


# The Application Environment Services

In this document, I list assembly language and C interfaces to the Intel GEM AES. The C prototypes are similar to those provided by the GEM Developer's Kit, version 3.0. Most of the C interfaces are declared to return WORDs; these will nearly always be the value returned in

int\_out[0]. The symbol  indicates that a function behaves differently in different GEM versions.

This is still a work in progress. Some information is missing or requires additional explanation. The information has been obtained by examining GEM source and bindings rather than published documentation. I have not been able to disassemble GEM/4 enough to find the purposes of various functions; only to note their existence.

The AES deals with drawing windows on the screen, and with managing controls (text fields, buttons etc.).

The AES lives in the file GEM.EXE (in ViewMAX, VIEWMAX.OVL). It is accessed by using INT 0EFh with CX=00C8h and ES:BX=address of parameter block. It also checks for CX=00C9h, but in the single-tasking version this is a no-op.

The parameter block format is:

DD	CONTROL	;Address of control array
DD	GLOBALS	;Address of globals array
DD	INT_IN	;Address of integer input array
DD	INT_OUT	;Address of integer output array
DD	ADDR_IN	;Address of far pointer input array
DD	ADDR_OUT	;Address of far pointer output array

The control array is:

```
CONTROL:
    DEFW    function    ;Input:  AES function, 10-134
    DEFW    #int_in      ;Input:  No. of integer parameters
    DEFW    #int_out     ;Input:  Max. no. of return words
    DEFW    #addr_in    ;Input:  No. of pointer parameters
    DEFW    #addr_out   ;Input:  Max no. of return pointers
                                ;(ignored in single-tasking GEM)
```

The globals array is used to hold necessary GEM state relating to the current process. It is almost entirely managed by GEM and there are very few occasions when programs need to modify it.

---

## Function 10 - Initialise AES

### C Interface:

- WORD appl\_init(VOID);
- WORD appl\_init([X\\_BUF\\_V2](#) \*xbuf);

### Entered with:

- *function*=10;
- #*int\_in*=0;
- #*addr\_in*=0 or 1.
- If #*addr\_in* is 1, *addr\_in*[0] must be the address of an initialised [X\\_BUF\\_V2](#) structure.



The extra parameter to an X\_BUF\_V2 is an extension in [the FreeGEM AES](#). If `addr_in[0]` is set to 1 and an initialised [X\\_BUF\\_V2](#) structure is passed in, then on return its `arch` member will be 0 for DRI GEM or ViewMAX, and 16 or 32 for FreeGEM.

### Returns:

- If failed, `int_out[0]` is -1.
- If succeeded, `int_out[0]` is the application's ID (process ID) and the `globals` array will contain:

```

        DEFW    version          ;eg 0x0300 for GEM 3.0 and all ViewMAX
versions
        DEFW    max_tasks       ;1 for single-tasking, >1 for multitasking
        DEFW    app_id         ;Same as int_out[0]
        DD      ap_private      ;Available for use by the program
                                ;The GEM v1 desktop assumes that on entry
                                ;this is the ob_spec of the desktop

window.
                                ;It uses this value to construct its own
                                ;custom desktop.
        DEFS    10              ;Not set at init time. Used to hold
                                ;pointers to the currently loaded resource

file.
        DEFW    nplanes        ;No. of planes in the display
        DD      global         ;Address of GEM global variables area
        DEFW    disks          ;Bitmap of available drives
        DEFW    hds            ;Bitmap of available hard drives

```



(all values after `[ap_private+4]` are variables used internally by GEM; their meaning may not be the same in all versions).

## Function 11 - Read message pipe

### C Interface:

- `WORD appl_read(WORD rwid, WORD length, VOID far *pbuff);`

### Entered with:

- `function=11;`
- `#int_in=2;`
- `#addr_in=1.`
- `int_in[0] = rwid, int_in[1] = length`
- `addr_in[0] = address of data to read.`

`rwid` is the application ID of the process that owns the message pipe (normally the calling process). This call will block until the required data have been returned.

See [the message list](#) for information on standard AES messages. All AES messages are 16 bytes long.

## Function 12 - Write message pipe

### C Interface:

- WORD appl\_write(WORD rwid, WORD length, VOID far \*pbuff);

### Entered with:

- *function*=12;
- *#int\_in*=2;
- *#addr\_in*=1.
- *int\_in*[0] = rwid, *int\_in*[1] = length
- *addr\_in*[0] = address of data to write.

rwid is the application ID of the process to which the data should be sent. This call will block until the data have been sent.

See [the message list](#) for information on standard AES messages. All AES messages are 16 bytes long.

---

## Function 13 - Find an application

### C Interface:

- WORD appl\_find(BYTE far \*name);;

### Entered with:

- *function*=13;
- *#int\_in*=0;
- *#addr\_in*=1.
- *addr\_in*[0] = address of process name (max. 8 bytes + terminating zero; if less than 8 bytes, pack it with spaces).

### Returns:

- If process found: Its application ID.
- If not: -1

GEM's internal window manager process is called SCRENMGR.

---

## Function 14 - Play back recorded events

### C Interface:

- WORD appl\_tplay(BYTE far \*buffer, WORD length, WORD scale);

### Entered with:

- *function*=14;
- *#int\_in*=2;
- *#addr\_in*=1.
- *int\_in*[0] = length of buffer
- *int\_in*[1] = Playback speed (percent of normal)
- *addr\_in*[0] = address of buffer

---

## Function 15 - Record events to memory

### C Interface:

- WORD appl\_trecord(BYTE far \*buffer, WORD length);

### Entered with:

- *function*=15;
- *#int\_in*=1;
- *#addr\_in*=1.
- *int\_in*[0] = length of buffer
- *addr\_in*[0] = address of buffer

Each record in the buffer is 6 bytes; a 2-byte event type and a 4-byte event body:

Type	Body
0 = timer event	LONG millis; (Time in milliseconds)
1 = button event	WORD down (1=down 0=up); WORD click_count;
2 = mouse movement event	WORD x; WORD y;
3 = keyboard event	WORD char; WORD keystate;

---

## Function 16 - Set bitmaps of available drives

### C Interface:

- WORD appl\_bvdisk(WORD bvdisk, WORD bvhard);

### Entered with:

- *function*=16;
- *#int\_in*=2;
- *#addr\_in*=0.
- *int\_in*[0] = bitmap of available drives
- *int\_in*[1] = bitmap of available hard drives

Not documented in GEM/1 programmer's guide.

In GEM/4 and GEM/5, use function 18 to access drives beyond P:

---

## Function 17 - Yield

### C Interface:

- WORD appl\_yield();

### Entered with:

- *function*=17;
- *#int\_in*=0;
- *#addr\_in*=0.

Allows other processes to run.



Not present in GEM/1; it can be simulated by [evnt\\_timer\(0,0\)](#).

---

## Function 18 - Extended available drive bitmaps

### C Interface:

- None.

### Entered with:

- *function*=18;
- *#int\_in*=1;
- *#int\_out*=4;
- *#addr\_in*=2.
- *int\_in*[0] = function code.  
If *int\_in*[0] is 1:
  - *addr\_in*[0] = bitmap of available drives
  - *addr\_in*[1] = bitmap of available hard drives

### Returns:

(only if *int\_in*[0] was 0)

- *int\_out*[0] = bitmap of available drives (drives Q:-Z:)
- *int\_out*[1] = bitmap of available drives (drives A:-P:)
- *int\_out*[2] = bitmap of available hard drives (drives Q:-Z:)
- *int\_out*[3] = bitmap of available hard drives (drives A:-P:)



This function is only present in GEM/4 and GEM/5.

In *addr\_in*, bit 31 corresponds to drive A:, bit 30 to drive B:, etc.

In *addr\_out*, bit 15 corresponds to drive A: or Q:, 16 to B: or R:, etc.

---

## Function 19 - Finish using AES

### C Interface:

- WORD *appl\_exit*();

### Entered with:

- *function*=19;
- *#int\_in*=0;
- *#addr\_in*=0.

Free all AES resources used by this program. On single-user versions, close all desk accessories.

---

## Function 20 - Await keyboard event

### C Interface:

- UWORD evnt\_keybd();

### Entered with:

- *function*=20;
- *#int\_in*=0;
- *#addr\_in*=0.

### Returns:

- int\_out[0] = keycode.

This call (like all evnt\_\* calls) will block the calling program until an event is received. If the low byte is 0, then the high byte holds a scancode; otherwise, the low byte holds an ASCII character.

---

## Function 21 - Await mouse click

### C Interface:

- WORD evnt\_button(WORD clicks, UWORD mask, UWORD state, WORD \*pmx, WORD \*pmy, WORD \*pmb, WORD \*pks);

### Entered with:

- *function*=21;
- *#int\_in*=3;
- *#addr\_in*=0;
- int\_in[0] = no. of clicks to watch for;
- int\_in[1] = mask of mouse buttons to watch;
- int\_in[2] = state these buttons must be in.

### Returns:

- int\_out[0] = mouse X coordinate
- int\_out[1] = mouse Y coordinate
- int\_out[2] = mouse button status
- int\_out[3] = shift state.

The *mask* and *state* parameters work like this: the call will return when all the appropriate bits of the mouse state match all the appropriate bits of the *state* parameter. As "Professional GEM" puts it:

It is important to notice that all of the target states in btn\_state must occur SIMULTANEOUSLY for the event to be triggered.

Note the limiting nature of this last statement. It prevents a program from waiting for EITHER the left or right button to be pressed. Instead, it must wait for BOTH to be pressed, which is a difficult operation at best.

The mouse button bits are:

- Bit 0: Left
- Bit 1: Right
- Bit 2: Middle

The keyboard shift bits are:

- Bit 0: Right shift
  - Bit 1: Left shift
  - Bit 2: Control
  - Bit 3: Alt
- 

## Function 22 - Await mouse enter/leave rectangle

### C Interface:

- WORD evnt\_mouse(WORD flags, WORD x, WORD y, WORD width, WORD height, WORD \*pmx, WORD \*pmy, WORD \*pmb, WORD \*pks);

### Entered with:

- *function*=22;
- *#int\_in*=5;
- *#addr\_in*=0;
- *int\_in*[0] = 0 to watch for entry, 1 to watch for exit;
- *int\_in*[1-4] = [rectangle to watch](#).

### Returns:

- *int\_out*[0] = mouse X coordinate
- *int\_out*[1] = mouse Y coordinate
- *int\_out*[2] = mouse button status
- *int\_out*[3] = shift state.

The returned values have the same meanings as in [evnt\\_button\(\)](#) above.

---

## Function 23 - Await message

### C Interface:

- WORD evnt\_mesag(WORD buff[8]);

### Entered with:

- *function*=23;
- *#int\_in*=0;
- *#addr\_in*=1;
- *addr\_in*[0] = address of message buffer.

Waits for a message to be received.

See [the message list](#) for information on standard AES messages. All AES messages are 16 bytes long.

In single-tasking GEM, this is the same operation as reading 16 bytes from the message pipe; see function [appl\\_read\(\)](#)

---

## Function 24 - Sleep

### C Interface:

- WORD evnt\_timer(WORD count\_low, WORD count\_high);

### Entered with:

- *function*=24;
- *#int\_in*=2;
- *#addr\_in*=0;
- *int\_in*[0] = low millisecond count;
- *int\_in*[1] = high millisecond count.

Waits until at least the specified time. It may be more, depending on when other processes are blocked.

---

## Function 25 - Monitor all events

### C Interface:

- WORD evnt\_multi(UWORD flags, UWORD bclk, UWORD bmsk, UWORD bst, UWORD m1flags, UWORD m1x, UWORD m1y, UWORD m1w, UWORD m1h, UWORD m2flags, UWORD m2x, UWORD m2y, UWORD m2w, UWORD m2h, WORD mepbuff[8], UWORD tlc, UWORD thc, UWORD \*pmx, UWORD \*pmy, UWORD \*pmb, UWORD \*pks, UWORD \*pkr, UWORD \*pbr);

### Entered with:

- *function*=25;
- *#int\_in*=16;
- *#addr\_in*=1;
- *int\_in*[0] = Event flags (which events are allowed);
- *int\_in*[1] = Mouse click count;
- *int\_in*[2] = Mouse mask;
- *int\_in*[3] = Mouse state;
- *int\_in*[4] = Mouse rectangle 1 flags;
- *int\_in*[5-8] = Mouse rectangle 1;
- *int\_in*[9] = Mouse rectangle 2 flags;
- *int\_in*[10-13] = Mouse rectangle 2;
- *int\_in*[14] = timer count, low;
- *int\_in*[15] = timer count, high;
- *addr\_in*[0] = address of message buffer;

### Returns:

- *int\_out*[0] = Event flags (what event was received);
- *int\_out*[1] = mouse X - coordinate;
- *int\_out*[2] = mouse Y - coordinate;
- *int\_out*[3] = mouse button flags;
- *int\_out*[4] = keyboard state;
- *int\_out*[5] = keycode;
- *int\_out*[6] = mouse click count;

The flags word should be set to a bitwise OR of some of:



MU_KEYBD	0x0001	Key event
MU_BUTTON	0x0002	Mouse click event
MU_M1	0x0004	Mouse rectangle 1 enter/leave
MU_M2	0x0008	Mouse rectangle 2 enter/leave
MU_MESAG	0x0010	Message receive
MU_TIMER	0x0020	Timeout
MU_KEYBD4	0x0100	Used internally in GEM/4 to check for a context-sensitive help request.

On return it will be the type of event that was received.

The parameters and results are as handled by the one-type `evnt_*`() calls listed above.

Some versions of the GEM Programmers Toolkit have an `evnt_evnt()` call that takes the `evnt_multi` parameters in a struct rather than as explicit parameters.

---

## Function 26 - Set double-click time

### C Interface:

- `WORD evnt_dclick(WORD rate, WORD setit);`

### Entered with:

- *function*=26;
- *#int\_in*=2;
- *#addr\_in*=0;
- *int\_in*[0] = new rate;
- *int\_in*[1] = 1 to set new rate, 0 to read old rate.

### Returns:

- *int\_in*[0] = rate now in use

The double-click rate is 0 (slow) to 4 (fast). It is the length of time GEM waits after a mouse click to see if another mouse click is coming.

---

## Function 30 - Create or remove menu bar

### C Interface:

- `VOID menu_bar(OBJECT far *tree, WORD showit);`

### Entered with:

- *function*=30;
  - *#int\_in*=1;
  - *#addr\_in*=1;
  - *int\_in*[0] = 0 to remove the menu bar, nonzero to create it.
  - *addr\_in*[0] = address of the object tree for the menu bar (only needed when the bar is being created).;
-

## Function 31 - Mark menu item as "checked"

### C Interface:

- VOID menu\_ichack([OBJECT](#) far \*tree, WORD item, WORD checkit);

### Entered with:

- *function*=31;
  - *#int\_in*=2;
  - *#addr\_in*=1;
  - *int\_in*[0] = Index of object within the menu tree.
  - *int\_in*[1] = 0 for unchecked; else checked.
  - *addr\_in*[0] = address of the object tree for the menu bar.
- 

## Function 32 - Mark menu item as enabled/disabled

### C Interface:

- VOID menu\_ienable([OBJECT](#) far \*tree, WORD item, WORD checkit);

### Entered with:

- *function*=32;
  - *#int\_in*=2;
  - *#addr\_in*=1;
  - *int\_in*[0] = Index of object within the menu tree.
  - *int\_in*[1] = 0 for disabled; else enabled.
  - *addr\_in*[0] = address of the object tree for the menu bar.
- 

## Function 33 - Mark menu item as selected/normal

### C Interface:

- VOID menu\_tnormal([OBJECT](#) far \*tree, WORD item, WORD normalit);

### Entered with:

- *function*=33;
  - *#int\_in*=2;
  - *#addr\_in*=1;
  - *int\_in*[0] = Index of object within the menu tree.
  - *int\_in*[1] = 0 for selected; else normal.
  - *addr\_in*[0] = address of the object tree for the menu bar.
- 

## Function 34 - Set menu item text

### C Interface:

- VOID menu\_text([OBJECT](#) far \*tree, WORD item, BYTE far \*text);

**Entered with:**

- *function*=34;
- *#int\_in*=1;
- *#addr\_in*=2;
- *int\_in*[0] = Index of object within the menu tree.
- *addr\_in*[0] = address of the object tree for the menu bar.
- *addr\_in*[1] = far pointer to text.

If the segment of the tree address is 0, the offset is the application ID for a desktop accessory. In this case, the text will be set for the accessory's menu item.

In GEM/3, this call actually sets the menu item's object specification; so if you have bitmaps in your menus, this call could be used to point them at a new BITBLK structure. However, it would be unwise to rely on this implementation detail; better to call this only for menu items that are string objects.

---

## Function 35 - Register a desk accessory

**C Interface:**

- WORD menu\_register(WORD pid, BYTE far \*pstr);

**Entered with:**

- *function*=35;
- *#int\_in*=1;
- *#addr\_in*=1;
- *int\_in*[0] = This program's application ID.
- *addr\_in*[0] = Text to appear in the menu option.

**Returns:**

- *int\_out*[0] = object number of menu item for this option.

Returns -1 if no spare menu options available.

---

## Function 36 - Unregister a desk accessory

**C Interface:**

- WORD menu\_unregister(WORD itemid);

**Entered with:**

- *function*=36;
- *#int\_in*=1;
- *#addr\_in*=0;
- *int\_in*[0] = Menu item ID returned from [menu\\_register\(\)](#).

Removes the appropriate desk accessory entry from the menu.



Not present in GEM/1.

---

## Function 37 - Set/get menu click options

### C Interface:

- WORD menu\_click(WORD click, WORD setit);

### Entered with:

- *function*=37;
- *#int\_in*=2;
- *#addr\_in*=0;
- *int\_in*[0] = 0 if click is not required, 1 if click required.
- *int\_in*[1] = 0 to read the current setting, 1 to set the value passed in *int\_in*[0].

### Returns:

- *int\_out*[0] = 0 if click is not required, 1 if it is.



Only present on GEM/3 and later. This call sets whether a click is needed to drop down menus, or whether the menus just appear when the mouse moves over the menu titles.

---

## Function 38



This function is only present in GEM/4, and its parameters and results are unknown. It is likely that it allows a menu bar to be attached to a window.

---

## Function 40 - Add an object to an object tree

### C Interface:

- WORD objc\_add([OBJECT](#) far \*tree, WORD parent, WORD newchild);

### Entered with:

- *function*=40;
- *#int\_in*=2;
- *#addr\_in*=1;
- *int\_in*[0] = index of parent of new object
- *int\_in*[1] = index of new object
- *addr\_in*[0] = address of root of this object tree.

Adds the object as the parent's last child.

---

## Function 41 - Delete an object from an object tree

### C Interface:

- WORD objc\_delete([OBJECT](#) far \*tree, WORD obj);

### Entered with:

- *function*=40;
- *#int\_in*=1;
- *#addr\_in*=1;
- *int\_in*[0] = index of object to delete
- *addr\_in*[0] = address of root of this object tree.

The object and all its children will be removed from the tree; unless it's the root object, in which case nothing happens.

---

## Function 42 - Draw an object or object tree

### C Interface:

- WORD objc\_draw([OBJECT](#) far \*tree, WORD object, WORD depth, WORD xc, WORD yc, WORD wc, WORD hc);

### Entered with:

- *function*=42;
  - *#int\_in*=6;
  - *#addr\_in*=1;
  - *int\_in*[0] = index of object to draw.
  - *int\_in*[1] = depth to draw. 0 means don't draw children. Depth can be up to 8.
  - *int\_in*[2-5]= clipping rectangle to use while drawing.
  - *addr\_in*[0] = address of root of this object tree.
- 

## Function 43 - Find an object from its coordinates

### C Interface:

- WORD objc\_find([OBJECT](#) far \*tree, WORD start\_object, WORD depth, WORD x, WORD y);

### Entered with:

- *function*=43;
- *#int\_in*=4;
- *#addr\_in*=1;
- *int\_in*[0] = index of first object to search. This object and its children will be tested.
- *int\_in*[1] = depth to search. 1-8.
- *int\_in*[2-3]= point the object must cover.
- *addr\_in*[0] = address of root of this object tree.

### Returns:

- *int\_out*[0] = index of object containing the point, -1 if none.
- 

## Function 44 - Find coordinates of an object

### C Interface:

- WORD objc\_offset([OBJECT](#) far \*tree, WORD object);

**Entered with:**

- *function*=44;
- *#int\_in*=1;
- *#addr\_in*=1;
- *int\_in*[0] = index of object to find.
- *addr\_in*[0] = address of root of this object tree.

**Returns:**

- *int\_out*[0] = ?
  - *int\_out*[1-2] = coordinates of object, based on coordinates of root object.
- 

## Function 45 - Change an object's order in the tree

**C Interface:**

- WORD objc\_order([OBJECT](#) far \*tree, WORD obj, WORD newposition);

**Entered with:**

- *function*=45;
- *#int\_in*=2;
- *#addr\_in*=1;
- *int\_in*[0] = index of object to move.
- *int\_in*[1] = position among the object's siblings. 0 => before first; -1 => after last.
- *addr\_in*[0] = address of root of this object tree.

The object and its siblings will be rearranged into the correct order.

---

## Function 46 - Handle a keypress in a text field

**C Interface:**

- WORD objc\_edit([OBJECT](#) far \*tree, WORD object, WORD char, WORD \*idx, WORD kind);

**Entered with:**

- *function*=46;
- *#int\_in*=4;
- *#addr\_in*=1;
- *int\_in*[0] = index of object being edited.
- *int\_in*[1] = keycode to process
- *int\_in*[2] = cursor position in field
- *int\_in*[3] = event type
- *addr\_in*[0] = address of root of this object tree.

**Returns:**

- *int\_out*[0] = ?
- *int\_out*[1] = new cursor position.

Event types are:

```
#define EDSTART 0      /* No-op in current PC versions */
#define EDINIT 1      /* Editing is starting; cursor position is set to
 * end of text */
#define EDCHAR 2      /* Character has been received, process it */
#define EDEND 3       /* Field is losing focus */
```

---

## Function 47 - Change an object's state

### C Interface:

- WORD objc\_change([OBJECT](#) far \*tree, WORD object, WORD depth, WORD xc, WORD yc, WORD wc, WORD hc, WORD newstate, WORD redraw);

### Entered with:

- *function*=47;
  - *#int\_in*=8;
  - *#addr\_in*=1;
  - *int\_in*[0] = index of object to change.
  - *int\_in*[1] = depth to draw. 0 means don't draw children. Depth can be up to 8.
  - *int\_in*[2-5]= clipping rectangle to use while drawing.
  - *int\_in*[6] = new [state](#).
  - *int\_in*[7] = nonzero to redraw the object
  - *addr\_in*[0] = address of root of this object tree.
- 

## Function 50 - Modal entry form

### C Interface:

- WORD form\_do([OBJECT](#) far \*tree, WORD object);

### Entered with:

- *function*=50;
- *#int\_in*=1;
- *#addr\_in*=1;
- *int\_in*[0] = index of object that starts with the focus.
- *addr\_in*[0] = address of root of the object tree that represents the form.

### Returns:

- *int\_out*[0] = index of object that caused loop to finish. Bit 15 will be set if the object was double-clicked.

This will enter a modal loop, and not leave it until either:

- An object with the EXIT flag has been selected;
- An object with the TOUCHEXIT flag has been clicked.

The source of form\_do() is normally provided with GEM developer kits, so that developers can construct versions with special behaviour.

---

## Function 51 - Create or destroy dialog{ue} box

### C Interface:

- WORD form\_dialog(WORD dtype, WORD x1, WORD y1, WORD w1, WORD h1, WORD x2, WORD y2, WORD w2, WORD h2);

### Entered with:

- *function*=51;
- *#int\_in*=9;
- *#addr\_in*=0;
- *int\_in*[0] = subfunction (see below);
- *int\_in*[1-4] = bounding rectangle of box
- *int\_in*[5-8] = second rectangle for zoom effects

The subfunctions are:

```
#define FMD_START 0      /* Save the screen area that will be overwritten.
                          Must be called before the box is drawn */
#define FMD_GROW 1      /* Show "zooming box" for form opening */
#define FMD_SHRINK 2    /* Show "zooming box" for form closing */
#define FMD_FINISH 3    /* Restore the screen area that was overwritten. */
```



The "grow" and "shrink" calls are not implemented in Digital Research's GEM; but [GROWBOX.ACC](#) can be used to provide the correct functionality.

---

## Function 52 - Show alert message

### C Interface:

- WORD form\_alert(WORD defbutton, BYTE FAR \*text);

### Entered with:

- *function*=52;
- *#int\_in*=1;
- *#addr\_in*=1;
- *int\_in*[0] = default button, 1-3 or 0 for none;
- *addr\_in*[0] = message box string.

### Returns

- *int\_out*[0] = button number chosen

The message box string is formed of three sections in square brackets: [*n*] [*line*|*line*|  
...] [*button*|*button*|...]

- The first section is the icon number; 0 for none, 1 for warning ("note"), 2 for question ("wait") and 3 for stop.
- The second section is up to five lines of text, separated by | marks.
- The third section is up to three button captions; again, they are separated by | marks.

So, a typical message could be: "[1][The printer is not responding][ Retry | Cancel ]".



ViewMAX/2 and later parse the "default button" parameter as two bytes; the low byte is the



default button and the high byte is the cancel button (ESC is a shortcut for it). This style of parameter will crash earlier GEM versions.

---

## Function 53 - Show system alert message

### C Interface:

- WORD form\_error(WORD err);

### Entered with:

- *function*=53;
- *#int\_in*=1;
- *#addr\_in*=0;
- *int\_in*[0] = DOS 2.x error code (see the Interrupt List, AH=59h);

### Returns:

- *int\_out*[0] = 0 to cancel, else retry.
- 

## Function 54 - Centre an object on the screen

### C Interface:

- WORD form\_center([OBJECT](#) far \*obj, WORD \*x, WORD \*y, WORD \*w, WORD \*h);

### Entered with:

- *function*=54;
- *#int\_in*=0;
- *#addr\_in*=1;
- *addr\_in*[0] = Address of object to centre.

### Returns:

- *int\_out*[1-4] = object bounding rectangle, centred.
- 

## Function 55 - Handle keyboard event in a modal form

### C Interface:

- WORD form\_keybd([OBJECT](#) far \*tree, WORD obj, WORD nxtobj, WORD char, WORD \*pnext, WORD \*pchar);

### Entered with:

- *function*=55;
- *#int\_in*=3;
- *#addr\_in*=1;
- *int\_in*[0] = Index of object with the focus.
- *int\_in*[1] = Index of object expected to be next with focus.
- *int\_in*[2] = Keycode to process
- *addr\_in*[0] = Address of root object of the form.

**Returns:**

- `int_out[0]` = 1 if modal loop should continue, else 0.
  - `int_out[1]` = Next object to have the focus.
  - `int_out[2]` = 0 if keycode processed, else keycode.
- 

**Function 56 - Handle click event in a modal form****C Interface:**

- `WORD form_button(OBJECT far *tree, WORD obj, WORD clicks, WORD *pnext);`

**Entered with:**

- `function=56;`
- `#int_in=2;`
- `#addr_in=1;`
- `int_in[0]` = Index of object with the focus.
- `int_in[1]` = No. of clicks received.
- `addr_in[0]` = Address of root object of the form.

**Returns:**

- `int_out[0]` = 1 if modal loop should continue, else 0.
  - `int_out[1]` = Next object to have the focus.
- 

**Function 57 - GEM/4 Modal entry form****C Interface:**

- None.

**Entered with:**

- `function=57;`
- `#int_in=2;`
- `#addr_in=1;`
- `int_in[0]` = index of object that starts with the focus.
- `int_in[1]` = GEM/4 context-sensitive help flag.
- `addr_in[0]` = address of root of the object tree that represents the form.

**Returns:**

- `int_out[0]` = index of object that caused loop to finish. Bit 15 will be set if the object was double-clicked.

If the context-sensitive help flag is set, pressing the [F1] key will trigger a 'help' mode. Clicking on an object when in 'help' mode then ends the modal loop, with an object number of -1 returned.



This function is only available in GEM/4 and GEM/5.

---

## Function 58 - GEM/4 alert

### C Interface:

- None.

### Entered with:

- *function*=58;
- *#int\_in*=2;
- *#addr\_in*=1;
- *int\_in*[0] = default button, 1-3 or 0 for none;
- *int\_in*[1] = Help flag, 1 if context-sensitive help supported.
- *addr\_in*[0] = message box string.

### Returns

- *int\_out*[0] = button number chosen, -1 for context-sensitive help request.



This function is only available in GEM/4 and GEM/5.

If context-sensitive help was chosen, WM\_REDRAW messages are sent out to some windows.

---



The following functions (60-67) are mentioned in the GEM source under the condition "MULTIAPP". They are not included in any single-tasking PC GEM or documented in any known GEM programmer's toolkit.

---

## Function 60 - Create a process

### C Interface:

- WORD proc\_create(VOID far \*address, LONG size, WORD is\_swap, WORD is\_gem, WORD \*num);

### Entered with:

- *function*=60;
- *#int\_in*=2;
- *#addr\_in*=2;
- *int\_in*[0] = "is swap" flag
- *int\_in*[1] = "is GEM" flag
- *addr\_in*[0] = Address to load process.
- *addr\_in*[1] = Space to allocate for process.

### Returns:

- *int\_out*[0] = ? 0 if failed, else succeeded ?
  - *int\_out*[1] = process ID of new process
- 

## Function 61 - Run process

### C Interface:

- WORD proc\_run(WORD pid, WORD is\_graphical, WORD is\_overlaid, BYTE far \*command, BYTE FAR \*tail);

**Entered with:**

- *function*=61;
- #*int\_in*=3;
- #*addr\_in*=2;
- int\_in[0] = Process ID
- int\_in[1] = "is graphical" flag
- int\_in[2] = "is overlaid" flag
- addr\_in[0] = ? Name of program
- addr\_in[1] = ? Command tail to pass to it.

**Returns:**

- int\_out[0] = ? 0 if failed, else succeeded ?
- 

## Function 62 - Delete process

**C Interface:**

- WORD proc\_delete(WORD pid);

**Entered with:**

- *function*=62;
- #*int\_in*=1;
- #*addr\_in*=0;
- int\_in[0] = Process ID to delete.

**Returns:**

- int\_out[0] = ? 0 if failed, else succeeded ?
- 

## Function 63 - Process information

**C Interface:**

- WORD proc\_info(WORD pid, WORD \*is\_swap, WORD \*is\_gem, (VOID far \*)\* address, LONG \*csize, (VOID far \*)\*endmem, LONG \*ssize, (VOID far \*)\* intaddr);

**Entered with:**

- *function*=63;
- #*int\_in*=1;
- #*addr\_in*=0;
- int\_in[0] = Process ID about which to get info.

**Returns:**

- int\_out[0] = ? 0 if failed, else succeeded ?
- int\_out[1] = "is swap" flag
- int\_out[2] = "is GEM" flag
- addr\_out[0] = beginning address

- `addr_out[1]` = "csize" - code size?
  - `addr_out[2]` = "end memory"
  - `addr_out[3]` = "ssize" - stack size?
  - `addr_out[4]` = "intaddr"
- 

## Function 64 - Allocate memory

### C Interface:

- `WORD proc_malloc(VOID far *address, LONG size);`

### Entered with:

- `function=65;`
- `#int_in=0;`
- `#addr_in=2;`
- `addr_in[0]` = base of memory
- `addr_in[1]` = length to allocate

### Returns:

- `int_out[0]` = ? 0 if failed, else succeeded ?
- 

## Function 65 - Free memory

### C Interface:

- `WORD proc_free(WORD pid);`

### Entered with:

- `function=65;`
- `#int_in=1;`
- `#addr_in=0;`
- `int_in[0]` = process ID.

### Returns:

- `int_out[0]` = ? 0 if failed, else succeeded ?
- 

## Function 66 - Switch to process?

### C Interface:

- `WORD proc_switch(WORD pid);`

### Entered with:

- `function=66;`
- `#int_in=1;`
- `#addr_in=0;`
- `int_in[0]` = process ID to switch to?.

**Returns:**

- `int_out[0]` = ? 0 if failed, else succeeded ?
- 

**Function 67 - Block a process?****C Interface:**

- `WORD proc_setblock(WORD pid);`

**Entered with:**

- `function=67;`
- `#int_in=1;`
- `#addr_in=0;`
- `int_in[0]` = process ID to block?.

**Returns:**

- `int_out[0]` = ? 0 if failed, else succeeded ?
- 

**Function 70 - Drag box for sizing****C Interface:**

- `WORD graf_rubbox(WORD x, WORD y, WORD mw, WORD mh, WORD *pw, WORD *ph);`

**Entered with:**

- `function=70;`
- `#int_in=4;`
- `#addr_in=0;`
- `int_in[0]` = X coordinate of box (top left corner)
- `int_in[1]` = Y coordinate of box (top left corner)
- `int_in[2]` = minimum width of box
- `int_in[3]` = minimum height of box.

**Returns:**

- `int_out[1]` = final width of box
- `int_out[2]` = final height of box

Returns when the mouse button is released.

---

**Function 71 - Drag box for drag/drop****C Interface:**

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

**Entered with:**

- *function*=71;
- *#int\_in*=8;
- *#addr\_in*=0;
- *int\_in*[0] = width of box
- *int\_in*[1] = height of box
- *int\_in*[2] = initial X coordinate of box, relative to mouse cursor X
- *int\_in*[3] = initial Y coordinate of box, relative to mouse cursor Y
- *int\_in*[4-7] = bounding rectangle outside which the box cannot be dragged.

**Returns:**

- *int\_out*[1] = final X coordinate of box
- *int\_out*[2] = final Y coordinate of box

Returns when the mouse button is released.

---

## Function 72 - Draw moving box

**C Interface:**

- WORD *graf\_mbox*(WORD *w*, WORD *h*, WORD *sx*, WORD *sy*, WORD *dx*, WORD *dy*);

**Entered with:**

- *function*=72;
- *#int\_in*=6;
- *#addr\_in*=0;
- *int\_in*[0] = width of box
- *int\_in*[1] = height of box
- *int\_in*[2] = initial X coordinate of box
- *int\_in*[3] = initial Y coordinate of box
- *int\_in*[4] = final X coordinate of box
- *int\_in*[5] = final Y coordinate of box

This suffers from a slight problem on modern PCs; it draws too fast to see! This problem has been corrected in FreeGEM.

---

## Function 73 - Draw expanding box

**C Interface:**

- WORD *graf\_growbox*(WORD *x1*, WORD *y1*, WORD *w1*, WORD *h1*, WORD *x2*, WORD *y2*, WORD *w2*, WORD *h2*);

**Entered with:**

- *function*=73;
- *#int\_in*=8;
- *#addr\_in*=0;
- *int\_in*[0-3] = "from" rectangle
- *int\_in*[4-7] = "to" rectangle

This animates the "from" rectangle moving to the centre of the "to" rectangle, and then expanding to

fill it.



Many versions of Digital Research GEM do not implement this function, and some development kits include dummy bindings that don't call the AES. [GROWBOX.ACC](#) implements it on non-equipped GEMs.

---

## Function 74 - Draw contracting box

### C Interface:

- WORD graf\_shrinkbox(WORD x1, WORD y1, WORD w1, WORD h1, WORD x2, WORD y2, WORD w2, WORD h2);

### Entered with:

- *function*=74;
- *#int\_in*=8;
- *#addr\_in*=0;
- *int\_in*[0-3] = ["from" rectangle](#)
- *int\_in*[4-7] = ["to" rectangle](#)

This performs the reverse animation of [graf\\_growbox\(\)](#).



Many versions of Digital Research GEM do not implement this function, and some development kits include dummy bindings for it. [GROWBOX.ACC](#) implements it on non-equipped GEMs.

---

## Function 75 - See if a mouse moves off a control

### C Interface:

- WORD graf\_watchbox([OBJECT](#) far \*tree, WORD obj, WORD instate, WORD outstate);

### Entered with:

- *function*=75;
- *#int\_in*=3;
- *#addr\_in*=1;
- *addr\_in*[0] = address of the object tree containing the object to watch.
- *int\_in*[0] = index of the object in the tree
- *int\_in*[1] = object state if the pointer is over it
- *int\_in*[2] = object state if the pointer isn't.

### Returns:

- *int\_out*[0] = 0 if the pointer is now inside the object, 1 if it is now outside it.

This will return when the mouse button is released.

---



## Function 76 - Handle vertical/horizontal drag

### C Interface:

- WORD graf\_slidebox([OBJECT](#) far \*tree, WORD parent, WORD obj, WORD isvertical);

### Entered with:

- *function*=76;
- *#int\_in*=3;
- *#addr\_in*=1;
- *addr\_in*[0] = address of the object tree containing the "thumb" object to drag.
- *int\_in*[0] = index of the object's parent in the tree, outside which the object may not be dragged;
- *int\_in*[1] = index of the object itself;
- *int\_in*[2] = 1 for vertical, 0 for horizontal.

### Returns:

- *int\_out*[0] = Position of "thumb" in parent, 0-1000 (Position difference / size difference \* 1000).

This will return when the mouse button is released.

---

## Function 77 - Get GEM's VDI handle

### C Interface:

- WORD graf\_handle(WORD \*pwchar, WORD \*phchar, WORD \*pwbox, WORD \*phbox);,

### Entered with:

- *function*=77;
- *#int\_in*=0;
- *#addr\_in*=0.

### Returns:

- *int\_out*[0] = The VDI handle (graphics context) used for all AES drawing operations;
  - *int\_out*[1] = [Character width](#)
  - *int\_out*[2] = [Character height](#)
  - *int\_out*[3] = width of box to surround a character (scaled from *int\_out*[4])
  - *int\_out*[4] = height of box to surround a character (*int\_out*[2] + 3)
- 

## Function 78 - Set mouse pointer

### C Interface:

- WORD graf\_mouse(WORD number, WORD FAR \*form);

### Entered with:

- *function*=78;
- *#int\_in*=1;
- *#addr\_in*=1;

- `int_in[0]` = cursor number. 0-7 are standard cursors; 255-257 are special.
- `addr_in[0]` = data for custom cursor (if number is 255).

The cursor numbers are:

```
#define ARROW          0          /* Standard pointer */
#define TEXT_CRSR     1          /* I-beam */
#define HOURLASS      2          /* Hourglass */
#define POINT_HAND    3          /* Pointing finger */
#define FLAT_HAND     4          /* Dragging hand */
#define THIN_CROSS    5          /* Thin crosshair */
#define THICK_CROSS   6          /* Thick crosshair */
#define OUTLN_CROSS   7          /* Outline crosshair */
#define USER_DEF      255       /* custom shape */
#define M_OFF         256       /* Hide mouse pointer */
#define M_ON          257       /* Show mouse pointer */
```

---

## Function 79 - Read mouse/keyboard state

### C Interface:

- `WORD graf_mkstate(WORD *px, WORD *py, WORD *pmouse, WORD *pkeyb);`

### Entered with:

- `function=79;`
- `#int_in=0;`
- `#addr_in=0.`

### Returns:

- `int_out[1]` = mouse X coordinate
  - `int_out[2]` = mouse Y coordinate
  - `int_out[3]` = [mouse button state](#)
  - `int_out[4]` = [keyboard shift state](#)
- 

## Function 80 - Get scrap directory name

### C Interface:

- `WORD scrp_read(BYTE far *pscrap);`

### Entered with:

- `function=80;`
- `#int_in=0;`
- `#addr_in=1;`
- `addr_in[0]` = address of buffer to receive scrap directory name.

### Returns:

- Buffer filled with "-"-terminated filename.
- `int_out[0]` = bitmap of files existing in that directory.



This function is not present in ViewMAX.

The file types returned by this function are:

Bit 0: SCRAP.CSV  
Bit 1: SCRAP.TXT  
Bit 2: SCRAP.GEM  
Bit 3: SCRAP.IMG  
Bit 4: SCRAP.DCA  
Bit 15: SCRAP.USR

The path returned will end with a "\".

---

## Function 81 - Set scrap directory name

### C Interface:

- WORD scrp\_write(BYTE far \*pscrap);

### Entered with:

- *function*=81;
- *#int\_in*=0;
- *#addr\_in*=1;
- *addr\_in*[0] = address of new scrap directory name.

### Returns:

- *int\_out*[0] = 1 if the directory exists, 0 if it doesn't. If 0 was returned, then a new scrap directory should be set.



This function is not present in ViewMAX.

---

## Function 82 - Empty scrap directory

### C Interface:

- WORD scrp\_clear();

### Entered with:

- *function*=82;
- *#int\_in*=0;
- *#addr\_in*=0.

### Returns:

- *int\_out*[0] = 1.

The files deleted are those mentioned earlier:

- SCRAP.CSV
- SCRAP.TXT
- SCRAP.GEM
- SCRAP.IMG
- SCRAP.DCA
- SCRAP.USR



This function is not present in GEM/1 or ViewMAX.

---

## Function 90 - File selector

### C Interface:

- WORD fsel\_input(BYTE far \*pipath, BYTE far \*pisel, WORD \*pbutton);

### Entered with:

- *function*=90;
- *#int\_in*=1;
- *#addr\_in*=2;
- *addr\_in*[0] = address of initial directory and wildcard, eg "C:\\*.GEM";
- *addr\_in*[1] = address of initial filename selection, eg "PICTURE.GEM".

### Returns:

- *int\_out*[0] = 0 if selector was not drawn, else 1.
- *int\_out*[1] = 0 if "cancel" was clicked, 1 if "OK".
- Buffers now contain selected directory and filename.



This function is not present in ViewMAX/3, and will crash ViewMAX/2.

---

## Function 91 - File selector with title

### C Interface:

- WORD fsel\_exinput(BYTE far \*pipath, BYTE far \*pisel, WORD \*pbutton, BYTE far \*title);

### Entered with:

- *function*=91;
- *#int\_in*=1 (error? No integers are passed)
- *#addr\_in*=3;
- *addr\_in*[0] = address of initial directory and wildcard, eg "C:\\*.GEM";
- *addr\_in*[1] = address of initial filename selection, eg "PICTURE.GEM";
- *addr\_in*[2] = address of form title for the selector.

### Returns:

- *int\_out*[0] = 0 if selector was not drawn, else 1.
- *int\_out*[1] = 0 if "cancel" was clicked, 1 if "OK".
- Buffers now contain selected directory and filename.



This is only present in FreeGEM. Note that if you are adding a binding for this function, the *addr\_in*[] array may need to be redeclared for 3 entries instead of 2.

To check for this feature, use [appl\\_init\(\)](#) and check that [xbuf.arch](#) is nonzero.



This function is GEM/4 and GEM/5 also have a function 91, which behaves differently and

is documented below. The FreeGEM function 91 is based on the Atari function of the same number and name.

---

## Function 91 - GEM/4, GEM/5 - File selector

### C Interface:

- None

### Entered with:

- *function*=91;
- *#int\_in*=1;
- *#addr\_in*=2;
- *int\_in*[0] = Context sensitive help flag
- *addr\_in*[0] = address of initial directory and wildcard, eg "C:\\*.GEM";
- *addr\_in*[1] = address of initial filename selection, eg "PICTURE.GEM";

### Returns:

- *int\_out*[0] = 0 if selector was not drawn, else 1.
- *int\_out*[1] = 0 if "cancel" was clicked, 1 if "OK".
- Buffers now contain selected directory and filename.



This is only present in GEM/4 and GEM/5. It stands in the same relation to the normal file selector as [function 57](#) does to [function 50](#).

---

## Function 100 - Create a window

### C Interface:

- WORD *wind\_create*(UWORD kind, WORD x, WORD y, WORD w, WORD h);

### Entered with:

- *function*=100;
- *#int\_in*=5;
- *#addr\_in*=0;
- *int\_in*[0] = window styles (see below);
- *int\_in*[1-4] = Rectangle for when the window is maximised.

### Returns:

- *int\_out*[0] = -1 if no windows available, else window handle.

Single-tasking GEM allows 7 windows (8 including the desktop window). Styles are:

```
#define NAME      0x0001      /* Titlebar */
#define CLOSER    0x0002      /* Close button */
#define FULLER    0x0004      /* Maximise button */
#define MOVER     0x0008      /* Window can be moved */
#define INFO      0x0010      /* Info bar (below the title bar) */
#define SIZER     0x0020      /* Resize button (bottom right-hand corner) */
#define UPARROW   0x0040      /* "up" button for vertical scroll bar */
#define DNARROW   0x0080      /* "down" button for vertical scroll bar */
#define VSLIDE    0x0100      /* Vertical scroll bar */
```

```
#define LFARROW 0x0200          /* "left" button for horizontal scroll bar */
#define RTARROW 0x0400          /* "right" button for horizontal scroll bar */
#define HSLIDE  0x0800          /* Horizontal scroll bar */
```



The following style is present in GEM/2 and later:

```
#define HOTCLOSE 0x1000          /* "Hot" close button */
```

The "hot" close button is used by the two-fixed-windows Desktop in GEM/2 and later. The difference between the "Hot" and normal close buttons is that you can move the mouse pointer out of the close button while holding it down, and WM\_CLOSE messages will still be sent to the window.



The following is not supported in PC GEM, but are in other GEMs:

```
#define SMALLER 0x4000          /* Iconify button */
```

---

## Function 101 - Open a window

### C Interface:

- WORD wind\_open(WORD hwind, WORD x, WORD y, WORD w, WORD h);

### Entered with:

- *function*=101;
- *#int\_in*=5;
- *#addr\_in*=0;
- *int\_in*[0] = Window handle;
- *int\_in*[1-4] = Window rectangle.

Opens a window that has previously been created.

---

## Function 102 - Close a window

### C Interface:

- WORD wind\_close(WORD hwind);

### Entered with:

- *function*=102;
- *#int\_in*=1;
- *#addr\_in*=0;
- *int\_in*[0] = Window handle.

Closes a window, but does not free its handle. The window can be reopened if desired.

---

## Function 103 - Delete a window

### C Interface:

- WORD wind\_delete(WORD hwind);

### Entered with:

- *function*=103;
- *#int\_in*=1;
- *#addr\_in*=0;
- *int\_in*[0] = Window handle.

Deletes the window, so its slot can be re-used.

---

## Function 104 - Get window properties

### C Interface:

- WORD wind\_get(WORD hwind, WORD field, WORD \*pw1, WORD \*pw2, WORD \*pw3, WORD \*pw4);

### Entered with:

- *function*=104;
- *#int\_in*=2;
- *#addr\_in*=0;
- *int\_in*[0] = Window handle;
- *int\_in*[1] = Property to read.

### Returns:

- *int\_out*[1-4] = returned values

The window properties which can be read are:

#### Window areas (return a rectangle)

```
#define WF_WXYWH      4      /* Work area. The area that can be drawn on. */
#define WF_CXYWH      5      /* Current window size/position */
#define WF_PXYWH      6      /* Previous window size/position */
#define WF_FXYWH      7      /* Full window size/position */
#define WF_FIRSTXYWH  11     /* First rectangle needing repainting */
#define WF_NEXTXYWH   12     /* Next rectangle needing repainting */
```

#### Sliders (return an integer)

```
#define WF_HSLIDE     8      /* Position of horizontal slider, 0-1000 */
#define WF_VSLIDE     9      /* Position of vertical slider, 0-1000 */
#define WF_HSLSIZ     15     /* Size of horizontal thumb, 1-1000 */
#define WF_VSLSIZ     16     /* Size of vertical thumb, 1-1000 */
```

#### Others (return various)

```
#define WF_TOP        10     /* Handle of topmost window, 0 if none open */
#define WF_SCREEN     17     /* Address of AES graphics buffer. */
#define WF_TATTRB     18     /* Window attributes. */
```

The rectangles WF\_CXYWH, WF\_PXYWH, WF\_FXYWH include window decorations.

WF\_SCREEN returns four values: Buffer offset, buffer segment, buffer length (low), buffer length(high). The buffer is used to back up the screen behind menus; if you use it, you should

disable menus with [wind\\_update\(\)](#).

WF\_TATTRB returns a bitmap in which two bits can be set:

```
#define WA_SUBWIN 0x01 /* The two windows on a GEM/3 desktop have this
* attribute set. If the active window has the
* WA_SUBWIN attribute, then other WA_SUBWIN windows
* on the screen can also get events. */
#define WA_KEEPPWIN 0x02 /* In multitasking GEM: the window should not be
* closed when the application runs a DOS program */
```



In some versions of the [FreeGEM AES](#), it is possible to read the window decorations using this call. To check for this feature, use [appl\\_init\(\)](#) and check that bit 2 of [xbuf.abilities](#) is set.

```
#define WF_OBFLAG 1001 /* Window tree: flag words */
#define WF_OBTYP 1002 /* Window tree: type words */
#define WF_OBSPEC 1003 /* Window tree: spec dwords */

wind_get(n, WF_OBFLAG, &a, &b, &c, &d) - a = object flags for decoration "n"
wind_get(n, WF_OBTYP, &a, &b, &c, &d) - a = object type of decoration "n"
wind_get(n, WF_OBSPEC, &a, &b, &c, &d) - (b<<16)|a = spec of decoration "n"

wind_set(n, WF_OBFLAG, a, b, c, d) - Set flags for decoration "n" to "a"
wind_set(n, WF_OBSPEC, a, b, c, d) - Set spec of decoration "n" to
(b<<16)|a
```

**The "n" parameter is one of:**

```
#define W_BOX 0 outline
#define W_TITLE 1 titlebar outline
#define W_CLOSER 2 close box
#define W_NAME 3 titlebar
#define W_FULLER 4 full-size box
#define W_INFO 5 info bar
#define W_DATA 6 work area outline
#define W_WORK 7 work area
#define W_SIZER 8 resize box
#define W_VBAR 9 vertical scroll bar outline
#define W_UPARROW 10 scroll up box
#define W_DNARROW 11 scroll down box
#define W_VSLIDE 12 vertical scroll bar
#define W_VELEV 13 scroll thumb
#define W_HBAR 14 horizontal scroll bar outline
#define W_LFARROW 15 scroll left box
#define W_RTARROW 16 scroll right box
#define W_HSLIDE 17 horizontal scroll bar
#define W_HELEV 18 horizontal scroll bar thumb
```

---

## Function 105 - Set window properties

### C Interface:

- WORD wind\_set(WORD hwind, WORD field, WORD w1, WORD w2, WORD w3, WORD w4);

### Entered with:

- *function*=105;



- `#int_in=6;`
- `#addr_in=0;`
- `int_in[0]` = Window handle;
- `int_in[1]` = Property to set.
- `int_in[2-5]` = Parameters as required

The window properties which can be set are:

**Sliders - take an integer parameter**

```
#define WF_HSLIDE      8      /* Position of horizontal slider, 0-1000 */
#define WF_VSLIDE      9      /* Position of vertical slider, 0-1000 */
#define WF_HLSIZ       15     /* Size of horizontal thumb, 1-1000 */
#define WF_VLSIZ       16     /* Size of vertical thumb, 1-1000 */
```

**Strings - these take two parameters, offset and segment**

```
#define WF_NAME        2      /* Window title */
#define WF_INFO        3      /* Info line */
```

**Window size - takes a rectangle as parameter**

```
#define WF_CXYWH       5      /* Window size*/
```

**Others**

```
#define WF_TOP          10     /* Top (active) window */
#define WF_NEWDESK     14     /* Set object tree to draw on desktop*/
#define WF_TATTRB      18     /* Set window attributes */
#define WF_SIZTOP      19     /* Resize window and move to front */
#define WF_COTOP       20     /* To make two windows look like one */
```

`WF_NEWDESK` takes three parameters; the first two are the offset and segment of an object tree, and the third is the index of the object within it (normally 0). This tree will then be drawn instead of the standard desktop.

`WF_COTOP` takes no extra parameters. It sets the window handle passed in `int_in[1]` as "co-top window" to the currently active window. You can pass a window handle of -1 for "none". When a window's co-top window is active, the window will also be drawn as active. This is for the tree view in ViewMAX, which uses two windows side by side; both must be drawn as active or inactive together.



In some versions of the [FreeGEM AES](#), it is possible to change the window decorations using this call; see [wind\\_get\(\)](#) for details.



`WF_COTOP` is only available in ViewMAX/2 and later.

## Function 106 - Find window from point

### C Interface:

- `WORD wind_find(WORD x, WORD y);`

### Entered with:

- `function=106;`
- `#int_in=2;`
- `#addr_in=0;`
- `int_in[0]` = X coordinate of point;
- `int_in[1]` = Y coordinate of point.

**Returns:**

- `int_out[0]` = window handle, 0 if no window contains the point.
- 

**Function 107 - Lock screen for drawing****C Interface:**

- `WORD wind_update(WORD begin);`

**Entered with:**

- `function=107;`
- `#int_in=1;`
- `#addr_in=0;`
- `int_in[0]` = update flag.

If the update flag is 1, then the AES is prevented from making changes to the screen (because the program is drawing on it). If the flag is 0, the AES is allowed to make changes. The update flag can also be 3 (take full control of mouse pointer - like a modal form does) or 2 (release mouse pointer).

---

**Function 108 - Calculate window geometry****C Interface:**

- `WORD wind_calc(WORD wctype, WORD kind, WORD x, WORD y, WORD w, WORD h, WORD *px, WORD *py, WORD *pw, WORD *ph);`

**Entered with:**

- `function=108;`
- `#int_in=6;`
- `#addr_in=0;`
- `int_in[0]` = Calculation type (0 = work area to outer rectangle, 1 = outer rectangle to work area).
- `int_in[1]` = [Window style](#)
- `int_in[2-5]` = input rectangle

**Returns:**

- `int_out[1-4]` = calculated rectangle
- 

**Function 110 - Load resources****C Interface:**

- `WORD rsrc_load(BYTE far *filename);`

**Entered with:**

- `function=110;`
- `#int_in=0;`
- `#addr_in=1;`

- `addr_in[0]` = filename of resource file.

**Returns:**

- `int_out[0]` = 0 if resource load failed, nonzero if success.
  - The `globals` array will contain (at `global+20`):
 

DD	object tree pointers
DD	resource file address
DD	resource file length
- 

## Function 111 - Free resources

**C Interface:**

- `WORD rsrc_free(VOID);`

**Entered with:**

- `function=111;`
- `#int_in=0;`
- `#addr_in=0.`

**Returns:**

- `int_out[0]` = 0 if failed to free resources, nonzero if success.
- 

## Function 112 - Get resource address

**C Interface:**

- `WORD rsrc_gaddr(WORD type, WORD id, (VOID far *) *addr);`

**Entered with:**

- `function=112;`
- `#int_in=2;`
- `#addr_in=0;`
- `int_in[0]` = resource type;
- `int_in[1]` = resource number.

**Returns:**

- `int_out[0]` = 0 if resource not found, else nonzero.
- `addr_out[0]` = resource address

This is the only function in single-tasking GEM which uses `addr_out`. The resource types that can be passed are:

```
#define R_TREE          0      /* Object tree
#define R_OBJECT       1      /* Objects (as one big array) */
#define R_TEDINFO      2      /* Text field
#define R_ICONBLK     3      /* Icon
#define R_BITBLK      4      /* Bitmap
#define R_STRING       5      /* Free-form ASCII string */
#define R_IMAGEDATA    6      /* Free-form image data */
#define R_OBSPEC       7      /* Pointer to ob_spec member of object */
```

```

#define R_TEPTEXT      8      /* Pointer to te_ptext member of text field */
#define R_TEPTEMPLT   9      /* Pointer to te_ptmplt member of text field */
#define R_TEPVALID    10     /* Pointer to te_pvalid member of text field */
#define R_IBPMASK     11     /* Pointer to ib_pmask member of icon */
#define R_IBPDATA     12     /* Pointer to ib_pdata member of icon */
#define R_IBPTEXT     13     /* Pointer to ib_ptext member of icon */
#define R_BIPDATA     14     /* Pointer to bi_pdata member of bitmap */
#define R_FRSTR       15     /* Pointer to free-form strings */
#define R_FRIMG       16     /* Pointer to free-form images */

```

The call types normally used are R\_TREE, R\_STRING and R\_IMAGEDATA. It is rare that a program needs to access a TEDINFO, ICONBLK or BITBLK directly rather than via its containing object tree.

---

## Function 113 - Set resource address

### C Interface:

- WORD rsrc\_saddr(WORD type, WORD id, (VOID far \*) addr );

### Entered with:

- *function*=113;
- *#int\_in*=2;
- *#addr\_in*=1;
- *int\_in*[0] = resource type;
- *int\_in*[1] = resource number;
- *addr\_in*[0] = address to write to the resource.

### Returns:

- *int\_out*[0] = 0 if resource not found, else nonzero.

This is called with one of the "pointer" resource types; R\_OBSPEC to R\_FRIMG. For example, it could set a new icon bitmap. In general programs manipulate resource pointers themselves rather than using this function.

---

## Function 114 - Fix object positions

### C Interface:

- WORD rsrc\_obfix([OBJECT](#) far \*tree, WORD obj);

### Entered with:

- *function*=114;
- *#int\_in*=1;
- *#addr\_in*=1;
- *int\_in*[0] = object number within tree;
- *addr\_in*[0] = root of the object tree.

This will convert an object's dimensions from the form used in the resource file (low byte is dimension in characters, high byte is offset in pixels) to screen coordinates. A height of 25 characters or a width of 80 is taken to mean as high/wide as possible.

---

## Function 115 - Load resources (supports EMS)

### C Interface:

- None.

### Entered with:

- *function*=115;
- *#int\_in*=0;
- *#addr\_in*=1;
- *addr\_in*[0] = filename of resource file.

### Returns:

- *int\_out*[0] = 0 if resource load failed, nonzero if success.
- The *globals* array will contain (at *global*+20):

DD	object tree pointers
DD	resource file address
DD	resource file length



This function is only present in GEM/4 and GEM/5. It behaves just like the normal version except that resources can be loaded into EMS memory. If resources are in EMS, then they should only be manipulated using GEM functions, not by trying to access them directly.

## Function 116



This function is used in GEM/4 and GEM/5 to manipulate resources in EMS, but exactly what it does is not known.

---

## Function 120 - Read shell command

### C Interface:

- WORD *shel\_read*(BYTE far \**pcmd*, BYTE far \**ptail*);

### Entered with:

- *function*=120;
- *#int\_in*=0;
- *#addr\_in*=2;
- *addr\_in*[0] = buffer for program name;
- *addr\_in*[1] = buffer for command tail.

If [shel\\_write\(\)](#) has not been used, this will return the information with which the program was started.

---

## Function 121 - Set next program to execute

### C Interface:

- WORD shel\_write(WORD doex, WORD isgem, WORD isover, BYTE far \*pcmd, BYTE far \*ptail);

### Entered with:

- *function*=121;
- #*int\_in*=3;
- #*addr\_in*=2;
- *int\_in*[0] = "doex" flag - 0 to quit GEM after running the program, 1 to return to the desktop;
- *int\_in*[1] = "isgem" - 0 if program runs in text mode, 1 for graphics mode;
- *int\_in*[2] = "isover" - 0 to keep current program in memory, 1 to load specified program instead of current, 2 to unload AES and restart it afterwards;
- *addr\_in*[0] = address of program name;
- *addr\_in*[1] = address of command tail.

### Returns:

- *int\_out*[0] = 0 if failed, 1 if OK.
- 

## Function 122 - Get shell settings

### C Interface:

- WORD shel\_get(VOID far \*buffer, WORD len);

### Entered with:

- *function*=122;
- #*int\_in*=1;
- #*addr\_in*=1;
- *int\_in*[0] = Length of data (up to 1k);
- *addr\_in*[0] = Destination for data.

This function is intended for use only by the Desktop, to save its state while other programs are executing. Some versions of the GEM PTK do not supply bindings for this function.

---

## Function 123 - Save shell settings

### C Interface:

- WORD shel\_put(VOID far \*buffer, WORD len);

### Entered with:

- *function*=123;
- #*int\_in*=1;
- #*addr\_in*=1;
- *int\_in*[0] = Length of data (up to 1k);
- *addr\_in*[0] = Address of data.

This function is intended for use only by the Desktop, to save its state while other programs are executing. Some versions of the GEM PTK do not supply bindings for this function.

---

## Function 124 - Find a program

### C Interface:

- WORD shel\_find(BYTE far \*spec);

### Entered with:

- *function*=124;
- *#int\_in*=0;
- *#addr\_in*=1;
- *addr\_in*[0] = address of buffer containing program name.

### Returns:

- *int\_out*[0] = 0 if file can't be found, 1 if OK.
- Buffer contains real name of program.

This will search the current directory and the search path.

---

## Function 125 - Read environment

### C Interface:

- WORD shel\_envrn(BYTE far \* far \*ptr, BYTE far \*string);

### Entered with:

- *function*=125;
- *#int\_in*=0;
- *#addr\_in*=2;
- *addr\_in*[0] = address of far pointer which will be set;
- *addr\_in*[1] = address of string to find (eg: "PATH=").

### Returns:

- The pointer is set to NULL if nothing was found, or the address of the next byte after the search string otherwise.
- 

## Function 126 - Get the default application name

### C Interface:

- WORD shel\_rdef(BYTE far \*pcmd, BYTE far \*pdir);

### Entered with:

- *function*=126;
- *#int\_in*=0;
- *#addr\_in*=2;

- `addr_in[0]` = address of buffer for command name;
- `addr_in[1]` = address of buffer for directory.

**Returns:**

- Buffers set accordingly.
- 

## Function 127 - Set the default application name

**C Interface:**

- `WORD shel_wdef(BYTE far *pcmd, BYTE far *pdir);`

**Entered with:**

- `function=127;`
  - `#int_in=0;`
  - `#addr_in=2;`
  - `addr_in[0]` = address of command name;
  - `addr_in[1]` = address of directory.
- 

## Function 130 - Calculate intermediate rectangle

**C Interface:**

- `WORD xgrf_stepcalc(WORD orgw, WORD orgh, WORD xc, WORD yc, WORD w, WORD h, WORD *pcx, WORD *pcy, WORD *pcnt, WORD *pxstep, WORD *pystep)`

**Entered with:**

- `function=130;`
- `#int_in=6;`
- `#addr_in=0;`
- `int_in[0]` = initial width of rectangle
- `int_in[1]` = initial height of rectangle
- `int_in[2-5]` = target size / location of rectangle

**Returns:**

- `int_out[1-2]` = coordinates of intermediate box
  - `int_out[3]` = no. of steps to draw between initial size & final size
  - `int_out[4]` = X step  $((\text{final X} - \text{initial X}) / \text{count})$
  - `int_out[5]` = Y step.
- 

## Function 131 - Animate transition from one rectangle to another

**C Interface:**

- `WORD xgrf_2box(WORD xc, WORD yc, WORD w, WORD h, WORD corners, WORD cnt, WORD xstep, WORD ystep, WORD doubled);`

**Entered with:**



- *function*=131;
- *#int\_in*=9;
- *#addr\_in*=0;
- *int\_in*[0] = No. of steps to draw;
- *int\_in*[1] = X step;
- *int\_in*[2] = Y step;
- *int\_in*[3] = "doubled" flag - if set, box changes size, otherwise position;;
- *int\_in*[4] = "corners" flag - if set, will draw the corners of the box correctly (slower);
- *int\_in*[5-8] = start rectangle coordinates.

This function and the previous one are used internally in the implementation of [graf\\_growbox\(\)](#) and [graf\\_shrinkbox\(\)](#).

---

## Function 132 - Set a colour category

### C Interface:

- WORD *xgrf\_colour*( WORD set, WORD fg, WORD bg, WORD style, WORD index);
- WORD *xgrf\_color*( WORD set, WORD fg, WORD bg, WORD style, WORD index);

### Entered with:

- *function*=132;
- *#int\_in*=5;
- *#addr\_in*=0;
- *int\_in*[0] = Colour set, 0-15;
- *int\_in*[1] = Foreground colour;
- *int\_in*[2] = Background colour;
- *int\_in*[3] = [Fill style](#);
- *int\_in*[4] = [Fill index](#).



This function is only present in ViewMAX/2 and later. To check for this feature in FreeGEM, use [appl\\_init\(\)](#) and check that [xbuf.arch](#) is nonzero. Unfortunately there is no similar test for ViewMAX/2.

The colour categories used by ViewMAX are:

```
#define CC_NAME          8          /* Titlebar */
#define CC_SLIDER       9          /* Scrollbar background */
#define CC_DESKTOP     10         /* Desktop */
#define CC_BUTTON      11         /* Button and other 3D objects */
#define CC_INFO        12         /* Information bar (below titlebar) */
#define CC_ALERT       13         /* Alert box (ignored) */
#define CC_SLCTDNAME   14         /* Inactive titlebar */
```



Later FreeGEM builds also allow:

```
#define CC_3DSHADOW    16         /* Foreground is 3D light colour */
                                /* Background is 3D dark colour */
#define CC_RADIO       17         /* Foreground and background of the */
                                /* radio button "dot" */
#define CC_CHECK       18         /* Colour of the checkbox tick */
```

To check for this feature, use [appl\\_init\(\)](#) and check that bit 3 of [xbuf.abilities](#) is set.

---

## Function 133 - Set desktop image

### C Interface:

- WORD xgrf\_dtimage([MFDB](#) far \*mfdb);

### Entered with:

- *function*=133;
- *#int\_in*=0;
- *#addr\_in*=1;
- *addr\_in*[0] = Address of MFDB of desktop image; 0 for none.



This call is only present in ViewMAX/3.

The "r1" member of the MFDB holds 1 to draw the image centred, 2 to draw it tiled.

---

## Function 1010 - Get property (setting)

### C Interface:

- WORD prop\_get(BYTE FAR \*program, BYTE FAR \*section, BYTE FAR \*buf, WORD buflen, WORD options);

### Entered with:

- *function*=1010;
- *#int\_in*=2;
- *#addr\_in*=3;
- *addr\_in*[0] = address of program name, eg "PTK.DEMO"
- *addr\_in*[1] = address of section name, eg "Pen.colour"
- *addr\_in*[2] = address of results buffer
- *int\_in*[0] = options
- *int\_in*[1] = size of results buffer (including terminating 0).

### Returns:

- *int\_out*[0] = 0 if successful, 1 if property not found, -1 if file error, -2 if out of memory.
- buffer filled if return value was 0.

"Options" should be 0 if the value is per user, 1 if it is global. Currently has no effect, because GEM does not support user profiles. It is recommended that global settings should be avoided.



This call is a compile-time option in recent FreeGEM versions. To check for this feature, use [appl\\_init\(\)](#) and check that bit 1 of [xbuf.abilities](#) is set.

---

## Function 1011 - Put property (setting)

### C Interface:

- WORD prop\_put(BYTE FAR \*program, BYTE FAR \*section, BYTE FAR \*buf, WORD

options);

**Entered with:**

- *function*=1011;
- *#int\_in*=1;
- *#addr\_in*=3;
- *addr\_in*[0] = address of program name, eg "PTK.DEMO"
- *addr\_in*[1] = address of section name, eg "Pen.colour"
- *addr\_in*[2] = address of results buffer
- *int\_in*[0] = options

**Returns:**

- *int\_out*[0] = 0 if successful, -1 if file error, -2 if out of memory.
- buffer filled if return value was 0.

[prop\\_get\(\)](#) will remove any leading spaces from returned values; so if you are writing a string that may start with spaces, you should guard it with quotation marks when calling [prop\\_put\(\)](#) and remove the quotation marks after the [prop\\_get\(\)](#). Carriage returns and linefeeds must not be written by [prop\\_put\(\)](#).

"Options" should be 0 if the value is per user, 1 if it is global. Currently has no effect, because GEM does not support user profiles. It is recommended that global settings should be avoided.



This call is a compile-time option in recent FreeGEM versions. To check for this feature, use [appl\\_init\(\)](#) and check that bit 1 of [xbuf.abilities](#) is set.

---

## Function 1012 - Delete property (setting)

**C Interface:**

- WORD [prop\\_del](#)(BYTE FAR \*program, BYTE FAR \*section, WORD options);

**Entered with:**

- *function*=1012;
- *#int\_in*=1;
- *#addr\_in*=2;
- *addr\_in*[0] = address of program name, eg "PTK.DEMO"
- *addr\_in*[1] = address of section name, eg "Pen.colour"
- *int\_in*[0] = options

**Returns:**

- *int\_out*[0] = 0 if successful, 1 if property not found, -1 if file error, -2 if out of memory.
- buffer filled if return value was 0.

"Options" should be 0 if the value is per user, 1 if it is global. Currently has no effect, because GEM does not support user profiles. It is recommended that global settings should be avoided.



This call is a compile-time option in recent FreeGEM versions. To check for this feature, use [appl\\_init\(\)](#) and check that bit 1 of [xbuf.abilities](#) is set.

## Function 1020 - Get extended AES information

### C Interface:

- WORD xapp\_getinfo(WORD section, WORD \*v1, WORD \*v2, WORD \*v3, WORD \*v4);
- #define appl\_getinfo xapp\_getinfo

### Entered with:

- *function*=1020;
- #*int\_in*=1;
- #*int\_out*=5;
- *int\_in*[0] = section to check

### Returns:

- *int\_out*[0] = 1 if successful, 0 if section number is not supported
- *int\_out*[1-4] = values for this section

This call is equivalent to the Atari call, `appl_getinfo()`. Most of its sections only have meanings on the Atari.

The values returned for each section are:

0

Information about the normal AES font:

1. font height
2. font id
3. font type (0=system, 1=font scale mechanism (FSM) )

1

information about the little AES font:

1. font height
2. font id
3. font type (0=system, 1=FSM )

2

Colours:

1. VDI device number
2. number of supported OBJECT colors.
3. Atari color icons are supported (1) or not (0)
4. Atari enhanced RSCs (>64kB) are possible (1) or not (0)

3

Language:

1. 0=English, 1=German, 2=French, 3=reserved, 4=Spanish, 5=Italian, 6=Swedish

4

general information 1:

1. preemptive multitasking (1) or not (0)
2. `appl_find()` converts MiNT/AES-IDs (1) or not (0)
3. `appl_search()` available (1) or not (0)
4. `rsrc_rcfix()` available (1) or not (0)

5

general information 2:

1. `objc_xfind()` available (1) or not (0)
2. reserved, always 0
3. `menu_click()` available (1) or not (0)
4. `shel_r/wdef()` available (1) or not (0)

6

general information 3:

1. appl\_read(-1,...) available (1) or not (0)
2. shel\_get(addr,-1) available (1) or not (0)
3. menu\_bar(tree,-1) available (1) or not (0)
4. menu\_bar(tree,100) available (1) or not (0)

7

general (MagiC) information:

- Bit-0 = wdlg\_xx() functions available (1)
- Bit-1 = lbox\_xx() functions available (1)
- Bit-2 = fnts\_xx() functions available (1)
- Bit-3 = fslx\_xx() functions available (1)
- Bit-4 = pdlg\_xx() functions available (1)

8

Mouse:

1. graf\_mouse(258-260, addr) available (1) or not (0)
2. The AES remember the mouse form of each application (1) or not (0)

9

Menus:

1. MultiTOS compatible submenus available (1) or not (0)
2. MultiTOS compatible popups available (1) or not (0)
3. MultiTOS compatible scrollmenus available (1) or not (0)
4. enhanced MN\_SELECTED message available (1) or not (0)

10

shel\_write():

1. available modes:
  - Bit 0..7: highest possible value for doex & 0x00ff
  - Bit 8..15: Bits of doex & 0xff00, that are treated like MultiTOS does.
  - 1: shel\_write(0,...) undoes previous shel\_write calls (-> the desktop will be started after the actual program)
  - 0: launching programm (MultiTOS like)
  - 1: shel\_write(1,...) starts programs after the actual one
  - 0: starts Programs immediately (MultiTOS like)
2. ARGV via iscr supported (1) or not (0)

11

Windows:

1. bits that are set represent supported functions:
  - Bit 0: WF\_TOP returns the second uppermost window
  - 1: wind\_get (WF\_NEWDESK)
  - 2: wind\_g/set (WF\_COLOR)
  - 3: wind\_g/set (WF\_DCOLOR)
  - 4: wind\_get (WF\_OWNER)
  - 5: wind\_g/set (WF\_BEVENT)
  - 6: WF\_BOTTOM
  - 7: WF\_ICONIFY
  - 8: WF\_UNICONIFY
  - 9..15: reserved, always 0
2. reserved, 0
3. available window buttons:
  - Bit 0: Iconifier
  - 1: Backdrop-Button (MagiC)
  - 2: Shift-Click for Backdrop

- 3: "Hot" Closebox (GEM/3 and MagiC)
  - 4..15: reserved, 0
4. wind\_update(256..257) 'check and set' available (1) or not (0)

12

#### Messages

1. Bits which are set represent supported messages:

- 0: WM\_NEWTOP
- 1: WM\_UNTOPPED
- 2: WM\_ONTOP
- 3: AP\_TERM
- 4: MultiTOS like resolution change
- 5: CH\_EXIT
- 6: WM\_BOTTOM
- 7: WM\_ICONIFY
- 8: WM\_UNICONIFY
- 9: WM\_ALLICONIFY

2. reserved, 0

3. WM\_ICONIFY delivers coordinates (1) or not (0)

13

#### OBJECTs

1. Atari-style 3D objects via ob\_flags available (1) or not (0)

2. objc\_sysvar() available (1) or not (0)

3. Speedo- and GDOS-Fonts allowed in the TEDINFO structure (1) or not (0)

- Bit 0: G\_SWBUTTON available
- 1: G\_POPUP available
- 2: WHITEBAK is used for underlining (MagiC like)
- 3: G\_SHORTCUT available

14

Formulars ( MagiC form\_xdo() and form\_xdial() )

1. MagiC like Flydials supported (1) or not (0)

2. MagiC like key tables supported (1) or not (0)

3. last cursor position will be returned (1) or not (0)

4. reserved, 0



This call is a compile-time option in recent FreeGEM versions. To check for this feature, use [appl\\_init\(\)](#) and check that bit 0 of [xbuf.abilities](#) is set.

This function corresponds to the Atari function number 130, [appl\\_getinfo\(\)](#).

## Function 1030 - Get shell name

### C Interface:

- WORD xshl\_getshell(BYTE far \*shell);

### Entered with:

- *function*=1030;
- *#addr\_in*=1;
- *#int\_out*=1;
- *addr\_in*[0] = buffer to receive shell filename

### Returns:

- `int_out[0]` = 0 if standard shell (DESKTOP.APP) in use; 1 if it was overridden by use of GEM.CFG or [xshl\\_setshell\(\)](#).
- Buffer contains shell filename (no path).



This call is a compile-time option in recent FreeGEM versions. To check for this feature, use [appl\\_init\(\)](#) and check that bit 4 of [xbuf.abilities](#) is set.

## Function 1031 - Set shell name

### C Interface:

- WORD `xshl_setshell(BYTE far *shell);`

### Entered with:

- *function*=1031;
- *#addr\_in*=1;
- *#int\_out*=1;
- `addr_in[0]` = name of shell (no path; it must always be in \GEMAPPS\GEMSYS).

### Returns:

- `int_out[0]` = 0 if name was too long; 1 if successful.



This call is a compile-time option in recent FreeGEM versions. To check for this feature, use [appl\\_init\(\)](#) and check that bit 4 of [xbuf.abilities](#) is set.

This text was originally created by John Elliott, and was located on his website [www.seasip.info](http://www.seasip.info).  
This version of the document was packaged by Shane M. Coughlan for the OpenGEM SDK.