OK

o fs_iinpath - The address of the buffer that holds the initial
  directory   specification displayed in the File Selector dialog
  box. This buffer will also hold the directory specification that
  was in the File Selector dialog box when the user selected OK or
  Cancel.

o fs_iinsel - The address of the buffer that holds the initial
  selection displayed in the File Selector dialog box. This buffer
  will also hold the selection that was in the File Selector dialog
  box when the user selected OK or Cancel.

Sample call to C language binding:

```
fs_ireturn = fsel_input(fs_iinpath, fs_iinsel,
                       &fs_iexbutton);
```

End of Section 10

Section 11

Window Library

11.1 Introduction

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

o the desktop window
o application windows

While the user is in the GEM AES environment, the desktop window is
always present on the screen. It contains the menu bar and the
desktop surface.   The desktop window is owned by the primary ap-
plication and serves as the   backdrop for the application's other
windows.

GEM applications use application windows whenever they need to
present data on the screen. In addition, GEM AES supports overlap-
ping 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.

Additional features of application windows include the following:

o They let the user view help information and the application data
  area at the same time.

o They let the user cut and paste data between applications.

o They make it possible to display status information from several
  background activities--for example, compiling, sorting, and
  transferring data.

GEM AES supports a maximum of eight (8) windows at a time. Of these
eight windows, the desktop window is always identified by a window
handle (identifier) of zero. A GEM application can use all eight
available windows, although this could result in no windows being
left available for desk accessories. An application can avoid this
problem in the manner adopted by the GEM Desktop application, which
sets a limit of four (4) windows for itself, leaving the remaining
four windows available for desk accessories.

11.2 Using the Window Library

An application window is made up of two components: a rectangular
work area surrounded by a border area.

Figure 11-1 illustrates the components of a typical window.

Close box   Title Bar   Information line   Full box
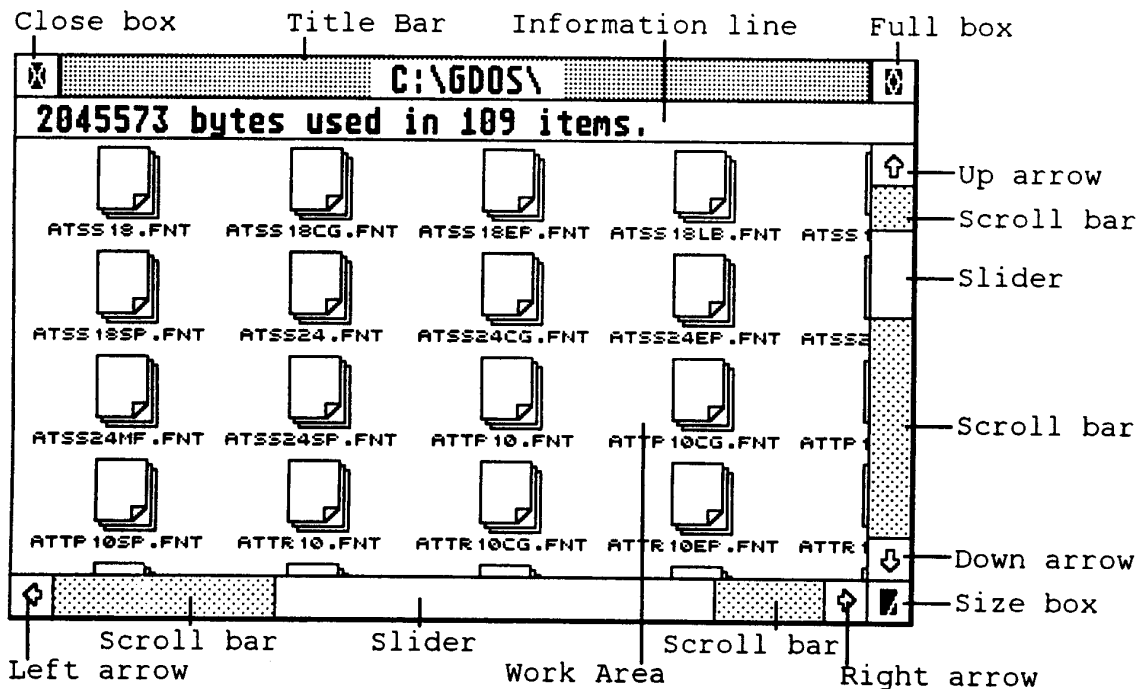
Figure 11-1. Components of a Typical Window

The work area has the following characteristics:

o It is the portion of the window's area that is available for use
  by the application.

o All user interactions inside the work area are managed by the ap-
  plication.

o 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:

o The border area can have several different components.   An ap-
  plication determines which components will appear in the border
  area but does not control them.

o GEM AES's Screen Manager displays the contents of each window's
  border and manages all user interactions with the border area.

11.2.1 Components of the Border Area

A window's border area has some or all of the following:

o title bar

  All windows have a title bar, which appears across the top of the
  window. It can contain a maximum of 80 text characters. The ap-
  plication provides this text as the name of the window.

o information line

  The application determines if its window has an information line.

  The information line appears directly under the title bar and can
  contain a maximum of 80 text characters.   The application
  provides this text to describe the contents of the window.

The remaining features of the border are known as "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.

o close box

  The close box is located at the left end of the title bar.

  When the user clicks the mouse on the close box, GEM AES sends the
  application a message that the user wants the window closed.

o full box

  The full box is located at the right end of the title bar.

  The full box acts as a toggle with which the user can change the
  window from its current size and location on the desktop to its
  greatest possible size, and back again.   In their greatest pos-
  sible size, most windows fill all of the desktop but the menu bar.

o move bar

   The move bar, when present, occupies the same space as the title
   bar. The move bar is invisible and to the user is indistinguish-
   able from the title bar.

   When the user presses the mouse button on the move bar, GEM 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 GEM
   AES sends the application a message that the user wants the window
   moved to the location indicated by the outline's last position.

o size box

   The size box is located in the lower right corner of the window.

   When the user presses the mouse button on the size box, GEM 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
   button.

   When the user releases the button, the outline disappears. GEM
   AES sends the application a message that the user wants the win-
   dow's size changed to match the size of the outline when the but-
   ton was released.

o up-arrow

   The up-arrow appears at the top of the right border.

   When the user clicks on the up-arrow, GEM AES moves one line (or
   an equivalent) toward the beginning of the directory or file.

o down-arrow

   The down-arrow appears at the bottom of the right border, directly
   above the size box.

   When the user clicks on the down-arrow, GEM AES moves one line (or
   an equivalent) toward the bottom of the directory or file.

o left-arrow

   The left-arrow appears at the left end of the bottom border.

   When the user clicks on the left-arrow, GEM AES moves the equiv-
   alent of one line toward the left side of the directory or file.

o right-arrow

   The right-arrow appears in the bottom border immediately to the
   left of the size box.

When the user clicks on the right-arrow, GEM AES moves the equiv-
alent of one line toward the right side of the directory or file.

o vertical scroll bar and slider

The vertical scroll bar is located between the up-arrow and down-
arrow. The vertical slider moves up and down in the scroll bar.

By clicking on the part of the scroll bar above the slider (when
visible), the user moves one page (or an equivalent) toward the
beginning of the directory or file.

By clicking on the part of the scroll bar below the slider (when
visible), the user moves one page (cr an equivalent) toward the
end of the directory or file.

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 length of the scroll bar represents all the window's data,
from top to bottom. The position of the slider in the scroll bar
indicates which part of the data is currently visible in the win-
dow. If the slider is at the top of the scroll bar, the top of
the window shows the top of the data. If the scroll bar is two-
thirds of the way down the scroll bar, the top of the window
shows a point two-thirds of the way through the data.

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.

o horizontal scroll bar and slider

The horizontal scroll bar is located between the left-arrow and
the right-arrow.   The horizontal slider moves back and forth in
the scroll bar.

By clicking on the part of the scroll bar to the left of the
slider (when   visible), the ·user moves to the left by the equiv-
alent of one page.

By clicking on the part of the scroll bar to the right of the
slider (when visible), the user moves to the right by the equiv-
alent of one page.

To move quickly anywhere to the left or right, the user can drag
the slider inside the scroll bar.

Like the vertical slider, the horizontal slider also shows the
location of the top of the window relative to the window's data
and how much of the data is visible.

The following sections describe concepts and procedures that apply in the GEM AES windowing environment.

11.2.2 Division of Labor

GEM AES and the application divide responsibility for window management.

GEM AES handles all user-mouse interactions that occur outside the window's work area, including the following:

o clicking on the close box or full box

o pressing the mouse button in the move bar to drag the window's outline

o pressing the mouse button in the size box to produce a larger or smaller window outline

o manipulating the scroll bars

GEM 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 GEM AES, the application has two choices:

o Make a Window Library call that causes the requested change to occur.

o Ignore the message.

This division of labor between GEM AES and the application has the following advantages:

o The application is not responsible for user interactions outside its window's work area.

o The application determines when and if user-requested window changes take place.

o 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.

11.2.3 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_WORKXYWH for window 0 (the desktop window).

   This call returns the 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 requires.

   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 code 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 code describing the window's border.

   The size given to WIND_CREATE determines the window's maximum possible size.

   GEM 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 GEM AES regarding user requests to close, full, size, scroll, or top (activate) the window. To support the overlapping window environment, GEM AES can also send a message requesting that part of the window be redrawn. The redraw procedure is described in Section 11.2.5.

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 may 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.

11.2.4 Support of Overlapping Windows

Application windows can overlap like sheets of paper on a desktop.
The topmost window is called the active window.

When the user clicks the mouse button outside the active window's
border area, GEM AES looks at what was under the tip of the mouse
pointer and acts as follows:

o If the pointer was over the desktop, GEM AES does nothing.

o If the pointer was over another window, GEM AES sends a message to
   the application owning the current active window.   The message
   informs the application that the user wants the other window
   brought to the top (activated). The application should respond
   to this message with a GEM AES call to bring the other window to
   the top.

There are two instances when part of a window may not be visible:

o when one window overlaps another

o 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 the portion of the physical screen that is
visible to the user. This selective drawing is called "clipping."

(GEM AES is responsible for drawing the border, but clipping applies
here as well.)

GEM AES uses a list of rectangular regions to keep track of the por-
tion of the physical screen belonging to each window.   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.

The application obtains the list by making a series of WIND_GET calls. The application must only draw to the physical screen in the area defined by the window's current rectangle list.

11.2.5 Redrawing and Updating

To use the windowing system most efficiently, an application should be able to respond quickly to redraw requests from GEM AES.

There are three reasons why an application might need to update a window's work area:

o to display new application-generated information to the user

o to respond to a message reporting a user request to scroll the contents of the window

o to respond to a request from GEM 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, GEM AES's 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.

3. If the width and height values of this rectangle are not zero, it continues to step 4. If the values are zero, it goes to step 8.

4. It calculates a "result rectangle," which is the intersection (if any) of the rectangle obtained from the window's rectangle list and the update rectangle.

5. If the result rectangle has width and height, 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 or height, the application doesn't draw anything. It continues to the next step.

6. It calls WIND_GET with a value of WF_NEXTXYWH, which asks for the next rectangle from the window's rectangle list.

7. It repeats steps 3, 4, 5, and 6.

8. 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.

11.3 Window Library Routines

The Window Library provides the following routines:

o WIND_CREATE - allocates the application's full-size window and
   returns a handle

o WIND_OPEN - opens the created window to a specified size

o WIND_CLOSE - closes an open window

o WIND_DELETE - de-allocates the application's window and handle

o WIND_GET - gets information on a particular window

o WIND_SET - sets new values for the fields that determine how a
   window is displayed

o WIND_FIND - determines which window is under the mouse's X,Y posi-
   tion

o WIND_UPDATE - notifies GEM 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

o WIND_CALC - calculates the X- and Y-coordinates and the width and
   height of a window's work area or border area

The following sections describe these routines.

Each Window Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

GEM AES Parameter Block
     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

Control Array
     control(0)   =   op_code
     control(1)   =   size in WORDS of int_in array
     control(2)   =   size in WORDS of int_out array
     control(3)   =   size in LONGS of addr_in array
     control(4)   =   size in LONGS of addr_out array

Global Array
```
   global(0)      =  ap_version
   global(1)      =  ap_count
   global(2)      =  ap_id
   global(3,4)    =  ap_private
   global(5,6)    =  ap_ptree
   global(7,8)    =  ap_1resv
   global(9,10)   =  ap_2resv
   global(11,12)  =  ap_3resv
   global(13,14)  =  ap_4resv
```

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

11.3.1 WIND_CREATE

Purpose:

 Allocates the application's full-size window and returns the win-
 dow'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.

Parameters:

```
control(0)  =  100
control(1)  =  5
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  wi_crkind
int_in(1)   =  wi_crwx
int_in(2)   =  wi_crwy
int_in(3)   =  wi_crww
int_in(4)   =  wi_crwh

int_out(0)  =  wi_crreturn
```


o wi_crkind - 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

o wi_crwx - the X-coordinate of the full-size window

o wi_crwy - the Y-coordinate of the full-size window

o wi_crww - the width (in pixels) of the full-size window

o wi_crwh - the height (in pixels) of the full-size window

o wi_crreturn - the handle (numeric identifier) that will identify
  this window in future calls. Values range from 0 to n. A nega-
  tive value indicates that GEM AES has no more windows available.

Sample call to C language binding:

```
wi_crreturn = wind_create(wi_crkind, wi_crwx, wi_crwy,
                        wi_crww, wi_crwh);
```

11.3.2 WIND_OPEN

Purpose:

  Opens a window in its initial size (not necessarily its full size) and location.

Parameters:

```
control(0)   =   101
control(1)   =   5
control(2)   =   1
control(3)   =   0
control(4)   =   0

int_in(0)    =   wi_ohandle
int_in(1)    =   wi_owx
int_in(2)    =   wi_owy
int_in(3)    =   wi_oww
int_in(4)    =   wi_owh

int_out(0)   =   wi_oreturn
```

o wi_ohandle - the handle of the window to be opened

o wi_owx - the X-coordinate of the window in its initial size

o wi_owy - the Y-coordinate of the window in its initial size

o wi_oww - the width (in pixels) of the window in its initial size

o wi_owh - the height (in pixels) of the window in its initial size

o wi_oreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

Sample call to C language binding:

```
wi_oreturn = wind_open(wi_ohandle, wi_owx, wi_owy, wi_oww,
                       wi_owh);
```

11.3.3 WIND_CLOSE

Purpose:

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.

Parameters:

```
control(0)  =  102
control(1)  =  1
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  wi_clhandle

int_out(0)  =  wi_clreturn
```

o wi_clhandle - the handle of the window to be closed

o wi_clreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

_____

Sample call to C language binding:

wi_clreturn = wind_close(wi_clhandle);

11.3.4 WIND_DELETE

Purpose:

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 before calling the WIND_OPEN routine.

Parameters:

```
control(0)  =  103
control(1)  =  1
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  wi_dhandle

int_out(0)  =  wi_dreturn
```

o wi_dhandle - the handle of the window to be deleted

o wi_dreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

_____

Sample call to C language binding:

wi_dreturn = wind_delete(wi_dhandle);

11.3.5 WIND_GET

Purpose:

   Depending on the information requested by the call, returns one of
   the following:

   o X- and Y-coordinates and width and height values for various
      aspects of the current and previous windows

   o slider location and size

   o the handle of the active window

   o X- and Y-coordinates, width, and height of the rectangles in the
      window's rectangle list

Parameters:

```
control(0)   =   104
control(1)   =   2
control(2)   =   5
control(3)   =   0
control(4)   =   0

int_in(0)    =   wi_ghandle
int_in(1)    =   wi_gfield

int_out(0)   =   wi_greturn
int_out(1)   =   wi_gw1
int_out(2)   =   wi_gw2
int_out(3)   =   wi_gw3
int_out(4)   =   wi_gw4
```


   o wi_ghandle - The handle of the window about which the application
      wants information.

   o wi_gfield - A numerical value identifying the field about which
      the application wants information. The value of wi_gfield deter-
      mines which of wi_gw1, wi_gw2, wi_gw3, and wi_gw4 is returned.

      1 - WF_RESVD1 - [RESERVED]

      4 - WF_WORKXYWH - the coordinates of the work area of the window
         - returns wi_gw1 (X-coordinate)
         - returns wi_gw2 (Y-coordinate)
         - returns wi_gw3 (width)
         - returns wi_gw4 (height)

5 - WF_CURRXYWH -   the coordinates of the entire current window,
     including borders, title bar, and information line
     - returns wi_gw1 (X-coordinate)
     - returns wi_gw2 (Y-coordinate)
     - returns wi_gw3 (width)
     - returns wi_gw4 (height)

6 -   WF_PREVXYWH - the coordinates of the previous window, in-
     cluding borders, title bar, and information line
     - returns wi_gw1 (X-coordinate)
     - returns wi_gw2 (Y-coordinate)
     - returns wi_gw3 (width)
     - returns wi_gw4 (height)

7 -   WF_FULLXYWH - the coordinates of the window at its fullest
     possible size,   including borders, title bar, and informa-
     tion line
     - returns wi_gw1 (X-coordinate)
     - returns wi_gw2 (Y-coordinate)
     - returns wi_gw3 (width)
     - returns wi_gw4 (height)

8 - WF_HSLIDE - a number between 1 and 1000, giving the relative
     position of the horizontal slider
     (1 = leftmost position; 1000 = rightmost position)
     - returns wi_gw1

9 - WF_VSLIDE - a number between 1 and 1000, giving the relative
     position of the vertical slider
     (1 = top position; 1000 = bottom position)
     - returns wi_gw1

10 - WF_TOP - the window handle of the window that is on top (ac-
     tive)
     - returns wi_gw1

11 - WF_FIRSTXYWH - the coordinates of the first rectangle in the
     window's rectangle list
     - returns wi_gw1 (X-coordinate)
     - returns wi_gw2 (Y-coordinate)
     - returns wi_gw3 (width)
     - returns wi_gw4 (height)

12 -   WF_NEXTXYWH - the coordinates of the next rectangle in the
     window's rectangle list
     - returns wi_gw1 (X-coordinate)
     - returns wi_gw2 (Y-coordinate)
     - returns wi_gw3 (width)
     - returns wi_gw4 (height)

13 - WF_RESVD2 - [RESERVED]

15 - WF_HSLSIZE - the size of the horizontal slider
             -1 = default minimum size (a square box)
          1 - 1000 = the slider's relative size compared to
                  the horizontal scroll bar
          - returns wi_gw1

16 - WF_VSLSIZE - the size of the vertical slider
             -1 = default minimum size (a square box)
          1 - 1000 = the slider's relative size compared to
                  the vertical scroll bar
          - returns wi_gw1

17 - WF_SCREEN - the address and length of the internal menu/alert
      buffers
          - returns wi_gw1 (low WORD of address)
          - returns wi_gw2 (high WORD of address)
          - returns wi_gw3 (low WORD of length)
          - returns wi_gw4 (high WORD of length)

o
  wi_greturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o wi_gw1 - The value returned depends on the field named in
   wi_gfield (see above).

o wi_gw2 - The value returned depends on the field named in
   wi_gfield (see above).

o wi_gw3 - The value returned depends on the field named in
   wi_gfield (see above).

o wi_gw4 - The value returned depends on the field named in
   wi_gfield (see above).

Sample call to C language binding:

wi_greturn = wind_get(wi_ghandle, wi_gfield, &wi_gw1,
                      &wi_gw2, &wi_gw3, &wi_gw4);

11.3.6 WIND_SET

Purpose:

   Changes the value in one of several fields that determine how a
   window is displayed.

Parameters:

```
control(0)  =  105
control(1)  =  6
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  wi_shandle
int_in(1)   =  wi_sfield
int_in(2)   =  wi_sw1
int_in(3)   =  wi_sw2
int_in(4)   =  wi_sw3
int_in(5)   =  wi_sw4

int_out(0)  =  wi_sreturn
```

o wi_shandle - The handle of the window whose fields the application
  wishes to change.

o wi_sfield - A numerical value identifying the field the applica-
  tion wishes to change.

     2 - WF_NAME - the address of the string containing the name of
         the window
         - takes wi_sw1 and wi_sw2. A pointer to the name is all
         that is stored.
           If you wish the name to be saved then keep a copy in your
         data space.

     3 - WF_INFO - the address of the string containing the informa-
         tion line
         - takes wi_sw1 and wi_sw2

     5 - WF_CURRXYWH - defined under WIND_GET

     8 - WF_HSLIDE - defined under WIND_GET

     9 - WF_VSLIDE - defined under WIND_GET

    10 - WF_TOP - defined under WIND_GET

      14 - WF_NEWDESK - the address of a new default GEM Desktop for
            GEM AES to draw
            - takes wi_sw1 (object tree LOW WORD)
            - takes wi_sw2 (object tree HIGH WORD)
            - takes wi_sw3 (starting object to draw in tree)
            The application should set the object tree to 0L before
            exiting to restore the desktop.

      15 - WF_HSLSIZE - defined under WIND_GET

      16 - WF_VSLSIZE - defined under WIND_GET

o wi_sw1 - The value depends on the field named in wi_sfield (see
   above).

o wi_sw2 - The value depends on the field named in wi_sfield (see
   above).

o wi_sw3 - The value depends on the field named in wi_sfield (see
   above).

o wi_sw4 - The value depends on the field named in wi_sfield (see
   above).

o wi_sreturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

Sample call to C language binding:

wi_sreturn = wind_set(wi_shandle, wi_sfield, wi_sw1,
                      wi_sw2, wi_sw3, wi_sw4);

11.3.7 WIND_FIND

Purpose:

Finds which window is under the mouse's X,Y position.

Parameters:

```
control(0)  =  106
control(1)  =  2
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_in(0)   =  wi_fmx
int_in(1)   =  wi_fmy

int_out(0)  =  wi_freturn
```

o wi_fmy - the X-coordinate of the mouse's position

o wi_fmy - the Y-coordinate of the mouse's position

o wi_freturn - the handle of the window under the mouse's X,Y position

---

Sample call to C language binding:

wi_freturn = wind_find(wi_fmx, wi_fmy);

11.3.8 WIND_UPDATE

Purpose:

    Does one of the following:

o Notifies GEM AES that the application is about to begin updating a
   window or has finished updating a window.

   During the update, GEM AES does not allow changes to take place in
   the portion of the screen belonging to the window.

o Notifies GEM 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.

Parameters:

    control(0)   =   107
    control(1)   =   1
    control(2)   =   1
    control(3)   =   0
    control(4)   =   0

    int_in(0)   =   wi_ubegend

    int_out(0)   =   wi_ureturn

o wi_ubegend - a code for the call's function

   0 - END_UPDATE
   1 - BEG_UPDATE
   2 - END_MCTRL
   3 - BEG_MCTRL

o wi_ureturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

_____

Sample call to C language binding:

wi_ureturn = wind_update(wi_ubegend);

11.3.9 WIND_CALC

Purpose:

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.

Parameters:

```
control(0)   =   108
control(1)   =   6
control(2)   =   5
control(3)   =   0
control(4)   =   0

int_in(0)    =   wi_ctype
int_in(1)    =   wi_ckind
int_in(2)    =   wi_cinx
int_in(3)    =   wi_ciny
int_in(4)    =   wi_cinw
int_in(5)    =   wi_cinh

int_out(0)   =   wi_creturn
int_out(1)   =   wi_coutx
int_out(2)   =   wi_couty
int_out(3)   =   wi_coutw
int_out(4)   =   wi_couth
```

o wi_ctype - the type of calculation to perform

0 - return border area X, Y, width, and height
1 - return work area X, Y, width, and height

o wi_ckind - 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

o wi_cinx - the input X-coordinate of the work area (if wi_ctype = 0) or border area (if wi_ctype = 1)

o wi_ciny - the input Y-coordinate of the work area (if wi_ctype = 0) or border area (if wi_ctype = 1)

o wi_cinw - the input width value of the work area (if wi_ctype = 0) or border area (if wi_ctype = 1)

o wi_cinh - the input height value of the work area (if wi_ctype = 0) or border area (if wi_ctype = 1)

o wi_creturn - a coded return message

0 - an error exists
n (positive integer) - no error exists

o wi_coutx - the output X-coordinate of the work area (if wi_ctype = 1) or border area (if wi_ctype = 0)

o wi_couty - the output Y-coordinate of the work area (if wi_ctype = 1) or border area (if wi_ctype = 0)

o wi_coutw - the output width value of the work area (if wi_ctype = 1) or border area (if wi_ctype = 0)

o wi_couth - the output height value of the work area (if wi_ctype = 1) or border area (if wi_ctype = 0)

Sample call to C language binding:

```
wi_creturn = wind_calc(wi_ctype, wi_ckind, wi_cinx,
                       wi_ciny, wi_cinw, wi_cinh,
                       &wi_coutx, &wi_couty,
                       &wi_coutw, &wi_couth);
```

End of Section 11

Section 12
Resource Library

12.1 Introduction

The Resource Library is the interface between an application and its resources, the collection of data used by the application. Types of resources include trees, objects, strings, icons, and pictures.

The purpose of a resource file is to isolate an application's device-, user-, and country-specific data from its code. This isolation provides the following advantages:

o machine-code portability

   To port the application across different environments, the programmer need only change the resource file data.

o customization of appearance

   A non-programmer can change the application's menu structure, the layout of dialog boxes, and error message text. In most cases the programmer need not be involved.

o internationalization of text messages

   To change text messages from one language to another, one need only change the text in the resource file.

o device-independent raster graphics

   Because they are stored as resources, GEM 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. GEM AES follows the convention that all resource files have the filetype .RSC.

The programmer creates a resource file by using the GEM Resource Construction Set.

12.2 Using the Resource Library

When an application calls RSRC_LOAD with the name of a resource file, the Resource Library does the following:

o It searches for the file and finds its total size in bytes.

o Using the DOS allocate call, it allocates enough memory space to hold the resource file.

o It opens the resource file, reads it into the allocated memory space, and closes the file.

o It makes any required updates to the file. These updates do the following:

  - make the file device-specific to the screen's resolution

  - link up all the OBJECT pointers, TEDINFO pointers, ICONBLK pointers, and BITBLK pointers

  - build the array of tree pointers

  - store the address of the tree array 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 out the address of the tree array in the Global Array.

12.3 Resource Library Routines

The Resource Library uses the following routines:

o RSRC_LOAD - loads an entire resource file into memory

o RSRC_FREE - frees the memory allocated during RSRC_LOAD

o RSRC_GADDR - gets the address of a data structure in memory

o RSRC_SADDR - stores an index to a data structure

o RSRC_OBFIX - converts an object's X- and Y-coordinates, width, and height from character coordinates to pixel coordinates

The following sections describe these routines.

Each Resource Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

GEM AES Parameter Block
    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

Control Array
    control(0) = op_code
    control(1) = size in WORDS of int_in array
    control(2) = size in WORDS of int_out array
    control(3) = size in LONGS of addr_in array
    control(4) = size in LONGS of addr_out array

Global Array
    global(0)   = ap_version
    global(1)   = ap_count
    global(2)   = ap_id
    global(3,4)   = ap_private
    global(5,6)   = ap_ptree
    global(7,8)   = ap_1resv
    global(9,10)   = ap_2resv
    global(11,12) = ap_3resv
    global(13,14) = ap_4resv

Global Array parameters are described in Section 3.

Each routine also contains some or all of the following arrays:

o Integer Input (int_in) - Unless otherwise noted, each parameter in
  this array is a WORD.

o Integer Output (int_out) - Unless otherwise noted, each parameter
  in this array is a WORD.

o Address Input (addr_in) - Unless otherwise noted, each parameter
  in this array is a POINTER.

o Address Output (addr_out) - Unless otherwise noted, each parameter
  in this array is a POINTER.

12.3.1 RSRC_LOAD

Purpose:

  Allocates memory and loads a resource file into memory.

Parameters:

```
control(0)  =  110
control(1)  =  0
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_out(0)  =  re_lreturn

addr_in(0)  =  re_lpfname
```

o re_lreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o re_lpfname - the address of the ASCII string of the resource
   filename

_____

Sample call to C language binding:

re_lreturn = rsrc_load(re_lpfname);

12.3.2 RSRC_FREE

Purpose:

  Frees the memory space allocated in RSRC_LOAD.

Parameters:

```
control(0)  =  111
control(1)  =  0
control(2)  =  1
control(3)  =  0
control(4)  =  0

int_out(0)  =  re_freturn
```

o re_freturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

Sample call to C language binding:

```
re_freturn = rsrc_free();
```

12.3.3 RSRC_GADDR

Purpose:

   Gets the address of a data structure in memory.

Parameters:

```
control(0)  =  112
control(1)  =  2
control(2)  =  1
control(3)  =  0
control(4)  =  1

int_in(0)   =  re_gtype
int_in(1)   =  re_gindex

int_out(0)  =  re-greturn

addr_out(0) =  re_gaddr
```


o re_gtype - the type of data structure


```
 0 tree
 1 OBJECT
 2 TEDINFO
 3 ICONBLK
 4 BITBLK
 5 string
 6 imagedata
 7 obspec
 8 te_ptext   (see Section 6.3.2)
 9 te_ptmplt  (see Section 6.3.2)
10 te_pvalid  (see Section 6.3.2)
11 ib_pmask   (see Section 6.3.3)
12 ib_pdata   (see Section 6.3.3)
13 ib_ptext   (see Section 6.3.3)
14 bi_pdata   (see Section 6.3.4)
15 ad_frstr - the address of a POINTER to a free string
16 ad_frimg - the address of a POINTER to a free image
```


o re_gindex - the index of the data structure

o re_greturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o re_gaddr - the address of the data structure specified by re_gtype
   and re_gindex

Sample call to C language binding:

```
re_greturn = rsrc_gaddr(re_gtype, re_gindex,
                        &re_gaddr);
```

12.3.4 RSRC_SADDR

Purpose:

  Stores the address of a data structure in memory.

Parameters:

    control(0) = 113
    control(1) = 2
    control(2) = 1
    control(3) = 1
    control(4) = 0

    int_in(0)   = re_stype
    int_in(1)   = re_sindex

    int_out(0) = re_sreturn

    addr_in(0) = re_saddr

o re_stype - the type of data structure

  15 ad_frstr - defined under RSRC_GADDR
  16 ad_frimg - defined under RSRC_GADDR

o re_sindex - the location in the data structure where re_saddr will
  be stored

o re_sreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o re_saddr - the address of the data structure

Sample call to C language binding:

re_sreturn = rsrc_saddr(re_stype, re_sindex,
                        re_saddr);

12.3.5 RSRC_OBFIX

Purpose:

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.

Parameters:

```
control(0)  =  114
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  re_oobject

int_out(0)  =  re_oresvd

addr_in(0)  =  re_otree
```

o re_oobject - The index of the object to be converted.

o re_oresvd - [RESERVED]; value always equals 1 (one).

o re_otree - The address of the tree that contains the object.

Sample call to C language binding:

```
re_oresvd = rsrc_obfix(re_otree, re_oobject);
```

12.4 Resource File Header Information

This is a brief document for the GEM Application Resource File Header.

Readers should have a knowledge of the meaning of TREES, OBJECTS, TEDINFO, ICONBLK, BITBLK, APPBLK etc which are described in the GEM programming manual from 6-1 to 6-10.

The easiest way to create a resource file is to use the resource construction program which is included in the Developer Tools Package from ATARI Corp. That program will generate a XXXX.rsc file that is recognizable by the GEM AES rscr_load routine.

Since everything is being done in a real time environment, the resource file can be loaded in anywhere in memory (relocatable). The file must contain some kind of header to describe the resource file itself in order for the GEM AES resource manager to do all of the necessary fix ups after the loading.

If you are using the resource construction program on the ST, then you don't have to worry about anything. But if you are using the resource construction file on the IBM PC, then you need to do an extra step to port that file across to the ST. Please read the document about the ST create, which is also included in the Developer Tools Package.

Now, if you are using a cross compiler on some other machine to generate a resource file, then you need to put a file header into that file. Assume you have all of the OBJECTS, TEDINFO, BITBLK, ... in a file, and everything links or points to their proper areas. You need an 18 word header to describe what that file has, and where the OBJECTS, TEDINFO, STRINGS ... are and patch it right in front of the file.

The following is a description of these 18 words (in order).

Important !!! All the offsets are related to the first byte of the file.

1. rsh_vrsn

This should be the first word of the file and is always zero.

2. rsh_object

Contains a 64k offset of where the first OBJECT structure is located at.

3. rsh_tedinfo

Contains a 64k offset of where the first TEDINFO structure is located at.

4. rsh_iconblk

Contains a 64k offset of where the first ICONBLK structure is located at.

5. rsh_bitblk

Contains a 64k offset of where the first BITBLK structure is located at.

6. rsh_frstr

Contains a 64k offset of where the strings pointer table is. Remember this table can be the same one that the OBJECTS or TEDINFOS are using. You can set offset pointer pointing to the middle or anywhere in the table if you want to reuse some of the strings in other ways.

7. rsh_string

Not used.

8. rsh_imdata

Contains a 64k offset of where the icon image data is located at.

9. rsh_frimg

Contains a 64k offset of where the cursor image data pointer table is located at.

10. rsh_trindex

Contains a 64k offset of where the tree pointer table is located at.

11. rsh_nobs

Contains the number of OBJECTS.

12. rsh_ntree

Contains the number of TREES.

13. rsh_nted

Contains the number of TEDINFOS.

14. rsh_nib

Contains the number of ICONBLKS.

15. rsh_nbb

Contains the number of BITBLKS.

16. rsh_nstring

Contains the number of indepedent strings.

17. rsh_nimages

Contains the number of ICON images.

18. rsh_rssize.

Contains the size of the whole resource file in bytes.

Use the BITBLK structure for any bit image or icon. Don't use the ICONBLK structure.

Section 13

Shell Library

13.1 Introduction

The Shell Library serves two major functions:

o It lets an application keep track of the command and command tail that invoked it.

o 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 application is itself an example of an application that uses the second Shell function to invoke GEM and TOS applications.

13.2 Using the Shell Library

Two Shell Library routines use a single buffer that contains the following:

o the command with which the Shell Library invoked the current application or will invoke the next application

o 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:

o 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 buffer.

13-1

o 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 applica-
   tion to run. It also indicates whether the requested application
   is graphic or character-based and whether it is a GEM or TOS ap-
   plication.

2. When the current application terminates, the Shell Library starts
   the application that was requested next.

To exit GEM AES, an application makes a SHEL_WRITE call, passing in
an sh_wdoex value of 0 (zero).

Two other Shell Library routines are SHEL_FIND and SHEL_ENVRN.

SHEL_FIND makes it possible for an application to locate a filename
by following the TOS search path.

SHEL_ENVRN lets an application search the TOS environment for a
predefined environment parameter like "COMSPEC=" or "VERIFY=". The
routine returns the address of the byte immediately following the
parameter. This byte contains the value of the parameter, and its
address is stored in a LONG value pointed at by one of the input
parameters of SHEL_ENVRN.

The last two Shell Library routines, SHEL_GET and SHEL_PUT, are used
by the desktop to write information to the desktop.inf file. This
file stores all the parameters you have setup and saved under the
SAVE DESKTOP option.

13.3 Shell Library Routines

The Shell Library uses the following routines:

o SHEL_READ - lets an application determine how it was invoked

o SHEL_WRITE - exits GEM AES or tells which application to run
  next

o SHEL_FIND - locates a filename by following the TOS search path

o SHEL_ENVRN - searches the TOS environment for a parameter and

returns the address of its value

o SHEL_GET - lets the application read data from the AES's internal
buffer

o SHEL_PUT - lets the application save data into the AES's internal
buffer

Each Shell Library routine has a GEM AES Parameter Block, Control
Array, and Global Array that contain the following information:

GEM AES Parameter Block
     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

Control Array
     control(0) = op_code
     control(1) = size in WORDS of int_in array
     control(2) = size in WORDS of int_out array
     control(3) = size in LONGS of addr_in array
     control(4) = size in LONGS of addr_out array

Global Array
     global(0)   = ap_version
     global(1)   = ap_count
     global(2)   = ap_id
     global(3,4)   = ap_private
     global(5,6)   = ap_ptree
     global(7,8)   = ap_1resv
     global(9,10) = ap_2resv
     global(11,12) = ap_3resv
     global(13,14) = ap_4resv

- Global Array parameters are described in Section 3.

   Each routine also contains some or all of the following arrays:

   o Integer Input (int_in) - Unless otherwise noted, each parameter in
     this array is a WORD.

   o Integer Output (int_out) - Unless otherwise noted, each parameter
     in this array is a WORD.

   o Address Input (addr_in) - Unless otherwise noted, each parameter
     in this array is a POINTER.

   o Address Output (addr_out) - Unless otherwise noted, each parameter
     in this array is a POINTER.

13.3.1 SHEL_READ

Purpose:

Lets an application identify the command that invoked it.

Parameters:

```
control(0)  =  120
control(1)  =  0
control(2)  =  1
control(3)  =  2
control(4)  =  0

int_out(0)  =  sh_rreturn

addr_in(0)  =  sh_rpcmd
addr_in(1)  =  sh_rptail
```

o sh_rreturn - a coded return message

0 - an error exists
n (positive integer) - no error exists

o sh_rpcmd - the address of a buffer that will hold the command that invoked the application

o sh_rptail - the address of a buffer that will hold the command tail invoked with the command

_____

Sample call to C language binding:

sh_rreturn = shel_read(sh_rpcmd, sh_rptail);

13.3.2 SHEL_WRITE

Purpose:

   Tells GEM AES whether to run another application and,   if so,
   which application to run.

Parameters:

   control(0)   =   121
   control(1)   =   3
   control(2)   =   1

   control(3)   =   2
   control(4)   =   0

   int_in(0)   =   sh_wdoex
   int_in(1)   =   sh_wisgr
   int_in(2)   =   sh_wiscr

   int_out(0)   =   sh_wreturn

   addr_in(0)   =   sh_wpcmd
   addr_in(1)   =   sh_wptail

o sh_wdoex - a coded instruction to exit the system or run another
  application when the user exits the current application

  0 - exit and return to the GEM DESKTOP
  1 - run another application

o sh_wisgr - a code for whether the next application is a graphic
  application

  0 - not graphic application
  1 - graphic application

o sh_wiscr - a code for whether the next application is a GEM AES
  application

  0 - not GEM application
  1 - GEM application

o sh_wreturn - a coded return message

  0 - an error exists
  n (positive integer) - no error exists

o sh_wpcmd - the address of the new command file to execute

o sh_wptail - the address of the command tail for the next program
* **Note:** The first byte of the command tail is its' length followed by
  the actual command tail (ie. at the second byte).

Sample call to C language binding:

```
sh_wreturn = shel_write(sh_wdoex, sh_wisgr, sh_wiscr,
                        sh_wpcmd, sh_wptail);
```

13.3.3 SHEL_FIND

Purpose:

   Searches for a filename in the current directory and in each
   directory in the search path; when it finds the filename, returns
   its full TOS file specification.

Parameters:

   control(0)   =   124
   control(1)   =   0
   control(2)   =   1
   control(3)   =   1
   control(4)   =   0

   int_out(0)   =   sh_freturn

   addr_in(0)   =   sh_fpbuff

o sh_freturn - a coded return message

   0 - an error exists
   n (positive integer) - no error exists

o sh_f pbuff - the address of a buffer with distinct input and out-
   put functions

   input: holds the filename the application is searching for
   output: holds the full TOS file specification of the filename's
           location in the search path

   The buffer must be long enough to hold the full TOS file
   specification (80 character minimum).

Sample call to C language binding:

sh_freturn = shel_get(sh_fpbuff);

13.3.4 SHEL_ENVRN

Purpose:

  Searches in the TOS environment for the occurrence of an environ-
  ment parameter string; stores the address of the byte immediately
  following the parameter string in a LONG value.

Parameters:

```
control(0)  =  125
control(1)  =  0
control(2)  =  1
control(3)  =  3
control(4)  =  0

int_out(0)  =  sh_eresvd

addr_in(0)  =  sh_epvalue
addr_in (1) =  sh_eparm
```


o sh_eresvd - [RESERVED]; value equals one (1)

o sh_epvalue - the address of a LONG value in which this routine
  will store the address of the byte immediately following the
  parameter string

o sh_eparm - the parameter string for which the application is sear-
  ching (includes the "=" sign)

Sample call to C language binding:

sh_eresvd = shel_envrn(sh_epvalue, sh_eparm);

13.3.5 SHEL_GET

Purpose:

Let the application read data from the AES's internal buffer.
Note: The length of the get data buffer should not be more than 1024 bytes.

Parameters:

```
control(0)  =  122
control(1)  =  1
control(2)  =  1
control(3)  =  1
control(4)  =  0

int_in(0)   =  sh_glen

addr_in(0)  =  sh_gbuff

int_out (0) =  sh_greturn
```

o sh_eresvd - a coded return message
         0 - an error exists
         n (positive integer) - no error exists

o sh_glen - the length of the buffer.

o sh_gbuff - the address of the buffer.

Sample call to C language binding:

sh_greturn = shel_get(sh_gbuff, sh_glen);

13.3.6 SHEL_PUT

Purpose:

   Let the application save data into the AES's shell internal buff-
   er.
   Note: Currently the AES's Desktop uses this buffer to store the
   desktop.inf data.   Any usage of this buffer may corrupt the data
   that are already stored there. The length of the data that goes
   into the buffer should be no more than 1024 bytes.

Parameters:

   control(0) = 123
   control(1) = 1
   control(2) = 1
   control(3) = 1
   control(4) = 0

   int_out(0) = sh_preturn

   addr_in(0) = sh_pbuff

   int_in (0) = sh_plen

o sh_return - a coded return message
         0 - an error exists
         n (positive integer) - no error exists

o sh_glen - the length of the buffer.

o sh_gbuff - the address of the buffer.

Sample call to C language binding:

sh_greturn = shel_put(sh_gbuff, sh_glen);

M

machine-code portability, 12-1
making a window active, 2-13
managing multiple windows 11-8
mask word, 4-2
maximum of eight windows, 11-1
me_bshow, 5-7
memory,
   freeing, 12-5, 12-2
     "required for desk
       accessories" ,1-5
menu
   making disappear, 5-3
   moving mouse out of, 5-3
   menu/alert buffer, 1-4, 1-9,
     7-10, 11-20
   menu bar
   displaying, 2-3
MENU_BAR, 2-4, 5-2, 5-5, 5-7
MENU_IENABLE, 2-9, 5-5, 5-9
Menu Library
   responsibilities, 5-2, 5-4
menu object tree, 2-4
MENU_REGISTER, 1-6, 4-8, 5-5,
   5-12
menu resource file, 5-4
menu selection, 2-5, 4-5
menu title
   highlighting with keystroke,
   2-8, 5-2
MENU_TNORMAL, 2-6, 5-5, 5-10
message, 2-6
message buffer, 3-2
message event, 4-2, 4-9
message pipe, 4-16, 5-3
meta file, 9-2
MN_SELECTED 2-5, 4-5
mouse button, 4-12
mouse button event, 2-8, 4-2,
   4-9
mouse button state, 2-8, 4-12,
   8-2, 8-20
mouse event, 4-2, 4-9, 4-14
mouse form, 4-3
move bar 2-9, 4-7, 11-4, 11-13
multiple event, 2-4, 4-9,
   4-18, 11-7

N

naming convention, 1-3
nil index, 6-3
nil pointer, 6-3
Not-Ready List, 1-6, 2-5
Note icon, 7-6
   radio buttons, 6-15, 7-2

O

ob_flags, 6-4
OBJC_ADD, 6-3, 6-18, 6-20
OBJC_CHANGE, 2-7, 6-18, 6-30
OBJC_DRAW, 2-4, 6-3, 6-18,
   6-22, 7-9
OBJC_FIND, 2-8, 6-18, 6-24
object
   child, 6-1, 7-7
   root, 6-1, 7-7
   sibling, 6-1, 6-4
   X-and Y-coordinates, 6-2

object colors, 6-13, 6-17
object color WORD, 6-17
object flags, 6-13, 6-15
object states, 6-13, 6-16
OBJECT structure, 6-3, 7-7
object tree, 6-1, 6-20, 7-7
object type, 6-4, 6-13
object type G_BOX, 6-2
ob_spec, 6-4
ob_spec pointer, 6-5
ob_state, 2-8, 6-4
ob_type, 6-4
one of many structure, 7-2
opening a window, 11-7, 11-17
overlapping windows 4-4, 11-1,
   11-7

P

parent-child
     relationships 6-1
parent object, 6-1, 6-4
PARMBLK structure, 6-3, 6-11
pasting data, 9-1, 11-1

V

validation string
    te-pvalid, 6-6, 7-5
validation string values, 6-6
vertical scroll bar, 4-6, 11-5
vertical slider, 2-11, 4-7,
    8-15, 11-13

W

WAIT icon, 7-6

WIND_CALC, 2-10, 11-7,
    11-11, 11-25
WIND_CLOSE, 11-7, 11-11,
    11-16
WIND_CREATE, 2-10, 11-7,
    11-11, 11-13
WIND_DELETE
    free the handle, 2-14, 11-8,
        11-11, 11-17
WIND_FIND, 2-13, 11-11, 11-23
WIND_GET, 2-12, 11-7, 11-9,
    11-11, 11-18
WIND_OPEN, 2-10, 11-7, 11-15
window
    activating, 2-9, 2-13,
        11-8, 11-15
    application responsibilities,
        2-9, 11-6
    closing, 2-13, 11-3,
        11-11, 11-16
    fulled, 4-6, 11-3
    GEM AES responsibilities, 11-6
    greatest possible size, 2-10,
        11-3, 11-13
    initial size and location,
        2-11, 11-7
    making active, 2-9, 11-8
    moving, 4-7, 11-4
    redrawing, 2-12, 4-5, 11-7, 11-9
    sizing, 2-11, 4-7, 11-4
window control areas, 2-11, 11-3
window handle, 2-10, 11-1, 11-7

window management
    responsibilities, 2-9, 11-6

WIND_SET, 2-11, 11-11, 11-21
WIND_UPDATE, 2-12, 11-11, 11-24
WM_ARROWED, 4-6

WM_CLOSED, 2-13, 4-5
WM_FULLED, 4-6
WM_HSLID, 4-6
WM_MOVED, 4-7
WM_REDRAW, 4-5
WM_REDRAW message, 2-11
WM_SIZED, 4-7
WM_TOPPED, 2-13, 4-5
WM_VSLID, 4-7

X

X- and Y-coordinates, 1-9, 2-8,
    6-2, 7-8, 11-9,
        11-11, 11-18, 11-25