Copyright and publication information

This manual reflects version 3.2 of MAUI.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.
Table of Contents

Chapter 1: Overview of MAUI

12 Animation API
12 Dependencies
12 Introduction
12 Sprite
12 Animation object
13 Animation Group
13 Function Reference
13 Initialization and Termination
13 Sprites
13 The Animation Group
14 The Animation Object
14 Data Type Reference
14 Enumerated Types
14 Data Types
14 Data Structures
15 Bit-BLT API
15 Dependencies
15 Introduction
15 Features
16 What This API Does Not Do
16 Bit-BLT Context
17 Function Reference
17 Initialization and Termination
17 The Bit-BLT Context
18 Block Transfer Operations
18 Data Reference
18 Enumerated Types
4 MAUI Programming Reference

18 Data Types
18 Data Structures
19 CDB API
19 Dependencies
19 Introduction
19 Example CDB
20 Device Description Record (DDR)
20 Device Type
21 Device Name
21 Device Parameter
22 CDB Modules
22 Function Reference
22 Initialization and Termination
22 CDB Functions
23 Data Reference
23 Defined Constants
24 Drawing API
24 Dependencies
24 Introduction
24 Function Reference
24 Initialization and Termination
25 The Drawing Context
25 Drawing Operations
26 Data Reference
26 Enumerated Types
26 Data Types
26 Data Structures
27 Graphics Device API
27 Dependencies
27 Introduction
27 Drawmaps
27 Viewports
28 Display and Drawmap Coordinate Systems
28 Function Reference
39  Dependencies
39  Introduction
40  Message Types
41  Processing Messages
41  Function Reference
41  Initialization and Termination
41  The Message Mailbox
41  Messages
42  Data Reference
42  Defined Constants
42  Enumerated Types
43  Data Types
43  Data Structures
44  Shaded Memory API
44  Dependencies
44  Introduction
45  Using Normal Shades
45  Using Pseudo Shades
47  Function Reference
47  Initialization and Termination
48  Creating and Destroying Shades
48  Allocating and De-allocating Memory Segments
48  Status and Debugging Functions
49  Data Reference
49  Defined Constants
49  Enumerated Types
49  Data Structures
50  Text API
50  Dependencies
50  Introduction
51  Character Attributes
52  Function Reference
52  Initialization and Termination
52  Fonts
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Drawing Functions</td>
<td>175</td>
</tr>
<tr>
<td>6</td>
<td>Graphics Functions</td>
<td>263</td>
</tr>
<tr>
<td>7</td>
<td>Input Functions</td>
<td>363</td>
</tr>
<tr>
<td>8</td>
<td>MAUI System Functions</td>
<td>399</td>
</tr>
<tr>
<td>9</td>
<td>Shaded Memory Functions</td>
<td>407</td>
</tr>
<tr>
<td>10</td>
<td>Messaging Functions</td>
<td>445</td>
</tr>
<tr>
<td>11</td>
<td>Text Functions</td>
<td>499</td>
</tr>
<tr>
<td>12</td>
<td>Windowing Functions</td>
<td>543</td>
</tr>
<tr>
<td>13</td>
<td>Animation Data Types</td>
<td>661</td>
</tr>
<tr>
<td>14</td>
<td>Bit-BLT Data Types</td>
<td>673</td>
</tr>
<tr>
<td>15</td>
<td>CDB Data Types</td>
<td>685</td>
</tr>
<tr>
<td>16</td>
<td>Drawing Data Types</td>
<td>727</td>
</tr>
<tr>
<td>17</td>
<td>Graphics Data Types</td>
<td>733</td>
</tr>
</tbody>
</table>
This chapter provides an overview of the MAUI (Multimedia Application User Interface) APIs and the functions and data types of each API.
Animation API

Dependencies

- Shaded Memory API
- Graphics Device API
- Bit-BLT API

Introduction

The animation API provides a set of functions for creating and manipulating sprites. Objects are created from sprites and are made visible in the output viewport using the API. Support for a background is provided so that animation may be performed on hardware that has only one graphics plane.

Sprite

A sprite is a multi-frame image used as the source for objects to be animated. See ANM_SPRITE for more details about this data structure.

Animation object

An animation object identified by ANM_OBJECT_ID is used to manage the animation of a sprite. Objects are drawn to the destination drawmap using Bit-BLT operations.
Animation Group

An animation group by `ANM_GROUP_ID` is a collection of animation objects. Animation objects are grouped together to make processing them more efficient. Objects are processed and drawn as a group. Usually, all objects are placed in the same group. The primary benefit of a group is that it creates a front-to-back order for your objects so that they appear correctly when they overlap.

Function Reference

Initialization and Termination

- `anm_init()` Initialize the Animation API
- `anm_term()` Terminate the Animation API
- `anm_set_error_action()` Set Action to Take in Error Handler

Sprites

- `anm_create_sprite()` Create a Sprite
- `anm_destroy_sprite()` Destroy a Sprite

The Animation Group

- `anm_create_group()` Create an Animation Group
- `anm_draw_group()` Draw Objects in a Group
- `anm_get_group()` Get Animation Group Parameters
- `anm_set_group_dst()` Set Destination Drawmap
- `anm_set_group_bkg()` Set Group Background
- `anm_destroy_group()` Destroy an Animation Group
- `anm_process_group()` Process Objects in a Group
The Animation Object

- `anm_create_object()`: Create an Animation Object
- `anm_destroy_object()`: Destroy an Animation Object
- `anm_get_object()`: Get Animation Object Parameters
- `anm_restack_object()`: Restack an Animation Object
- `anm_set_object_pos()`: Set Position for an Object
- `anm_set_object_state()`: Set State for an Object
- `anm_set_object_sprite()`: Set Sprite for an Object
- `anm_set_object_frame()`: Set Frame for an Object
- `anm_set_object_bhv()`: Set Behavior for an Object
- `anm_set_object_meth()`: Set Drawing Method for an Object

Data Type Reference

Enumerated Types

- `ANM_OBJECT_PLACEMENT`: Animation Object Placement
- `ANM_METHOD`: Drawing Method for an Object

Data Types

- `ANM_GROUP_ID`: Animation Group ID
- `ANM_OBJECT_ID`: Animation Object ID

Data Structures

- `ANM_SPRITE`: Sprite Structure
- `ANM_FRAME`: Sprite Frame Structure
- `ANM_OBJECT_PARAMS`: Animation Object Placement
- `ANM_GROUP_PARAMS`: Animation Group Parameters
Overview of MAUI

Bit-BLT API

Dependencies

- Shaded Memory API
- Graphics Device API
- Graphics Driver Interface

Introduction

All pixel manipulation in MAUI eventually filters down to the operations supported by this API. For this reason, functions in this API must be fast. In fact, this API provides very fast functions for doing everything from drawing individual pixels to copying blocks from one pixel depth to another (on-the-fly pixel expansion).

If a hardware Bit-BLT engine is available, the graphics driver should support a set of fast entry points so that this API can take advantage of it. This happens without any special efforts by the application.

Features

- This API supports functions for drawing blocks of pixels in a drawmap. Optimized functions are provided for dealing with block sizes with a width and/or a height of 1 pixel. These functions may be used as the foundation for higher shapes such as circles and polygons.

- You may use functions in this API to copy blocks of pixels between drawmaps. These functions may also be used to support higher level functions for text, sprites, windows, or image based widgets (for example push-buttons or menus).
• A special type of copy function supported by this API is named expand. Expand functions may be used to copy a block of pixels from a source drawmap that has a pixel depth smaller than the destination drawmap.

• If a hardware Bit-BLT engine is present, this API uses it (through the graphics driver) to greatly increase the execution speed of these functions. Since the higher level drawing functions are based on the functions in this API, their execution speed is also greatly enhanced.

• This API supports coding methods defined by the Graphics Device API. You may draw to a drawmap of any coding method. There are some limitations on the coding method of the source data when using expand functions.

What This API Does Not Do
Since this API is the foundation upon which all drawing is done, speed takes precedence over some conveniences. This API does not include:

• Support for clipping. In almost all cases it is more efficient to do clipping in the functions that call the Bit-BLT operations. Therefore, it is the responsibility of the caller to perform any necessary clipping. This includes clipping to the boundary of the drawmap.

• Rigorous error checking. Most API functions validate the parameters passed to them. This API does so on several functions. However, when it comes to the high-speed Bit-BLT operations, parameter checking is relaxed in favor of speed.

Bit-BLT Context
All the functions within this API that copy or draw pixels use a special type of object called a Bit-BLT Context. Bit-BLT parameters, such as the source drawmap, destination drawmap, and pixel value are stored in this object. The context object is also used to store context information between calls such as `blt_copy_block()` and `blt_copy_next_block()`.
Overview of MAUI

Function Reference

This section gives a detailed reference for each of the functions in this API. These functions are the complete and only interface to this API.

Initialization and Termination

blt_init() Initialize the Bit-BLT API
blt_term() Terminate the Bit-BLT API
blt_set_error_action() Set Action to Take in Error Handler

The Bit-BLT Context

blt_create_context() Create a Bit-BLT Context Object
blt_destroy_context() Destroy a Bit-BLT Context Object
blt_get_context() Get Bit-BLT Context Parameters
blt_set_context_cpymix() Set Mixing Mode for Copying
blt_set_context_expmix() Set Mixing Mode for Expanding
blt_set_context_drwmix() Set Mixing Mode for Drawing
blt_set_context_pix() Set Pixel Value for Drawing
blt_set_context_src() Set Source Drawmap
blt_set_context_exptbl() Set Pixel Expansion Table
blt_set_context_trans() Set Transparent Pixel Value
blt_set_context_mask() Set Mask Drawmap
blt_set_context_ofs() Set Offset Pixel Value
blt_set_context_dst() Set Destination Drawmap
Overview of MAUI

Block Transfer Operations

- `blt_copy_block()`  
  Copy Rectangular Block of Pixels
- `blt_copy_oblock()`  
  Copy Overlapping Blocks of Pixels
- `blt_copy_next_block()`  
  Copy Next Rectangular Block of Pixels
- `blt_expd_block()`  
  Expand a Block of Pixels
- `blt_expd_next_block()`  
  Expand the Next Block of Pixels
- `blt_draw_block()`  
  Draw Block of Pixels
- `blt_draw_hline()`  
  Draw Horizontal Line of Pixels
- `blt_draw_vline()`  
  Draw Vertical Line of Pixels
- `blt_draw_pixel()`  
  Draw a Single Pixel
- `blt_get_pixel()`  
  Get Pixel

Data Reference

Enumerated Types

- `BLT_MIX`  
  Mixing Mode

Data Types

- `BLT_CONTEXT_ID`  
  Bit-BLT Context ID

Data Structures

- `BLT_CONTEXT_PARAMS`  
  Bit-BLT Context Parameters
Overview of MAUI

CDB API

Dependencies

- Windowing API

Introduction

This API supports functions for reading information from the Configuration Description Block (CDB). The CDB allows applications to determine the configuration of the system on which they are running on.

The CDB is a textual description of the system. It contains a series of entries describing the capabilities and characteristics of each device. Each device in the system is represented in the CDB by a device description record (DDR).

Example CDB

The following is an example of a CDB:

```
0:sys:CP="68340":OS="OS9":RV="3.0":DV="2.0":SR#2048,128:GR#512,128:GR#512,129:
3:/gfx:AI="MAUI":
4:/nvr:
5:/rem/genrem.mpm:
9:/pipe:
20:/term:
20:/t1:
90:/mv:
91:/ma:
113:/sp0/lapb/x25:
114:/rt0:HD:
```
Device Description Record (DDR)

Each DDR corresponds to one device and has three parts.

Table 1-1  DDR Parts

<table>
<thead>
<tr>
<th>Parts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Type</td>
<td>An Unsigned base ten integer representing the general class of functional devices to which this device belongs (for example, a WAN, graphic overlay, printer, magnetic disk, pointing device, or keyboard).</td>
</tr>
<tr>
<td>Device Name</td>
<td>The name used to access the device.</td>
</tr>
<tr>
<td>Device Parameters</td>
<td>Identifies the device’s functional performance. The contents are specific to a particular device type.</td>
</tr>
</tbody>
</table>

Each DDR comprises bytes with values in the range 0x20 to 0x7E (International Standards Organization (ISO) 8859-1 character set). A DDR is terminated by a carriage return (0x0D) character.

Device Type

More than one device of the same type can be present within the CDB, but each device must be identified with a unique device name. See *MAUI Porting Guide* for a list of valid device types.
Device Name

The device name is a string of characters terminated by a colon (:). It must be less than CDB_MAX_DNAME (80) characters long.

For most devices, the operating system uses the device name to access the relevant device. This is the actual name of the device descriptor and begins with the forward slash (/) character. Therefore, an application can use the device type to determine the name of a device. For example, /gfx below identifies the graphics device:

3:/gfx:PL#1:

For devices not controlled by the operating system, the name only identifies the DDR to the application.

Device Parameter

The device parameter contains an open-ended list of parameters to identify and/or set the functional behavior of the device identified by its device type. These parameters are interpreted by an application or driver and are unique to each device. The format of the device parameter ensures that you can use a multitude of parameters and does not restrict future enhancements.

Each entry in the device parameter has one of the following formats:

- Boolean - A two-character parameter identifier used as a boolean flag to indicate one of two particular capability options. For example, in the CDB_TYPE_SYSTEM entry, the “LE” flag indicates that the target processor uses a least significant byte first ordering scheme (little-endian). If the “LE” flag is not present, the target processor uses a most significant byte first ordering scheme (big-endian).

- Numeric - A two-character parameter identifier followed by the pound sign character and a numeric parameter value. Optionally, this is followed by a comma character and another numeric parameter value. The value comprises a variable length string of characters in the range 0x30 through 0x39. For example, for a graphic overlay processor (CDB_TYPE_GRAPHIC) capable of displaying up to eight image planes, the relevant parameter is “PL#8”.

Overview of MAUI

- String - A two-character parameter identifier followed by the equal sign character and an alphanumeric string in double quotes. For example, an ‘OS="OS9” in the system (CDB_TYPE_SYSTEM) DDR indicates that the operating system is OS-9.

Each parameter entry is terminated by a colon character and must be less than CDB_MAX_PARAM (128) characters.

The parameter identifier for all three formats must comprise two upper case alphabetic characters (0x41 through 0x5A).

CDB Modules

The DDR entries are grouped into one or more CDB modules. Each module must have a M$TYPE of 5 and M$LANG of 1. The name of the module is not relevant, but it is recommended that it start with “cdb”.

The final DDR entry in a CDB module must be followed by a NULL character. This allows the contents to be treated as a single string.

Function Reference

Initialization and Termination

cdb_init() Initialize the CDB API
cdb_term() Terminates the CDB API
cdb_set_error_action() Set Action to Take in Error Handler

CDB Functions

cdb_get_ddr() Get Device Description By Type and Number
cdb_get_copy() Get Copy of the CDB
cdb_get_size() Get Size of the CDB
cdb_get_ncopy() Get an N Byte Copy of the CDB
Data Reference

Defined Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDB_MAX_DNAME</td>
<td>Maximum Length of a Device Name</td>
</tr>
<tr>
<td>CDB_MAX_PARAM</td>
<td>Maximum Length of Parameter String</td>
</tr>
<tr>
<td>CDB_TYPE</td>
<td>Device Type Names</td>
</tr>
</tbody>
</table>
Overview of MAUI

Drawing API

Dependencies

- Shaded Memory API
- Bit-BLT API

Introduction

This API provides a suite of functions for drawing geometric shapes to a drawmap. The shapes that can be drawn are point, line, polyline, polygon, rectangle, and circle.

Several attributes control drawing. These attributes are stored in a drawing context object. Attributes include fill mode, line style, dash pattern, and dash magnification.

Function Reference

Initialization and Termination

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drw_init()</td>
<td>Initialize the Drawing API</td>
</tr>
<tr>
<td>drw_term()</td>
<td>Terminate the Drawing API</td>
</tr>
<tr>
<td>drw_set_error_action()</td>
<td>Set Action to Take in Error Handler</td>
</tr>
</tbody>
</table>
The Drawing Context

- `drw_create_context()`: Create a Drawing Context
- `drw_destroy_context()`: Destroy a Drawing Context
- `drw_get_context()`: Get Drawing Context Parameters
- `drw_set_context_fm()`: Set Fill Mode
- `drw_set_context_ls()`: Set Line Style
- `drw_set_context_dash()`: Set Dash Pattern
- `drw_set_context_dst()`: Set Destination Drawmap
- `drw_set_context_expmix()`: Set Mixing Mode for Expanding
- `drw_set_context_mix()`: Set Mixing Mode for Drawing
- `drw_set_context_pix()`: Set Pixel Value for Drawing
- `drw_set_context_src()`: Set Source Drawmap
- `drw_set_context_exptbl()`: Set Pixel Expansion Table
- `drw_set_context_trans()`: Set Transparent Pixel Value
- `drw_set_context_mask()`: Set Mask Drawmap
- `drw_set_context_ofs()`: Set Offset Pixel Value
- `drw_set_context_dst()`: Set Destination Drawmap
- `drw_set_context_draw()`: Set Drawing Area
- `drw_set_context_origin()`: Set Drawing Origin
- `drw_set_context_clip()`: Set Clipping Area

Drawing Operations

- `drw_point()`: Draw a Point
- `drw_line()`: Draw a Line
- `drw_rectangle()`: Draw a Rectangle
- `drw_polyline()`: Draw a Polyline
- `drw_polygon()`: Draw a Polygon
Overview of MAUI

1. **drw_circle()**
   - Draw a Circle

2. **drw_arc()**
   - Draw a Circular Arc

3. **drw_earc()**
   - Draw an Elliptical Arc

4. **drw_ellipse()**
   - Draw an Ellipse

5. **drw_oval()**
   - Draw an Oval

6. **drw_oval_arc()**
   - Draw an Oval Arc

**Copy Operations**

7. **drw_copy_block()**
   - Copy a Block of Pixels

8. **drw_copy_oblock()**
   - Copy Overlapping Blocks of Pixels

9. **drw_expd_block()**
   - Expand a Block of Pixels

**Data Reference**

**Enumerated Types**

- **DRW_FM**
  - Fill Mode

- **DRW_LS**
  - Line Style

**Data Types**

- **DRW_CONTEXT_ID**
  - Drawing Context ID

**Data Structures**

- **DRW_CONTEXT_PARAMS**
  - Drawing Context Parameters
Overview of MAUI

Graphics Device API

Dependencies

- Shaded Memory API
- Graphics Driver API

Introduction

The graphics device API is responsible for insulating applications from differences in graphics hardware. The functionality in this API is based on the concepts of drawmaps and viewports.

Drawmaps

A drawmap is an object that defines a rectangular area of graphics memory. This object may be created by calling `gfx_create_dmap()` or created directly by the application (e.g. using initialized data). The structure name for a drawmap is `GFX_DMAP` and is fully explained in the Graphics Device API data reference section.

Viewports

A viewport is an object that allows you to map drawmaps to the physical display. Theoretically, this mapping allows any rectangular area of a drawmap to be mapped to any position on the display. However, the hardware usually imposes limits on the complexity of the viewport stack. The hardware may also limit the types of drawmaps that you may make visible. These limits are usually related to the coding method and the shade of memory used for the pixel memory.
Display and Drawmap Coordinate Systems

This API makes use of two coordinate systems. The display coordinate system refers to coordinates within the display. The drawmap coordinate system refers to coordinates within an individual drawmap. The translation between display and drawmap coordinates may vary based on the hardware you are using.

All coordinate systems have their origin (0,0) at the top-left corner. The x and y coordinates grow as you move to the right and down, respectively. The unit of measurement is a pixel. All coordinates specified relative to the display origin are considered to be within the display coordinate system. All coordinates specified relative to the drawmap origin are considered to be within the drawmap coordinate system.

Function Reference

Initialization and Termination

gfx_init()  Initialize the Graphics Device API
gfx_term()  Terminate the Graphics Device API
gfx_set_error_action()  Set Action to Take in Error Handler

The Graphics Device

gfx_open_dev()  Open a Graphics Device
gfx_close_dev()  Close a Graphics Device
gfx_clone_dev()  Clone a Graphics Device
gfx_restack_dev()  Restack a Device
gfx_get_dev_attribute()  Get Graphics Device Attribute
gfx_set_dev_attribute()  Set Graphic Device Attribute
Overview of MAUI

gfx_get_dev_cap() Get Graphics Device Capabilities
gfx_get_dev_capex() Get Graphics Device Extended Capabilities
gfx_get_dev_status() Get Graphics Device Status
gfx_set_display_size() Set Display Size
gfx_set_display_vpmix() Set Viewport Mixing On/Off
gfx_set_display_extvid() Set External Video On/Off
gfx_set_display_bkcol() Set Backdrop Color
gfx_set_display_transcol() Set Transparent Color
gfx_set_decode_dst() Set Destination for Video Decoding
/Graphics Memory

gfx_alloc_mem() Allocate Graphics Memory
gfx_dealloc_mem() De-allocate Graphics Memory

Dealing with Drawmaps

gfx_create_dmap() Create a Drawmap Object
gfx_destroy_dmap() Destroy a Drawmap Object
gfx_set_dmap_size() Set Coding Method and Size of Drawmap
gfx_set_dmap_pixmem() Set Pixel Memory Pointer in Drawmap

Viewports

gfx_create_vport() Create a Viewport
gfx_clone_vport() Clone a Viewport
gfx_destroy_vport() Destroy a Viewport
Overview of MAUI

- `gfx_get_vport_status()`: Get Viewport Status
- `gfx_set_vport_position()`: Set the Position of a Viewport
- `gfx_set_vport_size()`: Set the Size of a Viewport
- `gfx_set_vport_state()`: Set the State of a Viewport
- `gfx_set_vport_intensity()`: Set Viewport Intensity
- `gfx_set_vport_dmap()`: Set Drawmap to Use In a Viewport
- `gfx_set_vport_dmpos()`: Set Drawmap Position In a Viewport
- `gfx_set_vport_colors()`: Set the Colors for a Viewport
- `gfx_restack_vport()`: Re-stack a Viewport

Graphics Cursor

- `gfx_create_cursor()`: Create a New Hardware Cursor
- `gfx_destroy_cursor()`: Destroy a Hardware Cursor
- `gfx_get_cursor_cap()`: Get Information About a Hardware Cursor
- `gfx_set_cursor()`: Select Hardware Cursor
- `gfx_set_cursor_pos()`: Set the Hardware Cursor Position

Miscellaneous

- `gfx_calc_pixmem_size()`: Calculate Size of Pixel Memory
- `gfx_find_vport()`: Find the Viewport at the Specified Position
- `gfx_cvt_dppos_dmpos()`: Convert Display to Drawmap Position
- `gfx_cvt_dmpos_dppos()`: Convert Drawmap to Display Position
- `gfx_sync_retrace()`: Synchronize with Vertical Retrace
Data Reference

Defined Constants

- GFX_LINE_PAD: Line Padding
- GFX_POS_MIN: Minimum Value For GFX_POS
- GFX_POS_MAX: Maximum Value For GFX_POS
- GFX_DIMEN_MIN: Minimum Value for GFX_DIMEN
- GFX_DIMEN_MAX: Maximum Value For GFX_DIMEN
- GFX_OFFSET_MIN: Minimum Value For GFX_OFFSET
- GFX_OFFSET_MAX: Maximum Value For GFX_OFFSET
- GFX_MAX_DEV_NAME: Maximum Length of Device Name

Enumerated Types

- GFX_ATTR_MODE: Graphics Device Attribute Mode
- GFX_ATTR_TYPE: Graphics Device Attribute Types
- GFX_COLOR_TYPE: Color Type
- GFX_INTL_MODE: Interlace Mode
- GFX_VPC: Viewport Complexity
- GFX_VPDMC: Viewport Drawmap Complexity
- GFX_VPORT_PLACEMENT: Viewport Placement

Data Types

- GFX_DEV_ID: Graphics Device ID
1 Overview of MAUI

GFX_CM
Pixel Coding Method

GFX_POS
Pixel Position

GFX_DIMEN
Dimension In Pixels

GFX_OFFSET
Offset in Pixels

GFX_ANGLE
Angle in 64ths of a Degree

GFX_PIXEL
Pixel Value

GFX_VPORT_ID
Viewport ID

GFX_RGB
RGB Color

GFX_YUV
YUV Color

GFX_YCBCR
YCbCr Color

GFX_A1_RGB
RGB Color with Alpha Flag

Data Structures

GFX_POINT
Position of a Point Given by X and Y

GFX_DELTA
Delta Between Two Positions

GFX_RECT
Rectangle Defined by X, Y, Width, and Height

GFX_COLOR
Color Value of Specified Type

GFX_COLOR_VALUE
Untyped Color Value

GFX_CURSOR_CAP
Hardware Cursor Capabilities

GFX_CURSOR_ID
Hardware Cursor ID

GFX_CURSOR_INFO
Describes a Hardware Cursor Format

GFX_CURSOR_SPEC
Hardware Cursor Graphic Information

GFX_PALETTE
Color Palette

GFX_DMAP
Drawmap

GFX_VPORT_STATUS
Viewport Status
Overview of MAUI

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFX_DEV_CAP</td>
<td>Graphics Device Capabilities</td>
</tr>
<tr>
<td>GFX_DEV_CAPEXTEN</td>
<td>Graphics Device Extended Capabilities</td>
</tr>
<tr>
<td>GFX_DEV_RES</td>
<td>Graphics Device Resolution</td>
</tr>
<tr>
<td>GFX_DEV_CM</td>
<td>Graphics Device Coding Methods</td>
</tr>
<tr>
<td>GFX_DEV_STATUS</td>
<td>Graphics Device Status</td>
</tr>
</tbody>
</table>
Input Device API

Dependencies

- Shaded Memory API
- Messaging API
- MAUI Input Process

Introduction

Input in MAUI is not limited to a keyboard and mouse. In fact, any type of input device, such as a remote control, joystick, or game controller, may be used. Protocol modules are used to convert the raw SCF input received from the device into standardized pointer and key symbol messages. These messages are sent to the mailbox registered with the device.

This API deals with two general devices, pointer and key symbol. Pointer devices return information about X,Y movement and button press/release events. Key symbol devices return information about keys that are typed.

Using this API, an application may reserve specific keys on each key symbol device. For example, a process may reserve the POWER key so that it alone controls what happens when the POWER key is pressed.

This API is not limited to a single input device. An application may open as many devices as are allowed by the operating system. Multiple applications may also share a single device.
Function reference

Initialization and Termination

- `inp_init()`: Initialize the Input Device API
- `inp_term()`: Terminate use of the Input Device API
- `inp_set_error_action()`: Set Action to Take in Error Handler

The Input Device

- `inp_open_dev()`: Open an Input Device
- `inp_close_dev()`: Close an Input Device
- `inp_restack_dev()`: Re-stack an Input Device
- `inp_get_dev_cap()`: Get Input Device Capabilities
- `inp_get_dev_status()`: Get Input Device Status
- `inp_check_keys()`: Check a Range of Key Symbols
- `inp_set_ptr_limit()`: Set Pointer Limit
- `inp_set_ptr_pos()`: Set Pointer Position
- `inp_set_msg_mask()`: Set Mask for Queuing Messages
- `inp_set_callback()`: Set Callback for Queuing Messages

Key Reservation and Simulation

- `inp_reserve_key()`: Reserve a Key
- `inp_release_key()`: Release a Key
- `inp_set_sim_meth()`: Set Simulation Method
Overview of MAUI

Data Reference

Defined Constants

- INP_MAX_DEV_NAME: Maximum Length of Device Name
- INP_MAX_BUTTONS: Maximum Buttons for Pointer Device
- INP_KEY_*: Key Symbol Names

Enumerated Types

- INP_DEV_PLACEMENT: Device Placement
- INP_SIM_METH: Simulation Method
- INP_PTR_SUBTYPE: Pointer Message Subtype
- INP_KEY_SUBTYPE: Key Symbol Message Subtype
- INP_KEYMOD: Key Modifiers
- INP_DEV_CLASS: Input Device Classification
- INP_KEYS: Key Symbol Groups

Data Types

- INP_DEV_ID: Input Device ID

Data Structures

- INP_DEV_CAP: Input Device Capabilities
- INP_DEV_STATUS: Device Status
- INP_MSG: Union of All Input Message Structures
Overview of MAUI

MAUI System API

Dependencies

- Shaded Memory API
- CDB API
- Graphics Device API
- Bit-BLT API
- Text API
- Drawing API
- Animation API
- Messaging API
- Input Device API
- Windowing API

Introduction

This section describes the MAUI System API. The functions and data types defined in this API are referenced by all other MAUI APIs (common) or they make references to all other MAUI APIs (global).
Overview of MAUI

Function Reference

Initialization and Termination

maui_init() Initialize the MAUI APIs
maui_term() Terminate the MAUI APIs
maui_set_error_action() Set Action to Take in Error Handler

Data Reference

Defined Constants

MAUI_COMPAT_LEVEL Compatibility Level
MSBFIRST Big-Endian
LSBFIRST Little-Endian

Enumerated Types

BOOLEAN Boolean Type
MAUI_ERR_LEVEL Error Level

Data Structures

MAUI_MSG Union of All MAUI Message Types
Overview of MAUI

Messaging API

Dependencies

- Shaded Memory API

Introduction

MAUI, like virtually all other popular graphical user interface (GUI) platforms available, is tailored for applications that run in an event-driven manner. The messaging API is the part of MAUI which allows applications to work in this way. This API supports both inter- and intraprocess communications via high speed message packets.

This API supports functions for sending and receiving messages. Messages are sent via named mailboxes, so applications may use this mechanism to perform interprocess and intraprocess communication. This API does not enforce a certain format for messages other than requiring a common area at the beginning. You must correctly use this common area if you are using this API. See MSG_COMMON for a definition of this area.
Message Types

The message type is a bit-field, which means that it is limited to 32 unique values. For this reason, usage must be registered. The following table defines the assignments for different ranges of message types.

Table 1-2  Range of Bits

<table>
<thead>
<tr>
<th>(inclusive)</th>
<th>Defined By</th>
<th>Used By</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 19</td>
<td>MAUI APIs</td>
<td>Applications</td>
</tr>
<tr>
<td>20 - 23</td>
<td>Reserved by Microware</td>
<td>Microware internal</td>
</tr>
<tr>
<td>24 - 31</td>
<td>Applications</td>
<td>Applications</td>
</tr>
</tbody>
</table>

The following table lists all the registered message types. These are defined in the header file maui_msg.h. A union of these message types is defined as MAUI_MSG in maui.h.

Table 1-3  Registered Message Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TYPE_NONE</td>
<td>0</td>
<td>No message</td>
</tr>
<tr>
<td>MSG_TYPE_PTR</td>
<td>1</td>
<td>Pointer message</td>
</tr>
<tr>
<td>MSG_TYPE_KEY</td>
<td>2</td>
<td>Key symbol message</td>
</tr>
<tr>
<td>MSG_TYPE_ANY</td>
<td>0xffffffff</td>
<td>Any message</td>
</tr>
</tbody>
</table>
Overview of MAUI

Processing Messages

Messages may be processed in two ways. The application may process them directly or through callback functions. If your application wishes to process messages directly, simply call \texttt{msg\_read()} to read each message from the queue and look at its message type.

Callbacks may simplify the process of handling messages. In this case, your application calls \texttt{msg\_read()} to read the message, then calls \texttt{msg\_dispatch()} to call the callback function for it.

Function Reference

Initialization and Termination

\texttt{msg\_init()}
Initialize the Messaging API

\texttt{msg\_term()}
Terminate use of the Messaging API

\texttt{msg\_set\_error\_action()}
Set Action to Take in Error Handler

The Message Mailbox

\texttt{msg\_create\_mbox()}
Create a Mailbox

\texttt{msg\_open\_mbox()}
Open a Mailbox

\texttt{msg\_close\_mbox()}
Close a Mailbox

\texttt{msg\_get\_mbox\_status()}
Get Mailbox Status

Messages

\texttt{msg\_send\_sig()}
Send Signal on Message Ready

\texttt{msg\_release\_sig()}
Release Signal on Message Ready

\texttt{msg\_set\_mask()}
Set Mask for Queuing Messages
1 Overview of MAUI

- msg.peek()  
  Peek at a Message in a Mailbox
- msg.peekn()  
  Peek at N Bytes of a Message in a Mailbox
- msg.read()  
  Read a Message from a Mailbox
- msg.readn()  
  Read N Bytes of a Message from a Mailbox
- msg.unread()  
  Unread a Message to a Mailbox
- msg.unreadn()  
  Unread N Bytes of a Message to a Mailbox
- msg.write()  
  Write a Message to a Mailbox
- msg.writen()  
  Write N Bytes of a Message to a Mailbox
- msg.dispatch()  
  Dispatch Message
- msg.flush()  
  Flush Messages
- msg.set_filter()  
  Set Filter for Searching a Mailbox

Data Reference

Defined Constants

- MSG_TYPE  
  Message Type
- MSG_KEY  
  Key Symbol Message
- MSG_MAX_MBOX_NAME  
  Maximum Length of Mailbox Name

Enumerated Types

- MSG_BLOCK_TYPE  
  I/O Blocking Type
- MSG_PLACEMENT  
  Message Placement
- MSG_PTR  
  Pointer Message
Overview of MAUI

Data Types

msg_mbox_id  Mailbox ID

Data Structures

msg_common  Message Header
msg_mbox_status  Mailbox Status
Overview of MAUI

Shaded Memory API

Dependencies

- None

Introduction

Efficient memory management is a requirement for any graphical environment such as MAUI. Graphical environments are dynamic when allocating and de-allocating memory segments.

For this reason, unless an application takes specific steps to avoid it, memory fragmentation may become a serious problem. This API gives applications and other APIs the facilities required to manage multiple pools of memory.

This API builds upon the concept of colored memory by adding the concept of shades. Since colored memory areas are defined by the system, applications generally do not have enough control over how memory is allocated. This library gives applications the ability to define their own shades.

The OS-9/OS-9000 operating system supports the concept of colored memory. This allows OEMs to partition areas of memory that have different characteristics. The shaded memory API takes this a step further by adding the concept of shades. Two types of shades are defined by this API: normal and pseudo.

Normal shades are used to control the allocation and de-allocation of memory from an area that is accessible to the CPU. In this case, the operating system is usually used to supply the memory for the shade.

Pseudo shades are used to control the allocation and de-allocation of memory from an area that is not accessible to the CPU. In this case, the operating system cannot be used to supply the memory for the shade. Instead, application (or API) supplied allocation and de-allocation functions must be provided.
The following diagram shows the relationship between colored memory and shaded memory.

**Figure 1-1  Relationship between colored memory and pseudo memory.**

In this diagram the operating system has defined one color of memory (color 0x01). The application is using this color to define shades 0x01, 0x02, and 0x03.

In this diagram the application must provide allocation and de-allocation functions to support the pseudo memory identified by colors 0x30 and 0x31. The application uses color 0x30 to define shades 0x11 and 0x12, and uses color 0x31 to define shades 0x21 and 0x22.

**Using Normal Shades**

Before allocating memory from a normal shade, you must initialize the shaded memory functions and define the shade. The following example creates and then destroys three shades numbered 1 through 3.
# Overview of MAUI

```c
#include <MAUI/maui_mem.h>

main()
{
  /* Initialize API and create shades */
  mem_init();
  mem_create_shade(1,MEM_SHADE_NORMAL,0,8192,1024,
  MEM_OV_SEPARATE, TRUE);
  mem_create_shade(2,MEM_SHADE_NORMAL,0x80,0,1,
  MEM_OV_SEPARATE,TRUE);
  mem_create_shade(3,MEM_SHADE_NORMAL,0x81,0,4096,
  MEM_OV_SEPARATE,TRUE);

  /* Call mem_malloc(), mem_calloc() or 
   mem_realloc() to allocate each segment. */

  /* Call mem_free() to free each segment */

  /* Destroy shades and terminate the API */
  mem_destroy_shade(1);
  mem_destroy_shade(2);
  mem_destroy_shade(3);
  mem_term();
}
```

This example shows some of the diversity in the ways an application may choose to manage memory. Shade 1 uses color 0, which allows the system to satisfy the request from any color. The initial size of 8K forces the initial block to be allocated immediately. This block is not returned to the system until the shade is destroyed. The grow size of 1K forces the shade to grow by blocks that are at least 1K in size. For example, if there is no free memory in this shade and you try to allocate 50 bytes, then a 1K block is allocated from the system and the first 50 bytes are used. The remaining 974 bytes are put in the free list and used to satisfy future requests.

Shade 2 forces all allocations to be satisfied from system memory with color 0x80. Since the initial size is 0, no memory is allocated until a request is made. Since the grow size is 1K, this shade always grows by the size of the allocations being made by the application.
Shade 3 has no initial size, but its grow size is 4K. This means that blocks allocated from the system for this shade must always be multiples of 4K. This shade also requires the memory to come from color 0x81.

**Using Pseudo Shades**

Pseudo shades differ from normal shades in that memory is not allocated from the system to satisfy requests from the application. Instead, allocation and de-allocation functions are provided by the application.

Applications should use `mem_create_shade()` to create a pseudo shade. Most functions in this API operate on both normal shades and pseudo shades. The only significant difference is that the memory being managed by a pseudo shade is not accessible and is therefore never written to. Applications must call `mem_set_alloc()` and `mem_set_dealloc()` to set the allocator and de-allocator.

Pseudo shades are most commonly required by graphic devices where the CPU cannot directly access the graphics memory. In this case, the application can set `gfx_alloc_mem()` with `mem_set_alloc()` and `gfx_dalloc_mem()` with `mem_set_dealloc()`.

**Function Reference**

**Initialization and Termination**

- `mem_init()` Initialize the Shaded Memory API
- `mem_term()` Terminate Shaded Memory API
- `mem_set_error_action()` Set Action to Take in Error Handler
Overview of MAUI

Creating and Destroying Shades

mem_create_shade()  Create a Shade
mem_set_alloc()      Set Allocator Function for a Shade
mem_set_alloc_bndry() Set Memory Allocation Boundary
mem_set_dealloc()    Set De-allocator Function for a Shade
mem_destroy_shade()  Destroy a Shade of Memory
mem_set_grow_method() Set Grow Method for a Shade

Allocating and De-allocating Memory Segments

mem_malloc()         Allocate Shaded Memory
mem_calloc()         Allocate and Clear Shaded Memory Segment
mem_realloc()        Reallocate Shaded Memory
mem_free()           Free a Segment from a Normal Shade
mem_sfree()          Free a Segment from the Specified Shade
mem_sfree_all()      Free All Segments from the Specified Shade

Status and Debugging Functions

mem_get_shade_status() Get Shade Status
mem_list_segments()   Print a Listing of Allocated Segments
mem_list_tables()     Print a Listing of Memory Tables
mem_list_overflows()  Print a Listing of Overflows
## Data Reference

### Defined Constants

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM_DEF_SHADE</td>
<td>Default Shade</td>
</tr>
<tr>
<td>MEM_MIN_ALLOC</td>
<td>Minimum Allocation Size</td>
</tr>
</tbody>
</table>

### Enumerated Types

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM_OVTYPE</td>
<td>Overhead Type</td>
</tr>
<tr>
<td>MEM_GROW</td>
<td>Grow Method</td>
</tr>
<tr>
<td>MEM_SHADE_TYPE</td>
<td>Shade Type</td>
</tr>
</tbody>
</table>

### Data Structures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM_SHADE_STATUS</td>
<td>Shade Status</td>
</tr>
</tbody>
</table>
Text API

Dependencies

- Shaded Memory API
- Bit-BLT API

Introduction

The text API provides a suite of functions for writing multi-byte and wide-character strings to any drawmap. These functions provide support for multiple fonts (mono and proportional spaced), and methods for controlling the padding between characters.

In addition, you may control the padding (additional pixels) between glyphs (characters). This control is provided at two levels. First you may specify the glyph padding with `txt_set_context_cpad()`. This padding is automatically added when any text drawing function is called. Second, you may use the `pad_array` parameter, which is present in each of the text drawing functions. This parameter specifies the individual padding between each character in the output string. The first value specifies the padding between the first and second character. The next value specifies the padding between the second and third, and so forth.
Character Attributes

Figure 1-2 shows the character attributes supported by the MAUI Text API.

Note
The text API draws from the character's baseline, not the standard 0,0 coordinates.

Figure 1-2  Character Attributes

- ascent: the number of pixels in the character cell above the baseline
- descent: the number of pixels in the character cell below the baseline
- baseline: an imaginary line between pixels that make up the ascent and the descent
- height: the sum of ascent pixels and descent pixels
Initialization and Termination

- `txt_init()`: Initialize the Text API
- `txt_term()`: Terminate the Text API
- `txt_set_error_action()`: Set Action to Take in Error Handler

Fonts

- `txt_create_font()`: Create a Font Object
- `txt_destroy_font()`: Destroy a Font Object

The Text Context

- `txt_create_context()`: Create a Text Context Object
- `txt_destroy_context()`: Destroy a Text Context Object
- `txt_get_context()`: Get Text Context Parameters
- `txt_set_context_font()`: Set Font to Use
- `txt_set_context_cpad()`: Set Character Padding
- `txt_set_context_mix()`: Set Mixing Mode
- `txt_set_context_exptbl()`: Set Pixel Expansion Table
- `txt_set_context_trans()`: Set Transparent Pixel Value
- `txt_set_context_ofs()`: Set Offset Pixel Value
- `txt_set_context_dst()`: Set Destination Drawmap
- `txt_set_context_draw()`: Set Drawing Area
- `txt_set_context_origin()`: Set Drawing Origin
- `txt_set_context_clip()`: Set Clipping Area
Overview of MAUI

Text Drawing Operations

- `txt_draw_mbs()`  
  Draw a Multi-Byte String
- `txt_draw_wcs()`  
  Draw a Wide Character String
- `txt_get_mbs_width()`  
  Get Width of a Multi-Byte String
- `txt_get_wcs_width()`  
  Get Width of a Wide Character String

Data Type Reference

Defined Constants

- `TXT_NOGLYPH`  
  Unused Glyph

Data Types

- `TXTCONTEXT_ID`  
  Text Context ID

Enumerated Types

- `TXT_FONTTYPE`  
  Font Type

Data Structures

- `TXT_FONT`  
  Font Structure
- `TXT_RANGTBL`  
  Glyph Range Table
- `TXTCONTEXT_PARAMS`  
  Text Context Parameters
Overview of MAUI

Windowing API

Dependencies

- Messaging API
- Bit-BLT API
- Graphics Device API
- Shaded Memory API
- MAUI Windowing Process

Introduction

This API supports functions for creating, manipulating, and destroying windows on a windowing device.

A window is a rectangular area on the windowing device that lets you view graphic and/or text output. Applications can display overlapping and nested windows on one or more windowing devices. Windows can be moved, resized, and re-stacked.

The MAUI Windowing API manages multiple windows in a windowing device. A windowing device may display graphics and/or text in overlapping or nested windows.

Function Reference

Initialization and Termination

win_init() Initialize the Windowing API
win_term() Terminate the Windowing API
win_set_error_action() Set Action to Take in Error Handler
The Windowing Device

* win_create_dev()  
Create a Windowing Device  

* win_destroy_dev()  
Destroy a Windowing Device  

* win_open_dev()  
Open a Windowing Device  

* win_close_dev()  
Close a Windowing Device  

* win_open_inpdev()  
Open an Input Device  

* win_close_inpdev()  
Close an Input Device  

* win_get_dev_status()  
Get Windowing Device Status  

* win_set_focus()  
Set Window that has Keyboard Focus  

Windows

* win_create_win()  
Create a Window  

* win_destroy_win()  
Destroy a Window  

* win_reparent_win()  
Re-parent a Window  

* win_restack_win()  
Restack a Window  

* win_get_win_status()  
Get Window Status  

* win_move_win()  
Move Window  

* win_resize_win()  
Resize Window  

* win_set_callback()  
Set Callback for Queuing Messages  

* win_set_cursor()  
Set Cursor for a Window  

* win_set_cursor_state()  
Set Cursor Visibility  

* win_set_msg_mask()  
Set Mask for Queuing Messages  

* win_set_state()  
Set Window State  

* win_set_cmap()  
Set Colormap for a Window
Overview of MAUI

**Graphics Cursor**

- `win_create_cursor()`: Create a Cursor
- `win_destroy_cursor()`: Destroy a Cursor

**Pen and Ink**

- `win_set_ink_method()`: Set Inking Method
- `win_set_ink_pix()`: Set Pixel Value for Ink
- `win_erase_ink()`: Erase Ink from a Window

**Drawing**

- `win_set_txt_context()`: Set Text Context
- `win_set_drw_context()`: Set Drawing Context
- `win_set_drw_area()`: Set Drawing Area
- `win_lock_region()`: Lock a Region of a Window
- `win_unlock_region()`: Unlock a Region of a Window

**Colormaps**

- `win_create_cmap()`: Create a Colormap
- `win_destroy_cmap()`: Destroy a Colormap
- `win_get_cells_params()`: Get Info on the Color Cells
- `win_get_cmap_free()`: Get Free Space in a Colormap
- `win_get_cmap_cells()`: Get Colors From a Colormap
- `win_get_cmap_params()`: Get Info on the Colormap
- `win_alloc_cmap_color()`: Allocate a Color
- `win_alloc_cmap_colors()`: Allocate Array of Colors
- `win_alloc_cmap_cell()`: Allocate a Single Private Cell
- `win_alloc_cmap_cells()`: Allocate Group of Color Cells
Overview of MAUI

win_free_cmap_cells() Free Range of Color Cells
win_set_cmap_cells() Set Colors in Group of Cells
win_set_color_match() Set Color Matching Method

Data Type Reference

Defined Constants
WIN_MAX_DEV_NAME Maximum Length of Device Name

Enumerated Types
WIN_CMATCH Color match method
WIN_INK_METHOD Window Inking Method
WIN_MSG_MASK Window Message Mask
WIN_MSG_TYPE Window Message Type
WIN_PLACEMENT Window Placement

Data Type
WIN_CALLBACK Callback Function
WIN_CMAP_ID Colormap ID
WIN_DEV_ID Windowing Device ID
WIN_ID Window ID
1 Overview of MAUI

Data Structures

WIN_CELL_PARAMS
WIN_CMAP_PARAMS
WIN_CURSOR
WIN_DEV_STATUS
WIN_MSG
WIN_STATUS

Application Messages

MSG_WIN_BORDER
MSG_WIN_BUTTON
MSG_WIN_COMMON

MSG_WIN_CREATE
MSG_WIN_DESTROY
MSG_WIN_EXPOSE
MSG_WIN_FOCUS
MSG_WIN_INK_OFF
MSG_WIN_KEY
MSG_WIN_MOVE
MSG_WIN_PTR
MSG_WIN_REPARENT
MSG_WIN_RESIZE
MSG_WIN_RESTACK
MSG_WIN_STATE

Colormap cell parameters
Colormap parameters
Graphics Cursor
Windowing Device Status
Union of All Window Message Structures
Window Status

Border Enter/Leave Message
Button Down/Up Message
Common Part of Window Messages
Window Created Message
Window Destroyed Message
Expose Message
Focus In/Out Message
Inking Disabled Message
Key Down/Up Message
Window Move Message
Pointer Move Message
Window Re-parented Message
Window Re-sized Message
Window Re-stacked Message
Window State Change Message
**anm_create_group()**

Create an Animation Group

**Syntax**

```c
error_code
anm_create_group(ANM_GROUP_ID *ret_group,
                 GFX_DEV_ID gfxdev)
```

**Description**

`anm_create_group()` creates a new animation group. This object manages a group of animation objects. The group ID is returned in `ret_group`. A pointer to this variable should be passed to `anm_create_group()`. Use `anm_destroy_group()` to destroy this group when it is no longer needed.

If successful, this function returns `SUCCESS`.

The following table shows the default value for each parameter in the animation group `ANM_GROUP_ID`, and the functions for modifying them.

**Table 2-1  Default Values for Parameters in anm_create_group()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination viewport</td>
<td>0</td>
<td><code>anm_set_group_dst()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>NULL</td>
<td><code>anm_set_group_dst()</code></td>
</tr>
<tr>
<td>Destination drawmap x position</td>
<td>0</td>
<td><code>anm_set_group_dst()</code></td>
</tr>
<tr>
<td>Destination drawmap y position</td>
<td>0</td>
<td><code>anm_set_group_dst()</code></td>
</tr>
<tr>
<td>Double-buffered flag</td>
<td>FALSE</td>
<td><code>anm_set_group_dst()</code></td>
</tr>
</tbody>
</table>
## Animation Functions

### Attributes
- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters
- `*ret_group` — Pointer to animation group ID.
- `gfxdev` — Graphics device ID.

### Fatal Errors
- **010:036 EOS_MAUI_NOINIT** — This API has not been initialized with `anm_init()`.

### Indirect Errors
- `anm_set_group_bkg()`
- `anm_set_group_dst()`
- `blt_create_context()`
- `mem_calloc()`

### See Also
- `anm_destroy_group()`
- `ANM_GROUP_ID`
- `GFX_DEV_ID`

---

**Table 2-1 Default Values for Parameters in anm_create_group()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background drawmap</td>
<td>NULL</td>
<td><code>anm_set_group_bkg()</code></td>
</tr>
<tr>
<td>Background pixel value</td>
<td>0</td>
<td><code>anm_set_group_bkg()</code></td>
</tr>
</tbody>
</table>
anm_create_object()
Create an Animation Object

Syntax

```c
error_code anm_create_object(ANM_OBJECT_ID *ret_object,
                              ANM_GROUP_ID group,
                              ANM_OBJECT_PLACEMENT placement, ...)
```

Description

`anm_create_object()` creates a new animation object within the specified `group`. This object is used to manage an instance of a sprite on the display. The following table shows the default value for each parameter and the functions for modifying them.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>FALSE</td>
<td><code>anm_set_object_state()</code></td>
</tr>
<tr>
<td>Sprite</td>
<td>NULL</td>
<td><code>anm_set_object_sprite()</code></td>
</tr>
<tr>
<td>Frame</td>
<td>0</td>
<td><code>anm_set_object_frame()</code></td>
</tr>
<tr>
<td>Position</td>
<td>0,0</td>
<td><code>anm_set_object_pos()</code></td>
</tr>
<tr>
<td>Behavior function</td>
<td>NULL</td>
<td><code>anm_set_object_bhv()</code></td>
</tr>
<tr>
<td>Behavior function parameter</td>
<td>NULL</td>
<td><code>anm_set_object_bhv()</code></td>
</tr>
<tr>
<td>Drawing method</td>
<td>ANM_METH_DRAW</td>
<td><code>anm_set_object_meth()</code></td>
</tr>
</tbody>
</table>
The following table shows how placement may be used to specify the new position. The Parameter column shows the types of the additional parameters (represented by "...") above.

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Parameter</th>
<th>New Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANM_OBJECT_FRONT</td>
<td>None</td>
<td>In front of all objects</td>
</tr>
<tr>
<td>ANM_OBJECT_BACK</td>
<td>None</td>
<td>In back of all objects</td>
</tr>
<tr>
<td>ANM_OBJECT_FRONT_OF ref_object</td>
<td>ANM_OBJECT_ID</td>
<td>In front of ref_object</td>
</tr>
<tr>
<td>ANM_OBJECT_BACK_OF ref_object</td>
<td>ANM_OBJECT_ID</td>
<td>In back of ref_object</td>
</tr>
</tbody>
</table>

The object ID is returned in ret_object. A pointer to this variable should be passed to anm_create_object(). Use anm_destroy_object() to destroy this object when it is no longer needed. Use anm_get_object() to get the current settings in an object.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_object Pointer to animation object ID.
group Animation group ID.
placement Specifies the position of the animation group.

Optional additional parameters for placement.

**Fatal Errors**

010:008 EOS_MAUI_BADID
The ID specified by `group`, or the ID specified by `ref_object` (see `placement`) is not valid.

010:016 EOS_MAUI_BADVALUE
The value used for `placement` is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with `anm_init()`.

010:044 EOS_MAUI_NOTFOUND
The reference object `ref_object` is not in the group.

**Indirect Errors**

- `anm_set_object_bhv()`
- `anm_set_object_frame()`
- `anm_set_object_meth()`
- `anm_set_object_pos()`
- `anm_set_object_sprite()`
- `anm_set_object_state()`
- `blt_create_context()`
- `blt_set_context_dst()`
- `mem_calloc()`

**See Also**

- `anm_destroy_object()`
- `anm_get_object()`
- `anm_restack_object()`
- `ANM_GROUP_ID`
- `ANM_OBJECT_ID`
- `ANM_OBJECT_PLACEMENT`
**Syntax**

```c
error_code anm_create_sprite(ANM_SPRITE **ret_sprite,
   u_int16 num_frames, u_int32 shade)
```

**Description**

`anm_create_sprite()` creates a sprite structure with the specified number of frame structures attached. Since sprites are public data structures, you may create them in any way you want. This function is provided only as a convenience; its use is not mandatory.

`num_frames` indicates the number of entries in the array of frame structures that should be allocated and attached to the sprite structure. All allocations for the sprite are made from the specified `shade`. A pointer to the new sprite is returned in `ret_sprite`. A pointer to this variable should be passed to `anm_create_sprite()`. Use `anm_destroy_sprite()` to destroy this sprite when it is no longer needed.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `**ret_sprite` Pointer to the new sprite pointer.
- `num_frames` Number of entries in the array of frame structures.
- `shade` Memory shade for this sprite.
Fatal Errors

010:036 EOS_MAUI_NOINIT

This API has not been initialized with `anm_init()`.

Indirect Errors

`mem_calloc()`

See Also

`anm_destroy_sprite()`

`ANM_SPRITE`
**Animation Functions**

## anm_destroy_group()

### Destroy an Animation Group

**Syntax**

```c
error_code anm_destroy_group(ANM_GROUP_ID group)
```

**Description**

The `anm_destroy_group()` function destroys the specified animation group. Any remaining animation objects within this group are destroyed as part of this process.

If successful, this function returns `SUCCESS`.

**Attributes**

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**Parameters**

- **group**: Animation group ID.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `group` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `anm_init()`.

**Indirect Errors**

- `anm_destroy_object()`
- `blt_destroy_context()`
- `mem_free()`
2 Animation Functions

See Also

anm_create_group()
ANM_GROUP_ID
Animation Functions

anm_destroy_object()
Destroy an Animation Object

Syntax

error_code
anm_destroy_object(ANM_OBJECT_ID object)

Description

anm_destroy_object() destroys the specified animation object. If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

object Animation object ID.

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by object is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with anm_init().

Indirect Errors

blt_destroy_context()
mem_free()

See Also

anm_create_object()
ANM_OBJECT_ID
anm_destroy_sprite()

Destroy a Sprite

Syntax

error_code
anm_destroy_sprite(ANM_SPRITE *sprite)

Description

anm_destroy_sprite() destroys the specified sprite. Only call this function to destroy sprites created with anm_create_sprite(). If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

*sprite Pointer to sprite.

Non-Fatal Errors

010:036 EOS_MAUI_NOINIT This API has not been initialized with anm_init().

Indirect Errors

mem_free()

See Also

anm_create_sprite()
ANM_SPRITE
anm_draw_group()  
Draw Objects in a Group

Syntax
error_code
anm_draw_group(ANM_GROUP_ID group)

Description
anm_draw_group() draws all the active animation objects within the specified group. This function should be called after calling anm_process_group() so that the new state of each object is shown on the display.

The order of the drawing is from the back (lowest priority) to the front (highest priority) object. This is done so that higher priority objects get drawn on top of lower priority objects.

If any objects were deactivated since the last time this group was drawn, then the object is erased but not drawn. Likewise, if an object has been activated since the last time this group was drawn, it is drawn, but its old position is not erased.

If the background has been changed with anm_set_group_bkg() since the last time anm_draw_group() was called, then the background is drawn to the destination.

If the destination for the group is a double-buffered drawmap (see anm_set_group_dst()) then you must use gfx_update_display() after each call to anm_draw_group() to make the new state of the group visible.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Parameters

**group**

Animation group ID.

Non-Fatal Errors

010:008 EOS_MAUI_BADID

The ID specified by group is not valid.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with anm_init().

Indirect Errors

blt_copy_block()
blt_draw_block()

See Also

anm_process_group()
anm_set_group_dst()
gfx_update_display()
ANM_GROUP_ID


**anm_get_group()**  
Get Animation Group Parameters

**Syntax**

```c
error_code  
anm_get_group(ANM_GROUP_PARAMS *ret_group_params,  
               ANM_GROUP_ID group)
```

**Description**

`anm_get_group()` returns the current parameters for the specified animation group.

The group parameters are returned in `ret_group_params`. A pointer to this variable should be passed to `anm_get_group()`. The caller must ensure that `ret_group_params` points to storage large enough to hold the information.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `*ret_group_params` Pointer to animation group parameters.
- `group` Animation group ID.

**Non-Fatal Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:008 EOS_MAUI_BADID</td>
<td>The ID specified by <code>group</code> is not valid.</td>
</tr>
<tr>
<td>010:036 EOS_MAUI_NOINIT</td>
<td>This API has not been initialized with <code>anm_init()</code>.</td>
</tr>
</tbody>
</table>
2 Animation Functions

See Also

anm_create_group()
ANM_GROUP_ID
ANM_GROUP_PARAMS
**Syntax**

```c
error_code
anm_get_object(ANM_OBJECT_PARAMS *ret_object_params,
                ANM_OBJECT_ID object)
```

**Description**

`anm_get_object()` returns the current parameters for the specified animation object. The object parameters are returned in `ret_object_params`. A pointer to this variable should be passed to `anm_get_object()`. The caller must ensure that `ret_object_params` points to storage large enough to hold the information.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `*ret_object_params`: Pointer to animation object parameters.
- `object`: Animation object ID.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `object` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `anm_init()`.
See Also

- `anm_create_object()`
- `ANM_OBJECT_ID`
- `ANM_OBJECT_PARAMS`
**Syntax**

```c
error_code
anm_init(void)
```

**Description**

`anm_init()` initializes the animation API. This function must be called prior to calling any other animation function unless otherwise noted by that function.

This API depends on the Shaded Memory, Graphics Device, and Bit-BLT APIs. Therefore, `mem_init()`, `gfx_init()`, and `blt_init()` are called by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

None

**Indirect Errors**

- `blt_init()`
- `gfx_init()`
- `mem_init()`

**See Also**

- `anm_term()`
anm_process_group()
Process Objects in a Group

Syntax
error_code
anm_process_group(ANM_GROUP_ID group)

Description
anm_process_group() processes all the active animation objects
within the specified group. Processing an object consists of calling its
behavior function. Changes to the objects are not reflected on the
display until anm_draw_group() is called.

The order of the processing is from the front (highest priority) to the
back (lowest priority) object. This is done so that higher priority objects
move before (and avoid collisions with) lower priority objects.

If the behavior function returns an error, it is passed on to the caller of
this function. However, objects called after the one that generated the
error are still processed.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

group Animation group ID.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by group is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with anm_init().
Animation Functions

**Indirect Errors**

(`*bhv_func`())

**See Also**

anm_draw_group()
ANM_GROUP_ID
anm_restack_object()
Restack an Animation Object

Syntax
error_code
anm_restack_object(ANM_OBJECT_ID object,
                   ANM_OBJECT_PLACEMENT placement, ...)

Description
anm_restack_object() changes the placement of the specified object within the group to which it belongs. The effects of this call are not seen until the next time this group is drawn by anm_draw_group().

If successful, this function returns SUCCESS.

The following table shows how placement may be used to specify the new position. The Parameter column shows the types of the additional parameters (represented by "...") above.

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Prototype</th>
<th>New Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANM_OBJECT_FRONT</td>
<td>None</td>
<td>In front of all objects</td>
</tr>
<tr>
<td>ANM_OBJECT_BACK</td>
<td>None</td>
<td>In back of all objects</td>
</tr>
<tr>
<td>ANM_OBJECT_FRONT_OF</td>
<td>ANM_OBJECT_ID ref_object</td>
<td>In front of ref_object</td>
</tr>
<tr>
<td>ANM_OBJECT_BACK_OF</td>
<td>ANM_OBJECT_ID ref_object</td>
<td>In back of ref_object</td>
</tr>
</tbody>
</table>
**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

- **object**
  - Animation object ID.

- **placement**
  - Specifies the placement of the object within its group.
  - Optional additional parameters for placement.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**
  - The ID specified by object, or the ID specified by ref_object (see placement) is not valid.

- **010:016 EOS_MAUI_BADVALUE**
  - The value used for placement is not valid.

- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with anm_init().

- **010:044 EOS_MAUI_NOTFOUND**
  - The reference object ref_object is not in the group.

**See Also**

- anm_create_group()
- anm_draw_group()
- ANM_OBJECT_ID
- ANM_OBJECT_PLACEMENT
anm_set_error_action()

Set Action to Take in Error Handler

Syntax

error_code
anm_set_error_action(MAUI_ERR_LEVEL debug_level,
                     MAUI_ERR_LEVEL passback_level,
                     MAUI_ERR_LEVEL exit_level)

Description

anm_set_error_action() sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling anm_init(). Following is the table of error levels. The least severe error is listed first.

Table 2-5 Error Levels for anm_set_error_action()

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
</tbody>
</table>
debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default passback_level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to call exit(). In this case the program exits with the error code that caused the error handler to be called. The default passback_level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

### Attributes
- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters
- **debug_level** Minimum error level that causes the error handler to print a message to standard error.
- **passback_level** Minimum error level that causes the error handler to return the error.
- **exit_level** Minimum error level that causes the error handler to call exit().

### Table 2-5  Error Levels for anm_set_error_action()

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>

MAUI_ERR_AS_IS The status of the error handler is not changed.

MAUI_ERR_DEFAULT Restore the level to its default value.
Animation Functions

Non-Fatal Errors
None

See Also
anm_init()
anm_set_group_bkg()
Set Group Background

Syntax
error_code
anm_set_group_bkg(ANM_GROUP_ID group,
       const GFX_DMAP *bkgdmap,
       GFX_PIXEL bkgpixel)

Description
anm_set_group_bkg() sets the background drawmap or color for
the specified group. The effects of this call are not seen until the next
time the group is drawn by anm_draw_group().

If bkgdmap is NULL, then bkgpixel specifies the background color.

If bkgdmap is not NULL, then it specifies the drawmap to use as the
background image. If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>Animation group ID.</td>
</tr>
<tr>
<td>*bkgdmap</td>
<td>Pointer to background drawmap.</td>
</tr>
<tr>
<td>bkgpixel</td>
<td>Specifies background color if bkgdmap is NULL.</td>
</tr>
</tbody>
</table>

Non-Fatal Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:008</td>
<td>EOS_MAUI_BADID The ID specified by group is not valid.</td>
</tr>
<tr>
<td>010:036</td>
<td>EOS_MAUI_NOINIT This API has not been initialized with anm_init().</td>
</tr>
</tbody>
</table>
Indirect Errors
blt_set_context_pix()
blt_set_context_src()

See Also
anm_draw_group()
anm_get_group()
ANM_GROUP_ID
GFX_DMAP
GFX_PIXEL
anm_set_group_dst()
Set Destination Drawmap

Syntax

error_code
anm_set_group_dst(ANM_GROUP_ID group,
                      GFX_VPORT_ID vport,
                      const GFX_DMAP *dstdmap,
                      GFX_POS x,
                      GFX_POS y,
                      BOOLEAN dbl_buff)

Description

anm_set_group_dst() sets the destination drawmap and viewport for the specified group. The effects of this call are not seen until the next time gfx_update_display() is called.

vport specifies the destination viewport for the group. The destination drawmap dstdmap is automatically placed in this viewport.

dstdmap specifies the destination drawmap for the group. If the contents of this drawmap object are changed after calling this function, you must call it again to register the changes with this group. If you delete dstdmap, it must be removed from this group.

x and y indicate the offset to the first pixel of the destination drawmap *dstdmap that should be visible in the destination viewport vport.

If dbl_buff is TRUE, then the destination drawmap is divided into two pieces. The top half is displayed while the bottom is being updated, then the bottom half is displayed while the top is being updated. This technique is used to achieve flicker-free animation. Since the drawmap is divided in half, it is important to remember to create a drawmap twice as high as you would otherwise need.

If dbl_buff is TRUE, then you must use gfx_update_display() after each call to anm_draw_group() to make the new state of the group visible.

If successful, this function returns SUCCESS.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>Animation group ID.</td>
</tr>
<tr>
<td>vport</td>
<td>Destination viewport.</td>
</tr>
<tr>
<td>dst_dmap</td>
<td>Destination drawmap.</td>
</tr>
<tr>
<td>x</td>
<td>X-coordinate of upper-left pixel of destination drawmap that is visible in vport.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of upper-left pixel of destination drawmap that is visible in vport.</td>
</tr>
<tr>
<td>dbl_buff</td>
<td>Boolean TRUE enables double-buffering in destination drawmap.</td>
</tr>
</tbody>
</table>

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**
  - The ID specified by group is not valid.
- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with anm_init().

Indirect Errors

- blt_set_context_dst()
- gfx_set_vport_dmap()
See Also

anm_draw_group()
anm_get_group()
gfx_update_display()
ANM_GROUP_ID
BOOLEAN
GFX_DMAP
GFX_POS
GFX_VPORT_ID
anm_set_object_bhv()
Set Behavior for an Object

Syntax

```c
error_code
anm_set_object_bhv(ANM_OBJECT_ID object,
        error_code (*bhv_func) (ANM_OBJECT_ID,
                             const ANM_OBJECT_PARAMS *),
        void *bhv_param)
```

Description

`anm_set_object_bhv()` sets the behavior for the specified object. The effects of this call are not seen until the next time this object is drawn by `anm_draw_group()`.

`bhv_func` specifies the function to be called when the object is processed by `anm_process_group()`. If `bhv_func` is `NULL`, then no behavior function is called.

`bhv_param` is an application defined variable that is stored in the object so that it is available to the behavior function.

The behavior function `bhv_func` is defined by the caller and its prototype appears as follows:

```c
error_code bhv_func(ANM_OBJECT_ID object, 
                    const ANM_OBJECT_PARAMS *params)
```

In the behavior function, `object` is the ID for the object being processed, and `params` is a pointer to its parameter block. See `ANM_OBJECT_PARAMS` for a description of the data structure for `params`.

If successful, this function returns `SUCCESS`.

Attributes

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
</tbody>
</table>
Animation Functions

Parameters

object
Animation object ID.

*bhv_func
Function to be called when object is processed by `anm_process_group()`.

*bhv_param
Behavior parameter variable.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by object is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with `anm_init()`.

See Also

anm_draw_group()
anm_get_object()
anm_process_group()
ANM_OBJECT_ID
ANM_OBJECT_PARAMS
**anm_set_object_frame()**

*Set Frame for an Object*

**Syntax**

```c
error_code
anm_set_object_frame(ANM_OBJECT_ID object,
                     u_int16 frame)
```

**Description**

`anm_set_object_frame()` sets the frame for the specified object to `frame`. The effects of this call are not seen until the next time this object is drawn by `anm_draw_group()`.

`frame` is in the range 0 to \( n-1 \) where \( n \) is the number of frames supported by the current sprite for this object. The current sprite is specified by `anm_set_object_sprite()`. If the current sprite is set to `NULL`, then the only valid frame is 0.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **object**
  - Animation object ID.
- **frame**
  - The specific frame that is associated with this object.

**Non-Fatal Errors**

- **010:007 EOS_MAUI_BADFRAME**
  - The specified frame number is not legal for the current sprite in this object.

- **010:008 EOS_MAUI_BADID**
  - The ID specified by object is not valid.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with anm_init().

010:041 EOS_MAUI_NOSPRITE

No sprite has been assigned to this object and a non-zero value was specified for frame.

See Also
anm_draw_group()
anm_get_object()
anm_set_object_frame()
anm_set_object_sprite()
ANM_OBJECT_ID
### anm_set_object_meth()

**Set Drawing Method for an Object**

#### Syntax

```c
error_code
anm_set_object_meth(ANM_OBJECT_ID object,
                     ANM_METHOD method)
```

#### Description

`anm_set_object_meth()` sets the drawing method used by the specified object. The effects of this call are not seen until the next time the object is drawn by `anm_draw_group()`.

`method` indicates the method used for drawing the object. The following table shows the valid values for `method`.

<table>
<thead>
<tr>
<th>Value of Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANM_METH_DRAW</td>
<td>Draw the object without consideration for what is under it.</td>
</tr>
<tr>
<td>ANM_METH_DRAW_BKG</td>
<td>Same as ANM_METH_DRAW except that a background is supported.</td>
</tr>
<tr>
<td>ANM_METH_TDRAW</td>
<td>Draw the object using transparency. If the transparency mask is set, then it is used. Otherwise, the transparent pixel value is used.</td>
</tr>
<tr>
<td>ANM_METH_TDRAW_BKG</td>
<td>Same as ANM_METH_TDRAW except that a background is supported.</td>
</tr>
</tbody>
</table>

If successful, this function returns `SUCCESS`. 
### Attributes
- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

### Parameters
- **object**: Animation object ID.
- **method**: Drawing method for this object.

### Non-Fatal Errors
- **010:008 EOS_MAUI_BADID**: The ID specified by object is not valid.
- **010:016 EOS_MAUI_BADVALUE**: The method is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with anm_init().

### Indirect Errors
- **blt_set_context_cpymix()**

### See Also
- anm_draw_group()
- anm_get_object()
- ANM_METHOD
- ANM_OBJECT_ID
- ANM_SPRITE
anm_set_object_pos()
Set Position for an Object

Syntax
error_code
anm_set_object_pos(ANM_OBJECT_ID object,
                   GFX_POS x,
                   GFX_POS y)

Description
anm_set_object_pos() sets the position for the specified object to \(x,y\) relative to the origin of the destination drawmap. The effects of this call are not seen until the next time this object is drawn by anm_draw_group().

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
object Animation object ID.
x X coordinate of upper-left pixel of object.
y Y coordinate of upper-left pixel of object.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by object is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with anm_init().
See Also

- `anm_draw_group()`
- `anm_get_object()`
- `ANM_OBJECT_ID`
- `GFX_POS`
**anm_set_object_sprite()**

Set Sprite for an Object

**Syntax**

```c
error_code
anm_set_object_sprite(ANM_OBJECT_ID object,
                      const ANM_SPRITE *sprite)
```

**Description**

`anm_set_object_sprite()` sets the sprite for the specified object to `*sprite`. The effects of this call are not seen until the next time this object is drawn by `anm_draw_group()`.

If the value of `sprite` is `NULL`, then no sprite is specified for this object and the frame is automatically set to 0.

If the contents of the sprite object `sprite` are changed after calling this function, you must call it again to register the changes with this object. If you delete the `sprite`, it must be removed from this object.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `object` Animation object ID.
- `*sprite` Pointer to sprite object.

**Non-Fatal Errors**

- **010:007 EOS_MAUI_BADFRAME** The specified frame number is not valid for the specified sprite.
- **010:008 EOS_MAUI_BADID** The ID specified by `object` is not valid.
Animation Functions

010:024  EOS_MAUI_INUSE

010:036  EOS_MAUI_NOINIT

You cannot set the sprite to NULL if it is active.

This API has not been initialized with anm_init().

Indirect Errors
anm_set_object_meth()
blt_set_context_mask()
blt_set_context_src()
blt_set_context_trans()

See Also
anm_draw_group()
anm_get_object()
anm_set_object_frame()
ANM_OBJECT_ID
ANM_SPRITE
anm_set_object_state()
Set State for an Object

Syntax

error_code
anm_set_object_state(ANM_OBJECT_ID object,
                     BOOLEAN active)

Description

anm_set_object_state() sets the state for the specified object to state. The effects of this call are not seen until the next time this object is drawn by anm_draw_group().

If active is set to TRUE, then the object is activated (made visible); if set to FALSE, it is deactivated. If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

object Animation object ID.
active Sets the state of object to active (TRUE) or inactive (FALSE).

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by object is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with anm_init().
010:041 EOS_MAUI_NOSPRITE You must assign a sprite to the object before activating it.
See Also

- anm_draw_group()
- anm_get_object()
- anm_set_object_sprite()

ANM_OBJECT_ID
BOOLEAN
anm_term()

Terminate the Animation API

**Syntax**

```c
error_code
anm_term(void)
```

**Description**

`anm_term()` terminates use of the animation API. This API depends on the Shaded Memory, Graphics Device, and Bit-BLT APIs. Therefore, `mem_term()`, `gfx_term()`, and `blt_term()` are called by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

None

**Non-Fatal Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 010:036 EOS_MAUI_NOINIT | This API has not been initialized with `anm_init()`.

**Indirect Errors**

- `blt_term()`
- `gfx_term()`
- `mem_term()`

**See Also**

- `anm_init()`
**blt_copy_block()**

Copy Rectangular Block of Pixels

**Syntax**

```c
error_code blt_copy_block(BLT_CONTEXT_ID context,
                         GFX_POS dstx, GFX_POS dsty,
                         GFX_POS srcx, GFX_POS srcy,
                         GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`blt_copy_block()` copies a rectangular area of pixels using the Bit-BLT context specified by `context`.

The upper-left corner of this area is specified by `srcx` and `srcy` in the source drawmap and `dstx` and `dsty` in the destination drawmap. The dimensions of the area to copy are specified by `width` and `height`. The caller must ensure that the parameters passed to this function are within the bounds of their respective drawmaps.

The source and destination areas should not overlap. If they do, you may observe undesired side effects because the source data may be over-written before it is used.

The following table shows which parameters from the `context` object are used in this function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td><code>blt_set_context_cpymix()</code>.</td>
</tr>
<tr>
<td>Source drawmap</td>
<td><code>blt_set_context_src()</code>.</td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td><code>blt_set_context_trans()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns **SUCCESS**.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask drawmap</td>
<td><code>blt_set_context_mask()</code>.</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>blt_set_context_ofs()</code>.</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>blt_set_context_dst()</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Bit-BLT context ID.</td>
</tr>
<tr>
<td>dstx</td>
<td>X position of upper-left corner in destination drawmap.</td>
</tr>
<tr>
<td>dsty</td>
<td>Y position of upper-left corner in destination drawmap.</td>
</tr>
<tr>
<td>srcx</td>
<td>X position of upper-left corner of copy block in source drawmap.</td>
</tr>
<tr>
<td>srcy</td>
<td>Y position of upper-left corner of copy block in source drawmap.</td>
</tr>
<tr>
<td>width</td>
<td>Width of copy block in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>Height of copy block in pixels.</td>
</tr>
</tbody>
</table>
Bit-BLT Functions

Non-Fatal Errors

010:006 EOS_MAUI_BADDIMEN
The width and height of the source drawmap must be the same as the width and height of the mask drawmap.

010:009 EOS_MAUI_BADLINESIZE
The line size in the source and mask drawmaps must be the same.

010:022 EOS_MAUI_INCOMPATCM
The pixel depth for the source drawmap does not match that for the destination or the mask.

010:032 EOS_MAUI_NODSTDMAP
No destination drawmap.

010:037 EOS_MAUI_NOMASKDMAP
The mixing mode for copying specifies a mask is required, but a mask is not present in context.

010:042 EOS_MAUI_NOSRCDMAP
No source drawmap.

010:067 EOS_MAUI_NODVSUPPORT
Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

Indirect Errors

_gdv_ioblt_copyblk() Call into the graphics driver.

See Also
blt_copy_next_block()
BLT_CONTEXT_ID
BLT_MIX
GFX_DIMEN
GFX_POS
**blt_copy_next_block()**

Copy Next Rectangular Block of Pixels

### Syntax

```c
error_code blt_copy_next_block(BLT_CONTEXT_ID context, GFX_POS dstx, GFX_POS srcx, GFX_DIMEN height)
```

### Description

`blt_copy_next_block()` copies the next horizontal rectangular area of pixels using the Bit-BLT context specified by `context`.

This function operates exactly as `blt_copy_block()` except that the `dsty`, `srcy`, and `width` are not specified. They are assumed to be the same as the previous call to `blt_copy_block()` using this `context`.

The previous Bit-BLT function called with this `context` must have been `blt_copy_block()` or `blt_copy_next_block()`. Otherwise the behavior of this function is undefined.

This function is useful for repeated horizontal block copies—such as those required for text. This function is much faster than `blt_copy_block()`.

The caller must ensure that the parameters passed to this function are within the bounds of the respective drawmaps. The following table shows which parameters from the `context` object are used in this function.

### Table 3-2 Context Object Parameters Used in `blt_copy_next_block()`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td><code>blt_set_context_cpymix()</code></td>
</tr>
<tr>
<td>Source drawmap</td>
<td><code>blt_set_context_src()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns **SUCCESS**.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent pixel value</td>
<td><code>blt_set_context_trans()</code></td>
</tr>
<tr>
<td>Mask drawmap</td>
<td><code>blt_set_context_mask()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>blt_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>blt_set_context_dst()</code></td>
</tr>
</tbody>
</table>

### Table 3-2  Context Object Parameters Used in blt_copy_next_block()

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent pixel value</td>
<td><code>blt_set_context_trans()</code></td>
</tr>
<tr>
<td>Mask drawmap</td>
<td><code>blt_set_context_mask()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>blt_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>blt_set_context_dst()</code></td>
</tr>
</tbody>
</table>

If successful, this function returns **SUCCESS**.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- **context** Bit-BLT context ID.
- **dstx** X coordinate of top pixel placement of next block to copy to destination drawmap.
- **srcx** X coordinate of top pixel of next block to copy in source drawmap.
- **height** Height of next block to copy.

### Non-Fatal Errors

- **010:006 EOS_MAUİ_BADDIMEN** The width and height of the source drawmap must be the same as the width and height of the mask drawmap.
Bit-BLT Functions

010:009  EOS_MAUI_BADLINESIZE  The line size in the source and mask drawmaps must be the same.

010:022  EOS_MAUI_INCOMPATCM  The pixel depth for the source drawmap does not match that for the destination or the mask.

010:032  EOS_MAUI_NODSTDMAP  No destination drawmap.

010:037  EOS_MAUI_NOMASKDMAP  The mixing mode for copying specifies a mask is required, but a mask is not present in context.

010:042  EOS_MAUI_NOSRCDMAP  No source drawmap.

010:067  EOS_MAUI_NODVSUPPORT  Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

Indirect Errors

_gdv_ioblt_copynblk()  Call into the graphics driver.

See Also

blt_copy_block()
BLT_CONTEXT_ID
BLT_MIX
GFX_DIMEN
GFX_POS
blt_copy_oblock()
Copy Overlapping Blocks of Pixels

Syntax

error_code
blt_copy_oblock(BLT_CONTEXT_ID context,
    GFX_POS dstx, GFX_POS dsty,
    GFX_POS srcx, GFX_POS srcy,
    GFX_DIMEN width, GFX_DIMEN height)

Description

blt_copy_oblock() copies a rectangular area of pixels using the Bit-BLT context specified by context.

The upper-left corner of this area is specified by srcx and srcy in the source drawmap and dstx and dsty in the destination drawmap. The dimensions of the area to copy are specified by width and height. The caller must ensure that the parameters passed to this function are within the bounds of their respective drawmaps.

The source and destination areas can overlap. This function is not as efficient as blt_copy_block() because it has to check for overlapping copy areas and compensate for them.

The following table shows which parameters from the context object are used in this function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td>blt_set_context_cpymix()</td>
</tr>
<tr>
<td>Source drawmap</td>
<td>blt_set_context_src()</td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td>blt_set_context_trans()</td>
</tr>
</tbody>
</table>
If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `context`: Bit-BLT context ID.
- `dstx`: X position of upper-left corner in destination drawmap.
- `dsty`: Y position of upper-left corner in destination drawmap.
- `srcx`: X position of upper-left corner of copy block in source drawmap.
- `srcy`: Y position of upper-left corner of copy block in source drawmap.
- `width`: Width of copy block in pixels.
- `height`: Height of copy block in pixels.

---

**Table 3-3  Context Object Parameters Used in `blt_copy_block()`**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask drawmap</td>
<td><code>blt_set_context_mask()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>blt_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>blt_set_context_dst()</code></td>
</tr>
</tbody>
</table>
Non-Fatal Errors

010:006 EOS_MAUI_BADDIMEN
The width and height of the source drawmap must be the same as the width and height of the mask drawmap.

010:009 EOS_MAUI_BADLINESIZE
The line size in the source and mask drawmaps must be the same.

010:022 EOS_MAUI_INCOMPATCM
The pixel depth for the source drawmap does not match that for the destination or the mask.

010:032 EOS_MAUI_NODSTDMAP
No destination drawmap.

010:037 EOS_MAUI_NOMASKDMAP
The mixing mode for copying specifies a mask is required, but a mask is not present in context.

010:042 EOS_MAUI_NOSRCDMAP
No source drawmap.

010:045 EOS_MAUI_NOTIMPLEMENTED
A mixing mode other than BLT_MIX_REPLACE was specified for a fully overlapping copy.

010:067 EOS_MAUI_NODVSUPPORT
Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

Indirect Errors

blt_copy_block()
See Also

blt_copy_block()
BLT_CONTEXT_ID
BLT_MIX
GFX_DIMEN
GFX_POS
3 Bit-BLT Functions

blt_create_context()
Create a Bit-BLT Context Object

Syntax

```c
error_code blt_create_context(BLT_CONTEXT_ID *ret_context,
                            GFX_DEV_ID gfxdev)
```

Description

`blt_create_context()` creates a new Bit-BLT context object. The context object modifies the behavior of the copy, draw, and expand functions. The following table shows the default value for each setting and the functions for modifying them.

Table 3-4   Value of Context Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode for draw</td>
<td>BLT_MIX_REPLACE</td>
<td><code>blt_set_context_drwmix()</code></td>
</tr>
<tr>
<td>Mixing mode for copy</td>
<td>BLT_MIX_REPLACE</td>
<td><code>blt_set_context_cpymix()</code></td>
</tr>
<tr>
<td>Mixing mode for expand</td>
<td>BLT_MIX_REPLACE</td>
<td><code>blt_set_context_expmix()</code></td>
</tr>
<tr>
<td>Pixel value for drawing</td>
<td>1</td>
<td><code>blt_set_context_pix()</code></td>
</tr>
<tr>
<td>Source drawmap</td>
<td>NULL</td>
<td><code>blt_set_context_src()</code></td>
</tr>
<tr>
<td>Entries in expansion table</td>
<td>0</td>
<td><code>blt_set_context_exptbl()</code></td>
</tr>
</tbody>
</table>
Table 3-4  (continued) Value of Context Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel expansion table</td>
<td>NULL</td>
<td>blt_set_context_xptbl()</td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td>0</td>
<td>blt_set_context_trans()</td>
</tr>
<tr>
<td>Mask drawmap</td>
<td>NULL</td>
<td>blt_set_context_mask()</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td>0</td>
<td>blt_set_context_ofs()</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>NULL</td>
<td>blt_set_context_dst()</td>
</tr>
</tbody>
</table>

The context ID is returned in ret_context. A pointer to this variable should be passed to blt_create_context(). Use blt_destroy_context() to destroy this object when it is no longer needed. Use blt_get_context() to get the current settings in a context.

The variable gfxdev specifies the graphics device with which to associate the Bit-BLT context. If successful, it returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `*ret_context`   Pointer to context ID.
- `gfxdev`          Graphics device ID.
Fatal Errors

010:036 EOS_MAUI_NOINIT

This API has not been initialized with blt_init().

Indirect Errors

blt_set_context_cpymix()
blt_set_context_drwmix()
blt_set_context_dst()
blt_set_context_expmix()
blt_set_context_exptbl()
blt_set_context_ofs()
blt_set_context_mask()
blt_set_context_pix()
blt_set_context_src()
blt_set_context_trans()
mem_malloc()

_os_ss_blt_creatbc() Call into the graphics driver.

See Also

blt_destroy_context()
blt_get_context()
BLT_CONTEXT_ID
GFX_DEV_ID
**blt_destroy_context()**

Destroy a Bit-BLT Context Object

**Syntax**

```c
error_code blt_destroy_context(BLT_CONTEXT_ID context)
```

**Description**

`blt_destroy_context()` destroys the specified Bit-BLT context object `context`. If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `context` BLT context ID.

**Non-Fatal Errors**

- **010:036 EOS_MAUI_NOINIT** This API has not been initialized with `blt_init()`.
- **010:008 EOS_MAUI_BADID** The ID specified by `context` is not valid.

**Indirect Errors**

- `mem_free()` Driver does not support this function.
- `_os_ss_blt_destroybc()` Call into the graphics driver.

**See Also**

- `blt_create_context()`
- `BLT_CONTEXT_ID`
**blt_draw_block()**

**Draw Block of Pixels**

**Syntax**

```c
error_code blt_draw_block(BLT_CONTEXT_ID context,
                          GFX_POS dstx, GFX_POS dsty,
                          GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`blt_draw_block()` draws a block of pixels using the Bit-BLT context specified by `context`.

The upper-left corner is defined by `dstx` and `dsty` and the dimensions are defined by `width` and `height`.

All pixels within this block are drawn to the destination drawmap using the pixel value specified in the context object. The caller must ensure that the parameters passed to this function are within the bounds of the destination drawmap. The following table shows which parameters from the context object are used in this function.

**Table 3-5  Context Object Parameters Used in blt_draw_block()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td><code>blt_set_context_drwmix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>blt_set_context_pix()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>blt_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>blt_set_context_dst()</code></td>
</tr>
</tbody>
</table>

If successful, this function returns `SUCCESS`.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context BLT context ID.
dstx X coordinate of upper-left corner of block in destination drawmap.
dsty Y coordinate of upper-left corner of block in destination drawmap.
width Width of block in pixels.
height Height of block in pixels.

Non-Fatal Errors
010:032 EOS_MAUI_NODSTDMAP No destination drawmap.
010:067 EOS_MAUI_NODVSUPPORT Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

Indirect Errors
_gdv_iobltnblt_drawblk() Call into the graphics driver.

See Also
blt_set_context_drwmix()
blt_set_context_dst()
blt_set_context_ofs()
blt_set_context_pix()
BLT_CONTEXT_ID
BLT_MIX
GFX_DIMEN
GFX_POS
3 Bit-BLT Functions

blt_draw_hline()
Draw Horizontal Line of Pixels

Syntax

```c
error_code blt_draw_hline(BLT_CONTEXT_ID context, GFX_POS dstx, GFX_POS dsty, GFX_DIMEN width)
```

Description

`blt_draw_hline()` draws a horizontal line of pixels using the Bit-BLT context specified by `context`.

The left-most pixel is specified by `dstx` and `dsty` and the width is defined by `width`.

All pixels within this line are drawn to the destination drawmap using the drawing pixel value. The drawing pixel value is set with `blt_set_context_pix()`). It is the responsibility of the caller to ensure that the parameters passed to this function are within the bounds of the destination drawmap.

### Table 3-6 Context Object Parameters Used in blt_draw_hline()

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td><code>blt_set_context_drwmix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>blt_set_context_pix()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>blt_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>blt_set_context_dst()</code></td>
</tr>
</tbody>
</table>

If successful, this function returns `SUCCESS`.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context BLT context ID.
dstx X coordinate of left-most pixel in line.
dstx Y coordinate of left-most pixel in line.
width Width of line in pixels.

Non-Fatal Errors
010:032 EOS_MAUI_NODSTDMAP No destination drawmap.
010:067 EOS_MAUI_NODVSUPPORT Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

Indirect Errors
_gdv_ioblt_drawhline() Call into the graphics driver.

See Also
blt_set_context_drwmix()
blt_set_context_dst()
blt_set_context_ofs()
blt_set_context_pix()
BLT_CONTEXT_ID
BLT_MIX
GFX_DIMEN
GFX_POS
**blt_draw_pixel()**

*Draw a Single Pixel*

**Syntax**

```c
error_code blt_draw_pixel(BLT_CONTEXT_ID context, GFX_POS dstx, GFX_POS dsty)
```

**Description**

`blt_draw_pixel()` draws a single pixel using the Bit-BLT context specified by `context`.

The pixel at `dstx` and `dsty` in the destination drawmap is drawn using the drawing pixel value. The drawing pixel value is set with `blt_set_context_pix()`. The caller must ensure that the parameters passed to this function are within the bounds of the destination drawmap. The following table shows which parameters from the context object are used in this function.

**Table 3-7  Context Object Parameters Used in blt_draw_pixel()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td><code>blt_set_context_drwmix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>blt_set_context_pix()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>blt_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>blt_set_context_dst()</code></td>
</tr>
</tbody>
</table>

If successful, this function returns `SUCCESS`. 
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context BLT context ID.
dstx X coordinate of pixel to draw.
dsty Y coordinate of pixel to draw.

Non-Fatal Errors
010:032 EOS_MAU1_NODSTDMAP No destination drawmap.
010:067 EOS_MAU1_NODVSUPPORT Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

Indirect Errors
_gdv_ioblt_drawpixel() Call into the graphics driver.

See Also
blt_set_context_drwmix()
blt_set_context_dst()
blt_set_context_ofs()
blt_set_context_pix()
BLT_CONTEXT_ID
BLT_MIX
GFX_POS
3 Bit-BLT Functions

blt_draw_vline()

Draw Vertical Line of Pixels

Syntax

error_code
blt_draw_vline(BLT_CONTEXT_ID context, GFX_POS dstx, GFX_POS dsty, GFX_DIMEN height)

Description

blt_draw_vline() draws a vertical line of pixels using the Bit-BLT context specified by context.

The top-most pixel is specified by dstx and dsty and the height is defined by height.

All pixels within this line are drawn to the destination drawmap using the drawing pixel value. The drawing pixel value is set with blt_set_context_pix().

The caller must ensure that the parameters passed to this function are within the bounds of the destination drawmap. The following table shows which parameters from the context are used in this function.

Table 3-8 Context Object Parameters Used in blt_draw_vline()

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td>blt_set_context_drwmix().</td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td>blt_set_context_pix().</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td>blt_set_context_ofs().</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>blt_set_context_dst().</td>
</tr>
</tbody>
</table>

If successful, this function returns SUCCESS.
**Attributes**
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**
context
BLT context ID.
dstx
X coordinate of top of line.
dsty
Y coordinate of top of line.
height
Height of line in pixels.

**Non-Fatal Errors**
010:032 EOS_MAUI_NODSTDMAP
No destination drawmap.
010:067 EOS_MAUI_NODVSUPPORT
Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

**Indirect Errors**
_gdv_ioblt_drawvline() Call into the graphics driver.

**See Also**
blt_set_context_drwmix()
blt_set_context_dst()
blt_set_context_ofs()
blt_set_context_pix()
BLT_CONTEXT_ID
BLT_MIX
GFX_DIMEN
GFX_POS
3 Bit-BLT Functions

**blt_expd_block()**

Expand a Block of Pixels

**Syntax**

```c
error_code blt_expd_block(BLT_CONTEXT_ID context,
GFX_POS dstx, GFX_POS dsty,
GFX_POS srcx, GFX_POS srcy,
GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`blt_expd_block()` copies a rectangular area of pixels using the Bit-BLT context specified by `context`. This function is similar to `blt_copy_block()` except that it expands the pixels as it copies them. Currently the source drawmap must have pixels that are 1 bit deep, and the destination drawmap must have pixels that are 4, 8, 16, or 32 bits deep.

The upper-left corner of this area is specified by `srcx` and `srcy` in the source drawmap and `dstx` and `dsty` in the destination drawmap. The dimensions of the area to copy are specified by `width` and `height`. The caller must ensure that the parameters passed to this function are within the bounds of their respective drawmaps. The following table shows which parameters from the context object are used in this function.

**Table 3-9  Context Object Parameters Set in blt_expd_block()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode</td>
<td><code>blt_set_context_epxmix()</code></td>
</tr>
<tr>
<td>Source drawmap</td>
<td><code>blt_set_context_scr()</code></td>
</tr>
<tr>
<td>Expansion table</td>
<td><code>blt_set_context_exptbl()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns **SUCCESS**.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

- **context**: BLT context ID.
- **dstx**: X coordinate upper-left corner of block in destination drawmap.
- **dsty**: Y coordinate upper-left corner of block in destination drawmap.
- **srcx**: X coordinate of upper-left corner of block in source drawmap.
- **srcy**: Y coordinate of upper-left corner of block in source drawmap.
- **width**: Width of block in pixels.
- **height**: Height of block in pixels.

**Non-Fatal Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 010:022  | EOS_MAUİ_INCOMPATCM  
  The pixel depth for the source or destination is not valid. |
| 010:032  | EOS_MAUİ_NODSTDMAP  
  No destination drawmap. |
| 010:033  | EOS_MAUİ_NOEXPTABLE  
  No expansion table. |

---

**Table 3-9  Context Object Parameters Set in blt_expd_block()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset pixel value</td>
<td>blt_set_context_ofs()</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>blt_set_context_dst()</td>
</tr>
</tbody>
</table>
010:042 EOS_MAUI_NOSRCMDAP  
010:067 EOS_MAUI_NODVSUPPORT

No source drawmap.
Driver support for this type of Bit-BLT operation is required, but the driver does not support it.

Indirect Errors

_gdv_ioblt_expdblk()  Call into the graphics driver.

See Also

blt_copy_block()  
blt_expd_next_block()  
blt_set_context_dst()  
blt_set_context_expmix()  
blt_set_context_exptbl()  
blt_set_context_ofs()  
blt_set_context_src()  
BLT_CONTEXT_ID  
BLT_MIX  
GFX_DIMEN  
GFX_POS
**blt_expd_next_block()**

Expand the Next Block of Pixels

**Syntax**

```c
error_code blt_expd_next_block(BLT_CONTEXT_ID context,
                              GFX_POS dstx, GFX_POS srcx,
                              GFX_DIMEN height)
```

**Description**

`blt_expd_next_block()` copies the next horizontal rectangular area of pixels using the Bit-BLT context specified by `context`. Items in the `context` that are used to perform this operation depend upon the current mixing mode set by `blt_set_context_expmix()`.

This is similar to `blt_copy_next_block()` except that it expands the pixels as it copies them. This function operates exactly as `blt_expd_block()` except that the `dsty`, `srcy`, and `width` are not specified. They are assumed to be the same as the previous call to `blt_expd_block()` using this `context`.

The previous Bit-BLT function called with this `context` must have been `blt_expd_block()` or `blt_expd_next_block()`. Otherwise, the behavior of this function is undefined.

This function is useful for repeated horizontal block copies—such as those required for text. This function is much faster than `blt_expd_block()`.

The caller must ensure that the parameters passed to this function are within the bounds of their respective drawmaps.

For More Information

See **BLT_MIX** for additional information.
If successful, this function returns `SUCCESS`.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

- `context` BLT context ID.  
- `dstx` X coordinate of the left-most pixel in block in destination drawmap.  
- `srcx` X coordinate of the left-most pixel in block in source drawmap.  
- `height` Height of block in pixels.

**Non-Fatal Errors**

- 010:022 `EOS_MAUI_INCOMPATCM` The pixel depth for the source or destination is not valid.  
- 010:032 `EOS_MAUI_NODSTDMAP` No destination drawmap.  
- 010:033 `EOS_MAUI_NOEXPTABLE` No expansion table.  
- 010:042 `EOS_MAUI_NOSRCDMAP` No source drawmap.  
- 010:067 `EOS_MAUI_NODVSUPPORT` DRIVER SUPPORT FOR THIS type of Bit-BLT operation is required, but the driver does not support it.

**Indirect Errors**

- `_gdv_ioblt_expdnblk()` Call into the graphics driver.
Bit-BLT Functions

See Also

blt_copy_next_block()
blt_expd_block()
blt_set_context_dst()
blt_set_context_expmix()
blt_set_context_exptbl()
blt_set_context ofs()
blt_set_context_src()
BLT_CONTEXT_ID
BLT_MIX
GFX_DIMEN
GFX_POS
**blt_get_context()**

Get Bit-BLT Context Parameters

---

**Syntax**

```
error_code
blt_get_context(BLT_CONTEXT_PARAMS *ret_context_params,
                BLT_CONTEXT_ID context)
```

**Description**

`blt_get_context()` returns the current parameters for the specified Bit-BLT context.

The parameters are returned in `ret_context_params`. A pointer to this structure should be passed to `blt_get_context()`. The caller must ensure that `ret_context_params` points to storage large enough to hold the information.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `*ret_context_params` Pointer to structure storing context parameters.
- `context` BLT context ID.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID** The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT** This API has not been initialized with `blt_init()`.
See Also

blt_create_context()
blt_set_context_cpymix()
blt_set_context_drwmix()
blt_set_context_dst()
blt_set_context_expmix()
blt_set_context_exptbl()
blt_set_context_mask()
blt_set_context_ofs()
blt_set_context_pix()
blt_set_context_src()
blt_set_context_trans()
BLT_CONTEXT_ID
BLT_CONTEXT_PARAMS
**blt_get_pixel()**

Get Pixel

### Syntax

```c
error_code blt_get_pixel(GFX_PIXEL *ret_pixel,
                         BLT_CONTEXT_ID context, GFX_POS srcx,
                         GFX_POS srcy)
```

### Description

`blt_get_pixel()` gets the value of a single pixel using the Bit-BLT context specified by `context`.

The pixel at `srcx` and `srcy` is retrieved from the source drawmap. The caller must ensure that the parameters passed to this function are within the bounds of the source drawmap.

The pixel value is returned in `ret_pixel`. A pointer to this variable should be passed to `blt_get_pixel()`.

If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `*ret_pixel` Pointer to variable storing pixel value.
- `context` BLT context ID.
- `srcx` X coordinate of pixel in source drawmap.
- `srcy` Y coordinate of pixel in source drawmap.

### Non-Fatal Errors

- **010:042 EOS_MAUI_NOSRCDMAP** No source drawmap.
See Also

blt_set_context_src()
BLT_CONTEXT_ID
GFX_PIXEL
GFX_POS
**blt_init()**

Initialize the Bit-BLT API

**Syntax**

```c
error_code
blt_init(void)
```

**Description**

`blt_init()` initializes the Bit-BLT API. This function must be called prior to a call to any other Bit-BLT function unless otherwise noted by that function.

This API depends on the Shaded Memory and Graphics Device APIs. Therefore, `mem_init()` and `gfx_init()` are called by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Fatal Errors**

`gfx_init()`
`mem_init()`

**See Also**

`blt_term()`
**blt_set_context_cpymix()**

Set Mixing Mode for Copying

**Syntax**

```c
error_code blt_set_context_copy(BLT_CONTEXT_ID context, BLT_MIX cpymix)
```

**Description**

`blt_set_context_cpymix()` sets the mixing mode for copy operations in the specified Bit-BLT context to `cpymix`. The mixing mode specifies the way source pixels are transferred to the destination drawmap.

See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter. If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `context` : BLT context ID.
- `cpymix` : Mixing mode for copy operations.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `context` is not valid.

- **010:016 EOS_MAUI_BADVALUE**
  - An invalid mixing mode `cpymix` was specified.

- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `blt_init()`.
010:061 EOS_MAUI_DISABLED

The specified mixing mode cpymix has been disabled in this configuration of MAUI.

Indirect Errors

_gdv_blt_cpymix() Call into the graphics driver.

See Also

blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
**blt_set_context_drwmix()**

Set Mixing Mode for Drawing

### Syntax

```c
error_code blt_set_context_draw(BLT_CONTEXT_ID context,
                                BLT_MIX drwmix)
```

### Description

The `blt_set_context_drwmix()` function sets the mixing mode for draw operations in the specified Bit-BLT context to `drwmix`. The mixing mode specifies the way source pixels are transferred to the destination drawmap.

See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter. If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- **context**
  - BLT context ID.
- **drwmix**
  - Mixing mode for drawing.

### Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `context` is not valid.
- **010:016 EOS_MAUI_BADVALUE**
  - An invalid mixing mode `drwmix` was specified.
- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `blt_init()`.
The specified mixing mode `drwmix` has been disabled in this configuration of MAUI.

**Indirect Errors**

`_gdv_blt_drwmix()` Call into the graphics driver.

**See Also**

`blt_create_context()`  
`blt_get_context()`  
`BLT_CONTEXT_ID`  
`BLT_MIX`


**blt_set_context_dst()**

Set Destination Drawmap

**Syntax**

```c
error_code blt_set_context_dst(BLT_CONTEXT_ID context, const GFX_DMAP *dstdmap)
```

**Description**

`blt_set_context_dst()` sets the destination drawmap value for the specified Bit-BLT context to `dstdmap`. If set to NULL, then the destination drawmap becomes undefined and Bit-BLT operations that require a destination drawmap return the error `EOS_MAUI_NODSTDMAP`.

If the contents of the drawmap object `dstdmap` are changed after calling this function, you must call it again to register the changes with this context object. If you delete the `dstdmap`, it must be removed from this context.

See **BLT_MIX** for information about which Bit-BLT operations are affected by this parameter.

If the coding method in `srcdmap` requires driver-supported Bit-BLT and the driver does not support it, the Bit-BLT operations return `EOS_MAUI_NODVSUPPORT`.

**For More Information**

See **GFX_CM** for information about coding methods that require driver support for Bit-BLT.

If successful, this function returns `SUCCESS`. 
3 Bit-BLT Functions

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context BLT context ID.
*dstdmap Pointer to drawmap.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by context is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with blt_init().
010:039 EOS_MAUI_NOPIXMEM The destination drawmap dstdmap has no pixel memory.
010:043 EOS_MAUI_NOTALIGNED Either dstdmap->pixmem or dstdmap->line_size is not a multiple of GFX_LINE_PAD.

Indirect Errors
_gdv_blt_dst() Call into the graphics driver.

See Also
blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
GFX_CM
GFX_DMAP
### blt_set_context_expmix()

Set Mixing Mode for Expanding

#### Syntax

```c
error_code blt_set_context_expmix(BLT_CONTEXT_ID context, BLT_MIX expmix)
```

#### Description

`blt_set_context_expmix()` sets the mixing mode for expansion operations in the specified Bit-BLT context to `expmix`. The mixing mode specifies the way source pixels are transferred to the destination drawmap.

#### For More Information

See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter.

If successful, this function returns `SUCCESS`.

#### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

#### Parameters

- **context**
  - BLT context ID.
- **expmix**
  - Mixing mode for expansion.

#### Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `context` is not valid.
3 Bit-BLT Functions

010:016 EOS_M celui BADVALUE
An invalid mixing mode expmix was specified.

010:036 EOS_M celui NOINIT
This API has not been initialized with blt_init().

010:061 EOS_M celui DISABLED
The specified mixing mode expmix has been disabled in this configuration of MAUI.

Indirect Errors
_gdv_blt_expmix() Call into the graphics driver.

See Also
blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
`blt_set_context_exptbl()`
Set Pixel Expansion Table

**Syntax**
```c
error_code blt_set_context_exptbl(BLT_CONTEXT_ID context, u_int8 num_entries, const GFX_PIXEL exptbl[])
```

**Description**
`blt_set_context_exptbl()` sets the pixel expansion table for the specified Bit-BLT context to `exptbl`. `num_entries` specifies the number of values in the table `exptbl`.

If `exptbl` is set to `NULL`, then the pixel expansion table becomes undefined and Bit-BLT operations that require an expansion table return the error `EOS_MAUI_NOEXPTABLE`.

If the contents of the expansion table `exptbl` are changed after calling this function, you must call it again to register the changes with this context object. Since this function makes a copy of the data pointed to by `exptbl`, you may destroy `exptbl` immediately after calling `blt_set_context_exptbl()`.

**For More Information**
See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter.

If successful, this function returns `SUCCESS`.

**Attributes**
- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
3 Bit-BLT Functions

Parameters
context
BLT context ID.
num_entries
Number of values in expansion table.
exptbl[]
Points to expansion table.

Non-Fatal Errors
010:008 EOS_MAUI_BADID
The ID specified by context is not valid.
010:016 EOS_MAUI_BADVALUE
The value num_entries must be greater than or equal to 2, if exptbl is not NULL.
010:036 EOS_MAUI_NOINIT
This API has not been initialized with blt_init().

Indirect Errors
_gdv_blt_exptbl() Call into the graphics driver.

See Also
blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
GFX_PIXEL
blt_set_context_mask()

Set Mask Drawmap

**Syntax**

```c
error_code blt_set_context_mask(BLT_CONTEXT_ID context,
                               const GFX_DMAP *mask_dmap)
```

**Description**

`blt_set_context_mask()` sets the mask drawmap value for the specified Bit-BLT context to `mask_dmap`. If set to `NULL`, then the mask drawmap becomes undefined and Bit-BLT operations that require a mask (currently only copy operations) return the error `EOS_MAUI_NOMASKDMAP`.

If the pixel depth of the mask differs from the source, then `EOS_MAUI_INCOMPATCM` is returned from operations that use the mask.

If the line size of the mask is different from the source, then `EOS_MAUI_BADLINESIZE` is returned from operations that use the mask.

If the width and height of the mask is different from the source, then `EOS_MAUI_BADDIMEN` is returned from operations that use the mask.

The mask drawmap is used by copy operations as follows. When a source pixel is read from the source drawmap, the corresponding pixel is also read from the mask drawmap. For each bit in the mask that is 1, the corresponding bit in the source is transferred to the destination. For each bit in the mask that is 0, the corresponding bit in the source is ignored.

If the contents of the drawmap object `mask_dmap` are changed after calling this function, you must call it again to register the changes with this context object. If you delete the `mask_dmap`, it must be removed from this context.

See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter.
If successful, this function returns **SUCCESS**.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>BLT context ID.</td>
</tr>
<tr>
<td>*mask_dmap</td>
<td>Pointer to mask drawmap value.</td>
</tr>
</tbody>
</table>

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**  
  The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT**  
  This API has not been initialized with `blt_init()`.
- **010:039 EOS_MAUI_NOPIXMEM**  
  The drawmap `mask_dmap` has no pixel memory.

**Indirect Errors**

- `_gdv_blt_mask()`  
  Call into the graphics driver.

**See Also**

- `blt_copy_block()`  
- `blt_copy_next_block()`  
- `blt_create_context()`  
- `blt_get_context()`  
- `BLT_CONTEXT_ID`  
- `BLT_MIX`  
- `GFX_DMAP`
**blt_set_context_ofs()**

**Set Offset Pixel Value**

**Syntax**

```c
error_code blt_set_context_ofs(BLT_CONTEXT_ID context, GFX_PIXEL ofspixel)
```

**Description**

`blt_set_context_ofs()` sets the offset pixel value for the specified Bit-BLT context to `ofspixel`. The offset pixel value is added to the source pixels before they are transferred to the destination when `BLT_MIX_SPO` is used.

**For More Information**

See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context**
  - `BLT` context ID.
- **ofspixel**
  - Offset pixel value.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `context` is not valid.
010:036 EOS_MAUI_NOINIT

_gdv_blt ofs()

This API has not been initialized with blt_init().

Indirect Errors

Call into the graphics driver.

See Also
blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
GFX_PIXEL
blt_set_context_pix()
Set Pixel Value for Drawing

**Syntax**

```c
error_code blt_set_context_pix(BLT_CONTEXT_ID context, GFX_PIXEL drwpixel)
```

**Description**

`blt_set_context_pix()` sets the pixel value used for drawing with the specified Bit-BLT context to `drwpixel`.

**Note**

The contents of `drwpixel` should be in the endianess of the destination drawmap's coding method rather than that of the CPU.

See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter.

**For More Information**

See `GFX_CM` for coding method specific details for formatting pixels.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
3 Bit-BLT Functions

Parameters

context  BLT context ID.
drwpixel  Pixel value used for drawing.

Non-Fatal Errors

010:008 EOS_MAUI_BADID  The ID specified by context is not valid.
010:036 EOS_MAUI_NOINIT  This API has not been initialized with blt_init().

Indirect Errors

_gdv_blt_pix()  Call into the graphics driver.

See Also

blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
GFX_CM
GFX_PIXEL
**blt_set_context_src()**

Set Source Drawmap

**Syntax**

```c
error_code blt_set_context_src(BLT_CONTEXT_ID context,
                                const GFX_DMAP *srcdmap)
```

**Description**

`blt_set_context_src()` sets the source drawmap value for the specified Bit-BLT context to `srcdmap`. If set to NULL, then the source drawmap becomes undefined and Bit-BLT operations that require a source drawmap return the error `EOS_MAUI_NOSRCDMAP`.

If the contents of the drawmap object `srcdmap` are changed after calling this function, you must call it again to register the changes with this context object. If you delete the `srcdmap`, it must be removed from this context.

See `BLT_MIX` for information about which Bit-BLT operations are affected by this parameter. If the coding method in `srcdmap` requires driver supported Bit-BLT and the driver does not support it, then Bit-BLT operations return `EOS_MAUI_NODVSUPPORT`.

**For More Information**

See `GFX_CM` for information about coding methods that require driver support for Bit-BLT.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
3 Bit-BLT Functions

**Parameters**

context

BLT context ID.

*srcdmap

Pointer to source drawmap.

**Non-Fatal Errors**

010:008 EOS_MAUI_BADID

The ID specified by context is not valid.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with blt_init().

010:039 EOS_MAUI_NOPIXMEM

The source drawmap srcdmap has no pixel memory.

010:043 EOS_MAUI_NOTALIGNED

Either srcdmap->pixmem or srcdmap->line_size is not a multiple of GFX_LINE_PAD.

**Indirect Errors**

_gdv_blt_src() Call into the graphics driver.

**See Also**

blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
GFX_CM
GFX_DMAP
blt_set_context_trans()
Set Transparent Pixel Value

Syntax
error_code
blt_set_context_trans(BLT_CONTEXT_ID context,
GFX_PIXEL transpixel)

Description
blt_set_context_trans() sets the transparent pixel value for the specified Bit-BLT context to transpixel. The transparent pixel value is used to filter out transparent source pixels when they are transferred to the destination.

For More Information
See BLT_MIX for information about which Bit-BLT operations are affected by this parameter.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context BLT context ID.
transpixel Transparent pixel value.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by context is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with blt_init().

Indirect Errors

_gdv_blt_trans() Call into the graphics driver.

See Also
blt_create_context()
blt_get_context()
BLT_CONTEXT_ID
BLT_MIX
GFX_PIXEL
**blt_set_error_action()**

Set Action to Take in Error Handler

### Syntax

```c
error_code blt_set_error_action(MAUI_ERR_LEVEL debug_level,
                               MAUI_ERR_LEVEL passback_level,
                               MAUI_ERR_LEVEL exit_level)
```

### Description

`blt_set_error_action()` sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling `blt_init()`. Following is the table of error levels. The least severe error is listed first.

#### Table 3-10  Error Levels in `blt_set_error_action()`

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation completed but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error</td>
</tr>
</tbody>
</table>
### Bit-BLT Functions

**debug_level** sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is **MAUI_ERR_ANY**.

**passback_level** sets the minimum error level that causes the error handler to return the error. For less severe errors, **SUCCESS** is returned. The default pass-back level is **MAUI_ERR_NON_FATAL**.

**exit_level** sets the minimum error level that causes the error handler to call **exit()**. In this case the program exits with the error code that caused the error handler to be called. The default debug level is **MAUI_ERR_NONE**.

If successful, this function returns **SUCCESS**.

#### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

#### Parameters

- **debug_level**
  - Minimum error level that causes the error handler to print a message to standard error.

- **passback_level**
  - Minimum error level that causes the error handler to return the error.

- **exit_level**
  - Minimum error level that causes the error handler to call **exit()**.

---

### Table 3-10 Error Levels in blt_set_error_action()

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>

---

- **MAUI_ERR_AS_IS**: The status of the error handler is not changed.
- **MAUI_ERR_DEFAULT**: Restore the level to its default value.
**Non-Fatal Errors**

None

**See Also**

`blt_init()`
### blt_term()

**Terminate the Bit-BLT API**

#### Syntax

```c
error_code
blt_term(void)
```

#### Description

`blt_term()` terminates the Bit-BLT API.

This API depends on the Shaded Memory and Graphics Device APIs. Therefore, `mem_term()` and `gfx_term()` are called by this function.

If successful, this function returns `SUCCESS`.

#### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

#### Non-Fatal Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 010:036 EOS_MAUI_NOINIT | This API has not been initialized with `blt_init()`.

#### Indirect Errors

- `gfx_term()`, `mem_term()`

#### See Also

- `blt_init()`
cdb_get_copy()
Get Copy of the CDB

Syntax
error_code
cdb_get_copy(char **ret_buffer, u_int32 shade_id)

Description
cdb_get_copy() returns a pointer to a copy of the Configuration Description Block (CDB). ret_buffer is a pointer to a character pointer. Use mem_free() to return the memory allocated by this function. Use shade_id to specify from what shade to allocate the memory.

The returned copy of the CDB is formatted as a single NULL terminated string with each Device Description Record (DDR) separated by a carriage return (0x0D) character.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
**ret_buffer Pointer to the copy of the CDB.
shade_id Specifies what shade of memory to allocate where the copy of the CDB is to be stored.

Non-Fatal Errors
010:036 EOS_MAUI_NOINIT CDB API has not been initialized with cdb_init().
010:012 EOS_MAUI_BADPTR ret_buffer is set to NULL.
000:221 EOS_MNF No CDB modules found.
CDB Functions

**Indirect Errors**

    mem_malloc()
    cdb_get_ncopy()
    cdb_get_size()

**See Also**

    mem_free()
cdb_get_ddr()

Get Device Description By Type and Number

**Syntax**

```c
error_code
cdb_get_ddr(CDB_TYPE type, u_int32 num, char *name,
            char *param)
```

**Description**

cdb_get_ddr() searches for the device description record matching the device type and number of a device in the Configuration Description block (CDB).

*type* is an unsigned 32-bit integer representing the general class of device to which the device belongs.

*num* specifies the ordinal number of the device to return. For example, if *num = 1*, the name of the first device of the specified type is returned. If *num = 2*, the name of the second device of the specified type is returned.

*name* is a pointer to a character buffer that is at least CDB_MAX_DNAME bytes long. If *name* is not NULL, the device name is written into the buffer pointed to by *name*.

*param* is a pointer to a character buffer that is at least CDB_MAX_PARAM bytes long. If *param* is not NULL, the parameter string for device name is written into the buffer pointed to by *param*.

If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
Parameters

type
The general class of device.

num
The ordinal number of the device to return. For example, if num is 2, the device returned is the second device of the class.

*name
Points to a character buffer that is at least CDB_MAX_DNAME bytes long. If name is not NULL, the device name is written into the buffer pointed to by name.

*param
Points to a character buffer that is at least CDB_MAX_PARAM bytes long. If param is not NULL, the parameter string for the device name is written into the buffer pointed to by param.

Non-Fatal Errors

010:016 EOS_MAUI_BADVALUE
num is zero.

010:036 EOS_MAUI_NOINIT
CDB API has not been initialized with cdb_init().

010:044 EOS_MAUI_NOTFOUND
Entry with the specified type and num was not found.

000:221 EOS_MNF
No CDB modules found.

Indirect Errors

_os_chkmem()
=os_findpd()
cd_get_ncopy()
Get an N Byte Copy of the CDB

Syntax
error_code
cd_get_ncopy(char *buffer, size_t *size)

Description
cd_get_ncopy() copies the Configuration Description Block (CDB) into a caller supplied buffer. This call differs from cd_get_copy() in that it does not allocate memory that must be freed by the caller.

At most, size bytes are copied to buffer. If the call is successful, size is updated with the actual number of bytes copied including the NULL at the end (i.e. strlen()+1).

The CDB is formatted as a single NULL terminated string with each Device Description Record (DDR) separated by a carriage return (0x0D) character.

This function is new as of MAUI 3.1 and returns EOS_ITRAP on older systems.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*buffer Pointer to a *size byte buffer to copy the CDB.
*size Pointer to maximum size in bytes to copy, returns number of bytes copied.
Non-Fatal Errors

000:227 EOS_ITRAP
Function not supported by older shared libraries.

010:036 EOS_MAUI_NOINIT
CDB API has not been initialized with cdb_init().

010:012 EOS_MAUI_BADPTR
buffer or size is set to NULL.

000:221 EOS_MNF
No CDB modules found.

Indirect Errors

_os_chkmem()
_os_findpd()

See Also

cdb_get_copy()
cdb_get_size()
cdb_get_size()
Get Size of the CDB

Syntax

error_code
cdb_get_size(size_t *ret_size)

Description

cdb_get_size() returns the size of the Configuration Description Block (CDB) suitable for passing to cdb_get_ncopy().
The size is returned in ret_size. A pointer to a variable of type size_t should be passed to cdb_get_size().
This function is new as of MAUI 3.1 and returns EOS_ITRAP on older systems.
If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

*ret_size Pointer to returned size.

Non-Fatal Errors

000:227 EOS_ITRAP Function not supported by older shared libraries.
010:036 EOS_MAUI_NOINIT CDB API has not been initialized with cdb_init().
010:012 EOS_MAUI_BADPTR ret_size is set to NULL.
000:221 EOS_MNF No CDB modules found.
CDB Functions

**Indirect Errors**

`_os_findpd()`

**See Also**

`cdb_get_ncopy()`
**cdb_init()**

Initialize the CDB API

### Syntax

cdb_init(void)

### Description

cdb_init() initializes the Configuration Description Block (CDB) API. This function must be called prior to calling any other CDB function unless otherwise noted by that function.

Since this API depends on the Shaded Memory API, mem_init() is called by this function.

As of MAUI 3.1 this API also depends on the /mauidev device and mauidrvr driver.

If successful, this function returns **SUCCESS**.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

None

### Indirect Errors

- mem_init()
- _os_open()  
  To open the /mauidev device.

### See Also

cdb_term()
**Syntax**

```c
cdb_set_error_action(MAUI_ERR_LEVEL debug_level, 
                   MAUI_ERR_LEVEL passback_level, 
                   MAUI_ERR_LEVEL exit_level)
```

**Description**

This function sets the action taken in the error handler when a function in this API detects an error. This function may be called prior to calling `cdb_init()`. Following is the table of error levels. The least severe error is listed first.

**Table 4-1 Error Levels for cdb_set_error_action**

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation completed but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
</tbody>
</table>
#### Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

#### Parameters

- **debug_level**  
The minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

- **passback_level**  
The minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default passback level is MAUI_ERR_NON_FATAL.

- **exit_level**  
The minimum error level that causes the error handler to call `exit()`. In this case, the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

#### Direct Errors

None

#### See Also

- `cdb_init()`

---

### Table 4-1  Error Levels for cdb_set_error_action

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>

---

MAUI_ERR_DEFAULT  
Restore the level to its default value.
**Syntax**

error_code cdb_term(void)

**Description**

cdb_term() terminates use of the Configuration Description Block (CDB) API.

Since this API depends on the Shaded Memory API, mem_term() is called by this function.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

None

**Non-Fatal Errors**

010:036 EOS_MAUI_NOINIT  
CDB API has not been initialized with cdb_init().

**Indirect Errors**

mem_term()  
_os_close()  
To close the /mauidev device.

**See Also**

cdb_init()  
mem_term()
Chapter 5: Drawing Functions
**drw_arc()**

**Draw a Circular Arc**

**Syntax**

```c
error_code
drw_arc(DRW_CONTEXT_ID context, GFX_POS x, GFX_POS y,
       GFX_ANGLE start_angle, GFX_ANGLE end_angle,
       GFX_DIMEN radius)
```

**Description**

`drw_arc()` draws a circular arc using the specified drawing context. The arc is drawn with its center at the coordinate specified by `x` and `y`. It is drawn with the specified `radius`.

The arc is drawn in a counter-clockwise direction from `start_angle` to `end_angle`. These angles are in 64ths of a degree where 0 degrees is at 3 o’clock.

If the fill mode is `DRW_FM_SOLID` (see table 5-10), the arc is filled in pie-chart style.

If part or all of the arc does not fit on the destination drawmap, then that portion, or all of it, is clipped. The following table shows which parameters from the `context` object are used in this function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>File mode</td>
<td><code>drw_set_context_fm()</code></td>
</tr>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td><code>drw_set_context_trans()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>drw_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>drw_set_context_dst()</code></td>
</tr>
<tr>
<td>Origin</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
<tr>
<td>Drawing area</td>
<td><code>drw_set_context_draw()</code></td>
</tr>
<tr>
<td>Clipping areas</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
</tbody>
</table>

If successful, this function returns `SUCCESS`.

### Attributes

**Operating System:** OS-9 and OS-9 for 68K  
**State:** User  
**Threads:** Safe

### Table 5-1   Context Object Parameters Used in drw_arc() (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X screen coordinate of center of circle</td>
</tr>
<tr>
<td>y</td>
<td>Y screen coordinate of center of circle.</td>
</tr>
<tr>
<td>start_angle</td>
<td>Starting point of the arc expressed in 64ths of a degree. Zero degrees is at 3 o’clock.</td>
</tr>
<tr>
<td>end_angle</td>
<td>Ending point of the arc expressed in 64ths of a degree. Zero degrees is at 3 o’clock.</td>
</tr>
</tbody>
</table>
radius  

Radius of the arc in pixels.

**Non-Fatal Errors**

010:008 EOS_MAUI_BADID  
The ID specified by `context` is not valid.

010:016 EOS_MAUI_BADVALUE  
The radius is 0.

010:036 EOS_MAUI_NOINIT  
This API has not been initialized with `drw_init()`.

**Indirect Errors**

`blt_draw_hline()`  
`blt_draw_pixel()`  

**See Also**

`drw_create_context()`  
`DRW_CONTEXT_ID`  
`GFX_ANGLE`  
`GFX_DIMEN`  
`GFX_POS`
**drw_circle()**

*Draw a Circle*

**Syntax**

```c
error_code drw_circle(DRW_CONTEXT_ID context, GFX_POS x, GFX_POS y, GFX_DIMEN radius)
```

**Description**

`drw_circle()` draws a circle using the specified drawing context. The circle is drawn with its center at the coordinate specified by `x` and `y`, and it is drawn with the specified `radius`.

If part (or all) of the circle does not fit on the destination drawmap, then that portion (or all of it) is clipped.

The following table shows which parameters from the context object are used in this function.

**Table 5-2  Context Parameters Used in drw_circle()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill mode</td>
<td><code>drw_set_contxt_fm()</code></td>
</tr>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td><code>drw_set_context_trans()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset pixel value</td>
<td><code>drw_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>drw_set_context_dst()</code></td>
</tr>
<tr>
<td>Origin</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
<tr>
<td>Drawing area</td>
<td><code>drw_set_context_draw()</code></td>
</tr>
<tr>
<td>Clipping areas</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
</tbody>
</table>

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**  
  The ID specified by `context` is not valid.
- **010:016 EOS_MAUI_BADVALUE**  
  The radius is 0.
- **010:036 EOS_MAUI_NOINIT**  
  This API has not been initialized with `drw_init()`.

---

**Table 5-2  Context Parameters Used in `drw_circle()` (continued)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset pixel value</td>
<td><code>drw_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>drw_set_context_dst()</code></td>
</tr>
<tr>
<td>Origin</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
<tr>
<td>Drawing area</td>
<td><code>drw_set_context_draw()</code></td>
</tr>
<tr>
<td>Clipping areas</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
</tbody>
</table>
Drawing Functions

**Indirect Errors**

blt_draw_hline()
blt_draw_pixel()

**See Also**

drw_create_context()
DRW_CONTEXT_ID
GFX_DIMEN
GFX_POS
drw_copy_block()
Copy a Block of Pixels

**Syntax**

```c
error_code drw_copy_block(DRW_CONTEXT_ID context,
GFX_POS dstx, GFX_POS dsty,
GFX_POS srcx, GFX_POS srcy,
GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`drw_copy_block()` copies a rectangular area of pixels using the drawing context specified by `context`. The upper-left corner of this area is specified by `srcx` and `srcy` in the source drawmap and `dstx` and `dsty` in the destination drawmap. The size of the area to be copied is specified by `width` and `height`.

The source and destination areas should not overlap. If they do, you may observe undesired side effects because the source data may be over-written before it is used.

If part (or all) of the rectangle does not fit on the source drawmap or destination drawmap, then that portion (or all of it) is clipped. The following table shows which parameters from the `context` object are used in this function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode for copying</td>
<td><code>drw_set_context_cpymix()</code></td>
</tr>
<tr>
<td>Source drawmap</td>
<td><code>drw_set_context_src()</code></td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td><code>drw_set_context_trans()</code></td>
</tr>
</tbody>
</table>
Table 5-3  Context Parameters Used in drw_copy_block() (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask drawmap</td>
<td>drw_set_context_mask().</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td>drw_set_context_ofs().</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>drw_set_context_dst().</td>
</tr>
<tr>
<td>Origin</td>
<td>drw_set_context_origin().</td>
</tr>
<tr>
<td>Drawing area</td>
<td>drw_set_context_draw().</td>
</tr>
<tr>
<td>Clipping areas</td>
<td>drw_set_context_clip().</td>
</tr>
</tbody>
</table>

If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context**
  - Drawing context ID.
- **dstx**
  - $X$ coordinate of the upper-left corner of the destination drawmap
- **dsty**
  - $Y$ coordinate of the upper-left corner of the destination drawmap
- **srcx**
  - $X$ coordinate of the upper-left corner of the source drawmap
- **srcy**
  - $Y$ coordinate of the upper-left corner of the source drawmap
- **width**
  - Width of the area to be copied.
- **height**
  - Height of the area to be copied.
5 Drawing Functions

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by context is invalid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with drw_init().

Indirect Errors

blt_copy_block()

See Also

drw_create_context()
DRW_CONTEXT_ID
GFX_DIMEN
GFX_POS
**drw_copy_oblock()**

Copy Overlapping Blocks of Pixels

**Syntax**

```c
error_code drw_copy_oblock(DRW_CONTEXT_ID context,
GFX_POS dstx, GFX_POS dsty,
GFX_POS srcx, GFX_POS srcy,
GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

drw_copy_oblock() copies a rectangular area of pixels using the drawing context specified by context. The upper-left corner of this area is specified by srcx and srcy in the source drawmap and dstx and dsty in the destination drawmap. The size of the area to be copied is specified by width and height.

The source and destination areas can overlap. This function is not as efficient as drw_copy_block() because it has to check for overlapping copy areas and compensate for them.

If part (or all) of the rectangle does not fit on the source drawmap or destination drawmap, then that portion (or all of it) is clipped. The following table shows which parameters from the context object are used in this function.

**Table 5-4  Context Parameters Used in drw_copy_block()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode for copying</td>
<td>drw_set_context_cpymix().</td>
</tr>
<tr>
<td>Source drawmap</td>
<td>drw_set_context_src().</td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td>drw_set_context_trans().</td>
</tr>
</tbody>
</table>
Table 5-4  Context Parameters Used in drw_copy_block() (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask drawmap</td>
<td>drw_set_context_mask().</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td>drw_set_context_ofs().</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>drw_set_context_dst().</td>
</tr>
<tr>
<td>Origin</td>
<td>drw_set_context_origin().</td>
</tr>
<tr>
<td>Drawing area</td>
<td>drw_set_context_draw().</td>
</tr>
<tr>
<td>Clipping areas</td>
<td>drw_set_context_clip().</td>
</tr>
</tbody>
</table>

If successful, this function returns SUCCESS.

Attributes

Operating System:          OS-9 and OS-9 for 68K
State:                     User
Threads:                   Safe

Parameters

context                    Drawing context ID.
dstx                        X coordinate of the upper-left corner of the destination drawmap
dsty                        Y coordinate of the upper-left corner of the destination drawmap
srcx                        X coordinate of the upper-left corner of the source drawmap.
sr cy                       Y coordinate of the upper-left corner of the source drawmap.
width                       Width of the area to be copied.
height                      Height of the area to be copied.
Non-Fatal Errors

010:008  EOS_MAUI_BADID
The ID specified by context is invalid.

010:036  EOS_MAUI_NOINIT
This API has not been initialized with drw_init().

010:045  EOS_MAUI_NOTIMPLEMENTED
A mixing mode other than BLT_MIX_REPLACE was specified for a fully overlapping copy.

Indirect Errors

drw_copy_block()

See Also

drw_copy_block()
DRW_CONTEXT_ID
GFX_DIMEN
GFX_POS
**drw_create_context()**

Create a Drawing Context

**Syntax**

```c
error_code drw_create_context(DRW_CONTEXT_ID *ret_context,
                             GFX_DEV_ID gfxdev)
```

**Description**

`drw_create_context()` creates a new drawing context object. This object is used in all subsequent drawing functions. The following table shows the default value for each parameter and the functions for modifying them.

**Table 5-5  Default Value and Modifications for Drawing Context**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Values</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill mode</td>
<td>DRW_FM_OUTLINE</td>
<td>drw_set_context_fm()</td>
</tr>
<tr>
<td>Line style</td>
<td>DRW_LS_SOLID</td>
<td>drw_set_context_ls()</td>
</tr>
<tr>
<td>Dash pattern</td>
<td>0x55555555</td>
<td>drw_set_context_dash()</td>
</tr>
<tr>
<td>Dash magnification</td>
<td>4</td>
<td>drw_set_context_dash()</td>
</tr>
<tr>
<td>Mixing mode for copy</td>
<td>BLT_MIX_REPLACE</td>
<td>drw_set_context_c pymix()</td>
</tr>
<tr>
<td>Mixing mode for expand</td>
<td>BLT_MIX_REPLACE</td>
<td>drw_set_context_expmix()</td>
</tr>
<tr>
<td>Mixing mode for draw</td>
<td>BLT_MIX_REPLACE</td>
<td>drw_set_context_mix()</td>
</tr>
</tbody>
</table>
### Table 5-5  Default Value and Modifications for Drawing Context (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Values</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel value for drawing</td>
<td>1</td>
<td>drw_set_context_pix()</td>
</tr>
<tr>
<td>Pixel expansion table</td>
<td>NULL</td>
<td>drw_set_context_epxtbl()</td>
</tr>
<tr>
<td>Source drawmap</td>
<td>NULL</td>
<td>drw_set_context_src()</td>
</tr>
<tr>
<td>Entries in expansion table</td>
<td>0</td>
<td>drw_set_context_exptbl()</td>
</tr>
<tr>
<td>Pixel expansion table</td>
<td>NULL</td>
<td>drw_set_context_exptbl()</td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td>0</td>
<td>drw_set_context_trans()</td>
</tr>
<tr>
<td>Mask drawmap</td>
<td>NULL</td>
<td>drw_set_context_mask()</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td>0</td>
<td>drw_set_context_ofs()</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>NULL</td>
<td>drw_set_context_dst()</td>
</tr>
<tr>
<td>Drawing area</td>
<td>x=0, y=0</td>
<td>drw_set_context_drw()</td>
</tr>
<tr>
<td></td>
<td>w=GFX_DIMEN_MAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>h=GFX_DIMEN_MAX</td>
<td></td>
</tr>
</tbody>
</table>
The context ID is returned in `ret_context`. A pointer to this variable should be passed to `drw_create_context()`. Use `drw_destroy_context()` to destroy this object when it is no longer needed. Use `drw_get_context()` to get the current settings in a context.

If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `*ret_context` Pointer to context ID.
- `gfxdev` Graphics device ID.

### Fatal Errors

- **010:036 EOS_MAUI_NOINIT** This API has not been initialized with `drw_init()`.

#### Table 5-5  Default Value and Modifications for Drawing Context (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Values</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>0, 0</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
<tr>
<td>Number of clipping areas</td>
<td>0</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
<tr>
<td>Clipping areas</td>
<td>NULL</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
</tbody>
</table>
Indirect Errors

blt_create_context()
drw_set_context_clip()
drw_set_context_cpmix()
drw_set_context_dash()
drw_set_context_draw()
drw_set_context_dst()
drw_set_context_expmix()
drw_set_context_exptbl()
drw_set_context_fm()
drw_set_context_ls()
drw_set_context_mask()
drw_set_context_mix()
drw_set_context_ofs()
drw_set_context_origin()
drw_set_context_pix()
drw_set_context_src()
drw_set_context_trans()
mem_calloc()

See Also

drw_destroy_context()
DRW_CONTEXT_ID
**drw_destroy_context()**
Destroy a Drawing Context

**Syntax**
```
error_code drw_destroy_context(DRW_CONTEXT_ID context)
```

**Description**
`drw_destroy_context()` destroys the specified drawing context object `context`.
You should always destroy all `drw` and `txt` contexts associated with a `win` device before destroying the `win` device. Attempting to destroy one of these contexts after destroying the `win` device can result in an access to deallocated memory. On SSM systems, this can cause a bus trap.

If successful, this function returns `SUCCESS`.

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**
- `context` Drawing context ID.

**Non-Fatal**
- **010:008 EOS_MAUI_BADID** The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT** This API has not been initialized with `drw_init()`.

**Indirect Errors**
- `blt_destroy_context()`
- `mem_free()`
See Also

drw_create_context()
DRW_CONTEXT_ID
**drw_earc()**

**Draw an Elliptical Arc**

**Syntax**

```c
error_code drw_earc(DRW_CONTEXT_ID context,
                    GFX_POS x, GFX_POS y, GFX_ANGLE start_angle,
                    GFX_ANGLE end_angle, GFX_DIMEN xradius,
                    GFX_DIMEN yradius)
```

**Description**

`drw_earc()` draws an elliptical arc using the specified drawing context. The arc is drawn with its center at the coordinate specified by \( x \) and \( y \). It is drawn with the specified \( xradius \) and \( yradius \).

The arc is drawn in a counter-clockwise direction from \( start\_angle \) to \( end\_angle \). These angles are in 64ths of a degree where 0 degrees is at 3 o’clock.

If the fill mode is DRW_FM_SOLID (see table 5-10), the arc is filled in pie-chart style.

If part or all of the arc does not fit on the destination drawmap, then that portion, or all of it, is clipped. The following table shows which parameters from the context object are used in this function.

**Table 5-6   Context Object Parameters Used in drw_earc()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context**
  - Drawing context ID.
- **x**
  - X screen coordinate of center of ellipse.
- **y**
  - Y screen coordinate of center of ellipse.
- **start_angle**
  - Starting point of the arc expressed in 64ths of a degree. Zero degrees is at 3 o’clock.
- **end_angle**
  - Ending point of the arc expressed in 64ths of a degree. Zero degrees is at 3 o’clock.
- **xradius**
  - X radius of the arc in pixels.
yradius

Y radius of the arc in pixels.

**Non-Fatal Errors**

010:008 EOS_MAUI_BADID

The ID specified by context is not valid.

010:016 EOS_MAUI_BADVALUE

The xradius or yradius is 0.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with drw_init().

**Indirect Errors**

blt_draw_hline()
blt_draw_pixel()

**See Also**

drw_create_context()
DRW_CONTEXT_ID
GFX_ANGLE
GFX_DIMEN
GFX_POS
**drw_ellipse()**

**Draw an Ellipse**

**Syntax**

```
error_code drw_ellipse(DRW_CONTEXT_ID context,
                       GFX_POS x, GFX_POS y,
                       GFX_DIMEN xradius, GFX_DIMEN yradius)
```

**Description**

`drw_ellipse()` draws an ellipse using the specified drawing context. The ellipse is drawn with its center at the coordinate specified by `x` and `y`. The radius along the X and Y axis are specified by `xradius` and `yradius` respectfully.

If part or all of the ellipse does not fit on the destination drawmap, then that portion, or all of it, is clipped. The following table shows which parameters from the context object are used in this function.

**Table 5-7  Context Object Parameters Used in drw_ellipse()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill mode</td>
<td><code>drw_set_context_fm()</code></td>
</tr>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td><code>drw_set_context_trans()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context**: Drawing context ID.
- **x**: X screen coordinate of center of ellipse.
- **y**: Y screen coordinate of center of ellipse.
- **xradius**: X radius of the ellipse in pixels.
- **yradius**: Y radius of the ellipse in pixels.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by context is not valid.
- **010:016 EOS_MAUI_BADVALUE**: The xradius or yradius is 0.
This API has not been initialized with `drw_init()`.

**Indirect Errors**
- `blt_draw_hline()`
- `blt_draw_pixel()`

**See Also**
- `drw_create_context()`
- `DRW_CONTEXT_ID`
- `GFX_ANGLE`
- `GFX_DIMEN`
- `GFX_POS`
drw_expd_block()
Expand a Block of Pixels

Syntax

```c
error_code drw_expd_block(DRW_CONTEXT_ID context,
                           GFX_POS dstx, GFX_POS dsty,
                           GFX_POS srcx, GFX_POS srcy,
                           GFX_DIMEN width, GFX_DIMEN height)
```

Description

`drw_expd_block()` copies a rectangular area of pixels using the drawing context specified by `context`. This function is similar to `drw_copy_block()` except that it expands the pixels as it copies them. See `blt_expd_block()` for restrictions on the depth of the source and destination drawmaps.

The upper-left corner of this area is specified by `srcx` and `srcy` in the source drawmap and `dstx` and `dsty` in the destination drawmap. The size of the area to be copied is specified by `width` and `height`.

If part (or all) of the rectangle does not fit on the source drawmap or destination drawmap, then that portion (or all of it) is clipped. The following table shows which parameters from the `context` object are used in this function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode for expanding</td>
<td><code>drw_set_context_expmix()</code></td>
</tr>
<tr>
<td>Source drawmap</td>
<td><code>drw_set_context_src()</code></td>
</tr>
<tr>
<td>Expansion table</td>
<td><code>drw_set_context_exptbl()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `context` : Drawing context ID.
- `dstx` : \( x \) coordinate of the upper-left corner of the destination drawmap.
- `dsty` : \( y \) coordinate of the upper-left corner of the destination drawmap.
- `srcx` : \( x \) coordinate of the upper-left corner of the source drawmap.
- `srcy` : \( y \) coordinate of the upper-left corner of the source drawmap.

**Table 5-8  Context Object Parameters Used in `drw_expd_block()`**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent pixel value</td>
<td><code>drw_set_context_trans()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>drw_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>drw_set_context_dst()</code></td>
</tr>
<tr>
<td>Origin</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
<tr>
<td>Drawing area</td>
<td><code>drw_set_context_draw()</code></td>
</tr>
<tr>
<td>Clipping areas</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
</tbody>
</table>
width  \hspace{1em} \text{Width of the area to be copied.}
height  \hspace{1em} \text{Height of the area to be copied.}

**Non-Fatal Errors**

010:008  EOS_MAUI_BADID  \hspace{1em} \text{The ID specified by context is invalid.}

010:036  EOS_MAUI_NOINIT  \hspace{1em} \text{This API has not been initialized with drw_init().}

**Indirect Errors**

blt_expd_block()

**See Also**

drw_create_context()
DRW_CONTEXT_ID
GFX_DIMEN
GFX_POS
**drw_get_context()**

Get Drawing Context Parameters

**Syntax**

```c
error_code drw_get_context(DRW_CONTEXT_PARAMS *ret_context_params, DRW_CONTEXT_ID context)
```

**Description**

drw_get_context() returns the current parameters for the specified drawing context. The parameters are returned in ret_context_params. A pointer to this variable should be passed to drw_get_context(). The caller must ensure that ret_context_params points to storage large enough to hold the information.

If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `*ret_context_params` Pointer to parameters.
- `context` Drawing context ID.

**Non-Fatal Errors**

- **010:008 EOS_MAUÍ_BADID** The ID specified by context is not valid.
- **010:036 EOS_MAUÍ_NOINIT** This API has not been initialized with drw_init().
See Also

drw_create_context()
drw_set_context_clip()
drw_set_context_c pymix()
drw_set_context_dash()
drw_set_context_draw()
drw_set_context_dst()
drw_set_context_expmix()
drw_set_context_exptbl()
drw_set_context_fm()
drw_set_context_ls()
drw_set_context_mask()
drw_set_context_mix()
drw_set_context_ofs()
drw_set_context_origin()
drw_set_context_pix()
drw_set_context_src()
drw_set_context_trans()
DRW_CONTEXT_ID
DRW_CONTEXT_PARAMS
drw_init()
Initialize the Drawing API

Syntax
error_code
drw_init(void)

Description
drw_init() initializes the Drawing API. This function must be called prior to a call to any other drawing function unless otherwise noted by that function.

This API depends on the Shaded Memory and Bit-BLT APIs. Therefore, mem_init() and blt_init() are called by this function.
If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
None

Indirect Errors
blt_init()
mem_init()

See Also
drw_term()
**drw_line()**  
Draw a Line

**Syntax**

```c
error_code drw_line(DRW_CONTEXT_ID context, GFX_POS sx,
                     GFX_POS sy, GFX_POS ex, GFX_POS ey)
```

**Description**

`drw_line()` draws a line using the specified drawing context. The start point for the line is at `sx, sy` and the end point is at `ex, ey`.

The last pixel of the line `(ex, ey)` is not drawn. If the start point and the end point are the same, then no point is drawn.

The algorithm used to compute the points on the line will select the same pixels regardless of the direction it is drawn. For example, if you draw a solid line from point A to point B, the algorithm will pick the same pixels it would pick for drawing the line from point B to point A.

If part (or all) of the line does not fit on the destination drawmap, then that portion (or all of it) is clipped.

The following table shows which parameters from the context object are used in this function.

**Table 5-9  Context Parameters Used in drw_line()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

- **context**: Drawing context ID.
- **sx**: X coordinate of start point for line.
- **sy**: Y coordinate of start point for line.
- **ex**: X coordinate of end point for line.
- **ey**: Y coordinate of end point for line.

### Table 5-9  Context Parameters Used in drw_line() (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent pixel value</td>
<td>drw_set_context_trans().</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td>drw_set_context_ofs().</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>drw_set_context_dst().</td>
</tr>
<tr>
<td>Origin</td>
<td>drw_set_context_origin().</td>
</tr>
<tr>
<td>Drawing area</td>
<td>drw_set_context_draw().</td>
</tr>
<tr>
<td>Clipping areas</td>
<td>drw_set_context_clip().</td>
</tr>
</tbody>
</table>

Transparent pixel value: drw_set_context_trans().
Offset pixel value: drw_set_context_ofs().
Destination drawmap: drw_set_context_dst().
Origin: drw_set_context_origin().
Drawing area: drw_set_context_draw().
Clipping areas: drw_set_context_clip().
5 Drawing Functions

Non-Fatal Errors

010:008 EOS_MAUI_BADID

The ID specified by context is not valid.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with drw_init().

Indirect Errors

blt_draw_hline()
blt_draw_pixel()
blt_draw_vline()

See Also

drw_create_context()
DRW_CONTEXT_ID
GFX_POS


**Syntax**

```c
error_code
drw_oval(DRW_CONTEXT_ID context,
    GFX_POS x, GFX_POS y,
    GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

drw_oval() draws an oval using the specified drawing context. The parameters `x` and `y` are the coordinates of the top left corner of the bounding box. The bounding box is of size `width` by `height`. The oval is drawn within the bounding box.

If part or all of the oval does not fit on the destination drawmap, then that portion, or all of it, is clipped. The following table shows which parameters from the context object are used in this function.

**Table 5-10  Context Object Parameters Used in drw_oval()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill mode</td>
<td>drw_set_context_fm().</td>
</tr>
<tr>
<td>Line style</td>
<td>drw_set_context_ls().</td>
</tr>
<tr>
<td>Dash pattern</td>
<td>drw_set_context_dash().</td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td>drw_set_context_mix().</td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td>drw_set_context_pix().</td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td>drw_set_context_trans().</td>
</tr>
</tbody>
</table>
If successful, this function returns `SUCCESS`.

**Attributes**  
Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**  
- **context**: Drawing context ID.
- **x**: X screen coordinate of top left corner of the bounding box of the oval.
- **y**: Y screen coordinate of top left corner of the bounding box of the oval.
- **width**: Width of the bounding box of the oval in pixels.
- **height**: Height of the bounding box of the oval in pixels.

**Non-Fatal Errors**  
- **010:008 EOS_MAUI_BADID**: The ID specified by `context` is not valid.

---

**Table 5-10  Context Object Parameters Used in `drw_oval()` (continued)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset pixel value</td>
<td><code>drw_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>drw_set_context_dst()</code></td>
</tr>
<tr>
<td>Origin</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
<tr>
<td>Drawing area</td>
<td><code>drw_set_context_draw()</code></td>
</tr>
<tr>
<td>Clipping areas</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
</tbody>
</table>
Drawing Functions

010:016 EOS_MAUI_BADVALUE
010:036 EOS_MAUI_NOINIT

The width or height is 0.
This API has not been initialized with drw_init().

Indirect Errors
blt_draw_hline()
blt_draw_pixel()

See Also
drw_create_context()
DRW_CONTEXT_ID
GFX_DIMEN
GFX_POS
5  Drawing Functions

**drw_oval_arc()**

**Draw an Oval Arc**

**Syntax**

```c
error_code drw_oval_arc(DRW_CONTEXT_ID context,
                        GFX_POS x,
                        GFX_POS y,
                        GFX_ANGLE start_angle,
                        GFX_ANGLE end_angle,
                        GFX_DIMEN width,
                        GFX_DIMEN height)
```

**Description**

`drw_oval_arc()` draws an oval arc using the specified drawing context. The parameters `x` and `y` are the coordinates of the top left corner of the bounding box of size `width` by `height`. The arc is drawn within the bounding box.

The arc is drawn in a counter-clockwise direction from `start_angle` to `end_angle`. These angles are in 64ths of a degree where 0 degrees is at 3 o’clock. The arc is drawn counter-clockwise.

If the fill mode is `DRW_FM_SOLID` (see table 5-10), the arc is filled in pie-chart style.

If part or all of the arc does not fit on the destination drawmap, then that portion, or all of it, is clipped. The following table shows which parameters from the context object are used in this function.

**Table 5-11  Context Object Parameters Used in drw_oval_arc()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill mode</td>
<td><code>drw_set_context_fm()</code></td>
</tr>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns `SUCCESS`.

**Attributes**

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Drawing context ID.</td>
</tr>
<tr>
<td>x</td>
<td>X screen coordinate of top left corner of the bounding box of the arc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5-11</th>
<th>Context Object Parameters Used in <code>drw_oval_arc()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Modify With</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td><code>drw_set_context_trans()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>drw_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>drw_set_context_dst()</code></td>
</tr>
<tr>
<td>Origin</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
<tr>
<td>Drawing area</td>
<td><code>drw_set_context_draw()</code></td>
</tr>
<tr>
<td>Clipping areas</td>
<td><code>drw_set_context_clip()</code></td>
</tr>
</tbody>
</table>
5 Drawing Functions

\( y \)
\( \text{y screen coordinate of top left corner of the bounding box of the arc.} \)

start_angle
\( \text{Starting point of the arc expressed in 64ths of a degree. Zero degrees is at 3 o'clock.} \)

end_angle
\( \text{Ending point of the arc expressed in 64ths of a degree. Zero degrees is at 3 o'clock.} \)

width
\( \text{Width of the bounding box of the arc in pixels.} \)

height
\( \text{Height of the bounding box of the arc in pixels.} \)

**Non-Fatal Errors**

010:008 EOS_MAUI_BADID
\( \text{The ID specified by context is not valid.} \)

010:016 EOS_MAUI_BADVALUE
\( \text{The width or height is 0.} \)

010:036 EOS_MAUI_NOINIT
\( \text{This API has not been initialized with drw_init().} \)

**Indirect Errors**

blt_draw_hline()
blt_draw_pixel()

**See Also**

drw_create_context()
DRW_CONTEXT_ID
GFX_ANGLE
GFX_DIMEN
GFX_POS
**drw_point()**

**Draw a Point**

**Syntax**

```c
error_code drw_point(DRW_CONTEXT_ID context, GFX_POS x, GFX_POS y)
```

**Description**

`drw_point()` draws a point using the specified drawing context. The point is drawn at the coordinate specified by `x` and `y`.

If the point is beyond the edge of the destination drawmap, then it is clipped.

The following table shows which parameters from the context object are used in this function.

**Table 5-12  Context Parameters Used in drw_point()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td><code>drw_set_context_trans()</code></td>
</tr>
<tr>
<td>Offset pixel value</td>
<td><code>drw_set_context_ofs()</code></td>
</tr>
<tr>
<td>Destination drawmap</td>
<td><code>drw_set_context_dst()</code></td>
</tr>
<tr>
<td>Origin</td>
<td><code>drw_set_context_origin()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- **context**
  - Drawing context ID.
- **x**
  - X coordinate of point.
- **y**
  - Y coordinate of point.

### Non-Fatal Errors

- 010:008 EOS_MAUI_BADID: The ID specified by `context` is not valid.
- 010:036 EOS_MAUI_NOINIT: This API has not been initialized with `drw_init()`.

### Indirect Errors

- `blt_draw_pixel()`

### See Also

- `drw_create_context()`
- `DRW_CONTEXT_ID`
- `GFX_POS`
**draw_polygon()**  
Draw a Polygon

**Syntax**
```c
error_code draw_polygon(DRW_CONTEXT_ID context,
                      u_int32 num_vertices,
                      const GFX_POINT vertex[])
```

**Description**

`draw_polygon()` draws a polygon using the specified drawing context. `draw_polygon()` produces a closed polygon by automatically connecting the last entry in the vertex array to the first.

If the fill mode is `DRW_FM_SOLID` the polygon is filled.

Each entry in the `vertex` array specifies the `x` and `y` coordinates for the vertex. The number of such vertices is indicated by `num_vertices`.

If part (or all) of the polygon does not fit on the destination drawmap, then that portion (or all of it) is clipped. The following table shows which parameters from the `context` object are used in this function.

**Table 5-13  Context Parameters Used in `draw_polygon()`**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill mode</td>
<td><code>draw_set_context_fm()</code></td>
</tr>
<tr>
<td>Line style</td>
<td><code>draw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>draw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>draw_set_context_mix()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns **SUCCESS**.

### Attributes

**Operating System:** OS-9 and OS-9 for 68K  
**State:** User  
**Threads:** Safe

### Parameters

- **context**: Drawing context ID.  
- **num_vertices**: Number of vertices in polygon.  
- **vertex[]**: Array of X and Y coordinates for each vertex.

### Non-Fatal Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 010:008  | **EOS_MAUI_BADID**  
The ID specified by context is not valid. |
010:016  EOS_MAUI_BADVALUE
010:036  EOS_MAUI_NOINIT

This API has not been initialized with `drw_init()`. num_vertices is 0.

Indirect Errors

- `blt_draw_hline()`
- `blt_draw_pixel()`
- `blt_draw_vline()`
- `mem_calloc()`
- `mem_free()`

See Also

- `drw_create_context()`
- `drw_polyline()`
- `DRW_CONTEXT_ID`
- `GFX_POINT`
### drw_polyline()

#### Draw a Polyline

**Syntax**

```c
error_code drw_polyline(DRW_CONTEXT_ID context, u_int32 num_vertices, const GFX_POINT vertex[])
```

**Description**

`drw_polyline()` draws a polyline using the specified drawing context. Each entry in the `vertex` array specifies the `x` and `y` coordinates for the vertex. The number of such vertices is indicated by `num_vertices`. The last pixel of the last line segment is not drawn. `drw_polyline()` does not produce a closed polygon automatically.

If the fill mode is `DRW_FM_SOLID` it is ignored.

If part (or all) of the polyline does not fit on the destination drawmap, then that portion (or all of it) is clipped.

The following table shows which parameters from the `context` object are used in this function.

**Table 5-14  Context Parameters Used in drw_polyline()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
</tbody>
</table>
If successful, this function returns SUCCESS.

**Attributes**

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**Parameters**

- **context**: Drawing context ID.
- **num_vertices**: Number of vertices in the polyline.
- **vertex[]**: Array of x and y coordinates for each vertex.

**Non-Fatal Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:008</td>
<td>EOS_MAUI_BADID</td>
</tr>
<tr>
<td>010:016</td>
<td>EOS_MAUI_BADVALUE</td>
</tr>
<tr>
<td>010:036</td>
<td>EOS_MAUI_NOINIT</td>
</tr>
</tbody>
</table>
Indirect Errors
blt_draw_hline()
blt_draw_pixel()
blt_draw_vline()

See Also
drw_create_context()
drw_polygon()
DRW_CONTEXT_ID
GFX_POINT
Drawing Functions

**drw_rectangle()**

**Draw a Rectangle**

**Syntax**

```c
error_code drw_rectangle(DRW_CONTEXT_ID context,
                          GFX_POS x, GFX_POS y,
                          GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`drw_rectangle()` draws a rectangle using the specified drawing context. The upper-left corner of the rectangle is drawn at the coordinate specified by `x` and `y`. The `width` and `height` specify the size of the rectangle in pixels.

If part (or all) of the rectangle does not fit on the destination drawmap, then that portion (or all of it) is clipped.

The following table shows which parameters from the context object are used in this function.

**Table 5-15  Context Parameters Used in drw_rectangle()**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill mode</td>
<td><code>drw_set_context_fm()</code></td>
</tr>
<tr>
<td>Line style</td>
<td><code>drw_set_context_ls()</code></td>
</tr>
<tr>
<td>Dash pattern</td>
<td><code>drw_set_context_dash()</code></td>
</tr>
<tr>
<td>Mixing mode for drawing</td>
<td><code>drw_set_context_mix()</code></td>
</tr>
<tr>
<td>Pixel value to draw</td>
<td><code>drw_set_context_pix()</code></td>
</tr>
</tbody>
</table>
Table 5-15 Context Parameters Used in drw_rectangle() (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent pixel value</td>
<td>drw_set_context_trans().</td>
</tr>
<tr>
<td>Offset pixel value</td>
<td>drw_set_context_ofs().</td>
</tr>
<tr>
<td>Destination drawmap</td>
<td>drw_set_context_dst().</td>
</tr>
<tr>
<td>Origin</td>
<td>drw_set_context_origin().</td>
</tr>
<tr>
<td>Drawing area</td>
<td>drw_set_context_draw().</td>
</tr>
<tr>
<td>Clipping areas</td>
<td>drw_set_context_clip().</td>
</tr>
</tbody>
</table>

If successful, this function returns SUCCESS.

**Attributes**
- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**
- **context**
  - Drawing context ID.
- **x**
  - X coordinate of upper-left corner of rectangle.
- **y**
  - Y coordinates of upper-left corner of rectangle.
- **width**
  - Width of rectangle in pixels.
- **height**
  - Height of rectangle in pixels.
**Drawing Functions**

**Non-Fatal**

010:008 EOS_MAUI_BADID

The ID specified by `context` is not valid.

010:016 EOS_MAUI_BADVALUE

The width or height is 0.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with `drw_init()`.

**Indirect Errors**

- blt_draw_block()
- blt_draw_hline()
- blt_draw_pixel()
- blt_draw_vline()

**See Also**

- `drw_create_context()`
- `DRW_CONTEXT_ID`
- `GFX_DIMEN`
- `GFX_POS`
**drw_set_context_clip()**

**Set Clipping Area**

**Syntax**

```c
error_code drw_set_context_clip(DRW_CONTEXT_ID context,
                              u_int32 num_clip_areas,
                              const GFX_RECT clip_areas[])
```

**Description**

`drw_set_context_clip()` sets the clipping area for the specified drawing context to `clip_areas`.  

`clip_areas` is an array of `num_clip_areas` rectangles. These rectangles may overlap. Together, these rectangles define an area known as a clipping area. Because drawing is performed by functions using this context, pixels within this area are automatically clipped (not drawn).

If `clip_areas` is NULL or `num_clip_areas` is 0, then no clipping area is defined. In this case, drawing is clipped only if it is outside the bounds of the destination drawmap.  

In addition to the clipping defined above, all drawing outside the drawing area is clipped. See `drw_set_context_draw()` for information on setting the drawing area.

If successful, this function returns `SUCCESS`.

**Note**

Do not use this function if you are currently using the Windowing API.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
Parameters

context
Drawing context ID.

num_clip_areas
Specifies number of records defined in array.

clip_areas[]
An array.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by context is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with drw_init().

See Also

drw_create_context()
drw_get_context()
drw_set_context_draw()
GFX_RECT
DRW_CONTEXT_ID
5 Drawing Functions

**drw_set_context_cpymix()**

Set Mixing Mode for Copying

**Syntax**

```c
error_code drw_set_context_cpymix(DRW_CONTEXT_ID context,
                                BLT_MIX mixmode)
```

**Description**

`drw_set_context_cpymix()` sets the mixing mode for copy operations in the specified drawing context to `mixmode`. The mixing mode specifies the way source pixels are transferred to the destination drawmap.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- **context**: The drawing context ID.
- **mixmode**: The mixing mode for the copy operation.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `drw_init()`.

**Indirect Errors**

- `blt_set_context_cpymix()`
See Also

drw_create_context()
drw_get_context()
BLT_MIX
DRW_CONTEXT_ID
5 Drawing Functions

**drw_set_context_dash()**
Set Dash Pattern

**Syntax**
```
error_code drw_set_context_dash(DRW_CONTEXT_ID context,
                                u_int32 dash_pattern,
                                u_int16 dash_magnify)
```

**Description**

*drw_set_context_dash()* sets the parameters in the drawing context that affect dashed outlines. These are only used by the drawing functions when the fill mode is set to DRW_FM_OUTLINE and the line style is set to DRW_LS_DASHED.

*dash_pattern* specifies the pattern for dashed lines. One bit corresponds to one pixel. If the bit is set, then the pixel is drawn using the value set by *drw_set_context_pix()* If the bit is clear, no pixel is drawn.

*dash_magnify* specifies the magnification that should be applied to the dash pattern. A value of 1 means no magnification. A value of 2 means that each bit in *dash_pattern* corresponds to 2 pixels, and so forth.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

*context* Drawing context ID.
*dash_pattern* Specifies drawing pattern for dashed lines.
Drawing Functions

**dash_magnify**

Specifies magnification for dashed lines. (1=1X, 2=2X, etc.)

**Non-Fatal Errors**

010:008 EOS_MAUI_BADID

The ID specified by `context` is not valid.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with `drw_init()`.

**See Also**

`drw_create_context()`
`drw_get_context()`
`drw_set_context_pix()`
`DRW_CONTEXT_ID`
5 Drawing Functions

**drw_set_context_draw()**

*Set Drawing Area*

**Syntax**

```c
error_code
drw_set_context_draw(DRW_CONTEXT_ID context,
                      GFX_POS x,
                      GFX_POS y,
                      GFX_DIMEN width,
                      GFX_DIMEN height)
```

**Description**

`drw_set_context_draw()` sets the drawing area for the specified drawing context. All drawing outside this rectangle is clipped.

The upper-left corner of the rectangle is defined by `x` and `y`. The `width` and `height` define the size of the rectangle. Use 0 for `x` and `y`, and `GFX_DIMEN_MAX` for `width` and `height` to make the entire drawmap drawable.

If successful, this function returns `SUCCESS`.

**Note**

Do not use this function if you are currently using the Windowing API.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `context` - The drawing context ID.
- `x` - `x` coordinate of the upper-left corner of the drawing area.
Drawing Functions

y

\( y \) coordinate of the upper-left corner of the drawing area.

width

The width of the drawing area in pixels.

height

The height of the drawing area in pixels.

Non-Fatal Errors

010:008 EOS_MAUI_BADID

The ID specified by context is not valid.

010:016 EOS_MAUI_BADVALUE

The width or height is zero.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with drw_init().

See Also

drw_get_context()
drw_set_context_clip()
DRW_CONTEXT_ID
GFX_DIMEN
GFX_DIMEN_MAX
GFX_POS
drw_set_context_dst()
Set Destination Drawmap

Syntax
error_code
drw_set_context_dst(DRW_CONTEXT_ID context,
                   const GFX_DMAP *dstdmap)

Description
drw_set_context_dst() sets the destination drawmap value for the specified drawing context to dstdmap. If set to NULL, the destination drawmap becomes undefined and drawing operations that require it return the error EOS_MAUI_NODSTDMAP.

If the contents of the drawmap object dstdmap are changed after calling this function, you must call it again to register the changes with this context object. If you delete the dstdmap, it must be removed from this context.

If successful, this function returns SUCCESS.

Note
Do not use this function if you are currently using the Windowing API.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context       Drawing context ID.
*dstdmap      Pointer to destination drawmap.
Non-Fatal Errors

010:008 EOS_MAUI_BADID

The ID specified by context is not valid.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with drw_init().

Indirect Errors

blt_set_context_dst()

See Also

drw_create_context()
drw_get_context()

DRW_CONTEXT_ID

GFX_DMAP
**drw_set_context_expmix()**

**Set Mixing Mode for Expanding**

**Syntax**

```c
error_code
drw_set_context_expmix(DRW_CONTEXT_ID context,
                        BLT_MIX mixmode)
```

**Description**

`drw_set_context_expmix()` sets the mixing mode for expansion operations in the specified drawing context to `mixmode`. The mixing mode specifies the way source pixels are transferred to the destination drawmap.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- **context**: Drawing context ID.
- **mixmode**: Mixing mode.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `drw_init()`.

**Indirect Errors**

- `blt_set_context_expmix()`
See Also

drw_create_context()
drw_get_context()
BLT_MIX
BLT_CONTEXT_ID
**drw_set_context_exptbl()**

Set Pixel Expansion Table

**Syntax**

```c
error_code
drw_set_context_exptbl(DRW_CONTEXT_ID context,
                      u_int8 num_values,
                      const GFX_PIXEL exptbl[])
```

**Description**

This function sets the pixel expansion table for the specified drawing context to `exptbl`. `num_values` specifies the number of values in the table `exptbl`.

If `exptbl` is set to NULL, the pixel expansion table becomes undefined and drawing operations that require an expansion table return the error `EOS_MAUI_NOEXPTABLE`.

If the contents of the expansion table `exptbl` are changed after calling this function, you must call it again to register the changes with this context object. Since this function makes a copy of the data pointed to by `exptbl`, you may destroy `exptbl` immediately after calling `drw_set_context_exptbl()`.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `context`:
  - Drawing context ID.
- `num_values`:
  - Number of values in the expansion table.
- `exptbl`[]:
  - Table of expansion values.
Non-Fatal Errors

010:008  EOS_MAUI_BADID
The ID specified by context is not valid.

010:016  EOS_MAUI_BADVALUE
The value num_values must be greater than or equal to 2.

010:036  EOS_MAUI_NOINIT
This API has not been initialized with drw_init().

Indirect Errors

blt_set_context_exptbl()

See Also

drw_get_context()

DRW_CONTEXT_ID

GFX_PIXEL
drw_set_context_fm()
Set Fill Mode

Syntax
error_code
drw_set_context_fm(DRW_CONTEXT_ID context,
                   DRW_FM fill_mode)

Description
drw_set_context_fm() sets the fill mode for the specified drawing context to fill_mode.

If fill_mode is set to DRW_FM_SOLID, solid shapes are drawn. If set to DRW_FM_OUTLINE, only the outline of shapes are drawn. This attribute only affects closed shapes, such as rectangles and circles. Open shapes, such as lines and polylines, are not affected.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context Drawing context ID.
fill_mode Specifies fill mode for shape drawing as solid or outline.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by context is not valid.
010:016 EOS_MAUI_BADVALUE The fill_mode is not a valid value.
010:036 EOS_MAUI_NOINIT This API has not been initialized with drw_init().
See Also

drw_create_context()
drw_get_context()
DRW_CONTEXT_ID
DRW_FM
drw_set_context_ls()

Set Line Style

Syntax

```c
error_code
drw_set_context_ls(DRW_CONTEXT_ID context,
                   DRW_LS line_style)
```

Description

drw_set_context_ls() sets the line style for the specified drawing context to line_style.

This parameter is always used for lines and polylines. It is only used for solid shapes (such as circles) when the fill mode is set to DRW_FM_OUTLINE.

If line_style is set to DRW_LS_SOLID, shapes are drawn with a solid outline. If set to DRW_LS_DASHED, they are drawn with a dashed outline. The dash pattern is defined by drw_set_context_dash().

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- context: Drawing context ID.
- line_style: Specifies line style as solid or dashed.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**: The ID specified by context is not valid.
- **010:016 EOS_MAUI_BADVALUE**: The line_style is not a valid value.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with `drw_init()`.

See Also
- `drw_create_context()`
- `drw_get_context()`
- `drw_set_context_dash()`
- `DRW_CONTEXT_ID`
- `DRW_LS`
5 Drawing Functions

**drw_set_context_mask()**

Set Mask Drawmap

**Syntax**

```c
error_code
drw_set_context_mask(DRW_CONTEXT_ID context,
                     const GFX_DMAP *mask_dmap)
```

**Description**

*drw_set_context_mask()* sets the mask drawmap value for the specified drawing context to *mask_dmap*. If set to NULL, then the mask drawmap becomes undefined and drawing operations that require a mask return the error `EOS_MAUI_NOMASKDMAP`.

If the pixel depth of the mask is different from the source, then `EOS_MAUI_INCOMPATCM` is returned from operations that use the mask.

If the line size of the mask is different from the source, then `EOS_MAUI_BADLINESIZE` is returned from operations that use the mask.

If the width and height of the mask differ from the source, then `EOS_MAUI_BADDIMEN` is returned from operations that use the mask.

The mask drawmap is used by copy operations as follows. When a source pixel is read from the source drawmap, the corresponding pixel is also read from the mask drawmap. For each bit in the mask that is 1, the corresponding bit in the source is transferred to the destination. For each bit in the mask that is 0, the corresponding bit in the source is ignored.

If the contents of the drawmap object *mask_dmap* are changed after calling this function, you must call it again to register the changes with this context object. If you delete the *mask_dmap*, it must be removed from this context.

If successful, this function returns `SUCCESS`. 
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context Drawing context ID.
*mask_dmap Pointer to mask drawmap.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by context is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with drw_init().
010:039 EOS_MAUI_NOPIXMEM The source drawmap mask_dmap has no pixel memory.

Indirect Errors
blt_set_context_mask()

See Also
drw_create_context()
drw_get_context()
drw_copy_block()
BLT_MIX
DRW_CONTEXT_ID
GFX_DMAP
**drw_set_context_mix()**

**Set Mixing Mode for Drawing**

**Syntax**

```c
error_code drw_set_context_mix(DRW_CONTEXT_ID context,
                               BLT_MIX mixmode)
```

**Description**

`drw_set_context_mix()` sets the mixing mode for draw operations in the specified drawing context to `mixmode`. The mixing mode specifies the way source pixels are transferred to the destination drawmap.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context**
  - Drawing context ID.
- **mixmode**
  - Specifies mixing mode for drawing operations.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `context` is not valid.

- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `drw_init()`.

**Indirect Errors**

`blt_set_context_drwmix()`
Drawing Functions

See Also

drw_create_context()
drw_get_context()
BLT_MIX
BLT_CONTEXT_ID
**drw_set_context_ofs()**

**Set Offset Pixel Value**

**Syntax**

```c
error_code drw_set_context_ofs(DRW_CONTEXT_ID context, GFX_PIXEL ofspixel)
```

**Description**

`drw_set_context_ofs()` sets the offset pixel value for the specified drawing context to `ofspixel`. The offset pixel value is added to the source pixels before they are transferred to the destination when `BLT_MIX_SPO` is used.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `context` : Drawing context ID.
- `ofspixel` : Number of pixels to offset destination from source.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID** : The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT** : This API has not been initialized with `drw_init()`.

**Indirect Errors**

- `blt_set_context_ofs()`
Drawing Functions

See Also

drw_create_context()
drw_get_context()
BLT_MIX
DRW_CONTEXT_ID
GFX_PIXEL
drw_set_context_origin()
Set Drawing Origin

Syntax

```c
error_code
drw_set_context_origin(DRW_CONTEXT_ID context,
                        GFX_POS x, GFX_POS y)
```

Description

`drw_set_context_origin` sets the drawing origin for the specified drawing context. All coordinates used for drawing are relative to this position. The origin is specified by `x` and `y`. This is considered 0,0 for all drawing operations. If successful, this function returns `SUCCESS`.

Note

Do not use this function if you are currently using the Windowing API.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `context` Drawing context ID.
- `x` X coordinate of origin used for drawing.
- `y` Y coordinate of origin used for drawing.

Non-Fatal Errors

- `010:008 EOS_MAUI_BADID` The ID specified by `context` is not valid.
010:036 EOS_MAUI_NOINIT

This API has not been initialized by `drw_init()`.

**See Also**

`drw_get_context()`

`DRW_CONTEXT_ID`

`GFX_POS`
**drw_set_context_pix()**

Set Pixel Value for Drawing

---

**Syntax**

```c
error_code drw_set_context_pix(DRW_CONTEXT_ID context,
                                GFX_PIXEL drwpixel)
```

**Description**

`drw_set_context_pix()` sets the pixel value used for drawing with the specified drawing context to `drwpixel`.

**Note**

The contents of `drwpixel` should be in the endianess of the destination drawmap's coding method rather than that of the CPU.

If successful, this function returns `SUCCESS`.

---

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context**: Drawing context ID.
- **drwpixel**: Pixel value used for drawing.

---

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `drw_init()`.
Indirect Errors

blt_set_context_pix()

See Also

drw_create_context()
drw_get_context()
DRW_CONTEXT_ID
GFX_PIXEL
drw_set_context_trans()
Set Transparent Pixel Value

Syntax
error_code
drw_set_context_trans(DRW_CONTEXT_ID context,
                      GFX_PIXEL transpixel)

Description
drw_set_context_trans() sets the transparent pixel value for the specified drawing context to transpixel. The transparent pixel value is used to filter out source pixels when they are transferred to the destination.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context Drawing context ID.
transpixel Specifies transparent pixel value.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by context is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with drw_init().

Indirect Errors
blt_set_context_trans()
See Also

drw_create_context()
drw_get_context()
DRW_CONTEXT_ID
GFX_PIXEL
drw_set_context_src()
Set Source Drawmap

Syntax

eerror_code = drw_set_context_src(DRW_CONTEXT_ID context, 
const GFX_DMAP *srcdmap)

Description
drw_set_context_src() sets the source drawmap value for the 
specified drawing context to srcdmap. If set to NULL, the source 
drawmap becomes undefined and drawing operations that require a 
source drawmap return the error EOS_MAUI_NOSRCDMAP.

If the contents of the drawmap object srcdmap are changed after 
calling this function, you must call it again to register the changes with 
this context object. If you delete the srcdmap, it must be removed 
from this context.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context
Drawing context ID.
*srcdmap
Pointer to the source drawmap.

Non-Fatal Errors
010:008 EOS_MAUI_BADID
The ID specified by context is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with drw_init().
010:039 EOS_MAUI_NOPIXMEM

The source drawmap srcdmap has no pixel memory.

010:043 EOS_MAUI_NOTALIGNED

Either srcdmap->pixmem or srcdmap->line_size is not a multiple of GFX_LINE_PAD.

Indirect Errors

blt_set_context_src()

See Also

drw_create_context()

drw_copy_block()

drw_expd_block()

drw_get_context()

BLT_MIX

DRW_CONTEXT_ID

GFX_DMAP
**drw_set_error_action()**
Set Action to Take in Error Handler

**Syntax**

```c
error_code drw_set_error_action(MAUI_ERR_LEVEL debug_level,
                                MAUI_ERR_LEVEL passback_level,
                                MAUI_ERR_LEVEL exit_level)
```

**Description**

`drw_set_error_action()` sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling `drw_init()`. Following is the table of error levels. The least severe error is listed first.

**Table 5-16 Error Level for drw_set_error_action()**

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATALAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
</tbody>
</table>
debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default pass-back level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to call exit(). In this case the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

**Attributes**

**Operating System:** OS-9 and OS-9 for 68K

**State:** User

**Threads:** Safe

**Parameters**

debug_level Minimum error level that causes the error handler to print a message to standard error.

passback_level Minimum error level that causes the error handler to return the error.

exit_level Minimum error level that causes the error handler to call exit().

**Table 5-16  Error Level for drw_set_error_action()**

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>
5 Drawing Functions

Non-Fatal Errors
None

See Also

drw_init()
**drw_term()**

Terminate the Drawing API

**Syntax**

```c
error_code
drw_term(void)
```

**Description**

`drw_term()` terminates the Drawing API.

This API depends on the Shaded Memory and Bit-BLT APIs. Therefore, `mem_term()` and `blt_term()` are called by this function.

If successful, this function returns `SUCCESS`.

**Non-Fatal Errors**

010:036  **EOS_MAUI_NOINIT**

This API has not been initialized with `drw_init()`.

**Indirect Errors**

- `blt_term()`
- `mem_term()`

**See Also**

- `drw_init()`
Chapter 6: Graphics Functions
gfx_alloc_mem()
Allocate Graphics Memory

Syntax
error_code
gfx_alloc_mem(GFX_DEV_ID gfxdev, size_t *size,
     void **mem_ptr, u_int32 color)

Description
gfx_alloc_mem() allocates graphics memory from the specified color of memory. This memory is defined by the graphics driver. See the CDB for information about the color(s) of memory (if any) defined by the driver(s) you are using.

For More Information
See Chapter 4 and Chapter 5 in Using MAUI for a discussion of how to set up and use graphics memory.

size is the amount of memory to allocate. A pointer to size must be passed because it is updated by gfx_alloc_mem() with the actual size that is allocated.

A pointer to the allocated memory is returned in mem_ptr. A pointer to mem_ptr must be passed to gfx_alloc_mem(). Use gfx_dealloc_mem() to de-allocate this memory when it is no longer needed.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gfxdev</td>
<td>Graphics device ID.</td>
</tr>
<tr>
<td>*size</td>
<td>Amount of memory to allocate.</td>
</tr>
<tr>
<td><strong>mem_ptr</strong></td>
<td>Pointer to allocated memory.</td>
</tr>
<tr>
<td>color</td>
<td>Color of memory to allocate.</td>
</tr>
</tbody>
</table>

**Non-Fatal Errors**

- 010:008 EOS_MAUI_BADID: The ID specified by `gfxdev` is not valid.
- 010:036 EOS_MAUI_NOINIT: This API has not been initialized with `gfx_init()`.
- 010:060 EOS_MAUI_NOTOWNER: This is not the process that opened the device `gfxdev`.

**Indirect Errors**

-os_ss_gfx_allocmem()

**Driver Errors**

- 0:208 EOS_UNKSVC: This feature is not supported by the driver.

**See Also**

- cdb_get_ddr()
- gfx_dealloc_mem()
- GFX_DEV_ID
- CDB_TYPE_GRAPHIC
- CDB_TYPE_SYSTEM
gfx_calc_pixmem_size()
Calculate Size of Pixel Memory

Syntax

```c
error_code gfx_calc_pixmem_size(size_t *ret_size,
                                 GFX_CM coding_method,
                                 GFX_DIMEN width,
                                 GFX_DIMEN height)
```

Description

gfx_calc_pixmem_size() calculates the size (in bytes) of the memory required to hold a rectangular area of pixels with the specified width, height, and coding_method.

The calculations required take into consideration special requirements by coding methods such as GFX_CM_YCRCB420 and other factors such as line padding requirements indicated by coding_method.

The size is returned in ret_size. A pointer to this variable should be passed to gfx_calc_pixmem_size().

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

*ret_size Pointer to returned size.
coding_method Graphics coding method.
width Width of graphic image in pixels.
height Height of graphic image in pixels.
Non-Fatal Errors

010:002 EOS_MAUI_BADCODEMETH  coding_method is not valid.
010:006 EOS_MAUI_BADDIMEN     The width or height is zero.
010:036 EOS_MAUI_NOINIT       This API has not been initialized with gfx_init().

See Also

gfx_set_dmap_size()
GFX_CM
GFX_DIMEN
GFX_LINE_PAD
gfx_clone_dev()

Clone a Graphics Device

Syntax

error_code
gfx_clone_dev (GFX_DEV_ID *ret_gfxdev,
            GFX_DEV_ID gfxdev)

Description

gfx_clone_dev() clones the graphics device whose ID is indicated
by gfxdev. This allows more than one application to share the same
logical graphics device.

The device ID is returned in ret_gfxdev. A pointer to this variable
should be passed to gfx_clone_dev(). Use gfx_close_dev() when this device is no longer needed.

Since the original and cloned IDs both point to the same logical device,
this API enforces a mechanism that allows only one caller to make
changes to the device at any given time. Therefore, calls that change
the logical device parameters may block if another process is already
busy updating them.

Cloning a logical device has no effect on its placement in the stack of
logical devices currently open on the physical device.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

*ret_gfxdev The ID of the cloned device.
gfxdev The ID of the original device.
Graphics Functions

**Fatal Errors**

010:008  EOS_MAUI_BADID
The ID specified by gfxdev is not valid.

010:036  EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:058  EOS_MAUI_INCOMPATVER
The API has a newer compatibility level than the driver. You must obtain a newer driver.

**Indirect Errors**

mem_malloc()
_os_ev_link()
_os_gs_gfx_compat()
_os_open()
_os_ss_gfx_clonedev()

**For More Information**
See the OS-9 Technical Manual or the Ultra C Library Reference for all _os functions.

**Driver Errors**

010:058  EOS_MAUI_INCOMPATVER
The API is older than the driver and incompatible with driver functions.

**See Also**
gfx_close_dev()
gfx_get_dev_status()
gfx_restack_dev()
GFX_DEV_ID
gfx_clone_vport()  
Clone a Viewport

Syntax

```c
error_code gfx_clone_vport (GFX_VPORT_ID *ret_vport,
                            GFX_DEV_ID gfxdev,
                            GFX_VPORT_ID vport)
```

Description

gfx_clone_vport() clones the viewport whose ID is indicated by `vport`. This allows more than one application to share the same viewport.

When cloning a viewport, `gfx_clone_dev()` must be called first to obtain a valid `gfxdev`.

The viewport ID is returned in `ret_vport`. A pointer to this variable should be passed to `gfx_clone_vport()`. Use `gfx_destroy_vport()` when this viewport is no longer needed.

Since the original and cloned IDs both point to the same viewport, this API enforces a mechanism that allows only one caller to make changes to the viewport at any given time. Therefore, calls that change the viewport parameters may block if another process is already busy updating them.

Cloning a viewport has no effect on its placement in the stack of viewports currently open on the logical device.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Parameters

*ret_vport
The viewport ID of the cloned viewport.
gfxdev
The graphic device ID.
vport
The viewport ID of the original viewport.

Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by gfxdev is not valid.
010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().
010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.

Indirect Errors

mem_malloc()

Driver Errors

_os_ss_gfx_clonevp()
_os_permit()
_os_srqmem()

For More Information

See the OS-9 Technical Manual or the Ultra C Library Reference for all _os functions.

See Also

gfx_destroy_vport()
gfx_get_vport_status()
gfx_restack_vport()
GFX_DEV_ID
GFX_VPORT_ID
gfx_close_dev()

Close a Graphics Device

Syntax

```c
error_code
gfx_close_dev(GFX_DEV_ID gfxdev)
```

Description

gfx_close_dev() closes the logical graphics device `gfxdev`. This reduces the link count of processes that are sharing the logical device.

If this is the last process using the logical device, then any viewports still open for the device are automatically destroyed and the device is removed from the stack of logical devices on the physical device.

In this case, if the logical device was the top-most (visible) one, then the one immediately below it in the stack becomes visible on the physical device.

Referencing viewports after they have been destroyed can cause a bustrap error. Applications should destroy all viewports before closing the graphics device.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `gfxdev`: Graphics device ID

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**: The ID specified by `gfxdev` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `gfx_init()`.
010:060 EOS_MAUI_NOTOWNER  This is not the process that opened the device gfxdev.

Indirect Errors
gfx_destroy_vport()
mem_free()
_os_close()
_os_ev_unlink()

Driver Errors
_os_ss_gfx_closedev()
_os_protect()
_os_srtmem()

For More Information
See the OS-9 Technical Manual or the Ultra C Library Reference for all _os functions.

See Also
gfx_clone_dev()
gfx_open_dev()
GFX_DEV_ID
gfx_create_cursor()
Create a New Hardware Cursor

Syntax

```c
error_code gfx_create_cursor (GFX_CURSOR_ID * ret_cursor_id,
                              GFX_DEV_ID gfxdev,
                              GFX_CURSOR_SPEC * cursor)
```

Description

gfx_create_cursor() creates a new hardware cursor. This function pre-loads a cursor shape into the graphics device. Applications may define as many cursors as they wish up to the limits of the hardware.

The cursor image and attributes are defined by cursor. Use gfx_get_cursor_cap() to determine the cursor formats supported by the driver. Use gfx_set_cursor() to make this the active cursor.

This function calls the driver via the setstat _os_ss_gfx_cursor_create() to verify the cursor attributes and copy the cursor image to the hardware's cursor memory. Since this is a hardware cursor and not a software cursor, unlike the WIN API, the application does not need to keep cursor around after making this call.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `ret_cursor_id`: The cursor ID assigned by the system. Future references to the cursor should use this cursor ID.
- `gfxdev`: The graphic device ID.
- `cursor`: Pointer to the structure holding the cursor image and attributes information.
Non-Fatal Errors

**010:008 EOS_MAUI_BADID**
The ID specified by `gfxdev` is not valid.

**010:012 EOS_MAUI_BADPTR**
`ret_cursor_id` or `cursor` is `NULL`.

**010:035 EOS_MAUI_NOHWSUPPORT**
The hardware does not support a hardware cursor.

**010:036 EOS_MAUI_NOINIT**
The API has not been initialized with `gfx_init()`.

**010:037 EOS_MAUI_NOMASKDMAP**
The maskmap drawmap is missing.

**010:039 EOS_MAUI_NOPIXMEM**
The bitmap or maskmap has no pixel memory.

**010:042 EOS_MAUI_NOSRCDMAP**
The bitmap drawmap is missing.

Driver Errors

**010:015 EOS_MAUI_BADSIZE**
The size of the drawmap is not valid. Use `gfx_cursor_cap()` to determine valid drawmap sizes.

**010:022 EOS_MAUI_INCOMPATCM**
The coding method of bitmap or maskmap were not valid for the driver. Use `gfx_cursor_cap()` to determine valid coding methods.

**010:052 EOS_MAUI_NOPALETTE**
Either no palette or an incompatible palette was found in `cursor->bitmap` when one was required.

**0:201 EOS_BPNUM**
Bad path number. Path is invalid or is not currently open.
This driver does not support graphic hardware cursors.

See Also
GFX_CURSOR_SPEC
GFX_CURSOR_ID
GFX_DEV_ID
gfx_destroy_cursor()
gfx_get_cursor_cap()
gfx_set_cursor()
gfx_create_dmap()
Create a Drawmap Object

Syntax
error_code
gfx_create_dmap(GFX_DMAP **ret_dmap,
               u_int32 dmap_shade)

Description
gfx_create_dmap() creates a new drawmap object. dmap_shade specifies the shade of memory used for the new object. The following table shows how each member of the GFX_DMAP structure is initialized and the accepted methods for modifying each member.

Table 6-1  GFX_DMAP Initialization Values

<table>
<thead>
<tr>
<th>Member</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>coding_method</td>
<td>GFX_CM_UNKNOWN</td>
<td>gfx_set_dmap_size()</td>
</tr>
<tr>
<td>width</td>
<td>0</td>
<td>gfx_set_dmap_size()</td>
</tr>
<tr>
<td>height</td>
<td>0</td>
<td>gfx_set_dmap_size()</td>
</tr>
<tr>
<td>line_size</td>
<td>0</td>
<td>gfx_set_dmap_size()</td>
</tr>
<tr>
<td>pixmem</td>
<td>NULL</td>
<td>gfx_set_dmap_pixmem()</td>
</tr>
<tr>
<td>pixmem_shade</td>
<td>0</td>
<td>gfx_set_dmap_pixmem()</td>
</tr>
<tr>
<td>pixmem_size</td>
<td>0</td>
<td>gfx_set_dmap_pixmem()</td>
</tr>
<tr>
<td>palette</td>
<td>NULL</td>
<td>Directly</td>
</tr>
</tbody>
</table>
The drawmap object is allocated from dmap_shade and a pointer to it is returned in ret_dmap. A pointer to ret_dmap should be passed to gfx_create_dmap(). Use gfx_destroy_dmap() to destroy this drawmap when it is no longer needed.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

**ret_dmap**  Pointer to ret_dmap pointer.

dmap_shade  Shade from where drawmap object is allocated.

**Fatal Errors**

010:036 EOS_MAUI_NOINIT  This API has not been initialized with gfx_init().

**Indirect Errors**

mem_malloc()

**See Also**

gfx_destroy_dmap()
gfx_set_dmap_pixmem()
gfx_set_dmap_size()
GFX_DMAP
gfx_create_vport()

Create a Viewport

Syntax

error_code
gfx_create_vport(GFX_VPORT_ID *ret_vport,
    GFX_DEV_ID gfxdev, GFX_POS x,
    GFX_POS y, GFX_DIMEN width,
    GFX_DIMEN height,
    GFX_VPORT_PLACEMENT placement, ...)

Description

gfx_create_vport() creates a new viewport on the device specified by gfxdev. The following table shows the default value for each setting and the function for modifying them.

Table 6-2 gfxdev Settings

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewport position</td>
<td>x, y</td>
<td>gfx_set_vport_position()</td>
</tr>
<tr>
<td>Viewport size</td>
<td>width, height</td>
<td>gfx_set_vport_size()</td>
</tr>
<tr>
<td>Viewport state</td>
<td>False</td>
<td>gfx_set_vport_state()</td>
</tr>
<tr>
<td>Intensity</td>
<td>100</td>
<td>gfx_set_vport_intensity()</td>
</tr>
<tr>
<td>Drawmap in viewport</td>
<td>NULL</td>
<td>gfx_set_vport_dmap()</td>
</tr>
<tr>
<td>Palette</td>
<td>NULL</td>
<td>gfx_set_vport_dmap()</td>
</tr>
</tbody>
</table>
The position of the viewport on the display is given by \(x\) and \(y\). The size of the viewport is specified by \(\text{width}\) and \(\text{height}\). These coordinates are specified using the display coordinate system.

The following table shows how \textit{placement} may be used to specify the new position. The \textit{Parameter} column shows the types of the additional parameters (represented by “...” above).

### Table 6-3  Use of Placement to Specify New Position

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Parameter</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFX_VPORT_FRONT</td>
<td>None</td>
<td>In front of all viewports</td>
</tr>
<tr>
<td>GFX_VPORT_BACK</td>
<td>None</td>
<td>In back of all viewports</td>
</tr>
<tr>
<td>GFX_VPORT_FRONT_OF GFX_VPORT</td>
<td>GFX_VPORT ref_vport</td>
<td>In front of (\text{ref_vport})</td>
</tr>
<tr>
<td>GFX_VPORT_BACK_OF GFX_VPORT</td>
<td>GFX_VPORT ref_vport</td>
<td>In back of (\text{ref_vport})</td>
</tr>
</tbody>
</table>

The viewport ID is returned in \(\text{ret\_vport}\). A pointer to this variable should be passed to \texttt{gfx\_create\_vport()}. Use \texttt{gfx\_destroy\_vport()} to destroy this viewport when it is no longer needed. Use \texttt{gfx\_get\_vport()} to get the current settings in a viewport.

If successful, this function returns \texttt{SUCCESS}. 

### Table 6-2  gfxdev Settings

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drawmap position in viewport</td>
<td>0, 0</td>
<td>gfx_set_vport_dmap()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gfx_set_vport_dmpos()</td>
</tr>
</tbody>
</table>
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_vport
Viewport ID.
gfxdev
Graphics device ID.
x,y
Position of viewport on display.
width
Width of viewport in pixels.
height
Height of viewport in pixels.
placement
Position of viewport with respect to the reference viewport.

... Optional additional parameters for placement.

Fatal Errors
010:008 EOS_MAUI_BADID
ID specified by gfxdev, or the ID specified by ref_vport (see placement) is not valid.

010:016 EOS_MAUI_BADVALUE
The value used for placement is not valid.

010:036 EOS_MAUI_NOINIT
API not initialized with gfx_init().

010:044 EOS_MAUI_NOTFOUND
The reference object ref_vport is not in the viewport stack for the device gfxdev.

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.
6 Graphics Functions

**Indirect Errors**
- `gfx_set_vport_dmap()`
- `gfx_set_vport_intensity()`
- `gfx_set_vport_state()`
- `mem_calloc()`

**Driver Errors**

010:011 EOS_MAUI_BADPOS  
- `x` or `y` is not within the display.

010:006 EOS_MAUI_BADDIMEN  
- `width` or `height` is 0 or too big for the display.

- `_os_permit()`  
  - MFM binding to the graphics driver.
- `_os_srqmem`  
  - MFM binding to the graphics driver.
- `_os_os_gfx_createvp()`  
  - MFM binding to the graphics driver.

**See Also**
- `gfx_clone_vport()`
- `gfx_set_vport_position()`
- `gfx_destroy_vport()`
- `gfx_get_dev_cap()`
- `gfx_get_vport_status()`
- `gfx_restack_vport()`
- `gfx_set_vport_colors()`
- `gfx_set_vport_dmpos()`
- `gfx_set_vport_size()`
- `GFX_DEV_ID`
- `GFX_DIMEN`
- `GFX_POS`
- `GFX_VPORT_ID`
- `GFX_VPORT_PLACEMENT`
gfx_cvt_dmpos_dppos()
Convert Drawmap to Display Position

Syntax
error_code
gfx_cvt_dmpos_dppos(GFX_POS *ret_dpx,
                     GFX_POS *ret_dpy,
                     GFX_VPORT_ID vport,
                     GFX_POS dmx, GFX_POS dmy)

Description
gfx_cvt_dmpos_dppos() converts the position dmx and dmy in the
drawmap coordinate system to a position in the display coordinate
system using the viewport vport.

The display position is returned in ret_dpx and ret_dpy. A pointer to
these variables should be passed to gfx_cvt_dmpos_dppos().
If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_dpx Pointer to display position x coordinate.
*ret_dpy Pointer to display position y coordinate.
vport     Viewport ID.
dmx       Drawmap x coordinate.
dmy       Drawmap y coordinate.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by vport is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the viewport vport.

See Also

gfx_cvt_dppos_dmpos()
GFX_POS
GFX_VPORT_ID
gfx_cvt_dppos_dmpos()
Convert Display to Drawmap Position

Syntax

```c
error_code
gfx_cvt_dppos_dmpos(GFX_POS *ret_dmx,
                     GFX_POS *ret_dmy,
                     GFX_VPORT_ID vport,
                     GFX_POS dpx,
                     GFX_POS dpy)
```

Description

gfx_cvt_dppos_dmpos() converts the position dpx and dpy in the display coordinate system to a position in the drawmap coordinate system using the viewport vport.

The drawmap position is returned in ret_dmx and ret_dmy. A pointer to these variables should be passed to gfx_cvt_dppos_dmpos().

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

*ret_dmx Pointer to drawmap x coordinate.
*ret_dmy Pointer to drawmap y coordinate.
vport Viewport ID.
dpx Display x coordinate.
dpy Display y coordinate.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by \texttt{vport} is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with \texttt{gfx_init()}.  
This is not the process that opened the viewport \texttt{vport}.

010:060 EOS_MAUI_NOTOWNER

See Also
\texttt{gfx_cvtx_dmpos_dppos()}
\texttt{GFX_POS}
\texttt{GFX_VPORT_ID}
gfx_dealloc_mem()  
De-allocate Graphics Memory

Syntax

```c
error_code gfx_dealloc_mem(GFX_DEV_ID gfxdev, size_t size,
                              void *mem_ptr, u_int32 color)
```

Description

gfx_dealloc_mem() de-allocates graphics memory previously allocated by gfx_alloc_mem().

`size` is the size of the memory block being de-allocated and `mem_ptr` is a pointer to the memory block. `color` is the color of the memory block being de-allocated.

If successful, this function returns `SUCCESS`.

Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

Parameters

gfxdev  
Graphics device ID.

size  
Size of memory block being de-allocated.

*mem_ptr  
Pointer to memory block color.

color  
Color of memory block being de-allocated.
Non-Fatal Errors

010:008 EOS_MAUI_BADID  The ID specified by gfxdev is not valid.
010:036 EOS_MAUI_NOINIT  This API has not been initialized with gfx_init().
010:060 EOS_MAUI_NOTOWNER  This is not the process that opened the device gfxdev.

Indirect Errors

Driver Errors

0:208 EOS_UNKSVC  This feature is not supported by the driver.
_os_ss_gfx_deallocmem()  MFM binding to the graphics driver.

See Also

gfx_alloc_mem()
GFX_DEV_ID
gfx_destroy_cursor()
Destroy a Hardware Cursor

Syntax

```c
error_code
gfx_destroy_cursor (GFX_DEV_ID gfxdev,
                     GFX_CURSOR_ID * cursor_id)
```

Description

gfx_destroy_cursor() destroys the cursor `cursor_id`. If the
cursor was active (see gfx_set_cursor()) when destroyed, another
defined cursor will become active. If no cursors exist, the cursor will be
deactivated.

This function calls the driver via the setstat
_os_ss_cursor_destroy() to return the memory associated with
cursor.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `gfxdev` The graphic device ID.
- `cursor_id` The cursor ID of the cursor to destroy.

Non-Fatal Errors

- 010:008 EOS_MAUI_BADID The ID specified by `gfxdev` is not valid.
- 010:036 EOS_MAUI_NOINIT The API has not been initialized with `gfx_init()`.
Driver Errors

0:201 EOS_BPNUM  Bad path number. Path is invalid or is not currently open.
0:208 EOS_UNKSVC  This driver does not support graphic hardware cursors.
010:008 EOS_MAUI_BADID  The ID specified by cursor_id is not valid.

See Also

GFX_CURSOR_ID
GFX_DEV_ID
gfx_create_cursor()
gfx_destroy_dmap()

Destroy a Drawmap Object

**Syntax**

```c
error_code
gfx_destroy_dmap(GFX_DMAP *dmap)
```

**Description**

gfx_destroy_dmap() destroys the specified drawmap object `dmap`. Only call this function to destroy drawmaps created with `gfx_create_dmap()`.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `*dmap` — Pointer to drawmap object.

**Non-Fatal Errors**

- 010:036 EOS_MAUI_NOINIT — This API has not been initialized with `gfx_init()`.

**Indirect Errors**

- `mem_free()`
- `mem_sfree()`

**See Also**

- `gfx_create_dmap()`
- `GFX_DMAP`
gfx_destroy_vport()
Destroy a Viewport

Syntax

erreur_code = gfx_destroy_vport(GFX_VPORT_ID vport)

Description

gfx_destroy_vport() destroys the specified viewport vport for the current process. This reduces the link count of processes that are sharing the viewport. If this is the last process using the viewport, the vport is removed from the gfxdev.

If the viewport is currently active, you must call gfx_set_vport_state() and gfx_update_display() to deactivate it before calling gfx_destroy_vport().

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

vport Graphics viewport ID.

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by vport is not valid.
010:024 EOS_MAUI_INUSE The viewport is still being used.
010:036 EOS_MAUI_NOINIT This API has not been initialized with gfx_init().
010:060 EOS_MAUI_NOTOWNER This is not the process that opened the viewport vport.
Graphics Functions

**Indirect Errors**

`mem_free()`

**Driver Errors**

`_os_protect()`  See the *Ultra C Library Reference*.

`_os_srtmem()`  See the *Ultra C Library Reference*.

`_os_ss_gfx_destroyvp`  MFM binding to the graphics driver.

**See Also**

`gfx_clone_vport()`
`gfx_create_vport()`
`gfx_set_vport_state()`
`gfx_update_display()`
`GFX_VPORT_ID`
gfx_get_cursor_cap()
Get Information About a Hardware Cursor

Syntax

```c
error_code gfx_get_cursor_cap
    (const GFX_CURSOR_CAP ** ret_cursor_cap,
     GFX_DEV_ID gfxdev)
```

Description

gfx_get_cursor_cap() gets information about the existence and capabilities of the hardware cursor. This information may be used to adjust the operation of the application so that it runs properly on different hardware platforms.

This function calls the driver via the setstat _os_gs_gfx_cursor_cap() to query the driver as to the hardware cursor capabilities.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `ret_cursor_cap` A pointer to the buffer containing the cursor capabilities structure is returned in `ret_cursor_cap`. A pointer to this variable should be passed to `gfx_get_cursor_cap()`. Do not attempt to modify or free this buffer.
- `gfxdev` The graphic device ID.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID** The ID specified by `gfxdev` is not valid.
Graphics Functions

010:012 EOS_MAUI_BADPTR  ret_cursor_cap is NULL.
010:035 EOS_MAUI_NOHWSUPPORT  The hardware does not support a hardware cursor.
010:036 EOS_MAUI_NOINIT  The API has not been initialized with gfx_init().

Driver Errors
0:201 EOS_BPNUM  Bad path number. Path is invalid or is not currently open.
0:208 EOS_UNKSVC  This driver does not support graphic hardware cursors.

See Also
GFX_CURSOR_CAP
GFX_CURSOR_INFO
GFX_DEV_ID
gfx_find_vport()
Find the Viewport at the Specified Position

Syntax

```c
error_code
gfx_find_vport(GFX_VPORT_ID *ret_vport,
               GFX_DEV_ID gfxdev,
               GFX_POS dpx,
               GFX_POS dpy)
```

Description

gfx_find_vport() finds the front-most active viewport on gfxdev at the position in the display coordinate system indicated by dpx and dpy. The viewport ID is returned in ret_vport. A pointer to this variable should be passed to gfx_find_vport(). If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `*ret_vport`: Pointer to viewport ID.
- `gfxdev`: Graphics device ID.
- `dpx`: Display x coordinate.
- `dpy`: Display y coordinate.

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by gfxdev is not valid.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with \texttt{gfx\_init()}. 

010:044 EOS_MAUI_NOTFOUND

No active viewport at the specified display position. 

010:060 EOS_MAUI_NOTOWNER

This is not the process that opened the device \texttt{gfxdev}. 

**See Also**

\texttt{gfx\_cvt\_dppos\_dmpos()}

\texttt{GFX\_DEV\_ID}

\texttt{GFX\_POS}

\texttt{GFX\_VPORT\_ID}
gfx_get_dev_attribute()
Get Graphics Device Attribute

Syntax

```c
erreur_code
gfx_get_dev_attribute(GFX_DEV_ATTR *ret_dev_attr,
                       GFX_DEV_ID gfxdev,
                       GFX_ATTR_TYPE attr_type)
```

Description

`gfx_get_dev_attribute()` gets attribute information for the graphics device specified by `gfxdev`. This information may be to determine the current state and capabilities of the hardware platform. For instance, this function could be used to determine the current brightness level of an LCD panel.

The device attribute structure is returned in `ret_dev_attr`. A pointer to this variable should be passed to `gfx_get_dev_attribute()`. The caller must ensure that `ret_dev_attr` points to storage large enough to hold the information.

The attribute information to retrieve from the graphics driver is specified via `attr_type`.

If this feature is not supported by the hardware, then this function returns `EOS_MAUI_NOHWSUPPORT`.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `*ret_dev_attr`: Pointer to device attribute structure.
- `gfxdev`: Graphics device ID.
- `attr_type`: Attribute type.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by gfxdev is not valid.

010:012 EOS_MAUI_BADPTR
ret_dev_attr is not valid.

010:035 EOS_MAUI_NOHWSUPPORT
The hardware driver does not support any attributes.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

Indirect Errors
None

Driver Errors

010:044 EOS_MAUI_NOTFOUND
attr_type is not valid for this driver.

_os_gs_gfx_attribute()
MFM binding to the graphics driver.

See Also
gfx_set_dev_attribute()
GFX_ATTR_TYPE
GFX_DEV_ATTR
GFX_DEV_ID
gfx_get_dev_cap()
Get Graphics Device Capabilities

Syntax
error_code
gfx_get_dev_cap(const GFX_DEV_CAP **ret_dev_cap,
GFX_DEV_ID gfxdev)

Description
gfx_get_dev_cap() gets information about the capabilities of the graphics device specified by gfxdev. This information may be used to adjust the operation of the application so that it runs properly on different hardware platforms.

A pointer to the buffer containing the device capabilities structure is returned in ret_dev_cap. A pointer to this variable should be passed to gfx_get_dev_cap(). Do not attempt to modify or free this buffer.

The default resolution for the device is always returned as the first entry in the res_info member of ret_dev_cap. The default coding method is always returned as the first entry in the cm_info member.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
**ret_dev_cap Pointer to device capabilities structure.
gfxdev Graphics device ID.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by gfxdev is not valid.
010:036 EOS_MAUI_NOINIT  This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER  This is not the process that opened the device gfxdev.

Indirect Errors
None

Driver Errors
_os_gs gfx_devcap()  MFM binding to the graphics driver.

See Also
gfx_get_dev_capexten()  gfx_set_vport_intensity()
gfx_sync_retrace()  GFX_DEV_CAP
GFX_DEV_ID
gfx_get_dev_capexten()
Get Graphics Device Extended Capabilities

Syntax

error_code
gfx_get_dev_capexten(const GFX_DEV_CAPEXTEN **ret_dev_capexten,
                     GFX_DEV_ID gfxdev)

Description

gfx_get_dev_capexten() gets additional information about the
capabilities of the graphics device specified by gfxdev—beyond that
returned by gfx_get_dev_cap(). This information can be used to
adjust the operation of the application so that it runs properly on
different hardware platforms.

A pointer to the buffer containing the device capabilities structure is
returned in ret_dev_capexten. A pointer to this variable should be
passed to gfx_get_dev_capexten(). Do not attempt to modify or
free this buffer.

Note

This call was added in version 3.1 of MAUI. Older MAUI shared library
modules and drivers return an error when this function is called. In
addition, not all up-to-date drivers implement this function. Applications
should always check for errors from this function and be able to run if
the extended device capabilities information is not available.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
**Parameters**

*ret_dev_capexten*  
Pointer to an extended device capabilities structure.

*gfxdev*  
Graphics device ID.

**Non-Fatal Errors**

010:008 EOS_MAUI_BADID  
The ID specified by *gfxdev* is not valid.

010:036 EOS_MAUI_NOINIT  
This API has not been initialized with *gfx_init()*.

010:060 EOS_MAUI_NOTOWNER  
This is not the process that opened the device *gfxdev*.

**Indirect Errors**

000:227 EOS_ITRAP  
This call is not supported by the shared library module.

**Driver Errors**

_os_gs_gfx_devcapexten()  
MFM binding to the graphics driver.

000:208 EOS_UNKSVC  
This call is not supported by the driver common code. Driver is older than MAUI 3.1

010:067 EOS_MAUI_NODVSUPPORT  
This call is not supported by the driver specific code. Driver developer did not implement.

**See Also**

gfx_get_dev_cap()  
GFX_DEV_CAP  
GFX_DEV_CAPEXTEN  
GFX_DEV_ID
gfx_get_dev_status()
Get Graphics Device Status

Syntax
error_code
gfx_get_dev_status(GFX_DEV_STATUS *ret_dev_status,
                     GFX_DEV_ID gfxdev)

Description
gfx_get_dev_status() returns the current status of the specified
logical graphics device gfxdev. The current status includes any
queued up changes waiting for gfx_update_display() to be called.
The device status structure is returned in ret_dev_status. A pointer
to this variable should be passed to gfx_get_dev_status(). The
caller must ensure that ret_dev_status points to storage large
enough to hold the information.
If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_dev_status Pointer to status of graphics device.
gfxdev Graphics device ID.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by gfxdev is
not valid.
010:036 EOS_MAUI_NOINIT This API has not been
initialized with gfx_init().
010:060 EOS_MAUI_NOTOWNER This is not the process that
opened the device gfxdev.
See Also

gfx_set_dev_attribute()
gfx_set_display_extvid()
gfx_set_display_size()
gfx_set_display_transcol()
gfx_set_display_vpmix()
GFX_DEV_ID
GFX_DEV_STATUS
gfx_get_vport_status()
Get Viewport Status

Syntax

```c
error_code gfx_get_vport_status(GFX_VPORT_STATUS *ret_vport_status, GFX_VPORT_ID vport)
```

Description

gfx_get_vport_status() returns the current status of the specified vport. The viewport status structure is returned in ret_vport_status. A pointer to this variable should be passed to gfx_get_vport_status(). The caller must ensure that ret_vport_status points to storage large enough to hold the information.

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- *ret_vport_status: Pointer to viewport status structure.
- vport: Viewport ID.

Non-Fatal Errors

- 010:008 EOS_MAUI_BADID: The ID specified by vport is not valid.
- 010:036 EOS_MAUI_NOINIT: This API has not been initialized with gfx_init().
- 010:060 EOS_MAUI_NOTOWNER: This is not the process that opened the viewport vport.
See Also

gfx_clone_vport()
gfx_create_vport()
gfx_set_vport_intensity()
gfx_set_vport_position()
gfx_set_vport_size()
gfx_set_vport_state()
GFX_VPORT_ID
GFX_VPORT_STATUS
**gfx_init()**

Initialize the Graphics Device API

**Syntax**

```c
error_code
gfx_init(void)
```

**Description**

gfx_init() initializes the Graphics Device API. This function must be called prior to a call to any other graphics device function unless otherwise noted by that function.

Since this API depends on the Shaded Memory API, mem_init() is called by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Indirect Errors**

mem_init()

**See Also**

gfx_term()
gfx_open_dev()
Open a Graphics Device

Syntax

```c
error_code
gfx_open_dev(GFX_DEV_ID *ret_gfxdev,
             const char *device_name)
```

Description

gfx_open_dev() opens the graphics device device_name. The graphics device name (device_name) may be obtained from the CDB. Use cdb_get_ddr() with CDB_TYPE_GRAPHIC.

The device ID is returned in ret_gfxdev. A pointer to this variable should be passed to gfx_open_dev(). Use gfx_close_dev() when this device is no longer needed.

The device is opened in its default resolution. The resolution is determined by the graphics driver. Use gfx_set_display_size() to change the resolution and gfx_get_dev_status() to retrieve the current resolution.

Opening the logical device places it on the top of the stack of logical devices open on the physical device. However, it will not be visible until the first time you call gfx_update_display().

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- *ret_gfxdev Pointer to graphics device ID.
- *device_name Pointer to device name.
6 Graphics Functions

Fatal Errors

010:036 EOS_MAUI_NOINIT  This API has not been initialized with gfx_init().

010:058 EOS_MAUI_INCOMPATVER  The API has a newer compatibility level than the driver and they are not compatible. You must obtain a newer driver.

Indirect Errors

mem_malloc()
_os_ev_link()
_os_open()

Driver Errors

010:058 EOS_MAUI_INCOMPATVER  The API is newer than the driver and is incompatible.

_os_gs_gfx_compat()
_os_permit()
_os_srqmem()
_os_ss_gfx_opendev()  See the Ultra C Library Reference.

See Also

cdb_get_ddr()
gfx_clone_dev()
gfx_close_dev()
gfx_get_dev_status()
gfx_restack_dev()
gfx_set_display_size()
GFX_DEV_ID
CDB_TYPE_GRAPHIC
gfx_restack_dev()
Restack a Device

Syntax

error_code
gfx_restack_dev (GFX_DEV_ID gfxdev, GFX_DEV_PLACEMENT placement, ...)

Description

gfx_restack_dev() changes the placement of the logical graphics device gfxdev within the current stack of logical devices. The effects of this function are not seen until the next time you call gfx_update_display().

The following table shows how placement may be used to specify the new position. The prototype column shows the prototypes for additional parameters represented by "..." in the syntax.

Table 6-4 Use of Placement in gfx_restack_dev()

<table>
<thead>
<tr>
<th>Value of placement</th>
<th>Prototype</th>
<th>New position</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFX_DEV_FRONT</td>
<td>None</td>
<td>In front of all devices</td>
</tr>
<tr>
<td>GFX_DEV_BACK</td>
<td>None</td>
<td>In back of all devices</td>
</tr>
<tr>
<td>GFX_DEV_FRONT_OF GFX_DEV_ID ref gfxdev</td>
<td>GFX_DEV_ID ref gfxdev</td>
<td>In front of ref gfxdev</td>
</tr>
<tr>
<td>GFX_DEV_BACK_OF GFX_DEV_ID ref gfxdev</td>
<td>GFX_DEV_ID ref gfxdev</td>
<td>In back of ref gfxdev</td>
</tr>
</tbody>
</table>

If successful, this function return SUCCESS.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
gfxdev ID of the logical graphics device.
placement Specifies the new position relative to all devices or a reference device.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by gfxdev or the ID specified by ref_gfxdev is not valid.
010:016 EOS_MAUI_BADVALUE The value used for placement is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with gfx_init().
010:060 EOS_MAUI_NOTOWNER This is not the process that opened the device gfxdev.

Indirect Errors
_os_ss_gfx_restackdev() MFM binding to the graphics driver.

Driver Errors
010:044 EOS_MAUI_NOTFOUND The reference device ref_gfxdev is not in the stack of logical devices for this physical device.
Graphics Functions

See Also

gfx_clone_dev()
gfx_open_dev()
GFX_DEV_ID
GFX_DEV_PLACEMENT
gfx_restack_vport()
Re-stack a Viewport

Syntax

error_code
gfx_restack_vport(GFX_VPORT_ID vport,
                     GFX_VPORT_PLACEMENT placement, ...)

Description

gfx_restack_vport() changes the placement of the viewport vport within the current stack of viewports. The effects of this function are not seen until the next time you call gfx_update_display().

The following table shows how placement may be used to specify the new position. The Parameter column shows the types of additional parameters (represented by “...” in the syntax).

Table 6-5 Use of Placement in gfx_restack_vport()

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Parameters</th>
<th>New Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFX_VPORT_FRONT</td>
<td>None</td>
<td>In front of all viewports</td>
</tr>
<tr>
<td>GFX_VPORT_BACK</td>
<td>None</td>
<td>In back of all viewports</td>
</tr>
<tr>
<td>GFX_VPORT_FRONT_OF ref_vport</td>
<td>GFX_VPORT_ID ref_vport</td>
<td>In front of viewport ref_vport</td>
</tr>
<tr>
<td>GFX_VPORT_BACK_OF ref_vport</td>
<td>GFX_VPORT_ID ref_vport</td>
<td>In back of viewport ref_vport</td>
</tr>
</tbody>
</table>

If successful, this function returns SUCCESS.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
vport Graphics viewport ID.
placement Specifies placement of vport.
... Optional additional parameters for placement.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by vport or the ID specified by ref_vport is not valid.
010:016 EOS_MAUI_BADVALUE The value used for placement is not valid.
010:029 EOS_MAUI_MISSINGFEP Missing fast entry point in the graphics driver.
010:036 EOS_MAUI_NOINIT API has not been initialized with gfx_init().
010:044 EOS_MAUI_NOTFOUND The reference object ref_vport is not in the viewport stack for the device vport belongs to.
010:060 EOS_MAUI_NOTOWNER This is not the process that opened the viewport vport.

Indirect Errors
None
6 Graphics Functions

Driver Errors

010:049 EOS_MAUI_TOOCOMPLEX  

The viewport stack has become too complex for this hardware. Returned only if vport is active.

_fe_restack_vp()  

MFM binding to the graphics driver.

See Also

gfx_clone_vport()  
gfx_create_vport()  
gfx_update_display()  
GFX_VPORT_ID  
GFX_VPORT_PLACEMENT
gfx_set_cursor()
Select Hardware Cursor

Syntax

```c
error_code
gfx_set_cursor (GFX_DEV_ID gfxdev,
                GFX_CURSOR_ID * cursor_id)
```

Description

gfx_set_cursor() sets cursor_id as the current active cursor. This allows the application to select between several defined hardware cursors.

If cursor_id is equal to NULL, the cursor for the device is deactivated. This requires that gfxdev be passed as a parameter.

This function calls the driver via the fast entry point _fe_cursor_set() to set the hardware cursor.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- gfxdev: The graphic device ID.
- cursor_id: The ID of the cursor that is to be made active.

Non-Fatal Errors

- 010:008 EOS_MAUI_BADID: The ID specified by gfxdev or cursor_id is not valid.
- 010:036 EOS_MAUI_NOINIT: The API has not been initialized with gfx_init().
Graphics Functions

**Driver Errors**

- **0:201 EOS_BPNUM**: Bad path number. Path is invalid or is not currently open.
- **0:208 EOS_UNKSVC**: This driver does not support graphic hardware cursors.
- **010:008 EOS_MAUI_BADID**: The ID specified by `cursor_id` is not valid.

**See Also**

- `GFX_CURSOR_ID`
- `GFX_DEV_ID`
gfx_set_cursor_pos()
Set the Hardware Cursor Position

Syntax

```c
error_code
gfx_set_cursor_pos (GFX_DEV_ID gfxdev, GFX_POS x,
                    GFX_POS y)
```

Description

gfx_set_cursor_pos() sets the hardware cursor position of the
graphic device gfxdev to x and y. Note that there is one graphic cursor
position for all defined cursors, it does not matter what cursor shape is
active. Also note that when the cursor is displayed/drawn, it is relative to
the hit point, not the upper left corner of the drawmap.

This function calls _fe_cursor_pos() to set the hardware cursor
position.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `gfxdev`: The graphic device ID.
- `x`: Contains the x coordinate position of the
cursor, relative to the cursor's hit point.
- `y`: Contains the y coordinate position of the
cursor, relative to the cursor's hit point.

Non-Fatal Errors

- `010:008 EOS_MAUI_BADID`: The ID specified by gfxdev is
  not valid.
- `010:036 EOS_MAUI_NOINIT`: The API has not been
  initialized with gfx_init().
Driver Errors

0:201  EOS_BPNUM
      Bad path number. Path is invalid or is not currently open.

0:208  EOS_UNKSVC
      This driver does not support graphic hardware cursors.

See Also

GFX_DEV_ID
GFX_POS
gfx_set_decode_dst()
Set Destination for Video Decoding

Syntax

```c
error_code
gfx_set_decode_dst(GFX_DEV_ID gfxdev,
                    const GFX_DMAP *decode_dmap)
```

Description

gfx_set_decode_dst() sets the destination drawmap for video decoding to `decode_dmap`. The video is automatically scaled to the size of this drawmap. If `decode_dmap` is NULL, then video decoding is turned off.

If this feature is not supported by the hardware, then this function returns EOS_MAUI_NOHWSUPPORT.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

gfxdev Graphics device ID.
*decode_dmap Pointer to destination drawmap for video decoding.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**: The ID specified by `gfxdev` is not valid.
- **010:032 EOS_MAUI_NODSTDMAP**: The `decode_dmap->pixmem` is NULL.
- **010:035 EOS_MAUI_NOHWSUPPORT**: This function is not supported by the hardware.
010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.

Indirect Errors
None

Driver Errors
0:208 EOS_UNKSVC
This feature is not supported by the hardware.
_os_ss_gfx_decodedst()
MFM binding to the graphics driver.

See Also
gfx_create_dmap()
gfx_get_dev_cap()
gfx_get_dev_status()
GFX_DEV_ID
GFX_DMAP
gfx_set_dev_attribute()

Set Graphic Device Attribute

Syntax

```c
error_code
gfx_set_dev_attribute(GFX_DEV_ID gfxdev,
                        GFX_ATTR_TYPE attr_type,
                        GFX_ATTR_MODE mode, int32 value)
```

Description

gfx_set_dev_attribute() sets an attribute value for the graphics device specified by `gfxdev`. For instance, this function could be used to set the brightness level of an LCD panel.

`attr_type` specifies the type of attribute to set.

`mode` indicates how to interpret `value`.

<table>
<thead>
<tr>
<th>Value of <code>mode</code></th>
<th>Interpretation of <code>value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>GFX_ATTR_RESET</td>
<td><code>value</code> is ignored. The attribute is reset to its default value.</td>
</tr>
<tr>
<td>GFX_ATTR_ABSOLUTE</td>
<td>The attribute is set equal to <code>value</code>.</td>
</tr>
<tr>
<td>GFX_ATTR_RELATIVE</td>
<td>The attribute is incremented (or decremented if negative) by <code>value</code>.</td>
</tr>
</tbody>
</table>

If this feature is not supported by the hardware, then this function returns `EOS_MAUI_NOHWSUPPORT`.

If successful, this function returns `SUCCESS`. 
### Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

### Parameters

- **gfxdev**: Graphics device ID.  
- **attr_type**: Attribute type.  
- **mode**: Modification method.  
- **value**: Attribute value or offset.

### Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**: The ID specified by `gfxdev` is not valid.  
- **010:035 EOS_MAUI_NOHWSUPPORT**: The hardware driver does not support any attributes.  
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `gfx_init()`.

### Indirect Errors

None

### Driver Errors

- **010:044 EOS_MAUI_NOTFOUND**: The `attr_type` is not valid for this driver.  
- **010:016 EOS_MAUI_BADVALUE**: The `mode` is not valid for this driver.  

`_os_ss_gfx_attribute()`  

MFM binding to the graphics driver.
See Also

gfx_get_dev_attribute()
GFX_ATTR_MODE
GFX_ATTR_TYPE
GFX_DEV_ID
gfx_set_display_bkcol()
Set Backdrop Color

Syntax
error_code
gfx_set_display_bkcol(GFX_DEV_ID gfxdev,
                       const GFX_COLOR *bkcol)

Description
gfx_set_display_bkcol() sets the color of the backdrop for the
display indicated by gfxdev. The effects of this function are not seen
until the next time you call gfx_update_display().

If this feature is not supported by the hardware, then this function
returns EOS_MAUI_NOHWSUPPORT.

bkcol specifies the color for the backdrop. The default is black. The
backdrop is only seen if external video is OFF (see
gfx_set_display_extvid()).

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
gfxdev Graphics device ID.
*bkcol Specifies the backdrop color for the
display.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by gfxdev is
not valid.
010:035 EOS_MAUI_NOHWSUPPORT This function is not supported
by the hardware.
010:036  EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060  EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.

Indirect Errors
None

Driver Errors
0:208  EOS_UNKSVC
This feature is not supported by the hardware.

010:003  EOS_MAUI_BADCOLOTYPE
The color type bkcol is not supported by the driver.
_os_ss_gfx_bkcol()

See Also
gfx_clone_dev()
gfx_get_dev_cap()
gfx_get_dev_status()
gfx_open_dev()
gfx_set_display_extvid()
gfx_update_display()
GFX_COLOR
GFX_DEV_ID
gfx_set_display_extvid()

Set External Video On/Off

Syntax

error_code
gfx_set_display_extvid(GFX_DEV_ID gfxdev,
                  BOOLEAN extvid)

Description

gfx_set_display_extvid() sets external video on or off for the
display indicated by gfxdev. The effects of this function are not seen
until the next time you call gfx_update_display().

If this feature is not supported by the hardware, then this function
returns EOS_MAUI_NOHWSUPPORT.

If extvid is set to ON, then external video is visible in those areas of
the display that do not contain a viewport. External video is also visible
in transparent areas of viewports (see
gfx_set_display_transcol()).

If extvid is set to OFF (default), then a solid backdrop color is seen
instead of external video (see gfx_set_dev_attribute()).

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

gfxdev Graphics device.
extvid Sets external video On or Off.

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by gfxdev is
not valid.
Graphics Functions

010:035 EOS_MAUI_NOHWSUPPORT
This function is not supported by the hardware.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.

Indirect Errors
None

Driver Errors
0:208 EOS_UNKSVC
This feature is not supported by the hardware.

_os_ss_gfx_extvid()
MFM binding to the graphics driver.

See Also
gfx_clone_dev()
gfx_get_dev_cap()
gfx_get_dev_status()
gfx_open_dev()
gfx_set_dev_attribute()
gfx_set_display_transcol()
gfx_update_display()
BOOLEAN
GFX_DEV_ID
gfx_set_display_size()
Set Display Size

Syntax

error_code
gfx_set_display_size(GFX_DEV_ID gfxdev,
                      GFX_DIMEN width,
                      GFX_DIMEN height,
                      GFX_INTL_MODE intl_mode,
                      u_int16 refresh_rate)

Description

gfx_set_display_size() sets the parameters that affect the display size for gfxdev. The effects of this function are not seen until the next time you call gfx_update_display().

width and height set the horizontal and vertical dimensions respectively.

intl_mode sets the interlace mode and refresh_rate sets the refresh rate.

All supported display sizes are listed in the device capabilities.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

gfxdev
width
height
intl_mode
refresh_rate

Graphics device ID.
Horizontal size in pixels.
Vertical size in pixels.
Sets interlace mode.
Specifies the refresh rate.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by gfxdev is not valid.

010:016 EOS_MAUI_BADVALUE
An invalid value was specified for intl_mode.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.

Indirect Errors
None

Driver Errors

010:035 EOS_MAUI_NOHWSUPPORT
The combination of width, height, intl_mode, and refresh_rate is not supported by the hardware.

010:051 EOS_MAUI_CANTRESIZE
The display resolution cannot be changed if any viewports are active.

_os_ss_gfx_devres() MFM binding to the graphics driver.

See Also
gfx_clone_dev()
gfx_get_dev_cap()
gfx_get_dev_status()
gfx_open_dev()
gfx_update_display()
GFX_DEV_ID
GFX_DIMEN
GFX_INTL_MODE
gfx_set_display_transcol()
Set Transparent Color

Syntax

```c
error_code
gfx_set_display_transcol(GFX_DEV_ID gfxdev,
                        const GFX_COLOR *transcol)
```

Description

gfx_set_display_transcol() sets the transparent color for viewports on the display indicated by gfxdev. The effects of this function are not seen until the next time you call gfx_update_display().

If this feature is not supported by the hardware, then this function returns EOS_MAUI_NOHWSUPPORT.

If pixels in a viewport match the transparent color specified by transcol, then they are transparent. If transcol is NULL (this is the default), then no color is transparent.

The memory pointed to by transcol is not used by the API after returning from gfx_set_display_transcol().

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- gfxdev: Graphics device ID.
- *transcolor: Pointer to transparent color.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by gfxdev is not valid.

010:035 EOS_MAUI_NOHWSUPPORT
This function is not supported by the hardware.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.

Indirect Errors
None

Driver Errors

0:208 EOS_UNKSVC
This feature is not supported by the hardware.

010:003 EOS_MAUI_BADCOLORTYPE
The color type specified in transcol is not supported by the driver.

_os_ss_gfx_transcol()
MFM binding to the graphics driver.

See Also
gfx_clone_dev()
gfx_get_dev_cap()
gfx_open_dev()
gfx_get_dev_status()
gfx_update_display()
GFX_COLOR
GFX_DEV_ID
gfx_set_display_vpmix()
Set Viewport Mixing On/Off

Syntax

```c
error_code gfx_set_display_vpmix(GFX_DEV_ID gfxdev,
                                  BOOLEAN vpmix)
```

Description

gfx_set_display_vpmix() turns viewport mixing on or off for the display indicated by gfxdev. The effects of this function are not seen until the next time you call gfx_update_display().

If this feature is not supported by the hardware, then this function returns EOS_MAUI_NOHWSUPPORT.

If vpmix is set to ON, then when viewports overlap, the overlap area is a mix of all viewports in that area. The mixing is based on the relative intensities of the viewports. If set to OFF (default), then only the front-most viewport is seen in areas of overlap.

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- gfxdev: Graphics device ID.
- vpmix: Sets viewport mixing to On or Off.

Non-Fatal Errors

- 010:008 EOS_MAUI_BADID: The ID specified by gfxdev is not valid.
- 010:035 EOS_MAUI_NOHWSUPPORT: This function is not supported by the hardware.
Graphics Functions

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device gfxdev.

Indirect Errors
None

Driver Errors
0:208 EOS_UNKSVC
This feature is not supported by the hardware.

_os_ss_gfx_vpmix()
MFM binding to the graphics driver.

See Also
gfx_clone_dev()
gfx_get_dev_cap()
gfx_get_dev_status()
gfx_open_dev()
gfx_update_display()
BOOLEAN
GFX_DEV_ID
gfx_set_dmap_pixmem()

Set Pixel Memory Pointer in Drawmap

Syntax

```c
error_code
gfx_set_dmap_pixmem(GFX_DMAP *dmap,
                    GFX_PIXEL *pixmem,
                    u_int32 shade,
                    size_t size)
```

Description

gfx_set_dmap_pixmem() sets the pointer to the pixel memory used by the drawmap `dmap`. If the drawmap already has pixel memory assigned to it, you should deallocate it before calling this function.

If `pixmem` is `NULL`, then the pixel memory is allocated (by calling `mem_calloc()`) using the specified `shade` and `size`. In this case, if the `size` is zero, then it is computed using the current `width`, `height`, and coding method for the drawmap. These should be set with `gfx_set_dmap_size()` prior to calling this function.

If `pixmem` is not `NULL`, then it should point to the pixel memory to use for the drawmap. In this case `shade` and `size` should be set to the shade and size of the pre-allocated pixel memory.

It is the caller's responsibility to deallocate the memory allocated by this function (with `mem_free()` or `mem_sfree()`) before the drawmap is destroyed.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Parameters

*dmap
*pixmem
shade
size

Pointer to drawmap.
Pointer to pixel memory.
Memory shade for pixel memory.
Size of memory to allocate.

Fatal Errors

010:015 EOS_MAUI_BADSIZE
The size cannot be zero for pre-allocated pixel memory.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

Indirect Errors

gfx_calc_pixmem_size()
mem_calloc()

See Also

gfx_create_dmap()
gfx_set_dmap_size()
mem_free()
mem_sfree()
GFX_DMAP
GFX_PIXEL
gfx_set_dmap_size()
Set Coding Method and Size of Drawmap

Syntax
error_code
gfx_set_dmap_size(GFX_DMAP *dmap,
                GFX_CM coding_method,
                GFX_DIMEN width, GFX_DIMEN height)

Description
gfx_set_dmap_size() sets the coding method and size of a drawmap using the specified coding_method, width, and height in pixels.
dmap->line_size is computed using the parameters passed to this function. The line size is automatically rounded up to a multiple (in bytes) of GFX_LINE_PAD.
If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*dmap
Pointer to drawmap.
coding_method
Coding method for drawmap.
width
Horizontal size of drawmap in pixels.
height
Vertical size of drawmap in pixels.

Non-Fatal Errors
010:006 EOS_MAUI_BADDIMEN The specified height is zero.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with gfx_init().

**Indirect Errors**

gfx_calc_pixmem_size()

**See Also**

gfx_calc_pixmem_size()
gfx_create_dmap()
GFX_CM
GFX_DIMEN
GFX_DMAP
GFX_LINE_PAD
gfx_set_error_action()
Set Action to Take in Error Handler

Syntax

```c
error_code
gfx_set_error_action(MAUI_ERR_LEVEL debug_level,
                     MAUI_ERR_LEVEL passback_level,
                     MAUI_ERR_LEVEL exit_level)
```

Description

gfx_set_error_action() sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling gfx_init(). Following is the table of error levels. The least severe error is listed first.

Table 6-7  Error Levels for gfx_set_error_action()

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed, but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
</tbody>
</table>
debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default pass-back level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to call exit(). In this case the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- **debug_level**
  - Minimum error level that causes the error handler to print a message to standard error.
- **passback_level**
  - Minimum error level that causes the error handler to return the error.
- **exit_level**
  - Minimum error level that causes the error handler to call exit().

### Non-Fatal Errors

None
See Also

gfx_init()
gfx_set_vport_colors()
Set the Colors for a Viewport

Syntax

```c
error_code
gfx_set_vport_colors(GFX_VPORT_ID vport,
       u_int16 start_entry,
       u_int16 num_colors,
       GFX_COLOR_TYPE color_type,
       void *colors)
```

Description

gfx_set_vport_colors() sets the colors to be used within the viewport vport. This function is useful only for CLUT based coding methods. The effects of this function are not seen until the next time you call `gfx_update_display()`.

colors is a pointer to an array with `num_colors` entries. The format of each entry in the array is indicated by `color_type`. (e.g. If `color_type` is `GFX_RGB`, then the data pointed to by `colors` is of type `GFX_RGB`.)

`num_colors` entries in the hardware CLUT are updated starting with `start_entry`. Since different viewports may use the same CLUT entries for different colors, colors in other viewports may appear to be incorrect after calling this function.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Parameters

vport
  Viewport ID.

start_entry
  CLUT entry from which to begin updating.

num_colors
  Number of colors in the hardware CLUT to update.

color_type
  Format of color array.

*colors
  Array of color entries.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
  The ID specified by vport is not valid.

010:029 EOS_MAUI_MISSINGFEP
  Missing fast entry point in the graphics driver.

010:036 EOS_MAUI_NOINIT
  This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
  This is not the process that opened the viewport vport.

Indirect Errors

None

Driver Errors

_fe_set_vpcolors()
  MFM binding to the graphics driver.

See Also

gfx_create_vport()
gfx_update_display()
GFX_COLOR_TYPE
GFX_PALETTE
GFX_VPORT_ID
gfx_set_vport_dmap()
Set Drawmap to Use In a Viewport

Syntax

```
error_code
gfx_set_vport_dmap(GFX_VPORT_ID vport,
                    const GFX_DMAP *dmap,
                    GFX_POS x, GFX_POS y)
```

Description

gfx_set_vport_dmap() sets the drawmap dmap to be displayed in the viewport vport. If dmap is NULL, no drawmap is associated with the viewport.

The coordinates x and y specify the upper-left corner of the drawmap area to display. These coordinates are specified using the drawmap coordinate system. The effects of this function are not seen until the next time you call gfx_update_display().

If dmap->palette is not NULL then the hardware CLUT is updated with the complete range of entries defined by the palette. This ensures that the colors used by this drawmap are properly shown on the display. However, since different viewports may use the same CLUT entries for different colors, colors in other viewports may appear incorrectly.

If the contents of the drawmap object dmap are changed after calling this function, you must call it again to register the changes with this viewport object. If you delete the drawmap, it must be removed from this viewport.

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Parameters

vport
Graphics viewport ID.

*dmap
Pointer to drawmap.

x, y
X and Y drawmap coordinates of upper-left corner of drawmap to be displayed.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by vport is not valid.

010:029 EOS_MAUI_MISSINGFEP
Missing fast entry point in the graphics driver.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the viewport vport.

Indirect Errors
None

Driver Errors

010:002 EOS_MAUI_BADCODEMETH
The coding method for the drawmap dmap is not valid for this hardware.

010:003 EOS_MAUI_BADCOLORTYPE
A palette is specified for the drawmap, but its color type dmap->palette->color_type is not supported by the driver.

010:011 EOS_MAUI_BADPOS
The x and y position is not within the drawmap or is not valid for the hardware.
010:015 EOS_MAUİ_BADSIZE
The width or height of the drawmap dmap is not valid for this hardware.

010:018 EOS_MAUİ_CANTDISPLAY
vport is active and no pixel memory or the pixel memory with the drawmap cannot be displayed by this hardware.

010:021 EOS_MAUİ_DMAPTOOSMALL
vport is active and the width or height of dmap is not large enough to fill the viewport.

010:024 EOS_MAUİ_INUSE
dmap cannot be set to NULL if the vport is active.

010:043 EOS_MAUİ_NOTALIGNED
The pixel memory or line size in dmap is not aligned correctly for this hardware.

010:049 EOS_MAUİ_TOOCOMPLEX
vport is active and the viewport stack would become too complex for this hardware.

010:052 EOS_MAUİ_NOPALETTE
The drawmap assigned to the viewport does not contain a palette (dmap->palette is NULL), but one is required.

_ME_set_vpdmap()

See Also
gfx_create_dmap()
gfx_create_vport()
gfx_set_vport_colors()
gfx_update_display()
GFX_DMAP
GFX_POS
GFX_VPORT_ID
gfx_set_vport_dmpos()
Set Drawmap Position In a Viewport

Syntax

```c
error_code
gfx_set_vport_dmpos(GFX_VPORT_ID vport,
                     GFX_POS x, GFX_POS y)
```

Description

`gfx_set_vport_dmpos()` sets the position of the drawmap currently being displayed in the viewport `vport`. The coordinates `x` and `y` specify the upper-left corner of the drawmap area that should be displayed. These coordinates are specified using the drawmap coordinate system. The effects of this function are not seen until the next time you call `gfx_update_display()`.

If successful, this function returns `SUCCESS`.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

`vport` Graphics viewport ID.
`x`, `y` X and Y drawmap coordinates of the drawmap area displayed.

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by `vport` is not valid.
010:029 EOS_MAUI_MISSINGFEP Missing fast entry point in the graphics driver.
Graphics Functions

010:036 EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the viewport vport.

Indirect Errors
None

Driver Errors

010:011 EOS_MAUI_BADPOS
The x and y position is not within the drawmap or is not valid for the hardware.

010:021 EOS_MAUI_DMAPTOOSMALL
vport is active and the width or height of dmap is not large enough to fill the viewport.

010:031 EOS_MAUI_NODMAP
No drawmap is currently mapped to the viewport vport.

_fe_set_vpadmpos()

_MFM binding to the graphics driver.

See Also
gfx_create_vport()
gfx_update_display()
GFX_POS
GFX_VPORT_ID
gfx_set_vport_intensity()
Set Viewport Intensity

Syntax

type error_code
gfx_set_vport_intensity(GFX_VPORT_ID vport,
                    u_int8 intensity)

Description

gfx_set_vport_intensity() sets the intensity of the specified viewport vport. The intensity must be a value in the range 0 to 100 where 0 is black and 100 is full intensity. The effects of this function are not seen until the next time gfx_update_display() is called.

If this feature is not supported by the hardware, this function returns EOS_MAUI_NOHWSUPPORT.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

vport Graphics viewport ID.
intensity Sets intensity of viewport from 0 to 100.

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by gfxdev is not valid.
010:016 EOS_MAUI_BADVALUE The intensity is not in the range 0 to 100.
010:035 EOS_MAUI_NOHWSUPPORT This function is not supported by the hardware.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with `gfx_init()`.

010:060 EOS_MAUI_NOTOWNER

This is not the process that opened the viewport `vport`.

**Indirect Errors**

None

**Driver Errors**

 `_fe_set_vpinten()`  
MFM binding to the graphics driver.

**See Also**

`gfx_create_vport()`  
`gfx_get_dev_cap()`  
`gfx_get_vport_status()`  
`gfx_update_display()`  
`GFX_VPORT_ID`
gfx_set_vport_position()
Set the Position of a Viewport

Syntax

```c
error_code
gfx_set_vport_position(GFX_VPORT_ID vport,
                        GFX_POS x, GFX_POS y)
```

Description

gfx_set_vport_position() moves the viewport vport to the position on the display specified by x and y. This position is specified using the display coordinate system. The effects of this function are not seen until the next time you call gfx_update_display().

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `vport`: Viewport ID.
- `x, y`: Position of the viewport in display coordinate system.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**: The ID specified by vport is not valid.
- **010:029 EOS_MAUI_MISSINGFEP**: Missing fast entry point in the graphics driver.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with gfx_init().
- **010:060 EOS_MAUI_NOTOWNER**: This is not the process that opened the viewport vport.
Indirect Errors

_fe_set_vppos() MFM binding to the graphics driver.

Driver Errors

010:006 EOS_MAUI_BADDIMEN

_vport is active and the new position would make the current viewport dimensions too large for the display.

010:011 EOS_MAUI_BADPOS

The x and y position is not within the display.

010:049 EOS_MAUI_TOOCOMPLEX

_vport is active and the viewport stack would become too complex for this hardware.

See Also

gfx_create_vport()
gfx_get_vport_status()
gfx_update_display()
GFX_POS
GFX_VPORT_ID
gfx_set_vport_size()
Set the Size of a Viewport

Syntax
error_code
gfx_set_vport_size(GFX_VPORT_ID vport,
                      GFX_DIMEN width,
                      GFX_DIMEN height)

Description
gfx_set_vport_size() changes the size of viewport vport using width and height. The size is specified using the display coordinate system. The effects of this function are not seen until the next time you call gfx_update_display().

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
vport Graphics viewport ID.
width Horizontal size of viewport in pixels.
height Vertical size of viewport in pixels.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by vport is not valid.
010:029 EOS_MAUI_MISSINGFEP Missing fast entry point in the graphics driver.
010:036 EOS_MAUI_NOINIT This API has not been initialized with gfx_init().
010:060 EOS_MAUI_NOTOWNER

This is not the process that opened the viewport vport.

Indirect Errors
None

Driver Errors

010:006 EOS_MAUI_BADDIMEN

The width or height is zero or an illegal size for the display.

010:021 EOS_MAUI_DMAPTOOSMALL

vport is active and the viewport would become too large for the drawmap.

010:049 EOS_MAUI_TOOCOMPLEX

vport is active and the viewport stack would become too complex for this hardware.

_fe_set_vpsize()

MFM binding to the graphics driver.

See Also

gfx_create_vport()
gfx_get_vport_status()
gfx_update_display()
GFX_DIMEN
GFX_VPORT_ID
gfx_set_vport_state()
Set the State of a Viewport

Syntax
error_code
gfx_set_vport_state(GFX_VPORT_ID vport,
              BOOLEAN active)

Description
gfx_set_vport_state() sets the state of the viewport vport. The
position of vport within the viewport stack is not changed. The effects
of this function are not seen until the next time you call
gfx_update_display().
If active is set to TRUE, then the viewport is activated (made visible). If
active is set to FALSE, then the viewport is deactivated (no longer
visible).
If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
vport Graphics viewport ID.
active Activates (TRUE) or deactivates (FALSE)
the viewport.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by vport is
not valid.
010:029 EOS_MAUI_MISSINGFEP Missing fast entry point in the
graphics driver.
010:036  EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060  EOS_MAUI_NOTOWNER
This is not the process that opened the viewport vport.

Indirect Errors
None

Driver Errors
010:006  EOS_MAUI_BADDIMEN
vport is being activated and the dimensions of the viewport would make it too large to fit on the display.

010:021  EOS_MAUI_DMAPOOSMALL
vport is being activated and the drawmap within the viewport is too small to fill the viewport.

010:031  EOS_MAUI_NODMAP
vport is being activated and no drawmap is associated with it.

010:049  EOS_MAUI_TOOCOMPLEX
The viewport stack would become too complex for this hardware.

_fe_set_vpstate()

_MFM binding to the graphics driver.

See Also
gfx_create_vport()
gfx_get_vport_status()
gfx_update_display()
BOOLEAN
GFX_VPORT_ID
gfx_sync_retrace()

Synchronize with Vertical Retrace

Syntax

```c
error_code
gfx_sync_retrace(GFX_DEV_ID gfxdev)
```

Description

gfx_sync_retrace() is used to synchronize the caller with the vertical retrace period of the graphics device specified by gfxdev. This function waits until vertical retrace starts, then returns to the caller. If the hardware is already in vertical retrace, this function waits until the start of the next one.

If this feature is not supported by the hardware, this function returns EOS_MAUI_NOHWSUPPORT.

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

gfxdev Graphics device ID.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**
  - The ID specified by vport is not valid.
- **010:035 EOS_MAUI_NOHWSUPPORT**
  - This function is not supported by the hardware.
- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with gfx_init().
- **010:060 EOS_MAUI_NOTOWNER**
  - This is not the process that opened the viewport vport.
Indirect Errors

_os_ev_wait()
_os9_ev_wait()

See Also
gfx_get_dev_cap()
GFX_DEV_ID

See the *Ultra C Library Reference.*
gfx_term()

Terminate the Graphics Device API

Syntax

```c
error_code
gfx_term(void)
```

Description
gfx_term() terminates the Graphics Device API. All open graphic devices are closed by calling gfx_close_dev() for each one, then all other internal resources in use by the API are returned to the system.

Since this API depends on the Shaded Memory API, mem_term() is called by this function.

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Non-Fatal Errors

010:036 EOS_MAUI_NOINIT This API has not been initialized with gfx_init().

Indirect Errors
gfx_close_dev() mem_free() mem_term()

See Also
gfx_init()
gfx_update_display()

Update the Display

Syntax

```c
error_code
gfx_update_display(GFX_DEV_ID gfxdev, BOOLEAN sync)
```

Description

gfx_update_display() updates the display with any queued up changes. The list of functions that queue up changes to the display are shown in the See Also section that follows.

The only functions that queue up are changes to display parameters or changes to the configuration of a viewport. Changes to a drawmap within a viewport, such as drawing, do not queue up.

If `sync` is `TRUE`, then the update is synchronized with vertical retrace if such a feature is supported by the hardware. If `sync` is `FALSE` or the driver does not have the ability to synchronize to vertical retrace, then the update happens immediately.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `gfxdev`: Graphics device ID.
- `sync`: Sets the update to occur with the vertical retrace (`TRUE`) or immediately (`FALSE`).

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**
  The ID specified by `gfxdev` is not valid.
6 Graphics Functions

010:029 EOS_MAUI_MISSINGFEP

Missing fast entry point in the graphics driver.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with gfx_init().

010:060 EOS_MAUI_NOTOWNER

This is not the process that opened the device gfxdev.

000:233 EOS_SIGNAL

Signal error.

Indirect Errors
None

Driver Errors

_fe_update_dpy() MFM binding to the graphics driver.

See Also
gfx_open_dev()
gfx_restack_vport()
gfx_set_dev_attribute()
gfx_set_display_extvid()
gfx_set_display_size()
gfx_set_display_transcol()
gfx_restack_dev()
gfx_set_display_vpmix()
gfx_set_vport_colors()
gfx_set_vport_dmap()
gfx_set_vport_dmpos()
gfx_set_vport_intensity()
gfx_set_vport_position()
gfx_set_vport_size()
gfx_set_vport_state()
gfx_sync_retrace()
BOOLEAN
GFX_DEV_ID
inp_check_keys()
Check a Range of Key Symbols

Syntax

```c
error_code
inp_check_keys(BOOLEAN *ret_all_present,
               INP_DEV_ID inpdev, wchar_t min_key,
               wchar_t max_key)
```

Description

`inp_check_keys()` checks to confirm that all the keys in the range `min_key` to `max_key` (inclusive) exist on the device `inpdev`. The result of the check is returned in `ret_all_present`. A pointer to this variable should be passed to `inp_get_dev_cap()`. If all the keys in the specified range are present on the device, this value is set to `TRUE`. Otherwise it is set to `FALSE`. If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `*ret_all_present` Pointer to the result of the check.
- `inpdev` Device ID of the input device.
- `min_key` Minimum key to check.
- `max_key` Maximum key to check.

Non-Fatal Errors

- 010:001 EOS_MAUI_BADACK The command code is not understood by the protocol module.
- 010:008 EOS_MAUI_BADID The ID specified by `inpdev` is not valid.
010:013 EOS_MAUI_BADRANGE min_key is greater than max_key.
010:036 EOS_MAUI_NOINIT This API has not been initialized with inp_init().

**Indirect Errors**
msg_read()
msg_write()
MSG_CHECK_KEYS

**See Also**
inp_get_dev_cap()
INP_DEV_ID
BOOLEAN
inp_close_dev()
Close an Input Device

Syntax

```c
error_code
inp_close_dev(INP_DEV_ID inpdev)
```

Description

`inp_close_dev()` closes the device `inpdev`. Messages from this device are no longer written to the application’s mailbox. Messages already in the mailbox are unaffected.

If any keys have been reserved for this device ID, they are automatically released before the device is closed.

Closing the device removes this application from the stack of applications using the device. If this application was the top-most application (the one with the focus), the focus shifts to the process immediately below it in the stack.

If successful, this function returns `SUCCESS`.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

Parameters

- `inpdev`: Device ID of the input device.

Non-Fatal Errors

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from the MAUI Input Process.
- **010:008 EOS_MAUI_BADID**: The ID specified by `inpdev` is not valid.
- **010:019 EOS_MAUI_DAMAGE**: Data structures are damaged.
Input Functions

010:036 EOS_MAUI_NOINIT

This API has not been initialized with inp_init().

Indirect Errors

mem_free()
msg_close_mbox()
msg_read()
msg_write()
_os_close()
_os_unlink()
MSG_CLOSE_DEV

See Ultra C Library Reference.
See Ultra C Library Reference.
See MAUI Porting Guide

See Also

inp_open_dev()
inp_restack_dev()
INP_DEV_ID
inp_get_dev_cap()
Get Input Device Capabilities

Syntax

```
error_code
inp_get_dev_cap(INP_DEV_CAP *ret_dev_cap,
               INP_DEV_ID inpdev)
```

Description

`inp_get_dev_cap()` gets information about the capabilities of the input device `inpdev`. This information may be used to adjust the operation of the application so that it runs properly on different hardware platforms.

The device capabilities structure is returned in `ret_dev_cap`. A pointer to this structure should be passed to `inp_get_dev_cap()`. The caller must ensure that `ret_dev_cap` points to storage large enough to hold the information.

If successful, this function returns `SUCCESS`.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

* `*ret_dev_cap` Pointer to variable storing device capabilities.
* `inpdev` Device ID of the input device.

Non-Fatal Errors

010:001 EOS_MAUI_BADACK
The command code is not understood by the protocol module.
010:008  EOS_MAUI_BADID

The ID specified by inpdev is not valid.

010:036  EOS_MAUI_NOINIT

This API has not been initialized with inp_init().

Indirect Errors

msg_read()
msg_write()

See Also

inp_check_keys()
INP_DEV_CAP
INP_DEV_ID
inp_get_dev_status()
Get Input Device Status

Syntax
error_code
inp_get_dev_status(INP_DEV_STATUS *ret_dev_status,
                   INP_DEV_ID inpdev)

Description
inp_get_dev_status() returns the current status of the specified
device inpdev.

The device status is returned in ret_dev_status. A pointer to this
structure should be passed to inp_get_dev_status(). The caller
must ensure that ret_dev_status points to storage large enough to
hold the information.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_dev_status Pointer to variable storing the device
status.
inpdev Device ID of the input device.

Non-Fatal Errors
010:001 EOS_MAUI_BADACK The command code is not
understood by the protocol module.
010:008 EOS_MAUI_BADID The ID specified by inpdev is
not valid.
010:036  EOS_MAUI_NOINIT

This API has not been initialized with inp_init().

**Indirect Errors**

msg_read()
msg_write()

MSG_GET_DEV_STATUS  See *MAUI Porting Guide.*

**See Also**

inp_set_ptr_limit()
inp_set_ptr_pos()
inp_set_sim_meth()
INP_DEV_ID
INP_DEV_STATUS
inp_init()

Initialize the Input Device API

**Syntax**

```c
error_code
inp_init(void)
```

**Description**

`inp_init()` initializes the Input Device API. This function must be called prior to a call to any other input function unless otherwise noted by that function.

Communication with the MAUI Input Process is established by opening the command mailbox named `mp_mbox` and creating a reply mailbox named `mp%08x` where `%08x` is an 8 digit number (padded with zeros) whose value is the process ID of the process calling `inp_init()`.

This API depends on the Shaded Memory and Messaging APIs. Therefore, `mem_init()` and `msg_init()` are called by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Fatal Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 010:004  | **EOS_MAUI_BADCOMPATLEVEL**  
  Bad compatibility level reported by the MAUI Input Process. |
| 010:038  | **EOS_MAUI_NOMAUIP**  
  The MAUI Input Process is not running. |
**Indirect Errors**

- `mem_init()`
- `msg_create_mbox()`
- `msg_init()`
- `msg_open_mbox()`
- `msg_read()`
- `msg_write()`
- `_os_id()`

See [Ultra C Library Reference](#).

**See also**

- `inp_term()`
### inp_open_dev()

**Open an Input Device**

**Syntax**

```c
error_code
inp_open_dev(INP_DEV_ID *ret_inpdev,
             MSG_MBOX_ID mbox,
             const char *device_name)
```

**Description**

`inp_open_dev()` opens an input device and associates it with the mailbox named `mbox`. Messages generated from this device are written to the mailbox.

`device_name` specifies both the device and MAUI Input Process Protocol Module names. This protocol module translates raw input from the device into pointer and/or key symbol messages for the application. The format of this string is “/device/protocol”.

The mailbox must be opened using `msg_create_mbox()` or `msg_open_mbox()` prior to calling `inp_open_dev()`.

The device ID is returned in `ret_inpdev`. A pointer to this variable should be passed to `inp_open_dev()`. Use `inp_close_dev()` when this device is no longer needed.

Opening the device places this application at the top of the stack of applications using the device. This causes the focus for the device to move to the application calling `inp_open_dev()`.

If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
Input Functions

Parameters

*ret_inpdev
  Pointer to variable storing the device ID inpdev.

mbox
  Mailbox associated with this inpdev.

*device_name
  Pointer to the device specification and MAUI Input Process Protocol Module name.

Fatal Errors

010:001 EOS_MAUI_BADACK
  Bad acknowledgment from the MAUI Input Process.

010:004 EOS_MAUI_BADCOMPATLEVEL
  Bad compatibility level reported by the MAUI process protocol module.

010:019 EOS_MAUI_DAMAGE
  MAUI has detected that its data structures are damaged. This problem is usually caused by the use of un-initialized or improperly initialized pointers.

010:036 EOS_MAUI_NOINIT
  This API has not been initialized with inp_init().

010:050 EOS_MAUI_TOOLONG
  The device name device_name is too long. The maximum length is INP_MAX_DEV_NAME.

Non-Fatal Errors

010:040 EOS_MAUI_NOPMOD
  Could not find protocol module name in device_name.

Indirect Errors

mem_calloc()
mem_calloc()
Input Functions

msg_get_mbox_status()
msg_open_mbox()
msg_read()
msg_write()
_os_link()
_os_open()

See Ultra C Library Reference.

See Ultra C Library Reference.

See Also

cdb_get_ddr()
inp_close_dev()
msg_create_mbox()
msg_open_mbox()
MSG_RESTACK_DEV()

See Maui Porting Guide.

INP_DEV_ID
INP_MAX_DEV_NAME
MSG_MBOX_ID
**inp_release_key()**

Release a Key

### Syntax

```c
error_code
inp_release_key(INP_DEV_ID inpdev, wchar_t key)
```

### Description

`inp_release_key()` releases the key specified by `key`. If the key is not currently reserved, then `EOS_MAUI_NOTRESERVED` is returned by the MAUI Input Process.

If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- **inpdev**
  Device ID of the input device.
- **key**
  Identification of key to release.

### Non-Fatal Errors

- **010:001 EOS_MAUI_BADACK**
  The command code is not understood by the protocol module.

- **010:008 EOS_MAUI_BADID**
  The ID specified by `inpdev` is not valid.

- **010:035 EOS_MAUI_NOHWSUPPORT**
  The specified `key` is not supported by (present on) this device.

- **010:036 EOS_MAUI_NOINIT**
  This API has not been initialized with `inp_init()`.
010:047 EOS_MAUI_NOTRESERVED

The specified key is not currently reserved by this device.

**Indirect Errors**

msg_read()

msg_write()

**See Also**

inp_reserve_key()

INP_DEV_ID
# inp_reserve_key()

## Reserve a Key

**Syntax**

```c
error_code
inp_reserve_key(INP_DEV_ID inpdev, wchar_t key)
```

**Description**

`inp_reserve_key()` reserves the key specified by `key`. If the key is already reserved by another, then `EOS_MAUI_ISRESERVED` is returned by the MAUI Input Process.

The keys remain reserved by the mailbox associated with this device until they are released by calling `inp_release_key()` or the device is closed.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `inpdev`: Device ID of the input device.
- `key`: Identification of key to reserve.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**: The command code is not understood by the protocol module.
- **010:008 EOS_MAUI_BADID**: The ID specified by `inpdev` is not valid.
- **010:026 EOS_MAUI_ISRESERVED**: The specified `key` is already reserved by another device.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with inp_init().

Indirect Errors
msg_read()
msg_write()

See Also
inp_release_key()
INP_DEV_ID
inp_restack_dev()
Re-stack an Input Device

Syntax

```c
error_code
inp_restack_dev(INP_DEV_ID inpdev,
                INP_DEV_PLACEMENT placement, ...)
```

Description

This function changes the placement of the input device `inpdev` in the current stack of input devices.

The following table shows how `placement` specifies the new position. The Parameter column shows the types of the additional parameters (represented by “...” above).

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Parameter</th>
<th>New Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_DEV_FRONT</td>
<td>None</td>
<td>In front of all devices</td>
</tr>
<tr>
<td>INP_DEV_BACK</td>
<td>None</td>
<td>In back of all devices</td>
</tr>
<tr>
<td>INP_DEV_FRONT_OF</td>
<td>INP_DEV_ID</td>
<td>In front of device</td>
</tr>
<tr>
<td>ref_inpdev</td>
<td>ref_inpdev</td>
<td></td>
</tr>
<tr>
<td>INP_DEV_BACK_OF</td>
<td>INP_DEV_ID</td>
<td>In back of device</td>
</tr>
<tr>
<td>ref_inpdev</td>
<td>ref_inpdev</td>
<td></td>
</tr>
</tbody>
</table>

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Input Functions

Parameters
inpdev

Device ID of input device.

placement

Placement of inpdev in stack of input devices.

... Optional additional parameters for placement.

Non-Fatal Errors

010:001 EOS_MAUI_BADACK
Bad acknowledgment from the MAUI input process.

010:008 EOS_MAUI_BADID
The ID specified by inpdev is not valid.

010:016 EOS_MAUI_BADVALUE
The value used for placement is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with inp_init().

010:060 EOS_MAUI_NOTOWNER
This is not the process that opened the device inpdev.

Indirect Errors

MSG_RESTACK_DEV
See MAUI Porting Guide.

See Also
inp_open_dev() INP_DEV_ID INP_DEV_PLACEMENT
inp_set_callback()
Set Callback for Queuing Messages

Syntax

```c
error_code
inp_set_callback(INP_DEV_ID inpdev,
    void (*callback)(const void *))
```

Description

`inp_set_callback()` sets the callback used when queuing messages to the mailbox for the device `inpdev`. `callback` is placed in the `msg->callback` field when messages are written to the mailbox.

This only affects messages written by the MAUI Input Process on behalf of the specified device ID. Since each process does its own `inp_open_dev()`, each process is able to specify its own callback.

Calling this function only affects future writes to the mailbox. Messages that are already in the mailbox are not affected. The default value, `NULL`, is used if you do not call this function.

When the application reads messages (using `msg_read()`) generated from this device, they will have the specified `callback` present in the message. When this message is passed to `msg_dispatch()`, it calls the application-supplied `callback` function.

The `callback` function `callback` is defined by the caller and its prototype should appear as follows:

```c
error_code callback(const void *msg)
```

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Parameters

- `inpdev` : Device ID of the input device.
- `*callback` : Pointer to the callback to use.

Non-Fatal Errors

- **010:001 EOS_MAUI_BADACK**
  - The command code is not understood by the protocol module.

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `inpdev` is invalid.

- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `inp_init()`.

Indirect Errors

- `MSG_SET_MSG_CALLBACK` : See *Maui Porting Guide*.

See Also

- `inp_get_dev_status()`
- `inp_open_dev()`
- `msg_dispatch()`
- `msg_read()`
- `INP_DEV_ID`
- `MSG_COMMON`
**inp_set_error_action()**

Set Action to Take in Error Handler

---

**Syntax**

```c
error_code
inp_set_error_action(MAUI_ERR_LEVEL debug_level,
                     MAUI_ERR_LEVEL passback_level,
                     MAUI_ERR_LEVEL exit_level)
```

**Description**

`inp_set_error_action()` sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling `inp_init()`. Following is the table of error levels. The least severe error is listed first.

**Table 7-2 Error Levels for inp_set_error_action()**

<table>
<thead>
<tr>
<th>Error Levels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
</tbody>
</table>
debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default pass-back level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to call exit(). In this case the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

debug_level Minimum error level that causes the error handler to print a message to standard error.

passback_level Minimum error level that causes the error handler to return the error.

exit_level Minimum error level that causes the error handler to call exit().

---

### Table 7-2 Error Levels for inp_set_error_action()

<table>
<thead>
<tr>
<th>Error Levels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>
Non-Fatal Errors
None

See Also
inp_init()
inp_set_msg_mask()
Set Mask for Queuing Messages

Syntax

```c
error_code
inp_set_msg_mask(INP_DEV_ID inpdev, u_int32 mask)
```

Description

`inp_set_msg_mask()` sets the mask used when queuing messages to the mailbox for the device `inpdev`. Only message types included in this mask (see `MSG_TYPE`) are queued.

This only affects messages written by the MAUI Input Process on behalf of the specified device ID. Since each process does its own `inp_open_dev()`, each process is able to specify its own mask.

Calling this function only affects future writes to the mailbox. Messages that are already in the mailbox are not affected. The default mask, `MSG_TYPE_ANY`, is used if you do not call this function.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `inpdev` Device ID of the input device.
- `mask` `MSG_TYPE` mask to use when queuing messages to the mailbox.

Non-Fatal Errors

- `010:001 EOS_MAUI_BADACK` Bad acknowledgment from the MAUI Input Process.
- `010:008 EOS_MAUI_BADID` The ID specified by `inpdev` is not valid.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with `inp_init()`.

**Indirect Errors**

MSG_SET_MSG_MASK

**See Also**

`inp_get_dev_status()`
`inp_open_dev()`
`INP_DEV_ID`
`MSG_TYPE`
**inp_set_ptr_limit()**

**Set Pointer Limit**

**Syntax**

```c
error_code
inp_set_ptr_limit(INP_DEV_ID inpdev, GFX_POS minx,
                 GFX_POS miny, GFX_POS maxx,
                 GFX_POS maxy)
```

**Description**

`inp_set_ptr_limit()` limits the movement of the pointer to the area defined by `minx`, `miny` and `maxx`, `maxy`. By default `minx` and `miny` are set to `GFX_POS_MIN`, and `maxx` and `maxy` are set to `GFX_POS_MAX`. If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `inpdev`: Device ID of the input device.
- `minx`: X screen coordinate of upper-left pointer limit.
- `miny`: Y screen coordinate of upper-left pointer limit.
- `maxx`: X screen coordinate of lower-right pointer limit.
- `maxy`: Y screen coordinate of lower-right pointer limit.
Input Functions

**Non-Fatal Errors**

010:001 EOS_MAUI_BADACK
The command code is not understood by the protocol module.

010:008 EOS_MAUI_BADID
The ID specified by inpdev is not valid.

010:013 EOS_MAUI_BADRANGE
Either minx is greater than maxx or miny is greater than maxy.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with inp_init().

**Indirect Errors**

msg_read()
msg_write()
MSG_SET_PTR_LIMIT

See *Maui Porting Guide*.

**See Also**

inp_get_dev_status()
GFX_POS
INP_DEV_ID
inp_set_ptr_pos()

Set Pointer Position

**Syntax**

```c
error_code
inp_set_ptr_pos(INP_DEV_ID inpdev, GFX_POS x,
                GFX_POS y)
```

**Description**

`inp_set_ptr_pos()` moves the pointer to the position specified by `x, y`. The pointer position is updated automatically whenever cursor movement information is received from the device `inpdev`. This function is used to allow the application to reposition the pointer without requiring any input from the device.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **inpdev**
  - Device ID of the input device.
- **x**
  - X screen coordinate of pointer position.
- **y**
  - Y screen coordinate of pointer position.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**
  - The command code is not understood by the protocol module.
- **010:008 EOS_MAUI_BADID**
  - The ID specified by `inpdev` is not valid.
- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `inp_init()`.
Input Functions

**Indirect Errors**

msg_read()
msg_write()
MSG_SET_PTR_POS

See *Maui Porting Guide*.

**See Also**

inp_get_dev_status()
GFX_POS
INP_DEV_ID
Syntax

```c
error_code
inp_set_sim_meth(INP_DEV_ID inpdev,
                 u_int8 num_buttons,
                 const wchar_t keysyms[],
                 INP_SIM_METH sim_meth, ...)
```

Description

`inp_set_sim_meth()` sets the simulation method `sim_meth` to use for the specified device `inpdev`.

`num_buttons` specifies the number of pointer buttons that have associations with key symbols. If zero, no associations are made. However, if in the range 1 to `INP_MAX_BUTTONS`, it indicates the number of associations to make.

If `num_buttons` is non-zero, then `keysyms` must hold that number of entries. Each entry in the `keysyms` array specifies an association between buttons and key symbols. The first entry in the array maps to button one. The next entry to button two, and so forth.

If one or more elements in the `keysyms` array is zero, then those entries indicate that no association should be made for that button. This is also the method used to remove an association made with a previous call to this function.

If the contents of the `keysyms` array are changed after calling this function, you must call it again to register the changes with the MAUI Input Process. Since this function makes a copy of the data pointed to by `keysyms`, you may destroy `keysyms` immediately after calling `inp_set_sim_meth()`.
The following table shows how `sim_meth` may be used to specify the type of simulation to use. The Parameter column shows the types of the additional parameters (represented by “...” shown above).

<table>
<thead>
<tr>
<th>Value of sim_meth</th>
<th>Parameter</th>
<th>Type of Message Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_SIM_NATIVE</td>
<td>None</td>
<td>No translation (default)</td>
</tr>
<tr>
<td>INP_SIM_PTR</td>
<td>GFX_OFFSET horz_speed</td>
<td>Pointer messages</td>
</tr>
<tr>
<td></td>
<td>GFX_OFFSET vert_speed</td>
<td></td>
</tr>
<tr>
<td>INP_SIM_KEY</td>
<td>GFX_OFFSET horz_speed</td>
<td>Key symbol messages</td>
</tr>
<tr>
<td></td>
<td>GFXOFFSET vert_speed</td>
<td></td>
</tr>
</tbody>
</table>

**For More Information**

See `INP_SIM_METH` for details about the meaning of each of these parameters.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
Parameters

inpdev
Device ID of the input device.

num_buttons
Number of pointer buttons that have associations with key symbols.

keysyms[]
Array that specifies associations between buttons and key symbols.

sim_meth
Specifies the type of simulation to use.

Optional additional parameters for sim_meth.

Non-Fatal Errors

010:001 EOS_MAUI_BADACK
The command code is not understood by the protocol module.

010:008 EOS_MAUI_BADID
The ID specified by inpdev is invalid.

010:016 EOS_MAUI_BADVALUE
The value for sim_meth is not legal, or num_buttons is not in range.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with inp_init().

Indirect Errors

msg_read()
msg_write()
MSG_SET_SIM_METH
See MAUI Porting Guide.

See Also

inp_get_dev_status()
INP_SIM_METH
INP_DEV_ID
inp_term()

Terminate use of the Input Device API

Syntax

error_code
inp_term(void)

Description

inp_term() terminates the Input Device API and closes the command and reply mailboxes used to communicate with the MAUI Input Process. This function automatically closes all input devices that are still open for this process.

This API depends on the Shaded Memory and Messaging APIs. Therefore, mem_term() and msg_term() are called by this function.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Non-Fatal Errors

010:001 EOS_MAUI_BADACK Bad acknowledgment from the MAUI Input Process.
010:019 EOS_MAUI_DAMAGE Data structures are damaged.
010:036 EOS_MAUI_NOINIT This API has not been initialized with inp_init().

Indirect Errors

mem_term()
msg_close_mbox()
msg_read()
msg_term()
msg_write()
MSG_INP_TERM

See Also
inp_init()
Chapter 8: MAUI System Functions
maui_init()
Initialize the MAUI APIs

Syntax

error_code
maui_init(void)

Description

maui_init() initializes the MAUI APIs. This function calls the
_*_init()_ function of each MAUI API.

maui_init is a convenience function to initialize all the MAUI APIs with a
single call. Because this call cannot determine the specific needs of the
application, it may result in the allocation of system resources that are
beyond the actual needs of the application. It is a better programming
practice to initialize the individual API's used by the application.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe

Parameters

None

Non-Fatal Errors

010:025 EOS_MAUI_ISINIT This function has already been
called.

Indirect Calls

Errors from the following calls are not checked or returned by
maui_init(), but the following calls could print error messages to the
console. By not checking or returning errors, an application can use
maui_init() to initialize API's that are available while ignoring errors
from API's that are not available in a particular implementation. If an application is concerned about the return status of specific API's, then the application should call the individual API's init function before calling maui_init().

anm_init()
blt_init()
cdb_init()
drw_init()
gfx_init()
ingp_init()
mem_init()
msg_init()
txt_init()
win_init()

See Also
maui_term()
maui_set_error_action()
Set Action to Take in Error Handler

Syntax

error_code
maui_set_error_action(MAUI_ERR_LEVEL debug_level,
                      MAUI_ERR_LEVEL passback_level,
                      MAUI_ERR_LEVEL exit_level)

Description

maui_set_error_action() sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling maui_init().

After setting this API’s error levels, maui_set_error_action() calls the *_set_error_action() function of every other MAUI API with the parameters specified in maui_set_error_action(). Following is the table of error levels. The least severe error is listed first.

Table 8-1  Error Levels

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
</tbody>
</table>

debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default pass-back level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to exit(). In this case the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

### Attributes

**Operating System:** OS-9 and OS-9 for 68K  
**State:** User and System  
**Threads:** Safe

### Parameters

**debug_level**  
Minimum error level that causes the error handler to print a message to standard error.

### Table 8-1  Error Levels (continued)

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>
passback_level

exit_level

**Non-Fatal Errors**

None

**Indirect Calls**

- anm_set_error_action()
- blt_set_error_action()
- cdb_set_error_action()
- drw_set_error_action()
- gfx_set_error_action()
- inp_set_error_action()
- mem_set_error_action()
- msg_set_error_action()
- txt_set_error_action()
- win_set_error_action()

**See Also**

- maui_init()
- MAUI_ERR_LEVEL
maui_term()

Terminate the MAUI APIs

Syntax

error_code
maui_term(void)

Description

maui_term() terminates the MAUI System API. This function calls the *
_term() function of each MAUI API.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe

Parameters

None

Non-Fatal Errors

010:036 EOS_MAUI_NOINIT This API has not been initialized with maui_init().

Indirect Calls

anm_term()
blt_term()
cdb_term()
drw_term()
gfx_term()
inp_term()
mem_term()
msg_term()
txt_term()
win_term()
See Also

maui_init()
Chapter 9: Shaded Memory Functions
Syntax

```c
error_code
mem_calloc(void *ret_ptr, u_int32 shade_id,
             size_t num_entries, size_t entry_size)
```

Description

`mem_calloc()` allocates and clears (to zeros) space for an array. The size of the allocation is

\[
\text{(num\_entries} \times \text{entry\_size})
\]

rounded up to a multiple of `MEM_MIN_ALLOC`.

The allocation is satisfied from the specified `shade_id`. Since the CPU cannot write to a pseudo memory, if a shade is used, the memory is not cleared. In this case, `mem_calloc()` operates like `mem_malloc()`.

The memory is allocated from `shade_id` and a pointer to it is returned in `ret_ptr`. A pointer to `ret_ptr` should be passed to `mem_calloc()`. Use `mem_free()` or `mem_sfree()` to de-allocate this segment when it is no longer needed.

The segment allocated is guaranteed to start on, and have a length that is a multiple of, the boundary size for the shade. See `mem_set_alloc_bndry()` for setting the boundary size.

Allocation functions (`* alloc_func`) set with `mem_set_alloc()` may return errors as indicated below.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
Shaded Memory Functions

**Fatal Errors**

010:036 EOS_MAUI_NOINIT

This API has not been initialized with `mem_init()`.

010:014 EOS_MAUI_BADSHADE

The specified `shade_id` has not been defined or is not a normal shade.

010:015 EOS_MAUI_BADSIZE

Either `num_entries` or `entry_size` is zero.

010:030 EOS_MAUI_NOCALLBACK

`shade_id` is a pseudo shade, but no allocation function has been specified.

**Indirect Errors**

`(*alloc_func)()`

`_os_srqmem()`

See *Ultra C Library Reference*.

**See Also**

`mem_create_shade()`
`mem_free()`
`mem_malloc()`
`mem_realloc()`
`mem_set_alloc_bndry()`
`mem_sfree()`
`MEM_MIN_ALLOC`
mem_create_shade()
Create a Shade

Syntax

```c
error_code
mem_create_shade(u_int32 shade_id,
                 MEM_SHADE_TYPE shade_type,
                 u_int32 color,
                 size_t initial_size,
                 size_t grow_size,
                 MEM_OVTYPE ovtype,
                 BOOLEAN overflow_detect)
```

Description

`mem_create_shade()` creates the specified `shade_id`. The shade type is `shade_type`.

Memory blocks for this shade are provided by the allocator and deallocator functions for the shade. The default allocator is `_os_srqmem()`, and the default deallocator is `_os_srtmem()`. You may specify your own functions by calling `mem_set_alloc()` and `mem_set_dealloc()`.

If the `shade_type` is `MEM_SHADE_PSEUDO`, you must supply the allocator and deallocator functions before allocating segments from this shade.

`color` is passed to the allocator function when it is called to allocate a new block of memory for this shade. Memory color information may be found by examining the system's CDB.

If `initial_size` is not zero, then the allocator function is called immediately to allocate the initial block of memory for the shade. The block size is indicated by `initial_size`. If `shade_type` is `MEM_SHADE_PSEUDO`, or you want to set up an allocator function for a normal shade, then `initial_size` must be zero. The initial block is not returned (using the deallocator function) to the system until the shade is destroyed. `initial_size` is automatically rounded up to be a multiple of `MEM_MIN_ALLOC`. 
Shaded Memory Functions

Memory allocations are made from the initial block until it is exhausted, then the shade is expanded using the current grow method. If `grow_size` is zero, then the shade cannot grow beyond its initial size. See `mem_set_grow_method()` for details about the grow method. The default grow method is `MEM_GROW_MULTIPLIER`. `grow_size` is automatically rounded up to be a multiple of `MEM_MIN_ALLOC`.

`ovtype` specifies where the overhead for a segment is kept. For pseudo shades, `ovtype` must be `MEM_OV_SEPARATE`. See `MEM_OVTYPE` for an explanation of possible values.

If `overflow_detect` is `TRUE`, then a safe area is added immediately before and after each segment when it is allocated. A safe-area pattern is placed in this area and you may call `mem_list_overflow()` to print a list of segments whose safe areas have been written to. This information may be used to track down code that is improperly writing before or after an allocated segment. For pseudo shades, `overflow_detect` must be `FALSE`.

When all shaded segments within a block of colored memory have been de-allocated, the block is returned to the system using the deallocation function for the shade.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe

**Fatal Errors**

- **010:015 EOS_MAUI_BADSIZE**
  The `shade_type` is `MEM_SHADE_PSEUDO` and the `initial_size` is not zero.

- **010:016 EOS_MAUI_BADVALUE**
  The `shade_type` is not valid. See `MEM_SHADE_TYPE` for valid values. May also be caused by using incorrect values for `ovtype` or
overflow_detect when defining a pseudo shade (see description).

The specified shade_id has already been defined.

Shade 0 reserved by MAUI.

This API has not been initialized with mem_init().

Indirect Errors

_os_srqmem()

See Also

cdb_get_ddr()
mem_destroy_shade()
mem_free()
mem_set_alloc()
mem_set_alloc_bndry()
mem_set_dealloc()
mem_set_grow_method()
mem_sfree()
mem_sfree_all()
BOOLEAN
CDB_TYPE_GRAPHIC
CDB_TYPE_SYSTEM
MEM_GROW
MEM_OVTYPE
MEM_SHADE_TYPE

See Ultra C Library Reference.
Shaded Memory Functions

mem_destroy_shade()
Destroy a Shade of Memory

Syntax

```c
error_code
mem_destroy_shade(u_int32 shade_id)
```

Description

`mem_destroy_shade()` destroys the specified `shade_id` of memory. All memory allocated from this shade must be returned using `mem_free()`, `mem_sfree()`, or `mem_sfree_all()` before calling this function.

If successful, this function returns `SUCCESS`.

Non-Fatal Errors

**010:014 EOS_MAUI_BADSHADE** The specified `shade_id` has not been defined.

**010:036 EOS_MAUI_NOINIT** This API has not been initialized with `mem_init()`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe

Indirect Errors

- `(*dealloc_func)()`
- `_os_srtmem()`
- `mem_list_segments()`

See *Ultra C Library Reference.*
Note
A notice is printed for each segment that is still allocated by this shade. See `mem_list_segments()` for information about this list.

See Also
- `mem_create_shade()`
- `mem_free()`
- `mem_sfree()`
- `mem_sfree_all()`
mem_free()
Free a Segment from a Normal Shade

**Syntax**

```c
error_code
mem_free(void *ptr)
```

**Description**

`mem_free()` deallocates the memory segment pointed to by `ptr`. This pointer should point to memory previously allocated by `mem_calloc()`, `mem_malloc()` or `mem_realloc()`. You should only call `mem_free()` to free segments allocated from a normal shade. If the segment was allocated from a pseudo shade, you must call `mem_sfree()` instead.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe

**Non-Fatal Errors**

- **010:012 EOS_MAUI_BADPTR**
  The segment pointed to by `ptr` was not allocated from a normal shade.

- **010:036 EOS_MAUI_NOINIT**
  This API has not been initialized with `mem_init()`.

**Indirect Errors**

- `(*dealloc_func)()`
- `_os_srtmem()`
  See *Ultra C Library Reference*. 
Note
A notice is printed if an underflow and/or overflow is detected. See mem_list_overflows() for information about underflows and overflows.

See Also
mem_calloc()
mem_init()
mem_list_overflows()
mem_malloc()
mem_sfree()
mem_sfree_all()
mem_realloc()
mem_get_shade_status()
Get Shade Status

Syntax

```c
error_code
mem_get_shade_status(MEM_SHADE_STATUS *ret_shade_status,
                     u_int32 shade_id)
```

Description

`mem_get_shade_status()` returns the current status of the specified `shade_id`.

The shade status is returned in `ret_shade_status`. A pointer to this variable should be passed to `mem_get_shade_status()`. The caller must ensure that `ret_shade_status` points to storage large enough to hold the information. See `MEM_SHADE_STATUS` for information about this data structure.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe

Non-Fatal Errors

- `010:014 EOS_MAUI_BADSHADE`: The specified `shade_id` has not been defined.
- `010:036 EOS_MAUI_NOINIT`: This API has not been initialized with `mem_init()`.

See Also

- `mem_create_shade()`
- `mem_get_shade_status()`
- `MEM_SHADE_STATUS`
**mem_init()**

*Initialize the Shaded Memory API*

**Syntax**

```c
error_code mem_init(void)
```

**Description**

`mem_init()` initializes the shaded memory API. This function must be called prior to a call to any other shaded memory function unless otherwise noted by that function.

This function automatically creates the default shade using the following function call:

```c
    mem_create_shade(MEM_DEF_SHADE, MEM_ANY, 4096, 4096, MEM_OV_ATTACHED, TRUE);
```

The shade ID for the default shade is `MEM_DEF_SHADE`. Memory allocations for the default shade are satisfied from system color `MEM_ANY` (see `memory.h`).

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe

**Indirect Errors**

`mem_create_shade()`

**See Also**

- `mem_term()`
- `MEM_DEF_SHADE`
Shaded Memory Functions

mem_list_overflows()
Print a Listing of Overflows

Syntax

```c
error_code
mem_list_overflows(void)
```

Description

`mem_list_overflows()` prints a listing of the shaded memory segment underflows and overflows. An underflow is caused by an application writing to the safe area immediately before the segment. An overflow is caused by writing to the safe area immediately after the segment.

The presence of these safe areas is controlled by a parameter passed to `mem_create_shade()`. Following is a sample of the output produced by this function for an underflow and an overflow.

```
Memory underflow: shade=1, allocation=53,
                 start=0x96b400, size=0x10
Memory overflow: shade=5, allocation=42,
                 start=0x803400, size=0x400
```

If no underflows or overflows are detected, the following message is printed.

```
No memory underflows or overflows detected.
```

Otherwise, `EOS.Maui.Damage` is returned.

The allocation number is an ID that was assigned to the segment when it was allocated with `mem_calloc()`, `mem_malloc()`, or `mem_realloc()`. The first allocation is 1, the next is 2, and so forth.

If successful, this function returns `SUCCESS`.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
Non-Fatal Errors

010:019 EOS_MAUI_DAMAGE
A memory underflow or overflow detected.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with mem_init().

See Also
mem_calloc()
mem_create_shade()
mem_init()
mem_list_segments()
mem_list_tables()
mem_malloc()
mem_realloc()
mem_list_segments()
Print a Listing of Allocated Segments

Syntax
error_code
mem_list_segments(void)

Description
mem_list_segments() prints a listing to standard error of the memory segments that were allocated but never de-allocated. Following is a sample of the output produced by this function for an instance where there are three segments that were not deallocated.

```
Memory segment: shade=1, allocation=53,
               start=0x97a230, size=0x400
Memory segment: shade=1, allocation=68,
               start=0x96b500, size=0x100
Memory segment: shade=2, allocation=42,
               start=0xb80000, size=0x370
```

If no memory segments are currently allocated, the following message is printed.

```
No memory segments are currently allocated.
```

The allocation number is an ID that was assigned to the segment when it was allocated with mem_calloc(), mem_malloc() or mem_realloc(). The first allocation is 1, the next is 2, and so forth.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Non-Fatal Errors

010:036 EOS_MAUI_NOINIT

This API has not been initialized with `mem_init()`.

See Also

- `mem_calloc()`
- `mem_list_tables()`
- `mem_list_overflows()`
- `mem_malloc()`
- `mem_realloc()`
mem_list_tables()
Print a Listing of Memory Tables

Syntax
error_code
mem_list_tables(void)

Description

mem_list_tables() prints to standard error a listing of the current contents of the shaded memory tables. This list shows the shades that are currently defined, the blocks allocated for each shade, and a list of segments allocated from each block.

Following is a sample of the output produced by this function for an instance where one shade contains two blocks and a total of three allocated segments.

Internal Shaded Memory Tables
Shades:
   Id=0x1 Type=NORMAL Color=0x0 Initial-Size=0x400
   Grow=0x100
Blocks:
   Start=0x97a330 Size=0x400
   Allocated segments:
      Start=0x97a330 Size=0x50
      Start=0x97a390 Size=0x300
   Start=0x97a0d0 Size=0x200
   Allocated segments:
      Start=0x97a0d0 Size=0x150

End of Tables

If successful, this function returns SUCCESS

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
9 Shaded Memory Functions

Non-Fatal Errors

010:036 EOS_MAUI_NOINIT

This API has not been initialized with mem_init().

See Also

mem_list_segments()
mem_list_overflows()
mem_malloc()
Allocate Shaded Memory

Syntax

```c
error_code
mem_malloc(void *ret_ptr, u_int32 shade_id,
            size_t size)
```

Description

`mem_malloc()` allocates `size` (rounded up to a multiple of `MEM_MIN_ALLOC`) bytes from the specified `shade_id` of memory.

The memory is allocated from `shade_id` and a pointer to it is returned in `ret_ptr`. A pointer to `ret_ptr` should be passed to `mem_malloc()`. Use `mem_free()` or `mem_sfree()` to deallocate this segment when it is no longer needed.

The segment allocated is guaranteed to start on, and have a length that is a multiple of, the boundary size for the shade. See `mem_set_alloc_bndry()` for setting the boundary size. If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe

Fatal Errors

- `0:207 EOS_MEMFUL`  
  Out of memory in `shade_id` and the grow size is zero.

- `010:014 EOS_MAUI_BADSHADE`  
  The specified `shade_id` is not defined.

- `010:030 EOS_MAUI_NOCALLBACK`  
  `shade_id` is a pseudo shade, but no allocation function has been specified.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with \texttt{mem\_init()}. 

**Indirect Errors**

\begin{verbatim}
(*alloc_func)()
_os_srqmem()
\end{verbatim}

See \textit{Ultra C Library Reference}. 

**See Also**

\begin{verbatim}
mem\_create\_shade()
mem\_calloc()
mem\_free()
mem\_set\_alloc\_bndry()
mem\_sfree()
mem\_realloc()
MEM\_MIN\_ALLOC
\end{verbatim}
**mem_realloc()**

Reallocation of Shaded Memory

**Syntax**

```c
error_code
mem_realloc(void *ret_ptr, size_t size)
```

**Description**

The `mem_realloc()` function reallocates the memory segment pointed to by `ret_ptr`. The new size for this segment is specified by `size`. The `size` is rounded up to a multiple of MEM_MIN_ALLOC.

The new `size` may be smaller or larger than the current size. The contents of the current segment are copied to the new segment. The number of bytes copied is the smaller of the current size and new size.

The segment being reallocated must have been allocated from a normal shade. It is assumed that the CPU is not able to write to pseudo memory, and this function must copy the contents of the segment. Therefore, segments from a pseudo shade cannot be used.

The memory is re-allocated and a pointer to it is returned in `ret_ptr`. A pointer to `ret_ptr` should be passed to `mem_realloc()` and must originally point to the memory segment being reallocated. Use `mem_free()` or `mem_sfree()` to de-allocate this segment when it is no longer needed.

The segment allocated is guaranteed to start on, and have a length that is a multiple of, the boundary size for the shade. See `mem_set_alloc_bndry()` for setting the boundary size.

If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
Fatal Errors

010:012 EOS_MAUI_BADPTR
The pointer ret_ptr does not point to a segment allocated from a normal shade.

010:015 EOS_MAUI_BADSIZE
The specified size was zero.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with mem_init().

Indirect Errors

(*alloc_func)()
_os_srqmem() See Ultra C Library Reference.

Note
A notice is printed if an underflow and/or overflow is detected. See mem_list_overflows() for information about underflows and overflows.

See Also
mem_create_shade()
mem_calloc()
mem_free()
mem_set_alloc_bndry()
mem_sfree()
mem_malloc()
MEM_MIN_ALLOC
mem_set_alloc()

Set Allocator Function for a Shade

Syntax

```c
error_code
mem_set_alloc(u_int32 shade_id,
              error_code(*alloc_func)
              (void *, size_t *, void **, u_int32),
              void *alloc_data,
              size_t initial_size)
```

Description

`mem_set_alloc()` defines the allocation function for the specified shade_id. If `alloc_func` is NULL, then `_os_srqmem()` is used as the allocation function (this is the default). Since pseudo shades cannot use `_os_srqmem()`, you must provide an allocation function for shades of this type.

If you specified an initial size (non-zero) when you created the shade with `mem_create_shade()`, or any segments are still allocated from the shade, the error `EOS_MAUI_INUSE` is returned.

When this shade needs to grow, the allocation function (`*alloc_func`()) is called. The values for color and alloc_data passed to `mem_set_alloc()` are passed to `alloc_func()`. The prototype for `alloc_func()` appears as follows:

```c
error_code alloc_func(void *alloc_data,
                      size_t *size, void **mem_ptr, u_int32 color)
```

If `initial_size` is not zero, then the allocator function is called immediately to allocate the initial block of memory for the shade. The block size is indicated by `initial_size`. The initial block is not returned (using the deallocator function) to the system until the shade is destroyed.

If successful, this function returns `SUCCESS`. 
Shaded Memory Functions

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe

Fatal Errors
010:014 EOS_MAUI_BADSHADE
The specified shade_id has not been defined.

010:024 EOS_MAUI_INUSE
At least one block of memory is already attached to the shade. Make sure that all segments have been freed and that you do not specify an initial size when you create the shade.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with mem_init().

Indirect Errors
(*alloc_func)()
_os_srqmem() See Ultra C Library Reference.

See Also
mem_create_shade()
mem_destroy_shade()
mem_set_dealloc()
Shaded Memory Functions

**mem_set Alloc Bndry()**

Set Memory Allocation Boundary

**Syntax**

```c
error_code
mem_set_alloc_bndry(u_int32 shade_id,
size_t boundary_size)
```

**Description**

`mem_set_alloc_bndry()` sets the boundary size used for allocations by `mem_calloc()`, `mem_malloc()`, and `mem_realloc()`. This function must be called before any allocations are made or the error `EOS_MAUI_INUSE` is returned.

After calling `mem_set_alloc_bndry()`, each segment that is allocated is guaranteed to start on, and have a length that is a multiple of `boundary_size`.

The default boundary size of `MEM_MIN_ALLOC` is used if you do not call this function. The `boundary_size` is automatically rounded up to be a multiple of `MEM_MINALLOC`.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe

**Parameters**

- **shade_id**
  Shade of memory for this boundary allocation.

- **boundary_size**
  Start size and multiple to use for allocation.

**Non-Fatal Errors**

- **010:014 EOS_MAUI_BADSHADE**
  The specified `shade_id` is not defined.
At least one block of memory is already attached to the shade. Make sure that all segments have been freed and that you do not specify an initial size when you create the shade.

This API has not been initialized with `mem_init()`.

`shade_id` has overflow detection or attached overhead turned on. Boundary allocations are currently not implemented for shades that have any type of attached overhead.

**See Also**
- `mem_calloc()`
- `mem_create_shade()`
- `mem_malloc()`
- `mem_realloc()`
- `MEM_GROW`
- `MEM_MIN_ALLOC`
mem_set_dealloc()

Set De-alloactor Function for a Shade

Syntax

```c
error_code mem_set_dealloc(u_int32 shade_id,
    error_code (*dealloc_func)(void *, size_t, void *, u_int32),
    void *dealloc_data)
```

Description

`mem_set_dealloc()` defines the deallocation function for the specified `shade_id`.

If `dealloc_func` is NULL, `_os_srqmem()` is used as the deallocation function (default). Since pseudo shades cannot use `_os_srtmem()`, you must provide a deallocation function for shades of this type.

When a memory block previously allocated is no longer being used, it is deallocated by using the deallocation function `dealloc_func()`. The values for `color` and `dealloc_data` passed to `mem_set_dealloc()` are passed to `dealloc_func()`. The prototype for `dealloc_func()` appears as follows:

```c
error_code dealloc_func(void *dealloc_data,
    size_t size, void *mem_ptr, u_int32 color)
```

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
Shaded Memory Functions

**Fatal Errors**

010:014  EOS_MAUI_BADSHADE

The specified shade_id is not defined.

010:036  EOS_MAUI_NOINIT

API not initialized with mem_init().

**Indirect Errors**

_os_srtmem()  

See *Ultra C Library Reference*.

**See Also**

mem_create_shade()
mem_destroy_shade()
mem_set_dealloc()
mem_set_error_action()

Set Action to Take in Error Handler

Syntax

```c
error_code
mem_set_error_action(MAUI_ERR_LEVEL debug_level,
                     MAUI_ERR_LEVEL passback_level,
                     MAUI_ERR_LEVEL exit_level)
```

Description

`mem_set_error_action()` sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling `mem_init()`. Following is the table of error levels. The least severe error is listed first.

Error Levels in `mem_set_error_action()`

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed, but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
</tbody>
</table>
debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default pass-back level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to call exit(). In this case the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe

**Non-Fatal Errors**

None

**See Also**

mem_init()
Shaded Memory Functions

mem_set_grow_method()
Set Grow Method for a Shade

Syntax

```c
error_code mem_set_grow_method(u_int32 shade_id,
                               MEM_GROW grow_method)
```

Description

`mem_set_grow_method()` sets the grow method for the specified shade. Allocations previously made from this shade are not affected.

If `grow_method` is `MEM_GROW_LARGER`, then the size of the block requested from the system is the larger of the grow size for the shade, and the size being requested by the application.

If `grow_method` is `MEM_GROW_MULTIPLE`, then the size of the block requested from the system is a multiple of the grow size for the shade.

If successful, this function returns `SUCCESS`. Otherwise, the returned value is an error code. Error codes unique to this API are defined below.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe

Non-Fatal Errors

- 010:014 EOS_MAUI_BADSHADE: The specified `shade_id` is not defined.
- 010:036 EOS_MAUI_NOINIT: This API has not been initialized with `mem_init()`.

See Also

- `mem_create_shade()`
- `MEM_GROW`
### mem_sfree()

**Free a Segment from the Specified Shade**

#### Syntax

```c
error_code mem_sfree(u_int32 shade_id, void *ptr)
```

#### Description

`mem_sfree()` deallocates the memory segment pointed to by `ptr`. This pointer should point to a segment previously allocated from the specified `shade_id` using `mem_calloc()`, `mem_malloc()`, or `mem_realloc()`.

Use `mem_sfree()` to free segments allocated from a pseudo shade. `mem_free()` does not work with pseudo shades. If the memory segment was allocated from a normal shade, then you may use either `mem_free()` or `mem_sfree()`.

If successful, this function returns `SUCCESS`.

#### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe

#### Non-Fatal Errors

- **010:012 EOS_MAUI_BADPTR**
  
  The segment pointed to by `ptr` was not allocated from the specified `shade_id`.

- **010:014 EOS_MAUI_BADSHADE**
  
  The specified `shade_id` is not defined.

- **010:030 EOS_MAUI_NOCALLBACK**
  
  The `shade_id` is a pseudo shade, but no deallocation function has been set.

- **010:036 EOS_MAUI_NOINIT**
  
  This API has not been initialized with `mem_init()`.
Indirect Errors

(*dealloc_func)()
_os_srtmem()

See Ultra C Library Reference.

Note
A notice is printed if an underflow and/or overflow is detected. See mem_list_overflows() for information about underflows and overflows.

See Also
mem calloc()
mem free()
mem list_overflows()
mem malloc()
mem realloc()
mem set dealloc()
mem sfree all()
mem_sfree_all()
Free All Segments from the Specified Shade

Syntax
error_code
mem_sfree_all(u_int32 shade_id)

Description
mem_sfree_all() deallocates all memory segments from the specified shade shade_id. This includes all segments allocated by mem_calloc(), mem_malloc(), or mem_realloc(). If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe

Parameters
shade_id Shade of memory to free all segments.

Non-Fatal Errors
010:014 EOS_MAUI_BADSHADE The specified shade_id is not defined.
010:030 EOS_MAUI_NOCALLBACK shade_id is a pseudo shade, but no de-allocation function has been set.
010:036 EOS_MAUI_NOINIT This API has not been initialized with mem.init().

Indirect Errors
(*dealloc_func)()
_os_srtmem() See Ultra C Library Reference.
**Note**
A notice is printed if an underflow and/or overflow is detected. See `mem_list_overflows()` for information about underflows and overflows.

**See Also**
- `mem_calloc()`
- `mem_free()`
- `mem_list_overflows()`
- `mem_malloc()`
- `mem_realloc()`
- `mem_set_dealloc()`
- `mem_sfree()`
### mem_term()

**Terminate Shaded Memory API**

**Syntax**

```c
error_code
mem_term(void)
```

**Description**

`mem_term()` terminates the shaded memory API. All memory that was allocated (but not deallocated) is deallocated by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe

**Non-Fatal Errors**

- **010:030 EOS_MAUI_NOCALLBACK**
  
  At least one shade is a pseudo shade, but no deallocation function has been set for it.

- **010:036 EOS_MAUI_NOINIT**
  
  This API has not been initialized with `mem_init()`.

**Indirect Errors**

- `(*dealloc_func)()`
- `mem_list_segments()`
- `_os_srtmem()`

See [Ultra C Library Reference](#).
**Note**
A notice is printed for each segment that is still allocated by any shade. See `mem_list_segments()` for information about this list.

A notice is printed if an underflow and/or overflow is detected. See `mem_list_segments()` for information about underflows and overflows.

**See Also**
- `mem_init()`
- `mem_list_overflows()`
- `mem_list_segments()`
- `mem_set_dealloc()`
Chapter 10: Messaging Functions
msg_close_mbox()

Close a Mailbox

Syntax

```
error_code
msg_close_mbox(MSG_MBOX_ID mbox)
```

Description

`msg_close_mbox()` closes the mailbox `mbox` and unlinks the data module and event that were linked to when the mailbox was created or opened.

If you do not call this function to close the mailbox prior to exiting the process, the data module and event remain linked. This prevents the mailbox from being created the next time you execute your application.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

```
mbox
```
Message mailbox ID.

Non-Fatal Errors

```
010:008 EOS_MAUI_BADID
010:023 EOS_MAUI_INTERNAL
010:036 EOS_MAUI_NOINIT
```

- The ID specified by `mbox` is not valid.
- MAUI has detected an internal error. Please verify in a simple application and report the incident to Microware Customer Service.
- This API has not been initialized with `msg_init()`.
**Indirect Errors**

- `mem_free()`
- `_os_ev_unlink()`
- `_os_sema_term()`
- `_os_unlink()`

**See Also**

- `msg_create_mbox()`
- `msg_open_mbox()`
- `MSG_MBOX_ID`
msg_create_mbox()
Create a Mailbox

Syntax

```c
error_code
msg_create_mbox(MSG_MBOX_ID *ret_mbox,
                const char *mbox_name,
                u_int32 num_entries,
                size_t entry_size,
                u_int32 color)
```

Description

`msg_create_mbox()` creates and opens the mailbox named `mbox_name`. Other applications may link to this mailbox by calling `msg_open_mbox()`. The maximum length of the mailbox name is defined by `MSG_MAX_MBOX_NAME`.

This function creates a data module with the name `mbox_name`. If it already exists, `EOS_KWNMOD` is returned. This function creates an event with the name `mbox_name`. If it already exists, `EOS_EVBUSY` is returned.

`num_entries` specifies the maximum number of messages that the mailbox can hold at any given time. An attempt to write a message to a full mailbox results in an error.

`entry_size` specifies the size of the message structure used for reading and writing messages.

`color` is the memory color to use for the data module (currently ignored on OS-9000). Memory colors are defined by the operating system. A common color for system RAM is `MEM_ANY`. Do not confuse this with memory shades that are defined by the application using the Shaded Memory API.

The mailbox ID is returned in `ret_mbox`. A pointer to this variable should be passed to `msg_create_mbox()`. Use `msg_close_mbox()` when this mailbox is no longer needed. If you fail to close it, the data module and event remain linked. This may prevent the mailbox from being created the next time you execute your application.
If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_mbox Pointer to message mailbox ID.
*mbox_name Pointer to data module and event.
num_entries Maximum number of messages the mailbox can hold.
entry_size Size of message structure.
color Memory color to use for data module (currently ignored in OS-9000).

Fatal Errors
010:012 EOS_MAUI_BADPTR mbox_name is set to NULL.
010:036 EOS_MAUI_NOINIT This API has not been initialized with msg_init().
010:050 EOS_MAUI_TOOLONG The mailbox name mbox_name is too long. The maximum length is MSG_MAX_MBOX_NAME.

Indirect Errors
See Ultra C Library Reference.
mem_malloc() _os_datmod() _os_ev_creat() _os_mkmodule() _os_sema_init()
See Also
msg_close_mbox()
msg_open_mbox()
MSG_MBOX_ID
**msg_dispatch()**

Dispatch Message

**Syntax**

```c
error_code
msg_dispatch(const void *msg)
```

**Description**

`msg_dispatch()` dispatches the specified message `msg`. This causes the callback function for the message to be called.

The callback function called is `msg->callback`. If `msg->callback` is NULL, then the error `EOS_MAUI_NOCALLBACK` is returned. The prototype for `msg->callback` appears as follows:

```c
void callback(const void *msg)
```

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

`*msg`  
Pointer to the message to be dispatched.

**Non-Fatal Errors**

- **010:030 EOS_MAUI_NOCALLBACK**: No callback found for this message.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `msg_init()`.

**See Also**

- `msg_read()`
- `msg_readn()`
**msg_flush()**

Flush Messages

**Syntax**

```c
error_code
msg_flush(MSG_MBOX_ID mbox, u_int32 mask)
```

**Description**

`msg_flush()` flushes all messages in the mailbox `mbox` that are of a type specified by `mask` (see `MSG_TYPE`) and are validated by the filter function.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `mbox`: Message mailbox ID.
- `mask`: Type of message filter.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `mbox` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `msg_init()`.

**Indirect Errors**

- `(*filter)()`
- `_os_sema_p()`
- `_os_sema_v()`

See *Ultra C Library Reference*. 
See Also

msg_set_filter()
MSG_MBOX_ID
MSG_TYPE
msg_get_mbox_status()  
Get Mailbox Status

**Syntax**

```c
error_code  
msg_get_mbox_status(MSG_MBOX_STATUS *ret_mbox_status,  
                    MSG_MBOX_ID mbox)
```

**Description**

`msg_get_mbox_status()` returns the current status of the specified mailbox `mbox`.

The mailbox status is returned in `ret_mbox_status`. A pointer to this variable should be passed to `msg_get_mbox_status()`. The caller must ensure that `ret_mbox_status` points to storage large enough to hold the information.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `*ret_mbox_status` Pointer to status for mailbox specified in `mbox`.
- `mbox` Message mailbox ID.

**Non-Fatal Errors**

- 010:008 EOS_MAUI_BADID  
The ID specified by `mbox` is not valid.

- 010:036 EOS_MAUI_NOINIT  
This API has not been initialized with `msg_init()`.
See Also
msg_set_filter()
msg_set_mask()
MSG_MBOX_ID
MSG_MBOX_STATUS
msg_init()
Initialize the Messaging API

Syntax
error_code
msg_init(void)

Description
msg_init() initializes the messaging API. This function must be called prior to a call to any other messaging function unless otherwise noted by that function.

This API depends on the Shaded Memory API. Therefore, mem_init() is called by this function.

As of MAUI 3.1 this API also depends on the /mauidev device and mauidrvr driver.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
None

Indirect Errors
mem_init()
_os_open() To open the /mauidev device.

See Also
msg_term()
msg_open_mbox()

Open a Mailbox

Syntax

```c
error_code
msg_open_mbox(MSG_MBOX_ID *ret_mbox,
               const char *mbox_name,
               size_t *entry_size)
```

Description

`msg_open_mbox()` opens the mailbox named `mbox_name`. The maximum length of the mailbox name is defined by `MSG_MAX_MBOX_NAME`.

This function links to a data module with the name `mbox_name`. If it does not exist, `EOS_MNF` is returned. This function links to an event with the name `mbox_name`. If it does not exist, `EOS_EVNF` is returned. Both the data module and event are created by `msg_create_mbox()`.

The mailbox `mbox_name` must have already been created by `msg_create_mbox()`. This function is most often used to open a mailbox that was created by another application.

`entry_size` specifies the size of the message structure used for reading and writing messages. If it is different than the size specified by the mailbox creator, then the lesser of the two values is used when reading and writing messages using this mailbox ID. For this reason, this copy size is passed back to the caller in `entry_size`.

The mailbox ID is returned in `ret_mbox`. A pointer to this variable should be passed to `msg_open_mbox()`. You should use `msg_close_mbox()` when this mailbox is no longer needed. If you fail to close it, the data module and event remain linked. This may prevent the mailbox from being created the next time you execute your application.

If successful, this function returns `SUCCESS`. 
10 Messaging Functions

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**
- `*ret_mbox` Pointer to the mailbox ID.
- `*mbox_name` Pointer to the name of the mailbox.
- `*entry_size` Pointer to the size of message structure.

**Fatal Errors**
- **010:036 EOS_MAUI_NOINIT** This API has not been initialized with `msg_init()`.
- **010:058 EOS_MAUI_INCOMPATVER** This process is using a newer version of MAUI than the creator of the mailbox, and they are not compatible.

**Indirect Errors**
- `mem_malloc()`
- `_os_ev_link()`
- `_os_link()`
- `_os_sema_init()`

See *OS-9 Reference Manual*

**See Also**
- `msg_close_mbox()`
- `msg_create_mbox()`
- MSG_MBOX_ID
msg_peek()

Peek at a Message in a Mailbox

Syntax

```c
error_code msg_peek(MSG_MBOX_ID mbox, void *msg, u_int32 mask,
                     MSG_BLOCK_TYPE block_type)
```

Description

`msg_peek()` peeks at the next message in the mailbox `mbox` whose type is present in the specified `mask` (see `MSG_TYPE`) and is validated by the filter function. The search starts at the head of the queue and continues until a message is found or the tail of the queue is found.

If `block_type` is `MSG_BLOCK`, then this function blocks until a message in the specified `mask` is available in the queue. If set to `MSG_NOBLOCK`, the function returns a message immediately.

If `block_type` is `MSG_NOBLOCK` and a message is not found, then the message returned will have its type set to `MSG_TYPE_NONE`. In this case, `time_queued` is the current time, `callback` is `NULL`, and `pid` is the process ID for the process making the call to `msg_peek()`.

`msg` must point to a buffer large enough to hold the largest possible message (size specified in `msg_create_mbox()` or returned in `msg_open_mbox()`). This function copies the message to this buffer.

Although not common, it is possible for more than one process to read messages from the same queue (just not at the same time). Therefore, you may peek at a message, then try to read it, only to find that it is no longer there.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
10 Messaging Functions

Parameters

mbox
Message mailbox ID.

$msg
Pointer to message buffer.

mask
Type of message filter to use.

block_type
Type of blocking mechanism to use.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by mbox is not valid.

010:017 EOS_MAUI_BUSY
A read is already in progress on this mailbox. Try again later.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with msg_init().

Indirect Errors

(*filter)()
=os_ev_set()
=os_ev_wait()
=os_sema_p()
=os_sema_v()
=os_sigmask()


See Also

msg_create_mbox()
msg_open_mbox()
msg_peekn()
msg_read()
msg_readn()
msg_set_filter()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_TYPE
msg_peekn()
Peek at N Bytes of a Message in a Mailbox

Syntax

error_code
msg_peekn(MSG_MBOX_ID mbox, void *msg, size_t *size,
            u_int32 mask, MSG_BLOCK_TYPE block_type)

Description

msg_peekn() works exactly like msgpeek() except that
msg_peekn() allows you to specify the maximum number of bytes in
the message.

At most, size bytes are copied from the mailbox to msg. If the size of
the message in the mailbox is less than size, then the smaller size is
used. In this case, size is updated with the actual number of bytes
copied.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

mbox Message mailbox ID.
*msg Pointer to message buffer.
*size Pointer to maximum size of message in bytes.
mask Type of message filter to use.
block_type Type of blocking mechanism to use.
10 Messaging Functions

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by mbox is not valid.

010:017 EOS_MAUI_BUSY
A read is already in progress on this mailbox. Try again later.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with msg_init().

Indirect Errors

(*filter)()
_os_ev_set()
os_ev_wait()
os_sema_p()
os_sema_v()
os_sigmask()

See Ultra C Library Reference

See Also
msg_create_mbox()
msg_open_mbox()
msg_peek()
msg_read()
msg_readn()
msg_set_filter()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_TYPE
msg_read()
Read a Message from a Mailbox

Syntax

```c
error_code
msg_read(MSG_MBOX_ID mbox, void *msg, u_int32 mask, 
MSG_BLOCK_TYPE block_type)
```

Description

`msg_read()` reads the next message from the mailbox `mbox` whose type is present in the specified `mask` (see `MSG_TYPE`) and is validated by the filter function. The search starts at the head of the queue and continues until a message is found or the tail of the queue is found.

If `block_type` is `MSG_BLOCK`, this function blocks until a message in the specified `mask` is available in the queue. If set to `MSG_NOBLOCK` the function returns a message immediately.

If `block_type` is `MSG_NOBLOCK` and a message is not found, then the message returned will have its type set to `MSG_TYPE_NONE`. In this case, `time_queued` is the current time, `callback` is `NULL`, and `pid` is the process ID for the process making the call to `msg_read()`.

Use `MSG_SIGBLOCK` to stay in `msg_read()` even if you get a signal.

*`msg` must point to a buffer large enough to hold the largest possible message (size specified in `msg_create_mbox()` or returned in `msg_open_mbox()`). This function copies the message to this buffer.

- If successful, this function returns `SUCCESS`.
- When `msg_read()` reports (when it breaks out, due to a signal), it returns `EOS_MAUI_SIGNAL`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
10 Messaging Functions

Parameters

mbox
Message mailbox ID.

*msg
Pointer to message buffer.

mask
Type of message filter to use.

block_type
MSG_BLOCK or MSG_NOBLOCK.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by mbox is not valid.

010:016 EOS_MAUI_BADVALUE
The value of block_type is not valid.

010:017 EOS_MAUI_BUSY
A read (msg_peek() or msg_read()) is already in progress on this mailbox. Try again later.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with msg_init().

010:048 EOS_MAUI_SIGNAL
A signal was received while waiting for a message to become available.

Indirect Errors

(*filter)()

_os_ev_set()

_os_ev_wait()

_os_sema_p()

_os_sema_v()

os_sigmask()

See Also

msg_create_mbox()
msg_dispatch()
msg_open_mbox()
msg_peek()
msg_peekn()
msg_readn()
msg_set_filter()
msg_unread()
msg_unreadn()
msg_write()
msg_writen()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_TYPE
msg_readn()
Read N Bytes of a Message from a Mailbox

**Syntax**

```c
error_code
msg_readn(MSG_MBOX_ID mbox, void *msg, size_t *size,
    u_int32 mask, MSG_BLOCK_TYPE block_type)
```

**Description**

`msg_readn()` works exactly like `msg_read()` except that `msg_readn()` allows you to specify the maximum number of bytes in the message.

At most, `size` bytes are copied from the mailbox to `msg`. If the size of the message in the mailbox is less than `size`, then the smaller size is used. In this case, `size` is updated with the actual number of bytes copied.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `mbox` Mailbox ID.
- `*msg` Pointer to the message.
- `*size` Pointer to maximum bytes of message read.
- `mask` Message mask.
- `block_type` MSG_BLOCK or MSG_NOBLOCK.
Non-Fatal Errors

010:008  EOS_MAUI_BADID
The ID specified by mbox is not valid.

010:016  EOS_MAUI_BADVALUE
The value of block_type is not valid.

010:017  EOS_MAUI_BUSY
A read is already in progress on this mailbox. Try again later.

010:036  EOS_MAUI_NOINIT
This API has not been initialized with msg_init().

010:048  EOS_MAUI_SIGNAL
A signal was received while waiting for a message to become available.

Indirect Errors

(*filter)()
_os_ev_set()
_os_ev_wait()
_os_sema_p()
_os_sema_v()
_os_sigmask()

See Also
msg_create_mbox()
msg_dispatch()
msg_open_mbox()
msg.peek()
msg.peekn()
msg_read()
msg.set_filter()
msg.unread()
msg.unreadn()
msg.write()
msg.writen()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_TYPE
msg_release_sig()

Release Signal on Message Ready

Syntax

```c
error_code
msg_release_sig(MSG_MBOX_ID mbox)
```

Description

`msg_release_sig()` releases the pending request for a signal which was set by calling `msg_send_sig()`.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `mbox` Message mailbox ID.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID** The ID specified by `mbox` is not valid.
- **010:036 EOS_MAUI_NOINIT** This API has not been initialized with `msg_init()`.
- **010:046 EOS_MAUI_NOTPENDING** A request for a signal on message ready is not pending.

Indirect Errors

- `_os_sema_p()` See *OS-9 Reference Manual.*
- `_os_sema_v()`
See Also

msg_send_sig()
MSG_MBOX_ID
msg_release_watch()

Release Watch Signal

Syntax
error_code
msg_release_watch(process_id pid)

Description
msg_release_watch() releases the pending request for a watch signal which was set by calling msg_send_watch().

This function is new as of MAUI 3.1 and returns EOS_ITRAP on older systems.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
pid ID of the process to cancel watch.

Non-Fatal Errors
000:227 EOS_ITRAP Function not supported by older shared libraries.
010:044 EOS_MAUI_NOTFOUND Target pid is not initialized with msg_init(). It may have already quit.
010:036 EOS_MAUI_NOINIT This API has not been initialized with msg_init().
010:046 EOS_MAUI_NOTPENDING Now watch request pending for pid by the current thread/process.
10 Messaging Functions

**Indirect Errors**

mem_free()
_os_findpd()

**See Also**

msg_send_watch()
msg_send_sig()

Send Signal on Message Ready

Syntax

```c
error_code
msg_send_sig(MSG_MBOX_ID mbox, u_int32 mask,
               signal_code signal)
```

Description

`msg_send_sig()` causes this process/thread to receive the specified signal the next time a message whose type is present in `mask` (see `MSG_TYPE`) is queued to the mailbox `mbox` by any writer. If the message is already in the mailbox, the signal is sent immediately.

This signal is only received one time for each call to `msg_send_sig()`.

Although not common, it is possible for more than one process to read messages from the same queue (just not at the same time). Therefore, you may receive the signal, then try to read it, only to find that it is no longer there.

Only one process may have a pending request for a signal at any given time. If another process is waiting for a signal when you call `msg_send_sig()`, then `EOS_MAUI_BUSY` is returned.

If successful, this function returns `SUCCESS`.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `mbox`: Message mailbox ID.
- `mask`: Type of message filter to use.
- `signal`: Specifies what signal to send when messages are received.
10 Messaging Functions

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `mbox` is not valid.

- **010:017 EOS_MAUI_BUSY**
  - Another function has already requested a signal on this mailbox. Try again later.

- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `msg_init()`.

**Indirect Errors**

- `_os_sema_p()`
- `_os_sema_v()`
- `_os_send()`

See also **OS-9 Reference Manual**.

**See Also**

- `msg_release_sig()`
- `MSG_MBOX_ID`
- `MSG_MASK`
msg_send_watch()
Send Signal on msg_term()

Syntax
error_code
msg_send_watch(process_id pid, signal_code signal)

Description
msg_send_watch() causes this process/thread to receive the specified signal when process pid calls msg_term() or exits, causing an implied msg_term().

This call is useful when a process is communicating with another process via the Messaging API and has allocated resources on behalf of that other process. Even if the other process quits without informing anyone, the caller of msg_send_watch() can receive a signal and perform any necessary cleanup. To cancel a watch request, use msg_release_watch().

This signal is only received one time for each call to msg_send_watch(). Several processes can request signals for the same pid. The caller can only request one signal per pid. A process may not watch itself or any of its threads.

This function is new as of MAUI 3.1 and returns EOS_ITRAP on older systems.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
pid ID of the process to watch.
signal Specifies the signal to send.
Non-Fatal Errors

000:227 EOS_ITRAP
Function not supported by older shared libraries.

010:017 EOS_MAUI_BUSY
A watch is already registered by this thread/process for pid.

010:059 EOS_MAUI_NOTALLOWED
Caller tried to watch itself.

010:044 EOS_MAUI_NOTFOUND
Target pid is not initialized with msg_init().

010:036 EOS_MAUI_NOINIT
This API has not been initialized with msg_init().

Indirect Errors

mem_malloc()
=os_findpd()

See Also
msg_release_watch()
**msg_set_error_action()**

Set Action to Take in Error Handler

**Syntax**

```c
error_code
msg_set_error_action(MAUI_ERR_LEVEL debug_level,
                     MAUI_ERR_LEVEL passback_level,
                     MAUI_ERR_LEVEL exit_level)
```

**Description**

`msg_set_error_action()` sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling `msg_init()`. The following is the table of error levels. The least severe error is listed first.

**Table 10-1 msg_set_error_action() Error Levels**

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
</tbody>
</table>
Table 10-1  msg_set_error_action() Error Levels (continued)

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>

debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default pass-back level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to call exit(). In this case the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

debug_level Minimum error level that causes the error handler to print a message to standard error.

passback_level Minimum error level that causes the error handler to return the error.

exit_level Minimum error level that causes the error handler to call exit().
Messaging Functions

**Non-Fatal Errors**
None

**See Also**
msg_init()
msg_set_filter()

Set Filter for Searching a Mailbox

Note
Due to the performance and system stability implications of supporting this function, it is not supported by the default configuration of MAUI. System designers must explicitly modify the systems boot modules to enable this feature. Portable MAUI applications should not rely on this feature. See the MAUI Porting Guide for more information on how to configure MAUI to support Message Filtering.

Syntax
```c
error_code msg_set_filter(MSG_MBOX_ID mbox,
                         error_code (*filter)(BOOLEAN *, const void *, void *),
                         void *filter_data)
```

Description
msg_set_filter() sets the filter function used when searching the mailbox mbox. This application supplied filtering is applied automatically to any API function that searches the mailbox. These functions are msg_flush(), msg.peek(), msg.peekn(), msg_read(), and msg_readn().

filter is a pointer to the application supplied function that performs the filtering. If set to NULL, no application filtering takes place. This is the default value.

filter_data is passed by the API functions that do searching to the applications filter function. This parameter may be used by the application as a mechanism to pass information to the filter function.

The filter function filter is defined by the caller and its prototype should appear as follows:
```c
error_code filter(BOOLEAN *ret_use_msg,
                 const void *msg, void *filter_data)
```
When an API function (for example, `msg_read()`) calls the filter function it passes the address of a `BOOLEAN` named `ret_use_msg`. This value should be set to `TRUE` if the message `msg` should be used. Otherwise, it should be set to `FALSE`.

If successful, the filter function should return `SUCCESS`. Otherwise, the returned value is an error code. If an error code is returned by the filter function, it is returned by the API function (for example, `msg_read()`) that called it.

**Extreme care should be exercised when implementing filter functions.**

Filter functions are intended only for inspection of the message data in the queue. Activity in the filter function should be kept to a minimum. The filter function must not attempt to modify the messages in the queue. Programming errors or abnormal termination of an application while executing a filter function can cause damage to message queue and the other processes using that message queue.

The filter function should not use global variables as this feature is not portable to all implementations of MAUI. Use the `filter_data` field to access data external to the filter function.

The filter function should not make function calls or cause the application to block or abort. While the filter function is called, MAUI locks the message queue of the mailbox, blocking all other applications attempting to access the message queue. Delays or failures in the filter function can impact the stability of other applications accessing the message queue.

---

**Note**

Most systems disable `msg_set_filter()` to improve the speed, size, and security of their systems.

---

If successful, this function returns `SUCCESS`.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

mbox Message mailbox ID.
(*filter) Points to the function that performs the filtering.
*filter_data Pointer to information for filter.

Non-Fatal Errors

010:008 EOS_MAUI_BADID The ID specified by mbox is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with msg_init().

See Also
msg_flush()
msg.peek()
msg.peekn()
msg.read()
msg.readn()
BOOLEAN
MSG_MBOX_ID
msg_set_mask()
Set Mask for Queuing Messages

Syntax
error_code
msg_set_mask(MSG_MBOX_ID mbox, u_int32 mask)

Description
msg_set_mask() sets the mask used when queuing messages to the mailbox mbox. Only message types included in this mask (see MSG_TYPE) are queued. Message types not present in this mask are discarded.

This only affects messages written using this mailbox ID by this process. Since each process that plans to write to a mailbox does its own msg_open_mbox(), each process is able to specify its own mask.

Calling this function only affects future writes to the mailbox. Messages that are already in the mailbox are not affected. The default mask, MSG_TYPE_ANY is used if you do not call this function.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
mbox Message mailbox ID.
mask Message filter to use.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by mbox is not valid.
010:036 EOS_MAUI_NOINIT API not initialized with msg_init().
See Also

msg_open_mbox()
msg_read()
msg_readn()
msg_write()
msg_writen()
MSG_MBOX_ID
MSG_TYPE
**msg_term()**

Terminate use of the Messaging API

**Syntax**

```c
error_code
msg_term(void)
```

**Description**

`msg_term()` terminates use of the messaging API and closes all mailboxes that are still open.

This API depends on the Shaded Memory API. Therefore `mem_term()` is called by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Non-Fatal Errors**

- **010:023 EOS_MAUI_INTERNAL**
  
  MAUI has detected an internal error. Please verify in a simple application and report the incident to Microware Customer Service.

- **010:036 EOS_MAUI_NOINIT**
  
  This API has not been initialized with `msg_init()`.

**Indirect Errors**

- `msg_close_mbox()`
- `mem_term()`
- `_os_close()`

To close the `/mauidev` device.
10 Messaging Functions

_os_send()

Send watch signals. Target process may have died or it’s signal queue may be full.

See Also
msg_init()
msg_send_watch()
**msg_unread()**

Unread a Message to a Mailbox

### Syntax

```c
error_code msg_unread(MSG_MBOX_ID mbox, const void *msg, MSG_BLOCK_TYPE block_type, MSG_PLACEMENT placement)
```

### Description

`msg_unread()` un-reads the message `msg`. This places the message back in the mailbox `mbox`. The message time `msg->time` and the process ID `msg->pid` are not modified. The current message mask (see `msg_set_mask()`) has no effect on this function. The message is always queued.

`msg` must point to a buffer large enough to hold the largest possible message (size specified in `msg_create_mbox()` or returned in `msg_open_mbox()`). This function copies the contents of `msg` to an internal buffer, so the caller is free to modify it after this function returns.

If you previously called `msg_send_sig()` to request a signal when a message is available, the signal is sent before returning from this function. Since the message is queued before an attempt is made to send this signal, if `_os_send()` gets an error, the message remains in the queue.

If successful, this function returns `SUCCESS`. If `block_type` is set to `MSG_BLOCK`, then this function currently returns `EOS_MAUI_NOTIMPLEMENTED`.
10 Messaging Functions

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
mbox Message mailbox ID.
*msg Pointer to the message to unread.
block_type Must be MSG_NOBLOCK.
placement Specifies where in the message queue to insert msg.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by mbox is not valid.
010:016 EOS_MAUI_BADVALUE placement or block_type is not valid.
010:028 EOS_MAUI_MBOXFULL The mailbox is full.
010:036 EOS_MAUI_NOINIT This API has not been initialized with msg_init().
010:045 EOS_MAUI_NOTIMPLEMENTED block_type is set to MSG_BLOCK.

Indirect Errors
_os_sema_p()
_os_sema_v()
_os_sigmask()
_os_send()

See Also

msg_create_mbox()
msg_open_mbox()
msg_read()
msg_readn()
msg_send_sig()
msg_set_mask()
msg_unreadn()
msg_write()
msg_writen()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_PLACEMENT
msg_unreadn()
Unread N Bytes of a Message to a Mailbox

Syntax

```c
error_code msg_unreadn(MSG_MBOX_ID mbox, const void *msg,
    size_t *size, MSG_BLOCK_TYPE block_type,
    MSG_PLACEMENT placement)
```

Description

msg_unreadn() works exactly like msg_unread() except that msg_unreadn() allows you to specify the maximum number of bytes in the message.

At most, size bytes are copied from msg to the mailbox. If the maximum size of a message for the mailbox is less than size, then the smaller size is used. In this case, size is updated with the actual number of bytes copied.

If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- **mbox**: Message mailbox ID.
- **msg**: Pointer to the message to unread.
- **size**: Pointer to maximum size of the message in bytes.
- **block_type**: Must be MSG_NOBLOCK.
- **placement**: Specifies where to insert msg in the message queue.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by mbox is not valid.

010:016 EOS_MAUI_BADVALUE
placement or block_type is not valid.

010:028 EOS_MAUI_MBOXFULL
The mailbox is full.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with msg_init().

010:045 EOS_MAUI_NOTIMPLEMENTED
block_type is set to MSG_BLOCK.

Indirect Errors

=os_sema_p()
=os_sema_v()
=os_sigmask()
=os_send()


See Also
msg_create_mbox()
msg_open_mbox()
msg_read()
msg_readn()
msg_send_sig()
msg_set_mask()
msg_unreadn()
msg_write()
msg_writen()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_PLACEMENT
msg_write()

Write a Message to a Mailbox

Syntax

```c
error_code msg_write(MSG_MBOX_ID mbox,
                     const void *msg,
                     MSG_BLOCK_TYPE block_type,
                     MSG_PLACEMENT placement)
```

Description

msg_write() writes the specified message msg to the mailbox mbox.

block_type must be MSG_BLOCK or MSG_NOBLOCK. If block_type is set to MSG_BLOCK, then this function currently returns EOS_MAUI_NOTIMPLEMENTED.

placement specifies where the message is inserted in the queue. If set to MSG_AT_HEAD, then the message is inserted at the head of the queue and is the first one read by msg_read(). If set to MSG_AT_TAIL (typical) it is inserted at the tail of the queue and is read after all other messages.

The message time msg->time is modified to reflect the current system time. This is the time that the message is being queued. The process ID msg->pid is set to the ID of the process calling this function. The current message mask (see msg_set_mask()) may prevent this message from being queued. In this case the error EOS_MAUI_MASKED is returned.

*msg must point to a buffer large enough to hold the largest possible message (size specified in msg_create_mbox() or returned in msg_open_mbox()). This function copies the contents of msg to an internal buffer, so the caller is free to modify it after this function returns.

If you previously called msg_send_sig() to request a signal when a message is available, the signal is sent before returning from this function.
If an error occurs in `msg_write()` after the message has been placed in the queue, the message is left in the queue. Therefore, it is possible to receive an error even after the message has been sent.

`_os_send()` is called after placing the message in the queue. `_os_send()` will return an `EOS_SIGNAL` when the target process’ signal queue is full. In this case the message was written but the process which requested the `msg_sendsig()` (typically the reader of the mailbox is not notified. `_os_send()` could also return `EOS_IPRCID` if a process requesting the signal has died/quit without releasing the `msg_sendsig()`.

If an error is detected before the message is queued, processing stops and the error is returned. If successful, this function returns `SUCCESS`.

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**
- `mbox`: Message mailbox ID.
- `*msg`: Pointer to message to write.
- `block_type`: `MSG_BLOCK` or `MSG_NOBLOCK`.
- `placement`: Where to write `msg` in queue.

**Non-Fatal Errors**
- **010:008 EOS_MAUI_BADID**: The ID specified by `mbox` is not valid.
- **010:016 EOS_MAUI_BADVALUE**: `placement` or `block_type` is not valid.
- **010:027 EOS_MAUI_MASKED**: The message type `msg->type` is not allowed by the current message mask.
- **010:028 EOS_MAUI_MBOXFULL**: The mailbox is full.
10 Messaging Functions

010:036 EOS_MAUINI_NOINIT

This API has not been initialized with msg_init().

010:045 EOS_MAUINI_NOTIMPLEMENTED

block_type is set to MSG_BLOCK.

Indirect Errors

_os_ev_set()
_os_getsys()
_os_sema_p()
_os_sema_v()
_os_send()
_os_sigmask()


See Also

msg_create_mbox()
msg_open_mbox()
msg_read()
msg_readn()
msg_send_sig()
msg_set_mask()
msg_unread()
msg_unreadn()
msg_writen()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_PLACEMENT
msg_writen()
Write N Bytes of a Message to a Mailbox

Syntax

```c
error_code
msg_writen(MSG_MBOX_ID mbox, const void *msg,
    size_t *size, MSG_BLOCK_TYPE block_type,
    MSG_PLACEMENT placement)
```

Description

msg_writen() works exactly like msg_write() except that msg_writen() allows you to specify the maximum number of bytes in the message.

At most, `size` bytes are copied from `msg` to the mailbox. If the maximum size of a message for the mailbox is less than `size`, then the smaller `size` is used. In this case, `size` is updated with the actual number of bytes copied.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `mbox` Message mailbox ID.
- `*msg` Pointer to message to write.
- `*size` Maximum size of message in bytes.
- `block_type` `MSG_BLOCK` or `MSG_NOBLOCK`.
- `placement` Where to write `msg` in queue.
10 Messaging Functions

Non-Fatal Errors

010:008 EOS_MAUI_BADID  The ID specified by mbox is not valid.
010:016 EOS_MAUI_BADVALUE  placement or block_type is not valid.
010:027 EOS_MAUI_MASKED  The message type msg->type is not allowed by the current message mask.
010:028 EOS_MAUI_MBOXFULL  The mailbox is full.
010:036 EOS_MAUI_NOINIT  This API has not been initialized with msg_init().
010:045 EOS_MAUI_NOTIMPLEMENTED  block_type is set to MSG_BLOCK.

Indirect Errors

=os_ev_set()  See Ultra C Library Reference.
=os_getsys()  
=os_sema_p()
=os_sema_v()
=os_send()
=os_sigmask()
See Also

msg_create_mbox()
msg_open_mbox()
msg_read()
msg_readn()
msg_send_sig()
msg_set_mask()
msg_unread()
msg_unreadn()
msg_write()
MSG_BLOCK_TYPE
MSG_MBOX_ID
MSG_PLACEMENT
Chapter 11: Text Functions
**Syntax**

```c
error_code
txt_create_context(TXT_CONTEXT_ID *ret_context,
                   GFX_DEV_ID gfxdev)
```

**Description**

txt_create_context() creates a new text context object. It is used in all subsequent text drawing functions. The following table shows the default value for each parameter and the functions for modifying them.

**Table 11-1 Default Values of Text Context Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font</td>
<td>NULL</td>
<td>txt_set_context_font()</td>
</tr>
<tr>
<td>Character padding</td>
<td>0</td>
<td>txt_set_context_cpad()</td>
</tr>
<tr>
<td>Mixing mode</td>
<td>BLT_MIX_REPLACE</td>
<td>txt_set_context_mix()</td>
</tr>
<tr>
<td>Expansion table entries</td>
<td>0</td>
<td>txt_set_context_exptbl()</td>
</tr>
<tr>
<td>Pixel expansion table</td>
<td>NULL</td>
<td>txt_set_context_exptbl()</td>
</tr>
<tr>
<td>Transparent pixel value</td>
<td>0</td>
<td>txt_set_context_trans()</td>
</tr>
</tbody>
</table>
The context ID is returned in ret_context. A pointer to this variable should be passed to txt_create_context(). Use txt_destroy_context() to destroy this object when it is no longer needed. Use blt_get_context() to get the current settings in a context.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

*ret_context  
gfxdev

<table>
<thead>
<tr>
<th>Offset pixel value</th>
<th>0</th>
<th>txt_set_context_ofs()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination drawmap</td>
<td>NULL</td>
<td>txt_set_context_dst()</td>
</tr>
<tr>
<td>Origin</td>
<td>0, 0</td>
<td>txt_set_context_origin()</td>
</tr>
<tr>
<td>Drawing area</td>
<td>x=0, y=0, w=GFX_DIMEN_MAX, h=GFX_DIMEN_MAX</td>
<td>txt_set_context_draw()</td>
</tr>
</tbody>
</table>

| Number of clipping areas | 0 | txt_set_context_clip() |

**Table 11-1  Default Values of Text Context Parameters (continued)**

The context ID is returned in ret_context. A pointer to this variable should be passed to txt_create_context(). Use txt_destroy_context() to destroy this object when it is no longer needed. Use blt_get_context() to get the current settings in a context.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

*ret_context  
gfxdev
11 Text Functions

Fatal Errors
010:036 EOS_MAUI_NOINIT
This API has not been initialized with `txt_init()`.

Indirect Errors
- `blt_create_context()`
- `mem_malloc()`
- `txt_set_context_clip()`
- `txt_set_context_draw()`
- `txt_set_context_cpad()`
- `txt_set_context_dst()`
- `txt_set_context_exptbl()`
- `txt_set_context_font()`
- `txt_set_context_mix()`
- `txt_set_context_ofs()`
- `txt_set_context_origin()`
- `txt_set_context_trans()`

See Also
- `txt_destroy_context()`
- `BLT_MIX`
- `GFX_DEV_ID`
- `TXT_CONTEXT_ID`
txt_create_font()
Create a Font Object

Syntax

```c
error_code
txt_create_font(TXT_FONT **ret_font, u_int32 shade,
    u_int8 num_ranges)
```

Description

txt_create_font() creates a new font object with the specified number of glyph ranges. All entries in the object are cleared and must be filled in by the caller before the object is used. For most fonts, num_ranges is one. However, if more than one range of glyphs is required, such as 7/15 bit fonts, then this parameter may be larger. The font object is allocated from shade and a pointer to it is returned in ret_font. A pointer to this variable should be passed to txt_create_font(). Use txt_destroy_font() to destroy this object when it is no longer needed.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

**ret_font
Pointer to font object pointer.
shade
Memory shade allocated for the font object.
num_ranges
Number of ranges in glyphs.

Fatal Errors

010:016 EOS_MAUI_BADVALUE
The number of ranges num_ranges is 0.
This API has not been initialized with `txt_init()`.

### Indirect Errors

`mem_malloc()`

### See Also

- `txt_destroy_font()`
- `txt_set_context_font()`
- `TXT_FONT`
txt_destroy_context()

Destroy a Text Context Object

**Syntax**

```c
error_code
txt_destroy_context(TXT_CONTEXT_ID context)
```

**Description**

`txt_destroy_context()` destroys the text context object identified by `context`.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `context` Text context ID.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**  The ID specified by `context` is not valid.

- **010:036 EOS_MAUI_NOINIT**  This API has not been initialized with `txt_init()`.

**Indirect Errors**

- `blt_destroy_context()`
- `mem_free()`

**See Also**

- `txt_create_context()`
- `TXT_CONTEXT_ID`
txt_destroy_font()

Destroy a Font Object

Syntax

```c
error_code
txt_destroy_font(TXT_FONT *font)
```

Description

txt_destroy_font() destroys the specified font object font. Only call this function to destroy fonts created with txt_create_font(). If successful, this function returns SUCCESS.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

*font Pointer to font object.

Non-Fatal Errors

010:036 EOS_MAUI_NOINIT This API has not been initialized with txt_init().

Indirect Errors

mem_free()

See Also

txt_create_font() TXT_FONT
txt_draw_mbs()
Draw a Multi-Byte String

Syntax
error_code
txt_draw_mbs(GFX_DIMEN *ret_width,
    TXT_CONTEXT_ID context,
    const char *string,
    size_t *char_len,
    GFX_POS dstx,
    GFX_POS dsty,
    const int16 pad_array[])

Description
txt_draw_mbs() draws the specified multi-byte string at the pixel location dstx, dsty using the specified text context. The coordinate dstx, dsty specifies where the left-most pixel of the first character's baseline is drawn.

char_len specifies the maximum number of characters to be drawn. A pointer to char_len should be passed so that txt_draw_mbs() can update it with the actual number of characters processed.

The C library function mbtowc() extracts each multibyte character. Processing continues until char_len characters have been processed, or a NULL byte is detected.

If pad_array is not NULL, it should point to an array of values that indicate the additional padding between characters in the output string.

If ret_width is not NULL, then it should point to the location where txt_draw_mbs() writes the width (in pixels) of the text string drawn.

If part (or all) of the text string does not fit on the destination drawmap, then that portion (or all of it) is clipped.

If successful, this function returns SUCCESS.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_width Pointer to the width of text string.
context Text context ID.
*string Pointer to text string.
*char_len Pointer to number of characters.
dstx, dsty X and Y coordinates of baseline of first character on the display.
pad_array[] Array of values that indicate additional padding between characters in the output string.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by context is not valid.
010:010 EOS_MAUI_BADMBC Bad multi-byte character.
010:032 EOS_MAUI_NODSTDMAP No destination drawmap has been set in the specified context.
010:033 EOS_MAUI_NOEXPTABLE No expansion table has been set in the specified context, and one is required.
010:034 EOS_MAUI_NOFONT No font has been set in the specified context.
010:036 EOS_MAUI_NOINIT This API has not been initialized with txt_init().
Text Functions

**Indirect Errors**

- `blt_copy_block()`
- `blt_copy_next_block()`
- `blt_expd_block()`
- `blt_expd_next_block()`
- `blt_set_context_src()`

**See Also**

- `txt_drawwcs()`
- `txt_create_context()`
- `GFX_DIMEN`
- `GFX_POS`
- `TXT_CONTEXT_ID`
**txt_draw_wcs()**

**Draw a Wide Character String**

**Syntax**

```c
error_code
txt_draw_wcs(GFX_DIMEN *ret_width,
             TXT_CONTEXT_ID context,
             const wchar_t *string,
             size_t *char_len,
             GFX_POS dstx,
             GFX_POS dsty,
             const int16 pad_array[])
```

**Description**

txt_draw_wcs() **draws the specified wide character string** at the **pixel location** `dstx, dsty` **using the specified text context.** The **coordinate** `dstx, dsty` **specifies where the left-most pixel of the first character’s baseline is drawn.**

`char_len` **specifies the maximum number of characters to be drawn.** A **pointer to** `char_len` **should be passed so that txt_draw_wcs() can update it with the actual number of characters processed.**

Each wide character is processed until `char_len` **characters have been processed**, or the value 0 is detected.

If `pad_array` is not **NULL**, it should point to an array of values that **indicate the additional padding between characters in the output string.**

If `ret_width` is not **NULL**, then it should point to the location where **txt_draw_wcs()** writes the width (in pixels) of the text string drawn.

If part (or all) of the text string does not fit on the destination drawmap, **then that portion (or all of it) is clipped.**

If successful, this function returns **SUCCESS.**
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
*ret_width Pointer to the width of text string.
context Text context ID.
*string Pointer to text string.
*char_len Pointer to number of characters.
dstx Y coordinate of baseline of first character on the display.
dsty X coordinate of baseline of first character on the display.
pad_array[] Array of values indicating additional padding between characters in the output string.

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by context is not valid.
010:032 EOS_MAUI_NODSTDMAP No destination drawmap has been set in the specified context.
010:033 EOS_MAUI_NOEXPTABLE No expansion table has been set in the specified context, and one is required.
010:034 EOS_MAUI_NOFONT No font has been set in the specified context.
010:036 EOS_MAUI_NOINIT This API has not been initialized with txt_init().
Text Functions

Indirect Errors
blt_copy_block()
blt_copy_next_block()
blt_expd_block()
blt_expd_next_block()
blt_set_context_src()

See Also
txt_create_context()
txt_draw_mbs()
GFX_DIMEN
GFX_POS
TXT_CONTEXT_ID
txt_get_context()
Get Text Context Parameters

Syntax

```c
error_code
txt_get_context(TXT_CONTEXT_PARAMS *ret_context_params,
                   TXT_CONTEXT_ID context)
```

Description

`txt_get_context()` returns the current parameters for the specified text context.

The parameters are returned in `ret_context_params`. A pointer to this structure should be passed to `txt_get_context()`. The caller must ensure that `ret_context_params` points to storage large enough to hold the information. If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `*ret_context_params` Pointer to parameters.
- `context` Text context ID.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**
  The ID specified by `context` is not valid.

- **010:036 EOS_MAUI_NOINIT**
  This API has not been initialized with `txt_init()`.
See Also

txt_set_context_clip()
txt_set_context_cpad()
txt_set_context_draw()
txt_set_context_dst()
txt_set_context_mix()
txt_set_context_exptbl()
txt_set_context_font()
txt_set_context_ofs()
txt_set_context_origin()
txt_set_context_trans()
TXT_CONTEXT_ID
TXT_CONTEXT_PARAMS
txt_get_mbs_width()

Get Width of a Multi-Byte String

Syntax

error_code
txt_get_mbs_width(GFX_DIMEN *ret_swidth,
                  GFX_DIMEN ret_cwidth[],
                  TXT_CONTEXT_ID context,
                  const char *string,
                  size_t *max_chars)

Description

txt_get_mbs_width() is used to get the width of the specified
multi-byte string of characters.

If ret_swidth is not NULL, then it should point to the location where
txt_get_mbs_width() writes the width (in pixels) of the text string.

If ret_cwidth is NULL, then it is ignored. Otherwise, it should point to
an array (with max_chars entries) where the width (in pixels) of each
character is written.

max_chars specifies the maximum number of characters to be
processed. Since each character may consist of more than one byte,
max_chars should not be interpreted as the number of bytes in a
string. A pointer to max_chars should be passed so that
txt_get_mbs_width() can update it with the actual number of
characters processed.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

*ret_swidth Pointer to width of text string.
ret_cwidth[] Array containing width of each character.
context            Text context ID.
*string              Pointer to string.
*max_chars           Points to maximum number of characters to be processed.

Non-Fatal Errors

010:008  EOS_MAUI_BADID
The ID specified by context is not valid.

010:034  EOS_MAUI_NOFONT
No font has been set in the specified context.

010:036  EOS_MAUI_NOINIT
This API has not been initialized with txt_init().

See Also

txt_get_wcs_width()
GFX_DIMEN
TXT_CONTEXT_ID
**Syntax**

```c
error_code
txt_get_wcs_width(GFX_DIMEN *ret_swidth,
                GFX_DIMEN ret_cwidth[],
                TXT_CONTEXT_ID context,
                const wchar_t *string,
                size_t *max_chars)
```

**Description**

`txt_get_wcs_width()` is used to get the width of the specified wide character string of characters.

If `ret_swidth` is not NULL, then it should point to the location where `txt_get_wcs_width()` writes the width (in pixels) of the text string.

If `ret_cwidth` is NULL, then it is ignored. Otherwise, it should point to an array (with `max_chars` entries) where the width (in pixels) of each character is written.

`max_chars` specifies the maximum number of characters to be processed. Since each character consists of more than one byte, `max_chars` should not be interpreted as the number of bytes in a string. A pointer to `max_chars` should be passed so that `txt_get_wcs_width()` can update it with the actual number of characters processed.

If successful, this function returns `SUCCESS`.

**Attributes**

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
</tbody>
</table>

**Parameters**

- `*ret_swidth` Pointer to width of text string.
ret_cwidth[]
context
*string
*max_chars

Array containing width of each character.
Text context ID.
Pointer to string.
Points to maximum number of characters to be processed.

Non-Fatal Errors

010:008  EOS_MAUI_BADID  The ID specified by context is not valid.
010:034  EOS_MAUI_NOFONT  No font has been set in the specified context.
010:036  EOS_MAUI_NOINIT  This API has not been initialized with txt_init().

See Also

txt_get_mbs_width()
GFX_DIMEN
TXT_CONTEXT_ID
txt_init()

Initialize the Text API

Syntax

error_code
txt_init(void)

Description

txt_init() initializes the text API. This function must be called prior to a call to any other text function unless otherwise noted by that function.

This API depends on the Shaded Memory and Bit-BLT APIs. Therefore mem_init() and blt_init() are called by this function.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Indirect Errors

blt_init()
mem_init()

See Also

txt_term()
txt_set_context_clip()
Set Clipping Area

Syntax
error_code
txt_set_context_clip(TXT_CONTEXT_ID context,
                     u_int32 num_clip_areas,
                     const GFX_RECT clip_areas[])

Description
txt_set_context_clip() sets the clipping area for the specified
text context to clip_areas.
clip_areas is an array of num_clip_areas rectangles. These
rectangles may overlap. Together, these rectangles define an area
known as a clipping area. As text drawing is performed by functions
using this context, pixels within this area are automatically clipped (not
drawn).
If clip_areas is NULL or num_clip_areas is 0, then no clipping
area is defined. In this case, text drawing is clipped only if it is outside
the bounds of the destination drawmap.
In addition to the clipping defined above, all drawing outside of the
drawing area is clipped. See txt_set_context_draw() for
information on setting the drawing area. If successful, this function
returns SUCCESS.

Note
Do not use this function if you are currently using the Windowing API.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:             User
Threads:          Safe
Parameters

context

Text context ID.

num_clip_areas

Number of clipping areas.

clip_areas

Specifies clipping areas.

Non-Fatal Errors

010:008 EOS_MAUI_BADID

The ID specified by context is not valid.

010:036 EOS_MAUI_NOINIT

API not initialized with txt_init().

See Also

txt_create_context()
txt_get_context()
txt_set_context_draw()
GFX_RECT
TXT_CONTEXT_ID
**txt_set_context_cpad()**

Set Character Padding

**Syntax**

```c
error_code
txt_set_context_cpad(TXT_CONTEXT_ID context,
                     int16 cpad)
```

**Description**

`txt_set_context_cpad()` sets the character padding for the specified text context to `cpad`.

The character padding specifies the number of additional pixels to place between characters when they are drawn. If this number is negative, the characters will overlay by that many pixels.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `context` Text context ID.
- `cpad` Specifies character padding.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID** The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT** This API has not been initialized with `txt_init()`.

**See Also**

- `txt_get_context()`
- `TXT_CONTEXT_ID`
**Syntax**

```c
error_code txt_set_context_draw (TXT_CONTEXT_ID context,
                                 GFX_POS x,
                                 GFX_POS y,
                                 GFX_DIMEN width,
                                 GFX_DIMEN height)
```

**Description**

`txt_set_context_draw()` sets the drawing area for the specified text context. All drawing outside this rectangle is clipped (not drawn). The upper-left corner of the rectangle is defined by `x` and `y`. The `width` and `height` define the size of the rectangle. Use 0 for `x` and `y`, and `GFX_DIMEN_MAX` for `width` and `height` to make the entire drawmap available for drawing.

If successful, this function returns `SUCCESS`.

**Note**

Do not use this function if you are currently using the Windowing API.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `context`: Text context ID.
- `x, y`: Coordinates of upper-left corner of drawing area.
width, height

Width/height of drawing areas in pixels.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by context is not valid.

010:016 EOS_MAUI_BADVALUE
The width or height is zero.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with txt_init().

See Also

txt_get_context()
txt_set_context_clip()
GFX_DIMEN
GFX_DIMEN_MAX
GFX_POS
TXT_CONTEXT_ID
txt_set_context_dst()
Set Destination Drawmap

Syntax

```c
error_code
txt_set_context_dst(TXT_CONTEXT_ID context,
                    const GFX_DMAP *dstdmap)
```

Description
txt_set_context_dst() sets the destination drawmap value for the specified text context to dstdmap. If set to NULL, then the destination drawmap becomes undefined and text operations that require it return the error EOS_MAUI_NODSTDMAP.

If the contents of the drawmap object dstdmap are changed after calling this function, you must call it again to register the changes with this context object. If you delete the dstdmap, it must be removed from this context.

If successful, this function returns SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
context Text context ID.
*dstdmap Pointer to destination drawmap.

Non-Fatal Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:008 EOS_MAUI_BADID</td>
<td>The ID specified by context is not valid.</td>
</tr>
<tr>
<td>010:036 EOS_MAUI_NOINIT</td>
<td>This API has not been initialized with txt_init().</td>
</tr>
</tbody>
</table>
No pixel memory has been assigned to the destination drawmap dstdmap.

Indirect Errors

blt_set_context_dst()

See Also

txt_get_context()
GFX_DMAP
TXT_CONTEXT_ID
**txt_set_context_exptbl()**

Set Pixel Expansion Table

### Syntax

```c
error_code txt_set_context_exptbl(TXT_CONTEXT_ID context,
                                 u_int8 num_values,
                                 const GFX_PIXEL exptbl[])
```

### Description

`txt_set_context_exptbl()` sets the pixel expansion table for the specified text context to `exptbl`. `num_values` specifies the number of values in the table `exptbl`.

If `exptbl` is set to NULL, then the pixel expansion table becomes undefined and text operations that require an expansion table return the error `EOS_MAUI_NOEXPTABLE`.

If the contents of the expansion table `exptbl` are changed after calling this function, you must call it again to register the changes with this context object. Since this function makes a copy of the data pointed to by `exptbl`, you may destroy `exptbl` immediately after calling `txt_set_context_exptbl()`.

If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `context`: Text context ID.
- `num_values`: Number of values in expansion table.
- `exptbl[]`: Points to an expansion table.
Text Functions

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by `context` is not valid.

010:016 EOS_MAUI_BADVALUE
The value `num_values` must be equal to 2.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with `txt_init()`.

Indirect Errors

`blt_set_context_exptbl()`

See Also

`txt_get_context()`
`GFX_PIXEL`
`TXT_CONTEXT_ID`
**Text Functions**

**txt_set_context_font()**

Set Font to Use

**Syntax**

```c
error_code
txt_set_context_font(TXT_CONTEXT_ID context, const TXT_FONT *font)
```

**Description**

`txt_set_context_font()` sets the font for the specified text context to `font`. If `font` is set to `NULL`, then the font becomes undefined and drawing operations return the error `EOS_MAUI_NOFONT`.

If the contents of the drawmap object `font` are changed after calling this function, you must call it again to register the changes with this context object. If you delete the `font`, it must be removed from this context.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context** Text context ID.
- ***font** Pointer to font object.

**Non-Fatal Errors**

- **010:005 EOS_MAUI_BADDEFCHAR** Bad (missing) default character.
- **010:008 EOS_MAUI_BADID** The ID specified by `context` is not valid.
Incompatible coding methods in the bitmaps for different ranges of this font.

This API has not been initialized with `txt_init()`.

See Also
- `txt_get_context()`
- `TXT_CONTEXT_ID`
- `TXT_FONT`
**txt_set_context_mix()**

Set Mixing Mode

**Syntax**

```c
error_code
txt_set_context_mix(TXT_CONTEXT_ID context, BLT_MIX mixmode)
```

**Description**

`txt_set_context_mix()` sets the mixing mode for text operations in the specified text context to `mixmode`. The mixing mode specifies the way source pixels are transferred to the destination drawmap. If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **context**
  - Text context ID.
- **mixmode**
  - Specifies source to destination pixel mixing mode.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `context` is not valid.
- **010:016 EOS_MAUI_BADVALUE**
  - The `mixmode` value is not legal.
- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `txt_init()`.
11 Text Functions

Indirect Errors
blt_set_context_c pymix()
blt_set_context_exp mix()

See Also
txt_get_context()
BLT_MIX
TXT_CONTEXT_ID
**Syntax**

```c
error_code txt_set_context_ofs(TXT_CONTEXT_ID context, GFX_PIXEL ofspixel)
```

**Description**

`txt_set_context_ofs()` sets the offset pixel value for the specified text context to `ofspixel`. The offset pixel value is added to the source pixels before they are transferred to the destination when `BLT_MIX_SPO` is used.

If successful, this function returns `SUCCESS`.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

- `context`: Text context ID.  
- `ofspixel`: Offset pixel value added to source pixels before transfer.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `context` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `txt_init()`.

**Indirect Errors**

`blt_set_context_ofs()`
See Also

- `txt_get_context()`
- `BLT_MIX`
- `GFX_PIXEL`
- `TXT_CONTEXT_ID`
txt_set_context_origin()
Set Drawing Origin

Syntax
error_code
txt_set_context_origin(TXT_CONTEXT_ID context,
                     GFX_POS x, GFX_POS y)

Description
txt_set_context_origin() sets the drawing origin for the
specified text context. All coordinates used for drawing are relative to
this position. The origin is specified by x and y. This is considered 0, 0
for all drawing operations.
If successful, this function returns SUCCESS.

Note
Do not use this function if you are currently using the Windowing API.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
category context
x, y

Non-Fatal Errors
010:008 EOS_MAUI_BADID The ID specified by context is not valid.
010:036 EOS_MAUI_NOINIT

This API has not been initialized with txt_init().

See Also

txt_get_context()
GFX_POS
TXT_CONTEXT_ID
txt_set_context_trans()
Set Transparent Pixel Value

Syntax

```
error_code
txt_set_context_trans(TXT_CONTEXT_ID context, GFX_PIXEL transpixel)
```

Description

txt_set_context_trans() sets the transparent pixel value for the specified text context to transpixel. The transparent pixel value is used to filter out source pixels when they are transferred to the destination.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- context: Text context ID.
- transpixel: Sets transparent pixel value.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**: The ID specified by context is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with txt_init().

Indirect Errors

- blt_set_context_trans()
See Also

txt_get_context()
GFX_PIXEL
TXT_CONTEXT_ID
txt_set_error_action()
Set Action to Take in Error Handler

Syntax

```c
error_code
txt_set_error_action(MAUI_ERR_LEVEL debug_level,
                     MAUI_ERR_LEVEL passback_level,
                     MAUI_ERR_LEVEL exit_level)
```

Description

txt_set_error_action() sets the action to take in the error handler when a function in this API detects an error. This function may be called prior to calling txt_init(). Following is the table of error levels. The least severe error is listed first.

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed, but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
</tbody>
</table>
debug_level sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is MAUI_ERR_ANY.

passback_level sets the minimum error level that causes the error handler to return the error. For less severe errors, SUCCESS is returned. The default pass-back level is MAUI_ERR_NON_FATAL.

exit_level sets the minimum error level that causes the error handler to call exit(). In this case the program exits with the error code that caused the error handler to be called. The default debug level is MAUI_ERR_NONE.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

d debug_level Minimum error level that causes the error handler to print a message to standard error.

passback_level Minimum error level that causes the error handler to return the error.

exit_level Minimum error level that causes the error handler to call exit().

**Non-Fatal Errors**

None
See Also

txt_init()
txt_term()

Terminate the Text API

Syntax

error_code
txt_term(void)

Description

txt_term() terminates the text API. This API depends on the Shaded Memory and Bit-BLT APIs. Therefore, mem_term() and blt_term() are called by this function.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Non-Fatal Errors

010:036 EOS_MAUI_NOINIT
This API has not been initialized with txt_init().

Indirect Errors

blt_term()
mem_term()

See Also

txt_init()
Chapter 12: Windowing Functions
**win_alloc_cmap_cell()**

Allocate a Single Private Cell

**Syntax**

```c
error_code win_alloc_cmap_cell(GFX_PIXEL *ret_pixel,
                                WIN_CMAP_ID cmap,
                                GFX_COLOR_VALUE color)
```

**Description**

`win_alloc_cmap_cell()` allocates a single private cell in the colormap `cmap` according to the color value, specified by `color`.

The function allocates one new private cell. It does not re-use the existing ones, even if they belong to the calling process. The cell index is returned in `ret_pixel`. A pointer to this variable should be passed to `win_alloc_cmap_cell()`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `ret_pixel` : Pixel value returned.
- `cmap` : Colormap ID.
- `color` : Color to be assigned to the cell.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from the `maui_win` process.
- **010:008 EOS_MAUI_BADID**: The ID specified by `cmap` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `win_init()`.
No cells are available in colormap.

Indirect Errors
msg_read()
msg_write()

See Also
win_alloc_cmap_color()
win_alloc_cmap_cells()
win_alloc_cmap_colors()
WIN_CMAP_ID
GFX_PIXEL
GFX_COLOR_VALUE
win_alloc_cmap_cells()
Allocate Group of Color Cells

Syntax

```c
error_code
win_alloc_cmap_cells(GFX_PIXEL ret_pixels[],
                     WIN_CMAP_ID cmap,
                     u_int16 num_colors,
                     BOOLEAN contig,
                     const GFX_COLOR_VALUE colors[])
```

Description

`win_alloc_cmap_cells()` allocates a group of `num_colors` color cells from the colormap `cmap`. It does not allocate memory for the cells because they were already allocated by `win_create_cmap()`. These cells become private to the process that allocated them. Other processes are not allowed to read or modify.

If `colors` is not `NULL`, then it must contain at least `num_colors` entries. Each entry must contain the color to use to initialize the corresponding entry `ret_pixels`.

The color cells allocated are returned in `ret_pixels`. A pointer to this array should be passed to `win_alloc_cmap_cells()`. The caller must ensure that `ret_pixels` points to storage large enough to hold `num_colors` pixel values. You should free these cells with `win_free_cmap_cells()` when they are no longer needed.

If `contig` is `TRUE`, the pixel returned in `ret_pixels` is contiguous. `EOS_MAUI_CMAPFULL` is returned if the number of contiguous entries specified by `num_colors` is not available.

`win_alloc_cmap_cells()` assumes that the color type for `colors` matches that specified when the colormap was created with `win_create_cmap()`.

If successful, this function returns `SUCCESS`. 
**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

- `ret_pixels[]`: Array of color cells allocated.  
- `cmap`: Colormap containing the color cells.  
- `num_colors`: Number of color cells to allocate from `cmap`.  
- `contig`: BOOLEAN value indicating if array of color cells must be contiguous.  
- `colors[]`: Array of colors.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from the maui_win process.  
- **010:008 EOS_MAUI_BADID**: The ID specified by cmap is not valid.  
- **010:016 EOS_MAUI_BADVALUE**: The value for `num_colors` is zero.  
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `win_init()`.

**MAUI Win Errors**

- **010:064 EOS_MAUI_CMAPFULL**: The colormap is full or there are not enough contiguous free cells to satisfy the request.

**Indirect Errors**

- `msg_read()`  
- `msg_write()`
See Also

* win_free_cmap_cells()
* win_set_cmap_cells()
* BOOLEAN
* GFX_COLOR_VALUE
* GFX_PIXEL
* WIN_CMAP_ID
Windowing Functions

**win_alloc_cmap_color()**

Allocate a Color

**Syntax**

```c
error_code
win_alloc_cmap_color(GFX_PIXEL *ret_pixel,
                     WIN_CMAP_ID cmap,
                     GFX_COLOR_VALUE color)
```

**Description**

`win_alloc_cmap_color()` searches the colormap `cmap` for a color matching (or very close to) the specified `color`. If a close enough match cannot be found, then a new color cell is allocated and assigned the value `color`.

`win_alloc_cmap_color()` does not allocate memory for the cells because they were already allocated by `win_create_cmap()`.

The cell containing the specified `color` is returned in `ret_pixel`. A pointer to this variable should be passed to `win_alloc_cmap_color()`.

This cell is shared among all processes that request the stored color. Because the cell is shared, this process does not become the owner of the cell. A link count is maintained for shared colors to track the number of users. Use `win_free_cmap_cells()` to decrement the color's link count when finished with it.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
Windowing Functions

Parameters

*ret_pixel
Pointer to the cell containing the specified color.

cmap
Colormap ID.

color
Value of the color to allocate.

Non-Fatal Errors

010:001 EOS_MAUI_BADACK
Bad acknowledgment from the maui_win process.

010:008 EOS_MAUI_BADID
The ID specified by cmap is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init().

MAUI Win Errors

010:064 EOS_MAUI_CMAPFULL
A close match was not found and there are no free cells in the colormap.

Indirect Errors

msg_read()
msg_write()

See Also

win_alloc_cmap_cells()
win_free_cmap_cells()
GFX_COLOR_VALUE
GFX_PIXEL
WIN_CMAP_ID
Windowing Functions

win_alloc_cmap_colors()
Allocate Array of Colors

Syntax

```c
error_code
win_alloc_cmap_colors(GFX_PIXEL ret_pixels[],
                     WIN_CMAP_ID cmap,
                     u_int16 num_colors,
                     const GFX_COLOR_VALUE colors[])
```

Description

`win_alloc_cmap_colors()` allocates `num_colors` of shared cells in the colormap `cmap` according to the color array `colors`.

Depending on the current color matching method, the function either allocates new sharable cells or re-uses the existing ones. Use `win_set_color_match()` to set the color matching method. If no alternative color matching method is specified, exact matching is used.

Cell indices are returned in `ret_pixels` array. A pointer to this array should be passed to `win_alloc_cmap_colors()`. The caller must ensure that `ret_pixels` points to storage large enough to hold `num_colors` pixel values.

These cells are shared among all processors that request the stored color. Because the cell is shared, this process does not become the owner of the cell. A link count is maintained for shared colors to track the number of users. Use `win_free_cmap_cells()` to decrement the color’s link count when finished with it.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
12 Windowing Functions

Parameters

ret_pixels
Array of pixel values returned.
cmap
Colormap ID.
num_colors
Total number of colors to allocate.
colors
Colors to use for allocation.

Non-Fatal Errors

010:001 EOS_MAUI_BADACK
Bad acknowledgment from the mui_win process.
010:008 EOS_MAUI_BADID
The ID specified by cmap is not valid.
010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init().
010:064 EOS_MAUI_CMAPFULL
No cells available in the colormap.

Indirect Errors

msg_read()
msg_write()

See Also

win_alloc_cmap_color()
win_alloc_cmap_cells()
win_alloc_cmap_cell()
win_free_cmap_cells()
win_set_color_match()
WIN_CMAP_ID
GFX_PIXEL
GFX_COLOR_VALUE
win_close_dev()

Close a Windowing Device

Syntax

error_code
win_close_dev(WIN_DEV_ID windev)

Description

win_close_dev() closes the windowing device windev previously opened with win_open_dev(). Do not use win_close_dev() to close a device that you created with win_create_dev(); use win_desctroy_dev() instead.

All windows, colormaps, and cursors created by this process are destroyed and no further messages are written to the application mailbox. However, messages already in the mailbox are unaffected.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

windev Windowing device ID.

Non-Fatal Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:001 EOS_MAUI_BADACK</td>
<td>Bad acknowledgment from the maui_win process.</td>
</tr>
<tr>
<td>010:008 EOS_MAUI_BADID</td>
<td>The ID specified by windev is not valid.</td>
</tr>
<tr>
<td>010:036 EOS_MAUI_NOINIT</td>
<td>This API has not been initialized with win_init().</td>
</tr>
</tbody>
</table>
010:060 EOS_MAUI_NOTOWNER

This process did not open windev with win_open_dev().

Indirect Errors
gfx_clone_dev()
mem_free()
msg_close_mbox()
msg_read()
msg_write()
_os_sema_term()

See _Ultra C Library Reference_.

See Also
win_create_dev()
win_destroy_dev()
win_open_dev()
WIN_DEV_ID
win_close_inpdev()

Close an input Device

**Syntax**

```c
error_code
win_close_inpdev (WIN_DEV_ID windev,
                   INP_DEV_ID inpdev)
```

**Description**

`win_close_inpdev()` closes the input device `inpdev`. Messages from this input device are no longer written to any application mailboxes. Messages already in a mailbox are unaffected.

`win_close_inpdev()` must be called by the same process that called `win_open_inpdev()` to open it. Since the owner is the only process allowed to open the input device, it is also the only process allowed to close it.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `windev` — Windowing device ID.
- `inpdev` — Input device ID.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK** — Bad acknowledgment from the `maui_win` process.
- **010:008 EOS_MAUI_BADID** — The ID specified by `windev` is not valid.
- **010:036 EOS_MAUI_NOINIT** — This API has not been initialized with `win_init()`.
MAUI Win Errors

010:060 EOS_MAUI_NOTOWNER

This process did not open windev.

Indirect Errors

inp_close_dev()
msg_read()
msg_write()

See Also

win_create_dev()
win_open_dev()
win_open_inpdev()
INP_DEV_ID
WIN_DEV_ID
**win_copy_block()**

Copy Rectangular Block of Pixels

**Syntax**

```c
error_code
win_copy_block(WIN *dst_win, WIN *src_win,
                GFX_POS dstx, GFX_POS dsty,
                GFX_POS srcx, GFX_POS srcy,
                GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`win_copy_block()` copies a rectangular area of pixels from the `src_win` to the `dst_win`. The upper-left corner of this area is specified by `srcx` and `srcy` in the source drawmap and `dstx` and `dsty` in the destination drawmap. The dimensions of the area to copy are specified by `width` and `height`.

The source and destination window can be the same, and the source and destination areas can safely overlap if desired (although it can take significantly longer to perform the operation).

The copying takes into account window clipping. If any part of the block is covered by top windows, the `expose` message is sent to the covered part on the destination.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `dst_win` Destination window.
- `src_win` Source window.
- `dstx` X position of upper-left corner in destination drawmap.
Windowing Functions

dsty
Y position of upper-left corner in destination drawmap.

srcx
X position of upper-left corner of copy block in source drawmap.

srcy
Y position of upper-left corner of copy block in source drawmap.

width
Width of copy block in pixels.

height
Height of copy block in pixels.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by dst_win or src_win is not valid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init().

Indirect Errors

blt_copy_block()
blt_set_context_src()
blt_set_context_dst()
msg_read()
msg_write()

See Also

GFX_DIMEN
GFX_POS
**Syntax**

```c
error_code win_create_cmap(WIN_CMAP_ID *ret_cmap,
                          WIN_DEV_ID windev,
                          GFX_COLOR_TYPE color_type)
```

**Description**

`win_create_cmap()` creates a new colormap. The process that called `win_create_cmap()` becomes the owner of the colormap and is the only process allowed to destroy it.

When the colormap is created, color cells are allocated to hold the values. The number of cells is calculated automatically using the pixel depth of the device drawmap and the color type.

The color type for the colormap is `color_type`. Colors must be specified using this type when other colormap functions are called for this colormap.

The colormap ID is returned in `ret_cmap`. A pointer to this variable should be passed to `win_create_cmap()`. Use `win_destroy_cmap()` when this colormap is no longer needed. If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `*ret_cmap` Pointer to colormap ID.
- `windev` Windowing device ID.
- `color_type` Color type for colormap.
12 Windowing Functions

**Fatal Errors**

010:001 EOS_MAUI_BADACK
Bad acknowledgment from the maui_win process.

010:008 EOS_MAUI_BADID
The ID specified by windev is invalid.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init().

**MAUI Win Errors**

010:022 EOS_MAUI_INCOMPATCM
The color_type specified is not compatible with the coding method specified when windev was created with win_create_dev().

**Indirect Errors**

msg_read()
msg_write()

**See Also**

win_create_dev()
win_destroy_cmap()
win_get_cmap_cells()
win_get_cmap_free()
GFX_COLOR_TYPE
WIN_CMAP_ID
WIN_DEV_ID
Windowing Functions

**win_create_cursor()**

Create a Cursor

### Syntax

```c
error_code
win_create_cursor(WIN_DEV_ID windev,
                  u_int32 cursor_id,
                  WIN_CURSOR *cursor)
```

### Description

`win_create_cursor()` creates a new cursor. The process that called `win_create_cursor()` becomes the owner of the cursor and is the only process allowed to destroy it.

The cursor attributes are defined by `cursor`, and the `cursor_id` specifies the ID to be associated with it. This ID may be used by all applications that are using `windev`. Therefore, one application may create a cursor with a given ID and other applications may share it by using the same ID.

The contents of `cursor` should not be modified after `win_create_cursor()` is called and the memory associated with it should not be freed until `win_destroy_cursor()` is called.

If the number of colors specified in `cursor->bitmap->palette` is larger than two (2), `EOS_MAUI_BADISZE` is returned. If the width or height of the cursor is larger than 32 pixels, `EOS_MAUI_BADISZE` is also returned.

### Note

These are limitations of the `maui_win` software cursor. If a graphic hardware cursor is supported by the platform and graphic driver, then different limits may apply. For instance, some hardware supports 64x64 pixel cursors in some display modes.
The cursor coding method is `cursor->bitmap->coding_method` and must be compatible with the coding method used by the windowing device.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `windev` Windowing device ID.
- `cursor_id` Cursor ID.
- `*cursor` Pointer to cursor attributes.

**Fatal Errors**

- **010:001 EOS_MAUI_BADACK**
  Bad acknowledgment from the `maui_win` process.

- **010:008 EOS_MAUI_BADID**
  The ID specified by `windev` is not valid.

- **010:026 EOS_MAUI_ISRESERVED**
  The `cursor_id` may not be used because it is reserved by MAUI.

- **010:036 EOS_MAUI_NOINIT**
  This API has not been initialized with `win_init()`.

**MAUI Win Errors**

- **010:015 EOS_MAUI_BADSIZE**
  The number of colors specified by the cursor bitmap is more than allowed for the device, or the size of the cursor (width or height) is too large.
The specified `cursor_id` has already been defined for the device `windev`.

**Indirect Errors**

`msg_read()`  
`msg_write()`  

**See Also**

`win_create_dev()`  
`win_destroy_cursor()`  
`WIN_CURSOR`  
`WIN_DEV_ID`
win_create_dev()
Create a Windowing Device

Syntax

```c
error_code
win_create_dev(WIN_DEV_ID *ret_windev,
    GFX_DEV_ID *ret_gfxdev,
    WIN_ID *ret_root,
    const char *win_devname,
    const char *gfx_devname,
    u_int8 res_index,
    u_int8 cm_index,
    MSG_MBOX_ID mbox)
```

Description

`win_create_dev()` creates a windowing device named `win_devname`. The process calling this function becomes the owner of the windowing device. It also becomes the owner of the root window.

The graphics device named `gfx_devname` is opened and its resolution and coding method are set using `res_index` and `cm_index` respectively.

- `res_index` is an index into the `res_info` member of the device capability structure `GFX_DEV_CAP`. `cm_index` is an index into the `cm_info` member of the same structure.

The graphics device ID is returned in `ret_gfxdev`. A pointer to this variable should be passed to `win_create_dev()`. Do not close this device using `gfx_close_dev()`. It is automatically returned by `win_destroy_dev()`.

Initially, there are no input devices associated with the windowing device. You must call `win_open_inpdev()` to associate each input device with the windowing device.

After `win_create_dev()` succeeds, other processes may share the windowing device by calling `win_open_dev()` with the same device name.
The windowing device ID is returned in ret_windev. A pointer to this variable should be passed to win_create_dev(). Use win_destroy_dev() when this device is no longer needed.

The root window for the windowing device is created and its state is set to active. The ID for this window is returned in ret_root. A pointer to this variable should be passed to win_create_dev(). Do not destroy this window with win_destroy_win(). It is automatically destroyed by win_destroy_dev().

mbox specifies the mailbox that should receive windowing messages. The mailbox must be opened using msg_create_mbox() or msg_open_mbox() prior to calling win_create_dev().

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

*ret_windev Pointer to windowing device ID windev.
*ret_gfxdev Pointer to graphics device ID gfxdev.
*ret_root Pointer to windowing process ID.
*win_devname Windowing device name.
*gfx_devname Graphics device name.
res_index Index to res_info member of GFX_DEV_CAP structure.

**Fatal Errors**

010:001 EOS_MAUI_BADACK Bad acknowledgment from the mauli_win process.
12 Windowing Functions

010:004 EOS_MAUI_BADCOMPATLEVEL
Bad compatibility level reported by the maui_win process.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init().

010:038 EOS_MAUI_NOMAUIP
The maui_win module is not loaded.

010:050 EOS_MAUI_TOOLONG
The windowing device name win_devname is longer than WIN_MAX_DEVNAME or the graphics device name gfx_devname is longer than GFX_MAX_DEV_NAME.

**MAUI Win Errors**

010:002 EOS_MAUI_BADCODEMETH
The coding method in the cm_index entry of the device capabilities structure is not valid.

010:016 EOS_MAUI_BADVALUE
Either res_index, cm_index, or num_cursor_colors is too large.

**Indirect Errors**
gfx_clone_dev()
mem_malloc()
msg_get_mbox_status()
msg_open_mbox()
msg_read()
msg_write()

For _os_* functions, see *Ultra C Library Reference*

_os_exec()
_os_fork()
_os_sema_init
See Also

msg_create_mbox()
msg_open_dev
win_close_dev()
win_destroy_win()
win_open_dev()
win_open_inpdev()
GFX_DEV_ID
MSG_MBOX_ID
WIN_DEV_ID
WIN_ID
WIN_MAX_DEV_NAME
**win_create_win()**

Create a Window

**Syntax**

```c
error_code win_create_win(WIN_ID *ret_win, WIN_ID parent_win,
GFX_POS x, GFX_POS y, GFX_DIMEN width, GFX_DIMEN height,
WIN_MSG_MASK mask, WIN_PLACEMENT placement, ...)
```

**Description**

`win_create_win()` creates a new window as a child of the window `parent_win`. `parent_win` must be the root window or a window owned by this process. The new window must be wholly contained within the parent window.

The following table shows the default value for each parameter and the functions for modifying them:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Modify With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>x, y</td>
<td><code>win_move_win()</code></td>
</tr>
<tr>
<td>Size</td>
<td>width, height</td>
<td><code>win_resize_win()</code></td>
</tr>
<tr>
<td>State</td>
<td>Inactive</td>
<td><code>win_set_state()</code></td>
</tr>
<tr>
<td>Cursor</td>
<td>0</td>
<td><code>win_set_cursor()</code></td>
</tr>
<tr>
<td>Colormap</td>
<td>NULL</td>
<td><code>win_set_cmap()</code></td>
</tr>
<tr>
<td>Text context</td>
<td>NULL</td>
<td><code>win_set_txt_context()</code></td>
</tr>
</tbody>
</table>
The following table shows how `placement` specifies the position of the window relative to its siblings. The `Parameter` column shows the types for additional parameters (represented by “...” in the Syntax section).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drawing context</td>
<td><code>NULL</code></td>
<td><code>win_set_drw_context()</code></td>
</tr>
<tr>
<td>Drawing area</td>
<td><code>x=0, y=0, w=GFX_DIMEN_MAX, h=GFX_DIMEN_MAX</code></td>
<td><code>win_set_drw_area()</code></td>
</tr>
<tr>
<td>Ink method</td>
<td><code>WIN_INK_OFF</code></td>
<td><code>win_set_ink_method()</code></td>
</tr>
<tr>
<td>Pixel value for ink</td>
<td><code>0</code></td>
<td><code>win_set_ink_pix()</code></td>
</tr>
<tr>
<td>Message mask</td>
<td><code>mask</code></td>
<td><code>win_set_msg_mask()</code></td>
</tr>
<tr>
<td>Callback</td>
<td><code>NULL</code></td>
<td><code>win_set_callback()</code></td>
</tr>
<tr>
<td>User data</td>
<td><code>NULL</code></td>
<td><code>win_set_callback()</code></td>
</tr>
</tbody>
</table>

Table 12-1 win_create_win() Parameter Default Values
The position of the window relative to its parent is given by \( x \) and \( y \). The size of the window is specified by \( \text{width} \) and \( \text{height} \). The initial message mask for the window is set to \( \text{mask} \).

The window ID is returned in \( \text{ret_win} \). A pointer to this variable should be passed to \( \text{win_create_win()} \). Use \( \text{win_destroy_win()} \) to destroy this window when it is no longer needed. Use \( \text{win_get_win_status()} \) to get the current status of a window.

After the window is created, a \( \text{WIN_MSG_CREATE} \) message is sent to the process that owns \( \text{parent_win} \).

If successful, this function returns \( \text{SUCCESS} \).

### Attributes

**Operating System:** OS-9 and OS-9 for 68K  
**State:** User  
**Threads:** Safe

### Parameters

- \(*\text{ret_win}\)  
  - Window ID.  
- \(\text{parent_win}\)  
  - Root window or window owned by this process.

---

**Table 12-2 Use of Placement in win_create_win()**

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Parameter</th>
<th>New Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN_FRONT</td>
<td>None</td>
<td>In front of all siblings</td>
</tr>
<tr>
<td>WIN_BACK</td>
<td>None</td>
<td>In back of all siblings</td>
</tr>
<tr>
<td>WIN_FRONT_OF</td>
<td>WIN_ID ref_win</td>
<td>In front of sibling ref_win</td>
</tr>
<tr>
<td>WIN_BACK_OF</td>
<td>WIN_ID ref_win</td>
<td>In back of sibling ref_win</td>
</tr>
</tbody>
</table>

The position of the window relative to its parent is given by \( x \) and \( y \). The size of the window is specified by \( \text{width} \) and \( \text{height} \). The initial message mask for the window is set to \( \text{mask} \).
Windowing Functions

X, Y

Position of window relative to parent_win.

width, height

Size of window in pixels.

mask

Initial message mask.

placement

Placement of window relative to other sibling windows.

... Optional additional parameters for placement.

Fatal Errors

010:001 EOS_MAUI_BADACK

Bad acknowledgment from the maui_win process.

010:008 EOS_MAUI_BADID

The ID specified by parent_win is not valid or the ID specified by ref_win (see placement) is not valid.

010:016 EOS_MAUI_BADVALUE

The value used for placement is not valid.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with win_init().

Indirect Errors

msg_read()

msg_write()

MAUI Win Errors

010:006 EOS_MAUI_BADDIMEN

The width or height is zero.

010:044 EOS_MAUI_NOTFOUND

The reference window ref_win is not a sibling (child of parent_win).

010:060 EOS_MAUI_NOTOWNER

This process does not own parent_win, and parent_win is not the root window.
See Also

win_destroy_win()
win_get_win_status()
win_move_win()
win_reparent_win()
win_resize_win()
win_restack_win()
win_set_callback()
win_set_cursor()
win_set_drw_context()
win_set_drw_area()
win_set_ink_method()
win_set_ink_pix()
win_set_msg_mask()
win_set_state()
win_set_txt_context()
GFX_DIMEN
GFX_POS
MSG_WIN_CREATE
WIN_ID
WIN_PLACEMENT
Windowing Functions

win_destroy_cmap()

Destroy a Colormap

Syntax

```c
error_code
win_destroy_cmap(WIN_CMAP_ID cmap)
```

Description

`win_destroy_cmap()` destroys the colormap `cmap`. The owner of the process that called `win_create_cmap()` is the only owner that can destroy a colormap.

If successful, this function returns `SUCCESS`.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

cmap

Window colormap ID.

Non-Fatal Errors

- **010:001 EOS_MAUI_BADACK**
  - Bad acknowledgment from the maui_win process.

- **010:008 EOS_MAUI_BADID**
  - The ID specified by cmap is not valid.

- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `win_init()`.

MAUI Win Errors

- **010:060 EOS_MAUI_NOTOWNER**
  - This process is not the owner of cmap.
Windowing Functions

Indirect Errors
msg_read()
msg_write()

See Also
win_create_cmap()
WIN_CMAP_ID
**win_destroy_cursor()**

Destroy a Cursor

**Syntax**

```c
error_code
win_destroy_cursor(WIN_DEV_ID windev,
                   u_int32 cursor_id)
```

**Description**

`win_destroy_cursor()` destroys the cursor identified as `cursor_id`. The owner (the process that called `win_create_cursor()`) is the only process that may destroy the cursor.

Any window that is currently using the cursor is modified so that the cursor is no longer associated with it. The cursor becomes 0, and those windows inherit the cursor of the parent window.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `windev` Windowing device ID.
- `cursor_id` Cursor ID.

**Non-Fatal Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:001</td>
<td>EOS_MAUI_BADACK Bad acknowledgment from the mauli_win process.</td>
</tr>
<tr>
<td>010:008</td>
<td>EOS_MAUI_BADID The ID specified by <code>windev</code> is not valid.</td>
</tr>
</tbody>
</table>
| 010:036 | EOS_MAUI_NOINIT This API has not been initialized with `win_init()`.

---

MAUI Programming Reference 575
12 Windowing Functions

MAUI Win Errors

010:044  EOS_MAUI_NOTFOUND

No cursor with the specified cursor_id was found.

010:060  EOS_MAUI_NOTOWNER

This process is not the owner of cmap.

Indirect Errors

msg_read()
msg_write()

See Also

win_create_cursor()
WIN_DEV_ID
win_destroy_dev()

Destroy a Windowing Device

Syntax

```c
error code
win_destroy_dev(WIN_DEV_ID windev)
```

Description

**win_destroy_dev()** destroys the windowing device `windev` previously created with `win_create_dev()`. Do not use `win_destroy_dev()` to destroy a device you opened with `win_open_dev()`; use `win_close_dev()` instead.

All windows, colormaps, and cursors created by any process using this device are destroyed and all input devices are closed. No further messages are written to any application mailbox. Messages already in a mailbox are unaffected.

Any processes that currently have the device open behave as if they each individually called `win_close_dev()` and the device becomes inaccessible.

If successful, this function returns **SUCCESS**.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

Parameters

- **windev** Windowing device ID.

Non-Fatal Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:001</td>
<td>EOS_MAIUI_BADACK Bad acknowledgment from the mauli_win process.</td>
</tr>
<tr>
<td>010:008</td>
<td>EOS_MAIUI_BADID The ID specified by windev is not valid.</td>
</tr>
</tbody>
</table>
010:036 EOS_MAUI_NOINIT

This API has not been initialized with \texttt{win\_init()}. 

010:060 EOS_MAUI_NOTOWNER

This process did not create \texttt{windev} with \texttt{win\_create\_dev}.

\textbf{Non-Fatal Errors}

- \texttt{gfx\_close\_dev()}
- \texttt{mem\_free()}
- \texttt{msg\_close\_mbox()}
- \texttt{msg\_read()}
- \texttt{msg\_write()}
- \texttt{\_os\_sema\_term()}

See \textit{Ultra C Library Reference}.

\textbf{See Also}

- \texttt{win\_create\_dev()}
- \texttt{win\_close\_dev()}
- \texttt{win\_open\_dev()}
- \texttt{WIN\_DEV\_ID}
**win_destroy_win()**

**Destroy a Window**

### Syntax

```c
error_code
win_destroy_win(WIN_ID win)
```

### Description

`win_destroy_win()` destroys the specified window `win` and all its descendent even if the descendents are owned by other processes.

If one or more descendent windows have a region locked with `win_lock_region()` for drawing, the destruction of the window is postponed until `win_unlock_region()` is called.

If `win` currently has the keyboard focus, the focus becomes unassigned and keyboard input is discarded until the focus is assigned to a window using `win_set_focus()`.

If the pointer is currently in the window being destroyed, then a `WIN_MSG_BORDER_ENTER` message is sent to the window receiving the pointer.

If the destruction of this window causes other window areas to be exposed, a `WIN_MSG_EXPOSE` message is sent for each area exposed.

After the window is destroyed, a `WIN_MSG_DESTROY` message is sent to the process that owns `win` and the process that owns the parent of `win`.

If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `win` Window ID.
Non-Fatal Errors

010:001 EOS_MAUI_BADACK
   Bad acknowledgment from the maui_win process.

010:008 EOS_MAUI_BADID
   The ID specified by win is not valid.

010:036 EOS_MAUI_NOINIT
   This API has not been initialized with win_init().

MAUI Win Errors

010:059 EOS_MAUI_NOTALLOWED
   The root window cannot be destroyed.

010:060 EOS_MAUI_NOTOWNER
   This process is not the owner of win.

Indirect Errors

msg_read()
msg_write()

See Also

win_create_win()
win_set_focus()
MSG_WIN_BORDER
MSG_WIN_DESTROY
MSG_WIN_EXPOSE
WIN_ID
Windowing Functions

**win_erase_ink()**

Erase Ink from a Window

**Syntax**

error_code

win_erase_ink(WIN_ID win)

**Description**

`win_erase_ink()` erases the ink from the window specified by `win`. The method used to erase the ink is determined by the current ink method. See `WIN_INK_METHOD` for more information. If necessary, the window is restored by sending a `WIN_MSG_EXPOSE` message to repaint the entire contents of the window. If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

`win` Window ID.

**Non-Fatal Errors**

010:001 EOS_MAUI_BADACK Bad acknowledgment from the maui_win process.
010:008 EOS_MAUI_BADID The ID specified by `win` is not valid.
010:036 EOS_MAUI_NOINIT API not initialized with `win_init()`.

**MAUI Win Errors**

010:060 EOS_MAUI_NOTOWNER This process does not own `win`.
12 Windowing Functions

Indirect Errors
msg_read()
msg_write()

See Also
win_set_ink_method()
win_set_ink_pix()
MSG_WIN_EXPOSE
WIN_ID
WIN_INK_METHOD
**Syntax**

```c
error_code
win_free_cmap_cells(WIN_CMAP_ID cmap,
                    GFX_PIXEL start_pix,
                    u_int16 num_colors)
```

**Description**

`win_free_cmap_cells()` frees color cells previously allocated with `win_alloc_cmap_cell()` or `win_alloc_cmap_cells()`, and decrements the link count of cells linked by `win_alloc_cmap_color()` or `win_alloc_cmap_colors()`. `num_colors` cells are freed starting at the cell `start_pix`.

Any range of currently owned cells can be freed. Cells do not have to be freed using the original range, or all at once.

Always individually free each cell allocated by `win_alloc_cmap_color()` or `win_alloc_cmap_colors()`.

These cells do not have owners, only link counts. It is possible, but not recommended, for a process to free a color allocated by a different process. Free cells individually to avoid inadvertently freeing cells of other processes.

If you originally allocated the cells as a non-contiguous group, using `win_alloc_cmap_cells()` you should free each cell individually using `win_free_cmap_cells()` to avoid attempting to free cells you do not own. Cells allocated by `win_alloc_cmap_cell()` and `win_alloc_cmap_cells()` may only be freed by the process that allocated them.

If this process is not the owner (allocated by another process using `win_alloc_cmap_cell()` or `win_alloc_cmap_cells()`) of all the cells in the specified range, `EOS_MAUI_NOTOWNER` is returned. If a cell in the specified range is not currently allocated, `EOS_MAUI_NOTFOUND`.
is returned. If both errors occur, only one error code is returned. Neither error stops the execution of this function, all eligible cells will be either freed or their link count decremented.

If successful, this function returns SUCCESS.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**

cmap     Colormap ID.
start_pix Index number of starting pixel.
um_colors Number of cells to free.

**Non-Fatal Errors**

010:001 EOS_MAUI_BADACK     Bad acknowledgment from the maui_win process.
010:008 EOS_MAUI_BADID      The ID specified by cmap is not valid.
010:016 EOS_MAUI_BADVALUE   The value for num_colors is 0.
010:036 EOS_MAUI_NOINIT     This API has not been initialized with win_init().

**MAUI Win Errors**

010:044 EOS_MAUI_NOTFOUND   One or more of the color cells you are trying to free is not allocated (it is free) or is out of range.
010:060 EOS_MAUI_NOTOWNER  This process does not own one or more of the color cells you are attempting to free.
Windowing Functions

Indirect Errors
msg_read()
msg_write()

See Also
win_alloc_cmap_cells()
GFX_PIXEL
WIN_CMAP_ID
**win_get_cells_params()**

Get Info on the Color Cells

### Syntax

```c
error_code
win_get_cells_params(WIN_CELL_PARAMS ret_params[],
                      WIN_CMAP_ID cmap,
                      u_int16 num_cells,
                      GFX_PIXEL pixels[])
```

### Description

`win_get_cells_params()` returns detailed information about `num_cells` of color cells, specified by `pixels` array, for colormap `cmap`. The information is placed into `ret_params` array.

The caller must ensure that `ret_params` points to storage large enough to hold `num_cells WIN_CELL_PARAMS` values.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `ret_params`: Array of `WIN_CELL_PARAMS` where the function returns the information requested.
- `cmap`: Colormap ID.
- `num_cells`: The size of the cell range.
- `pixels`: The array of cell indices.
**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**
  The ID specified by `cmap` is not valid.

- **010:016 EOS_MAUI_BADVALUE**
  `num_cells` is equal to 0 or pixel value is out of the cell range.

- **010:036 EOS_MAUI_NOINIT**
  This API has not been initialized with `win_init()`.

**See Also**
- `win_get_cmap_cells()`
- `win_get_cmap_params()`
- `WIN_CELL_PARAMS`
- `WIN_CMAP_ID`
- `GFX_PIXEL`
### Syntax

```c
error_code win_get_cmap_cells(GFX_COLOR_VALUE ret_colors[],
                                 WIN_CMAP_ID cmap,
                                 u_int_16 num_colors,
                                 GFX_PIXEL pixels[])
```

### Description

`win_get_cmap_cells()` returns the colors currently assigned to the specified group of cells in the colormap `cmap`.

`num_colors` is the total number of cells to read. `pixels` is an array of `num_colors` cell numbers.

For each cell specified by an entry in `pixels`, the corresponding entry in `ret_colors` is set with the color value from that cell. The color values are returned in `ret_colors`. A pointer to this variable should be passed to `win_get_cmap_cells()`. The caller must ensure `ret_colors` points to storage large enough to hold `num_colors` color values.

If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `ret_colors[]`: Array of color values returned.
- `cmap`: Colormap ID.
- `num_colors`: Total number of cells to read.
- `pixels[]`: Array of cell numbers.
Non-Fatal Errors

010:001 EOS_MAUI_BADACK
Bad acknowledgment from the maui_win process.

010:008 EOS_MAUI_BADID
The ID specified by cmap is not valid.

010:016 EOS_MAUI_BADVALUE
The value for num_colors is 0.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init().

MAUI Win Errors

010:044 EOS_MAUI_NOTFOUND
One or more of the color cells you are trying to read is not allocated (it is free) or is out of range.

Indirect Errors

msg_read()
msg_write()

See Also

win_create_cmap()
WIN_CMAP_ID
GFX_COLOR_VALUE
GFX_PIXEL
12 Windowing Functions

**win_get_cmap_free()**

Get Free Space in a Colormap

**Syntax**

```c
error_code
win_get_cmap_free(WIN_CMAP_ID cmap,
                 u_int16 *ret_free_cells,
                 u_int16 *ret_largest_block,
                 u_int16 *ret_num_blocks)
```

**Description**

This function returns information about the amount of free (un-allocated) cells in the colormap specified by cmap.

- `ret_free_cells` is set to the total number of cells currently free in the colormap. `ret_largest_block` returns the size of the largest contiguous block. `ret_num_blocks` returns the number of blocks in the colormap containing free space. A block is one or more contiguous cells.

For each return value, a pointer should be passed to `win_get_cmap_free()`. If the pointer is NULL, the corresponding return value is not set.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `cmap` Colormap ID.
- `*ret_free_cells` Pointer to total number of free cells.
- `*ret_largest_block` Pointer to size of largest contiguous block.
*ret_num_blocks*  
Pointer to number of blocks containing free space.

**Non-Fatal Errors**

010:001 EOS_MAUI_BADACK  
Bad acknowledgment from the mauli_win process.

010:008 EOS_MAUI_BADID  
The ID specified by cmap is not valid.

010:036 EOS_MAUI_NOINIT  
This API has not been initialized with win_init().

**Indirect Errors**

msg_read()  
msg_write()

**See Also**

win_create_cmap()  
WIN_CMAP_ID
**win_get_cmap_params()**

Get Info on the Colormap

## Syntax

```c
error_code
win_get_cmap_params (WIN_CMAP_PARAMS *ret_param,
                     WIN_CMAP_ID cmap)
```

## Description

`win_get_cmap_params()` returns detailed information about colormap `cmap`. The information is placed into `ret_param`. The caller must ensure that `ret_param` points to storage large enough to hold `WIN_CMAP_PARAMS` structure.

## Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

## Parameters

- `ret_param`: Pointer to `WIN_CMAP_PARAMS` object where the function returns the information requested.
- `cmap`: Colormap ID.

## Non-Fatal Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:008 EOS_MAUI_BADID</td>
<td>The ID specified by <code>cmap</code> is not valid.</td>
</tr>
</tbody>
</table>
| 010:036 EOS_MAUI_NOINIT | This API has not been initialized with `win_init()`.


Windowing Functions

**See Also**

- `win_get_cmap_cells()`
- `win_get_cells_params()`
- `WIN_CMAP_PARAMS`
- `WIN_CMAP_ID`
### win_get_dev_status()

**Get Windowing Device Status**

#### Syntax

```
error_code
win_get_dev_status(WIN_DEV_STATUS *ret_dev_status,
                   WIN_DEV_ID windev)
```

#### Description

`win_get_dev_status()` returns the current status of the specified windowing device `windev`.

The device status structure is returned in `ret_dev_status`. A pointer to this variable should be passed to `win_get_dev_status()`. The caller must ensure that `ret_dev_status` points to storage large enough to hold the information.

If successful, this function returns `SUCCESS`.

#### Attributes

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User
- **Threads**: Safe

#### Parameters

- `*ret_dev_status` Pointer to device status.
- `windev` Windowing device ID.

#### Non-Fatal Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:008</td>
<td>EOS_MAUI_BADID The ID specified by <code>windev</code> is not valid.</td>
</tr>
<tr>
<td>010:036</td>
<td>EOS_MAUI_NOINIT This API has not been initialized with <code>win_init()</code></td>
</tr>
</tbody>
</table>
See Also

win_create_dev()
win_open_dev()
WIN_DEV_ID
WIN_DEV_STATUS
**win_get_win_status()**

Get Window Status

**Syntax**

```c
error_code
win_get_win_status(WIN_STATUS *ret_win_status,
                    WIN_ID win)
```

**Description**

`win_get_win_status()` returns the current status of the specified `win`. You do not have to own `win` to get its status. The window status structure is returned in `ret_win_status`. A pointer to this variable should be passed to `win_get_win_status()`. The caller must ensure that `ret_win_status` points to storage large enough to hold the information.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `*ret_win_status` Pointer to window status.
- `win` Window ID.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**
  
  Bad acknowledgment from the `maui_win` process.

- **010:008 EOS_MAUI_BADID**

  The ID specified by `win` is not valid.

- **010:036 EOS_MAUI_NOINIT**

  This API has not been initialized with `win_init()`.
Windowing Functions

**Indirect Errors**

msg_read()
msg_write()

**See Also**

win_create_win()
win_move_win()
win_reparent_win()
win_resize_win()
win_restack_win()
win_set_callback()
win_set_cursor()
win_set_drw_area()
win_set_drw_context()
win_set_ink_method()
win_set_ink_pix()
win_set_msg_mask()
win_set_state()
win_set_txt_context()
WIN_ID
WIN_STATUS
12 Windowing Functions

win_grab_ptr()

Explicitly capture the cursor

Syntax

```c
error_code
win_grab_ptr(WIN *win)
```

Description

\texttt{win\_grab\_ptr()} is used to perform an explicit capture (grab) of the cursor. The focus remains on the window even when the pointer is moved out of it.

If the window is visible and there is no explicit grab in effect, \texttt{maui\_win} checks whether the window already has a pointer focus. If not, the focus is moved to this window and the colormap and the cursor are reinstalled.

If successful, this function returns \texttt{SUCCESS}.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

\texttt{win} Window ID.

Non-Fatal Errors

- **010:008 EOS_MAUI_BADID**  
The ID specified by \texttt{win} is not valid.

- **010:017 EOS_MAUI_BUSY**  
Cursor has already been explicitly grabbed.

- **010:036 EOS_MAUI_NOINIT**  
This API has not been initialized with \texttt{win\_init()}.
Windowing Functions

010:060 EOS_MAUI_NOTOWNER
This process does not own win.

010:068 EOS_MAUI_NOTVISIBLE
The window is not visible.

Indirect Errors
None

See Also
win_ungrab_ptr()
WIN_ID
**win_init()**

Initialize the Windowing API

---

**Syntax**

```c
error_code
win_init(void)
```

**Description**

`win_init()` initializes the Windowing API. This function must be called prior to a call to any other window function unless otherwise noted by that function. This API depends on the Shaded Memory, Graphics, Bit-BLT, and Messaging APIs. Therefore, `mem_init()`, `gfx_init()`, `blt_init`, and `msg_init()` are called by this function.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

None

**Fatal Errors**

- `blt_init()`
- `gfx_init()`
- `mem_init()`
- `mem_malloc()`
- `msg_create_mbox()`
- `msg_init()`
- `_os_id()`

See *Ultra C Library Reference*.

**See Also**

- `win_term()`

---
win_lock_region()
Lock a Region of a Window

Syntax
error_code
win_lock_region(WIN_ID win,
    GFX_POS x,
    GFX_POS y,
    GFX_DIMEN width,
    GFX_DIMEN height)

Description
win_lock_region() locks the specified region of the window win so that drawing can be done safely. After all drawing is done,
win_unlock_region() should be called to unlock the region.

Note
Do not lock a region for an extended period of time. This will interfere with other processes (such as clipping region calculations) and erase the cursor. (To restore the cursor after the draw, use the
win_unlock_region() function.) Moreover, do not call any other Windowing API functions until win_unlock_region() is called.

The region to be locked is specified by upper-left corner x and y and by the specified width and height. No drawing should be performed outside of the area because it may interfere with other windows or the graphics cursor.
If the graphics cursor is visible within the specified region, it is turned off by win_lock_region().
If successful, this function returns SUCCESS.
12 Windowing Functions

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Windowing Functions

**Parameters**

- `win`  
  Window ID.
- `x, y`  
  X and Y coordinates of the top-left corner of the locked region with respect to `win`.
- `width, height`  
  Width and height in pixels of the locked region.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**  
  The ID specified by `win` is not valid.
- **010:036 EOS_MAUI_NOINIT**  
  This API has not been initialized with `win_init()`.
- **010:060 EOS_MAUI_NOTOWNER**  
  This process does not own `win`.

**Indirect Errors**

- `drw_set_context_clip()`  
- `drw_set_context_draw()`  
- `drw_set_context_origin()`  
- `mem_realloc()`  
- `txt_set_context_clip()`  
- `_os_sema_p()`  
- `_os_sema_v()`  
- `txt_set_context_draw()`  
- `txt_set_context_origin()`  

  See [Ultra C Library Reference](#).

**See Also**

- `win_unlock_region()`  
- `GFX_DIMEN`  
- `GFX_POS`  
- `WIN_ID`
**win_move_win()**

**Syntax**

```c
error_code
win_move_win(WIN_ID win, GFX_POS x, GFX_POS y)
```

**Description**

`win_move_win()` moves the upper-left corner of window `win` to the position specified by `x` and `y`. Either `win` or the parent of `win` must be owned by this process.

The new position is specified relative to the parent window that contains `win`. The new position must place the window wholly within the parent window.

If the process that owns the parent of `win` has asked to be notified (see `win_set_msg_mask()`) about move operations on `win`, the window is not moved. Instead, a `WIN_MSG_MOVE_REQ` message is sent to the parent and this function returns. The process that owns the parent window then decides what to do with the request (honor the request or ignore it.)

If the parent did not ask for notification, then the window is moved and a `WIN_MSG_MOVE` message is sent.

If the movement of the window causes the pointer to move to a different window, a `WIN_MSG_BORDER_LEAVE` message is sent to `win` and a `WIN_MSG_BORDER_ENTER` message is sent to the window receiving the pointer.

If the movement of the window causes other window areas to be exposed, a `WIN_MSG_EXPOSE` message is sent for each area exposed.

If successful, this function returns `SUCCESS`.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe
Windowing Functions

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>win</code></td>
<td>Window ID.</td>
</tr>
<tr>
<td><code>x, y</code></td>
<td>Upper-left corner of new window position.</td>
</tr>
</tbody>
</table>

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from the `maui_win` process.
- **010:008 EOS_MAUI_BADID**: The ID specified by `win` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `win_init()`.

**MAUI Win Errors**

- **010:060 EOS_MAUI_NOTOWNER**: This process does not own either `win` or the parent of `win`.
- **010:059 EOS_MAUI_NOTALLOWED**: The root window cannot be moved.

**Indirect Errors**

- `msg_read()`
- `msg_write()`

**See Also**

- `win_create_win()`
- `win_get_win_status()`
- `win_resize_win()`
- `win_set_msg_mask()`
- `GFX_POS`
- `MSG_WIN_BORDER`
- `MSG_WIN_EXPOSE`
- `MSG_WIN_MOVE`
- `WIN_ID`
**win_open_dev()**

Open a Windowing Device

**Syntax**

```c
error_code
win_open_dev(WIN_DEV_ID *ret_windev,
             GFX_DEV_ID *ret_gfxdev,
             WIN_ID *ret_root,
             const char *device_name,
             MSG_MBOX_ID mbox)
```

**Description**

`win_open_dev()` opens the windowing device named `device_name`. The process that calls `win_open_dev()` does not become the owner of the windowing device or the root window. The owner is the process that calls `win_create_dev()`.

The windowing device named `device_name` must already exist. It is created by the owner of the windowing device when that process calls `win_open_dev()`.

The windowing device ID is returned in `ret_windev`. A pointer to this variable should be passed to `win_open_dev()`. Use `win_close_dev()` when this device is no longer needed.

The graphics device ID is returned in `ret_gfxdev`. A pointer to this variable should be passed to `win_open_dev()`. Do not close this device using `gfx_close_dev()`.

The root window for the windowing device is returned in `ret_root`. A pointer to this variable should be passed to `win_open_dev()`. Do not attempt to destroy this window.

`mbox` specifies the mailbox that should receive windowing messages. The mailbox must be opened using `msg_create_mbox()` or `msg_open_mbox` prior to calling `win_open_dev()`.

If successful, this function returns `SUCCESS`. 
## Windowing Functions

### Attributes
- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters
- `*ret_windev` - Pointer to windowing device ID.
- `*ret_gfxdev` - Pointer to graphics device ID.
- `*ret_root` - Pointer to root ID.
- `*device_name` - Pointer to windowing device.
- `mbox` - Message mailbox.

### Fatal Errors
- **010:001 EOS_MAUI_BADACK** - Bad acknowledgment from the `maui_win` process.
- **010:004 EOS_MAUI_BADCOMPATLEVEL** - Bad compatibility level reported by the `maui_win` process.
- **010:036 EOS_MAUI_NOINIT** - This API has not been initialized with `win_init()`.
- **010:050 EOS_MAUI_TOOLONG** - The device name `device_name` is too long. The maximum length is `WIN_MAX_DEV_NAME`.
- **010:063 EOS_MAUI_DEVNOTFOUND** - The device named `device_name` could not be found.

### Indirect Errors
- `gfx_clone_dev()`
- `mem_malloc()`
- `msg_get_mbox_status()`
- `msg_open_mbox()`
Windowing Functions

msg_read()
msg_write()
_os_sema_init()

See Also
msg_create_mbox()
msg_open_dev()
win_close_dev()
win_create_dev()
win_destroy_dev()
MSG_MBOX_ID
WIN_DEV_ID
WIN_ID
WIN_MAX_DEV_NAME

See Ultra C Library Reference
WinOpenInpdev()  
Open an Input Device

Syntax

```c
error_code win_open_inpdev(INP_DEV_ID *ret_inpdev, 
WIN_DEV_ID windev, 
const char *device_name)
```

Description

`win_open_inpdev()` opens the input device named `device_name` and associates it with the windowing device `windev`.

The input device ID is returned in `ret_inpdev`. A pointer to this variable should be passed to `win_open_inpdev()`. Use `win_close_inpdev()` when this device is no longer needed.

The input device ID is not directly usable by functions in the Input API. Do not try to use `ret_inpdev` in any Input API function.

By default, the keyboard focus is not assigned to any window. It must be assigned to a window using `win_set_focus()` before any key messages from any input device are generated.

If successful, this function returns `SUCCESS`. Otherwise, the returned value is an error code. Error codes unique to this API are defined below.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `*ret_inpdev`: Pointer to input device ID.
- `windev`: Windowing device ID.
- `*device_name`: Input device name.
12 Windowing Functions

**Fatal Errors**

- **010:001 EOS_MAUİ.BADACK**
  - Bad acknowledgment from the maui_win process.
- **010:008 EOS_MAUİ.BADID**
  - The ID specified by windev is not valid.
- **010:036 EOS_MAUİ.NOINIT**
  - This API has not been initialized with win_init().

**MAUI Win Errors**

- **010:060 EOS_MAUİ.NOTOWNER**
  - This process is not the owner of cmap.

**Indirect Errors**

- `inp_open_dev()`
- `msg_read()`
- `msg_write()`

**See Also**

- `win_close_inpdev()`
- `win_set_focus()`
- `INP_DEV_ID`
- `WIN_DEV_ID`
**Syntax**

```c
error_code win_reparent_win(WIN_ID win,
WIN_ID new_parent_win,
GFX_POS new_x,
GFX_POS new_y,
WIN_PLACEMENT placement,
...)
```

**Description**

`win_reparent_win()` removes the window `win` from its current parent and makes it a child of `new_parent_win`. `win` and/or the current parent of `win` must be owned by this process.

The position in the new parent is specified by `new_x` and `new_y`. The size of `win` remains unchanged. The window must fit wholly within the new parent window.

The following table shows how `placement` may be used to specify the position of the window relative to its new siblings. The Parameter column shows the types of the additional parameters (represented by “...” in the Syntax section).

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Parameter</th>
<th>New Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN_FRONT</td>
<td>None</td>
<td>In front of all siblings</td>
</tr>
<tr>
<td>WIN_BACK</td>
<td>None</td>
<td>In back of all siblings</td>
</tr>
</tbody>
</table>
After the window is re-parented, a `WIN_MSG_REPARENT` message is sent to the process that owns `win`. If successful, this function returns `SUCCESS`.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

**Parameters**

- `win`: Window ID.
- `new_parent_win`: Window ID of new parent window.
- `new_x, new_y`: New x and y coordinates.
- `placement`: Placement of window with respect to other windows.
- `...`: Optional additional parameters for `placement`.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from the `maui_win` process.
- **010:008 EOS_MAUI_BADID**: The ID specified by `win` or `parent_win` is not valid or the ID specified by `ref_win` (see `placement`) is not valid.

---

**Table 12-3 Use of Placement in win_reparent_win()**

<table>
<thead>
<tr>
<th>Placement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN_FRONT_OF WIN_ID ref_win</td>
<td>In front of sibling ref_win</td>
</tr>
<tr>
<td>WIN_BACK_OF WIN_ID ref_win</td>
<td>In back of sibling ref_win</td>
</tr>
</tbody>
</table>

---
Windowing Functions

010:016 EOS_MAUI_BADVALUE

The value used for placement is invalid.

010:036 EOS_MAUI_NOINIT

This API has not been initialized with `win_init()`.

Indirect Errors
msg_read()
msg_write()

MAUI Win Errors

010:044 EOS_MAUI_NOTFOUND

The reference window `ref_win` is not a child of `new_parent_win`.

010:059 EOS_MAUI_NOTALLOWED

Root window can’t be reparented.

010:060 EOS_MAUI_NOTOWNER

This process does not own `new_parent_win` and the current parent of `win`.

See Also

`win_create_win()`
`win_restack_win()`
`MSG_WIN_REPARENT`
`WIN_ID`
`WIN_PLACEMENT`
**Syntax**

```
error_code
win_resize_win(WIN_ID win, GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`win_resize_win()` changes the size of window `win` to the specified width and height. Either `win` or the parent of `win` must be owned by this process.

The new size must allow the window to be wholly contained within its parent.

If the process that owns the parent of `win` has asked to be notified (see `win_set_msg_mask()`) about resize operations on `win`, the window is not resized. Instead, a `WIN_MSG_RESIZE_REQ` message is sent to the parent and this function returns. The process that owns the parent window then decides what to do with the request (honor the request or ignore it).

If the parent did not ask for notification, then the window is resized and a `WIN_MSG_RESIZE` message is sent.

If the resize of the window causes the pointer to move to a different window, a `WIN_MSG_BORDER_LEAVE` message is sent to `win` and a `WIN_MSG_BORDER_ENTER` message is sent to the window receiving the pointer.

If the resize of the window causes other window areas to be exposed, a `WIN_MSG_EXPOSE` message is sent for each area exposed.

If successful, this function returns `SUCCESS`. Otherwise, the returned value is an error code. Error codes unique to this API are defined below.
Windowing Functions

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
win Window ID.
width, height Width and height of window in pixels.

Non-Fatal Errors
010:001 EOS_MAUI_BADACK Bad acknowledgment from maui_win.
010:008 EOS_MAUI_BADID The ID specified by win is not valid.
010:036 EOS_MAUI_NOINIT This API has not been initialized with win_init.

MAUI Win Errors
010:006 EOS_MAUI_BADDIMEN The width or height is zero.
010:059 EOS_MAUI_NOTALLOWED The root window cannot be resized.
010:060 EOS_MAUI_NOTOWNER This process does not own either win or the parent of win.

Indirect Errors
msg_read()
msg_write()
See Also

win_create_win()
win_get_win_status()
win_move_win()
win_set_msg_mask()
GFX_DIMEN
MSG_WIN_BORDER
MSG_WIN_EXPOSE
MSG_WIN_RESIZE
WIN_ID
Win_restack_dev()

Restack a Device

Syntax

```c
error_code win_restack_dev (WIN_DEV *windev,
                          WIN_DEV_PLACEMENT placement)
```

Description

`win_restack_dev()` changes the placement of the windowing device with ID defined by `windev` according to `placement` within the current stack of windowing devices.

The following table shows how `placement` may be used to specify the new position.

<table>
<thead>
<tr>
<th>Value of placement</th>
<th>New position</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN_DEV_FRONT</td>
<td>In front of all devices</td>
</tr>
<tr>
<td>WIN_DEV_BACK</td>
<td>In back of all devices</td>
</tr>
</tbody>
</table>

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `windev`: ID of the windowing device.
- `placement`: Specifies the new position relative to all devices or a reference device.
Non-Fatal Errors

010:008  EOS_MAUI_BADID
The ID specified by windev is not valid.

010:016  EOS_MAUI_BADVALUE
The value used for placement is not valid.

010:036  EOS_MAUI_NOINIT
This API has not been initialized with gfx_init().

010:060  EOS_MAUI_NOTOWNER
This is not the process that opened the device windev.

Indirect Errors

gfx_restack_dev()
inp_restack_dev()

Driver Errors
None

See Also

GFX_DEV_ID
GFX_DEV_PLACEMENT
**Syntax**

```c
error_code
win_restack_win(WIN_ID win,
                 WIN_PLACEMENT placement, ...)
```

**Description**

`win_restack_win()` changes the placement of the window `win` within the current stack of sibling windows. Either `win` or the parent of `win` must be owned by this process.

The following table shows how `placement` may be used to specify the new position. The Parameter column shows the types of the additional parameters (represented by “...” in the Syntax section).

<table>
<thead>
<tr>
<th>Value of Placement</th>
<th>Parameter</th>
<th>New Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN_FRONT</td>
<td>None</td>
<td>In front of all windows</td>
</tr>
<tr>
<td>WIN_BACK</td>
<td>None</td>
<td>In back of all windows</td>
</tr>
<tr>
<td>WIN_FRONT_OF</td>
<td>WIN_ID ref_win</td>
<td>In front of ref_win</td>
</tr>
<tr>
<td>WIN_BACK_OF</td>
<td>WIN_ID ref_win</td>
<td>In back of ref_win</td>
</tr>
</tbody>
</table>

If the process that owns the parent of `win` has asked to be notified (see `win_set_msg_mask()`) about re-stack operations on `win`, then the window is not re-stacked. Instead, a `WIN_MSG_RESTACK_REQ` message is sent to the parent and this function returns. The process that owns the parent window then decides what to do with the request (honor the request or ignore it).
If the parent did not ask for notification, the window is re-stacked and a \texttt{WIN\_MSG\_RESTACK} message is sent.

If the re-stack operation causes the pointer to leave the window it is currently in and move to a new window, a \texttt{WIN\_MSG\_BORDER\_LEAVE} message is sent to the window losing the pointer and a \texttt{WIN\_MSG\_BORDER\_ENTER} message is sent to the window receiving the pointer.

If the re-stack operation causes other window areas to be exposed, a \texttt{WIN\_MSG\_EXPOSE} message is sent for each area exposed.

If successful, this function returns \texttt{SUCCESS}.

\textbf{Attributes}

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

\textbf{Parameters}

- \texttt{win}: Window ID.
- \texttt{placement}: Position of window.
- \ldots: Optional additional parameters for \texttt{placement}.

\textbf{Non-Fatal Errors}

\begin{itemize}
  \item \texttt{010:001 EOS\_MAUI\_BADACK}: Bad acknowledgment from \texttt{maui\_win}.
  \item \texttt{010:008 EOS\_MAUI\_BADID}: The ID specified by \texttt{win} or the ID specified by \texttt{ref\_win} is not valid.
  \item \texttt{010:016 EOS\_MAUI\_BADVALUE}: The value used for \texttt{placement} is invalid.
  \item \texttt{010:036 EOS\_MAUI\_NOINIT}: API not initialized with \texttt{win\_init}().
\end{itemize}
Windowing Functions

MAUI Win Errors

010:044 EOS_MAUI_NOTFOUND
The reference window ref_win is not a sibling of win.

010:059 EOS_MAUI_NOTALLOWED
Root window cannot be re-stacked.

010:060 EOS_MAUI_NOTOWNER
This process does not own either win or the parent of win.

Indirect Errors

msg_read()
msg_write()

See Also

win_create_win()
win_reparent_win()
win_set_msg_mask()
MSG_WIN_BORDER
MSG_WIN_EXPOSE
MSG_WIN_RESTACK
WIN_ID
WIN_PLACEMENT
**win_set_callback()**

**Set Callback for Queuing Messages**

**Syntax**

```c
error_code
win_set_callback(WIN_ID win, WIN_CALLBACK callback,
                 void *user_data)
```

**Description**

`win_set_callback()` sets the callback used when queuing messages for the window `win`. Either `win` or the parent of `win` must be owned by this process. When messages are written by `maui_win`, `callback` is placed in the `callback` member of the message. When the message is dispatched, `user_data` is passed as a parameter to the callback function.

Calling this function only affects future writes to the mailbox. Messages that are already queued are not affected. The default callback is `NULL`. If you attempt to dispatch a message with no callback using `msg_dispatch()`, you get the error `EOS_MAUI_NOCALLBACK`.

When the application reads messages (using `msg_read()`) generated from this device, they will have the specified `callback` present in the message. When this message is passed to `msg_dispatch()` it calls the application supplied `callback` function.

The `callback` function `callback` is defined by the caller and its prototype should appear as follows:

```c
void callback(const void *msg, void *user_data)
```

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
Windowing Functions

**Parameters**

- **win**: Window ID.
- **callback**: Specifies the callback function.
- **user_data**: Any data needed by callback function.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from `maui_win`.
- **010:008 EOS_MAUI_BADID**: The ID specified by `win` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `win_init`.

**MAUI Win Errors**

- **010:060 EOS_MAUI_NOTOWNER**: This process does not own either `win` or the parent of `win`.

**Indirect Errors**

- `msg_read()`
- `msg_write()`

**See Also**

- `win_create_win()`
- `win_open_dev()`
- `win_get_win_status()`
- `msg_dispatch()`
- `msg_read()`
- `WIN_CALLBACK`
- `WIN_ID`
**Syntax**

```c
error_code win_set_cmap(WIN_ID win, WIN_CMAP_ID cmap)
```

**Description**

`win_set_cmap()` sets the colormap to use for the specified window `win`. If a colormap is not assigned to a window it inherits the colormap of its parent.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `win` Window ID.
- `cmap` Colormap ID for this window.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**  
  Bad acknowledgment from `maui_win`.

- **010:008 EOS_MAUI_BADID**  
  The ID specified by `win` is not valid or the ID specified by `cmap` is not valid.

- **010:036 EOS_MAUI_NOINIT**  
  This API has not been initialized with `win_init`. 
Windowing Functions

**MAUI Win Errors**

010:060 EOS_MAUI_NOTOWNER

This process does not own win.

**Indirect Errors**

msg_read()
msg_write()

**See Also**

win_create_cmap()
WIN_CMAP_ID
WIN_ID
**win_set_cmap_cells()**

Set Colors in Group of Cells

**Syntax**

```c
error_code
win_set_cmap_cells(WIN_CMAP_ID cmap,
                    const GFX_PIXEL pixels[],
                    u_int16 num_colors,
                    GFX_COLOR_VALUE colors[])
```

**Description**

`win_set_cmap_cells()` sets a group of cells in the colormap `cmap` with the colors specified by `colors`. `num_colors` specifies the number of color cells to set.

Each entry in `colors` is used to set a colormap cell identified by the corresponding entry in `pixels`.

Only the owner of a color cell may modify it. You must use `win_alloc_color_cells()` to allocate a group of cells before you use them.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `cmap`  
  Colormap ID for this window.

- `pixels[]`  
  Array of pixels to assign to colormap cells.

- `num_colors`  
  Number of colors in group.

- `colors[]`  
  Array of color values.
**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**
  - Bad acknowledgment from `maui_win`.

- **010:008 EOS_MAUI_BADID**
  - The ID specified by `cmap` is not valid.

- **010:016 EOS_MAUI_BADVALUE**
  - The value for `num_colors` is zero.

- **010:036 EOS_MAUI_NOINIT**
  - This API has not been initialized with `win_init`.

- **010:044 EOS_MAUI_NOTFOUND**
  - One or more of the color cells you are trying to set is not allocated (it is free) or it is out of range.

**MAUI Win Errors**

- **010:060 EOS_MAUI_NOTOWNER**
  - This process is not the owner of one or more of the specified cells in the colormap `cmap`.

**Indirect Errors**

- `msg_read()`
- `msg_write()`

**See Also**

- `win_alloc_cmap_cells()`
- `GFX_COLOR_VALUE`
- `GFX_PIXEL`
- `WIN_CMAP_ID`
win_set_color_match()
Set Color Matching Method

Syntax
```
error_code
win_set_color_match(WIN_CMAP_ID cmap,
                   WIN_CMATCH method)
```

Description
win_set_color_match() sets the method used in color allocation for colormap cmap. method is the comparison function which decides whether to reuse existing colors or to allocate a new cell.

The method that is currently set is used in color allocation until it is set again.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

cmap
Colormap ID.

method
Color matching method; can assume the following values:

- WIN_CMATCH_NONE (always allocate a new cell - never reuse),
- WIN_CMATCH_CLOSEST (never allocate a new cell - always reuse),
- WIN_CMATCH_CLOSE (reuse cell only if there is a close enough match),
WIN_CMATCH_EXACT
(reuse cell only if there is an exact match). Default value is
WIN_CMATCH_EXACT.

Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by cmap is not valid.

010:016 EOS_MAUI_BADVALUE
Invalid method value was specified.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init().

See Also
win_alloc_cmap_color()
win_alloc_cmap_colors()
WIN_CMAP_ID
WIN_CMATCH

Note
WIN_CMATCH_CLOSE is implemented only for RGB and YCBCR color types. All other color types result in WIN_CMATCH_EXACT while using WIN_CMATCH_CLOSE.
win_set_cursor()
Set Cursor for a Window

Syntax

```c
error_code win_set_cursor(WIN_ID win, u_int32 cursor_id)
```

Description

`win_set_cursor()` sets the cursor for the specified window `win`. The cursor is specified by the ID `cursor_id`. If `cursor_id` is 0 then no cursor is assigned to `win`. In this case, `win` inherits the cursor assigned to its parent window.

The cursor must be defined by this process or another process (using `win_create_cursor()`) before this function is used to assign it to a window.

If successful, this function returns `SUCCESS`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `win` Window ID.
- `cursor_id` Cursor ID number.

Non-Fatal Errors

- **010:001 EOS_MAUI_BADACK**
  Bad acknowledgment from `maui_win`.
- **010:008 EOS_MAUI_BADID**
  The ID specified by `win` or `cursor_id` is not valid.
- **010:036 EOS_MAUI_NOINIT**
  This API has not been initialized with `win_init()`.
MAUI Win Errors

010:060 EOS_MAUI_NOTOWNER  This process does not own win.

Indirect Errors

msg_read()
msg_write()

See Also

win_create_cursor()
win_create_dev()
WIN_ID
**win_set_cursor_pos()**

**Set Cursor Position Relative to a Window**

**Syntax**

```c
error_code
win_set_cursor_pos(WIN *win, GFX_POS x, GFX_POS y)
```

**Description**

`win_set_cursor_pos()` sets the cursor position to `(x, y)` relative to the window `win`. `win_set_cursor_pos()` behaves differently depending on whether the application owns the windowing device or it does not.

If the application owns the windowing device, the cursor is set immediately to the new position.

If the application does not own the windowing device, the request to set the cursor to its new position is sent asynchronously to the application that owns the windowing device (e.g. a window manager). The application that owns the windowing device can either honor the request by setting the new cursor position or reject it.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `*win` — Window ID.
- `x` — New x coordinate for the cursor.
- `y` — New y coordinate for the cursor.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID** — The ID specified by `win` is not valid.
This API has not been initialized with `win_init()`.

**Indirect Errors**
- `msg_read()`
- `msg_write()`

**See Also**
- `GFX_POS`
- `WIN`
**win_set_cursor_state()**

Set Cursor Visibility

**Syntax**

```
error_code
win_set_cursor_state(WIN_DEV* windev,
                      BOOLEAN active)
```

**Description**

`win_set_cursor_state()` sets the visibility setting of a cursor. The cursor is either hidden or visible depending on the value of `active`. If `active` is `TRUE`, then the cursor is visible. If `active` is `FALSE`, then the cursor is hidden.

The cursor is specified by `windev`, the pointer to the windowing device.

The cursor must be defined by this process or another process (using `win_create_cursor()`) before this function is used to assign it to a window.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `windev` Pointer to the windowing device.
- `active` State of the cursor (hidden or visible).

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK** Bad acknowledgment from `maui_win`.
- **010:008 EOS_MAUI_BADID** The ID specified by `win` is not valid.
Windowing Functions

010:036  EOS_MAUI_NOINIT

This API has not been initialized with `win_init()`.

MAUI Win Errors

010:060  EOS_MAUI_NOTOWNER

This process does not own `win`.

Indirect Errors

msg_read()
msg_write()

See Also

gfx_create_dmap()
win_create_cursor()
win_create_dev()
win_get_dev_status()
win_get_win_status()
GFX_DMAP
WIN_ID
**Syntax**

```c
error_code win_set_drw_area(WIN_ID win, GFX_POS x, GFX_POS y,
                         GFX_DIMEN width, GFX_DIMEN height)
```

**Description**

`win_set_drw_area()` sets the drawing area for the specified window `win`. All drawing outside this drawing area is clipped (not drawn).

The upper-left corner of the drawing area is specified by `x` and `y`. The `width` and `height` define the size of the drawing area. If either `width` or `height` is 0, all drawing is clipped (no drawable area).

If successful, this function returns `SUCCESS`.

**Note**

Do not use `win_set_drw_area()` between `win_lock_region()` and `win_unlock_region()`. When drawing, you should always lock before the draw. Locking a region erases the cursor; you need the unlock to restore the cursor after the draw. Therefore, complete the setup process before locking the region and draw inside the lock only.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
Windowing Functions

Parameters

```
win        Window ID.
x, y       Upper-left corner of drawing area.
width, height Width and height of drawing area.
```

Non-Fatal Errors

```
010:008 EOS_MAUI_BADID       The ID specified by win is not valid.
010:036 EOS_MAUI_NOINIT      This API has not been initialized with win_init.
```

Indirect Errors

```
drw_set_context_draw()
txt_set_context_draw()
_os_sema_p()
_os_sema_v()       See Ultra C Library Reference.
```

See Also

```
drw_set_context_draw()
txt_set_context_draw()
win_create_win()
win_get_win_status()
GFX_DIMEN
GFX_POS
WIN_ID
```

Ultra C Library Reference
**win_set_drw_context()**

Set Drawing Context

**Syntax**

```c
error_code
win_set_drw_context(WIN_ID win,
                   DRW_CONTEXT_ID context)
```

**Description**

`win_set_drw_context()` sets the drawing context for the specified window `win`. You must assign the drawing context to the window before you use the context object to perform any drawing.

Drawing is performed with the Drawing API, however, the Windowing API maintains the origin, drawing area, and clipping area in the drawing context. Therefore, you must not call `drw_set_context_origin()`, `drw_set_context_draw()`, `drw_set_context_clip()`, or `drw_set_context_dst()`.

The drawing context must be removed from the window before it is deleted. Call `win_set_drw_context()` with `context` set to `NULL` to do so.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `win`: Window ID.
- `context`: Drawing context ID.

**Non-Fatal Errors**

- **010:008 EOS_MAUI_BADID**: The ID specified by `win` is not valid.
010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init.

010:060 EOS_MAUI_NOTOWNER
This process does not own win.

Indirect Errors
- drw_set_context_draw()
- drw_set_context_clip()
- drw_set_context_origin()
- mem_realloc()
- _os_sema_p()
- _os_sema_v() See Ultra C Library Reference.

See Also
- drw_set_context_clip()
- drw_set_context_draw()
- drw_set_context_dst()
- drw_set_context_origin()
- win_create_win()
- win_get_win_status()
- DRW_CONTEXT_ID
- WIN_ID
win_set_error_action()
Set Action to Take in Error Handler

Syntax
error_code
win_set_error_action(MAUI_ERR_LEVEL debug_level,
                      MAUI_ERR_LEVEL passback_level,
                      MAUI_ERR_LEVEL exit_level)

Description

win_set_error_action() sets the action to take in the error
handler when a function in this API detects an error. This function may
be called prior to calling win_init(). Following is the table of error
levels. The least severe error is listed first.

Table 12-6  Error Levels for win_set_error_action()

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_NONE</td>
<td>No error will cause the handler to perform the specified operation.</td>
</tr>
<tr>
<td>MAUI_ERR_NOTICE</td>
<td>Prints a message, but is not severe enough for an error code.</td>
</tr>
<tr>
<td>MAUI_ERR_WARNING</td>
<td>Least severe error code. The operation is completed, but something may be wrong.</td>
</tr>
<tr>
<td>MAUI_ERR_NON_FATAL</td>
<td>The operation did not complete, but a cascade failure is not likely.</td>
</tr>
<tr>
<td>MAUI_ERR_FATAL</td>
<td>The operation did not complete and a cascade failure is likely.</td>
</tr>
<tr>
<td>MAUI_ERR_ANY</td>
<td>Any error.</td>
</tr>
</tbody>
</table>
Windowing Functions

**debug_level** sets the minimum error level that causes the error handler to print a message to standard error. The default debug level is **MAUI_ERR_ANY**.

**passback_level** sets the minimum error level that causes the error handler to return the error. For less severe errors, **SUCCESS** is returned. The default pass-back level is **MAUI_ERR_NON_FATAL**.

**exit_level** sets the minimum error level that causes the error handler to call **exit()**. In this case the program exits with the error code that caused the error handler to be called. The default debug level is **MAUI_ERR_NONE**.

If successful, this function returns **SUCCESS**.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **debug_level**
  - Minimum error level that causes the error handler to print a message to standard error.

- **passback_level**
  - Minimum error level that causes the error handler to return the error.

- **exit_level**
  - Minimum error level that causes the error handler to call **exit()**.

**Table 12-6  Error Levels for win_set_error_action() (continued)**

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAUI_ERR_AS_IS</td>
<td>The status of the error handler is not changed.</td>
</tr>
<tr>
<td>MAUI_ERR_DEFAULT</td>
<td>Restore the level to its default value.</td>
</tr>
</tbody>
</table>
12 Windowing Functions

Non-Fatal Errors
None

See Also
win_init()
Windowing Functions

**win_set_focus()**
Set Window that has Keyboard Focus

**Syntax**

```c
error_code
win_set_focus(WIN_ID win)
```

**Description**

`win_set_focus()` sets the keyboard focus to the window `win`. All keyboard messages are delivered on behalf of this window until the focus is moved to another window.

If `win` is set to `NULL` then no window receives keyboard messages. In this case keyboard input is discarded until the focus is assigned to a window.

If the keyboard focus is currently assigned to a window other than `win`, a `WIN_MSG_FOCUS_OUT` message is sent to the window that is losing focus and a `WIN_MSG_FOCUS_IN` message is sent to `win`.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `win` Window ID.

**Non-Fatal Errors**

- 010:001 EOS_MAUI_BADACK: Bad acknowledgment from `maui_win`.
- 010:008 EOS_MAUI_BADID: The ID specified by `win` is not valid.
- 010:036 EOS_MAUI_NOINIT: This API has not been initialized with `win_init`. 
12 Windowing Functions

MAUI Win Errors

010:060 EOS_MAUI_NOTOWNER

This process does not own win.

Indirect Errors

msg_read()
msg_write()

See Also

win_get_dev_status()
win_get_win_status()
MSG_WIN FOCUS
WIN_ID
**Windowing Functions**

### win_set_ink_method()

**Set Inking Method**

**Syntax**

```c
error_code win_set_ink_method(WIN_ID win,
                              WIN_INK_METHOD method)
```

**Description**

`win_set_ink_method()` sets the inking method for the window specified by `win`. The method used to draw the ink is determined by the current ink method. See `WIN_INK_METHOD` for more information.

Ink is automatically drawn by `maui_win` as follows. Inking starts when button 1 on the pointer device is depressed within the inking window. Ink continues to be drawn while the pointer is moved until the button is released. Ink is drawn using a 3 x 3 pixel square pen with a hit-point at its center.

If the inking window is obscured in any way by another window, then inking is not performed. Instead, a `WIN_MSG_INK_OFF` message is sent each time button 1 on the pointer device is pressed.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `win` Window ID.
- `method` Current ink method.

**Non-Fatal Errors**

010:001 EOS_MAUI_BADACK Bad acknowledgment from `maui_win`. 
12 Windowing Functions

010:008 EOS_MAUI_BADID
The ID specified by win is not valid.

010:016 EOS_MAUI_BADVALUE
An invalid inking method was specified.

010:036 EOS_MAUI_NOINIT
This API has not been initialized with win_init.

MAUI Win Errors
010:060 EOS_MAUI_NOTOWNER
This process does not own win.

Indirect Errors
msg_read()
msg_write()

See Also
win_create_win()
win_erase_ink()
win_get_win_status()
win_set_ink_pix()
WIN_ID
WIN_INK_METHOD
MSG_WIN_INK_OFF
## win_set_ink_pix()

Set Pixel Value for Ink

### Syntax

```c
error_code win_set_ink_pix(WIN_ID win, GFX_PIXEL ink_pixel)
```

### Description

`win_set_ink_pix()` sets the pixel value to use when drawing ink to the window `win`. The method used to draw the ink is specified by the inking method. See `WIN_INK_METHOD` for more information.

If successful, this function returns `SUCCESS`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `win` Window ID.
- `ink_pixel` Ink pixel value.

### Non-Fatal Errors

- **010:001 EOS_MAUI_BADACK**
  
  Bad acknowledgment from the `maui_win` process.

- **010:008 EOS_MAUI_BADID**
  
  The ID specified by `win` is not valid.

- **010:036 EOS_MAUI_NOINIT**
  
  This API has not been initialized with `win_init`.

### MAUI Win Errors

- **010:060 EOS_MAUI_NOTOWNER**
  
  This process does not own `win`.
12 Windowing Functions

**Indirect Errors**
msg_read()
msg_write()

**See Also**
win_create_win()
win_get_win_status()
win_set_ink_method()
GFX_PIXEL
WIN_ID
WIN_INK_METHOD
Windowing Functions

**win_set_msg_mask()**

Set Mask for Queuing Messages

---

**Syntax**

```c
error_code win_set_msg_mask(WIN_ID win, WIN_MSG_MASK mask)
```

**Description**

The `win_set_msg_mask()` function sets the mask used when queuing messages to the window `win`. Only message types included in this mask are queued for this process.

Calling this function only affects future writes to the mailbox. Messages that are already queued are not affected.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `win`: Window ID.
- `mask`: Message types to write to mailbox.

**Non-Fatal Errors**

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from the mauli_win process.
- **010:008 EOS_MAUI_BADID**: The ID specified by `win` is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `win_init`. 
MAUI Win Errors

010:060 EOS_MAUI_NOTOWNER

This process does not own win.

Indirect Errors

msg_read()
msg_write()

See Also

win_create_win()
win_get_win_status()
win_move_win()
win_resize_win()
win_restack_win()
win_set_state()
WIN_ID
WIN_MSG_MASK
Windowing Functions

**win_set_state()**
Set Window State

**Syntax**

```c
error_code
win_set_state(WIN_ID win, BOOLEAN active)
```

**Description**

`win_set_state()` sets the state of the window `win`. The position of `win` within the window stack is not changed. Either `win` or the parent of `win` must be owned by this process.

If the process that owns the parent of `win` has asked to be notified (see `win_set_msg_mask()`) about state change operations on `win`, then the state of the window is not changed. Instead, a `WIN_MSG_STATE_REQ` message is sent to the parent and this function returns. The process that owns the parent window then decides what to do with the request (honor the request or ignore it.)

If the parent did not ask for notification, the state of the window is changed and a `WIN_MSG_STATE` message is sent.

If `active` is set to `TRUE`, the window is activated (made visible). If `active` is set to `FALSE`, the window is deactivated.

If the state change causes the pointer to move to a different window, then a `WIN_MSG_BORDER_LEAVE` message is sent to the window losing the pointer and a `WIN_MSG_BORDER_ENTER` message is sent to the window receiving the pointer.

If the state change causes other window areas to be exposed, a `WIN_MSG_EXPOSE` message is sent for each area exposed.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
## 12 Windowing Functions

### Parameters

- **win**: Window ID.
- **active**: Boolean value of window state (true or false).

### Non-Fatal Errors

- **010:001 EOS_MAUI_BADACK**: Bad acknowledgment from the maui_win process.
- **010:008 EOS_MAUI_BADID**: The ID specified by win is not valid.
- **010:036 EOS_MAUI_NOINIT**: This API has not been initialized with `win_init`.

### MUAI Win Errors

- **010:059 EOS_MAUI_NOTALLOWED**: The state of the root window cannot be changed.
- **010:060 EOS_MAUI_NOTOWNER**: This process does not own either win or the parent of win.

### Indirect errors

- `msg_read()`
- `msg_write()`

### See Also

- `win_create_win()`
- `win_get_win_status()`
- `win_set_msg_mask()`
- `BOOLEAN`
- `MSG_WIN_BORDER`
- `MSG_WIN_EXPOSE`
- `MSG_WIN_STATE`
- `WIN_ID`
Windowing Functions

**win_set_txt_context()**

Set Text Context

**Syntax**

```c
error_code win_set_txt_context(WIN_ID win,
                                TXT_CONTEXT_ID context)
```

**Description**

`win_set_txt_context()` sets the text context for the specified window `win`. You must assign the text context to the window before you use the context object to perform any drawing.

Drawing is performed with the Text API, however, the Windowing API maintains the origin, drawing area, and clipping area in the text context. Therefore, you must not call `txt_set_context_origin()`, `txt_set_context_draw()`, or `txt_set_context_clip()`.

The text context must be removed from the window before it is deleted. Call `win_set_txt_context()` with `context` set to `NULL` to do so.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **win**
  - Window ID.
- **context**
  - Text context ID.
Non-Fatal Errors

010:008 EOS_MAUI_BADID
The ID specified by `win` is not valid.

010:036 EOS_MAUI_NOINIT
API not initialized with `win_init`.

010:060 EOS_MAUI_NOTOWNER
This process does not own `win`.

Indirect Errors

- `_os_sema_p()` See [Ultra C Library Reference](#)
- `_os_sema_v()` See [Ultra C Library Reference](#)
- `mem_realloc()`
- `txt_set_context_draw()`
- `txt_set_context_dst()`
- `txt_set_context_clip()`
- `txt_set_context_origin()`

See Also

- `txt_set_context_clip()`
- `txt_set_context_draw()`
- `txt_set_context_origin()`
- `win_create_win()`
- `win_get_win_status()`
- `DRW_CONTEXT_ID`
- `WIN_ID`
Windowing Functions

\section*{win_term()}

\noindent Terminate the Windowing API

\begin{verbatim}
Syntax

error_code
win_term(void)

Description

win_term() terminates the Windowing API. All internal resources in use by the API are returned to the system.

All windowing devices created with win_create_dev() are destroyed by calling win_destroy_dev() for each one. All windowing devices opened with win_open_dev() are closed by calling win_close_dev() for each.

Since this API depends on the Shaded Memory, Messaging, Graphics, and Bit-BLT APIs, mem_term(), msg_term(), gfx_term(), and blt_term are called by this function.

If successful, this function returns SUCCESS.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

None

Non-Fatal Errors

010:036 EOS_MAUI_NOINIT This API has not been initialized with win_init().

Indirect Errors

blt_term()
gfx_term()
mem_free()
\end{verbatim}
Windowing Functions

mem_term()
msg_close_mbox()
msg_term()
win_close_dev()
win_destroy_dev()

See Also

win_init()
Windowing Functions

win_ungrab_ptr()

Release explicit cursor capture

**Syntax**

```c
error_code win_ungrab_ptr(WIN *win)
```

**Description**

`win_ungrab_ptr()` releases the explicit cursor capture of `win_grab_ptr()`. After locking the device, `maui_win` checks whether an explicit grab is in effect. If an explicit grab is in effect, `maui_win` sets the grab state to none. It then checks where the cursor landed. If the cursor landed in a different window, the colormap and the cursor are reinstalled.

If successful, this function returns `SUCCESS`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **win** Window ID.

**Non-Fatal Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010:008 EOS_MAUI_BADID</td>
<td>The ID specified by <code>win</code> is not valid.</td>
</tr>
<tr>
<td>010:036 EOS_MAUI_NOINIT</td>
<td>This API has not been initialized with <code>win_init()</code>.</td>
</tr>
<tr>
<td>010:054 EOS_MAUI_NOTBUSY</td>
<td>Cursor has not been explicitly grabbed.</td>
</tr>
<tr>
<td>010:060 EOS_MAUI_NOTOWNER</td>
<td>This process does not own <code>win</code>.</td>
</tr>
</tbody>
</table>
12 Windowing Functions

See Also

- `win_grab_ptr()`
- `WIN_ID`
### win_unlock_region()

#### Unlock a Region of a Window

**Syntax**

```c
error_code
win_unlock_region(WIN_ID win)
```

**Description**

`win_unlock_region()` unlocks the lock for window `win` that was put in place when `win_lock_region()` was called. No drawing should be done with the window unlocked because it could interfere with other windows or the graphics cursor.

If the graphics cursor was turned off by `win_lock_region()` it is turned back on by `win_unlock_region()`.

If successful, this function returns `SUCCESS`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `win` Window ID.

**Non-Fatal Errors**

- 010:001 EOS_MAUI_BADACK: Bad acknowledgment from the `maui_win` process.
- 010:008 EOS_MAUI_BADID: The ID specified by `win` is not valid.
- 010:036 EOS_MAUI_NOINIT: This API has not been initialized with `win_init()`.
- 010:060 EOS_MAUI_NOTOWNER: This process does not own `win`. 
12 Windowing Functions

Indirect Errors
msg_read()
msg_write()
_os_sema_p()
_os_sema_v()

See Also
win_lock_region()
WIN_ID

See Ultra C Library Reference
See Ultra C Library Reference
Chapter 13: Animation Data Types
ANM_FRAME

Sprite Frame Structure

Syntax

typedef struct _ANM_FRAME {
    GFX_RECT src_area; /* Area of source drawmap */
    GFX_RECT bound_area; /* Bounding area */
    GFX_POINT hit_point; /* Hit point */
    void *user_data; /* User defined data */
} ANM_FRAME;

Description

This data structure defines a rectangular area of the source drawmap called a frame. This source drawmap is pointed to by the sprite data structure ANM_SPRITE.

src_area defines the rectangular area of the source drawmap that makes up the frame.

bound_area defines the bounding area. This is the area of the frame that includes visible pixels. This area is specified relative to the src_area. The bound_area is useful for collision detection in the applications behavior functions.

hit_point defines the hit point for the frame. When you position an object (using anm_set_object_pos()), this point corresponds to the objects position. This point is specified relative to the src_area.

user_data is user defined data. You may use this for any purpose, but it is usually used to extend the frame information with application specific information.

See Also

anm_create_sprite()
anm_set_object_pos()
ANM_SPRITE
GFX_POINT
GFX_RECT
Syntax

typedef void * ANM_GROUP_ID;

Description

This data type defines an animation group ID. This ID is returned by anm_create_group() and is used in subsequent calls to functions that require an animation group identifier.

See Also

anm_create_group()
ANM_GROUP_PARAMS
Animation Group Parameters

Syntax

typedef struct _ANM_GROUP_PARAMS {
    GFX_DEV_ID gfxdev; /* Graphics device ID */
    GFX_VPORT_ID vport; /* Destination viewport */
    const GFX_DMAP *dstdmap; /* Destination drawmap */
    GFX_POS dstx; /* X coordinate in dstdmap */
    GFX_POS dsty; /* Y coordinate in dstdmap */
    BOOLEAN dbl_buff; /* Double buffered if TRUE */
    const GFX_DMAP *bkgdmap; /* Background drawmap */
    GFX_PIXEL bkgpixel; /* Background pixel value */
} ANM_GROUP_PARAMS;

Description

This data structure is used by anm_get_group() to return the current settings for a group.

See Also
anm_get_group()
BOOLEAN
GFX_DEV_ID
GFX_DMAP
GFX_PIXEL
GFX_POS
GFX_VPORT_ID
Animation Data Types

ANM_METHOD
Drawing Method for an Object

Syntax

typedef enum {
    ANM_METH_DRAW,     /* Draw the object */
    ANM_METH_TDRAW,    /* Draw with transparency */
    ANM_METH_DRAW_BKG, /* Draw the object over a */
                     /* background */
    ANM_METH_TDRAW_BKG /* Draw the object with trans- */
                     /* parency over a background */
} ANM_METHOD;

Description

This enumerated type defines how objects are drawn. The following table shows the valid values for the animation method.

Table 13-1 Value of Method in ANM_METHOD

<table>
<thead>
<tr>
<th>Value of Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANM_METH_DRAW</td>
<td>Draw the object without consideration for what is under it.</td>
</tr>
<tr>
<td>ANM_METH_DRAW_BKG</td>
<td>Same as ANM_METH_DRAW except that a background is supported.</td>
</tr>
<tr>
<td>ANM_METH_TDRAW</td>
<td>Draw the object using transparency. If the transparency mask</td>
</tr>
<tr>
<td></td>
<td>is set, then it is used. Otherwise, the transparent pixel</td>
</tr>
<tr>
<td></td>
<td>value is used.</td>
</tr>
<tr>
<td>ANM_METH_TDRAW_BKG</td>
<td>Same as ANM_METH_TDRAW except that a background is supported.</td>
</tr>
</tbody>
</table>

13 Animation Data Types

See Also
anm_set_object_meth()
ANM_SPRITE
Animation Data Types

**ANM_OBJECT_ID**

Animation Object ID

**Syntax**

typedef void * ANM_OBJECT_ID;

**Description**

This data type defines an animation object ID. This ID is returned by `anm_create_object()` and is used in subsequent calls to functions that require an animation object identifier.

**See Also**

`anm_create_object()`
ANM_OBJECT_PARAMS
Animation Object Parameters

Syntax

typedef struct _ANM_OBJECT_PARAMS {
    BOOLEAN active;       /* Active if TRUE or ON */
    const ANM_SPRITE *sprite;  /* Pointer to current sprite */
    u_int16 frame;          /* Current frame */
    GFX_POINT position;    /* Current position */
    error_code (*bhv_func)(ANM_OBJECT_ID, const ANM_OBJECT_PARAMS *);  /* Behavior function */
    void *bhv_param;       /* Parameter for behavior func */
    ANM_METHOD method;     /* Draw method */
} ANM_OBJECT_PARAMS;

Description

This data structure is used by anm_get_object() to return the current settings for an object.

See Also

anm_get_object()
anm_set_object_bhv()
ANM_METHOD
ANM_SPRITE
BOOLEAN
GFX_POINT
ANM_OBJECT_PLACEMENT
Animation Object Placement

Syntax

typedef enum {
    ANM_OBJECT_FRONT,    /* In front of all objects */
    ANM_OBJECT_BACK,     /* In back of all objects */
    ANM_OBJECT_FRONT_OF, /* In front of another object */
    ANM_OBJECT_BACK_OF   /* In back of another object */
} ANM_OBJECT_PLACEMENT;

Description

This enumerated type defines how an object should be ordered within a group. Objects are ordered from front to back.

- ANM_OBJECT_FRONT places the object in front of all other objects.
- ANM_OBJECT_BACK places the object in back of all other objects.
- ANM_OBJECT_FRONT_OF places the object in front of another object.
- ANM_OBJECT_BACK_OF places the object in back of another object.

See Also

anm_create_object()
anm_restack_object()
ANM_SPRITE
Sprite Structure

Syntax

typedef struct _ANM_SPRITE {
    GFX_DMAP *srcdmap; /* Source drawmap */
    GFX_PIXEL trans_pixel; /* Transparent pixel value */
    GFX_DMAP *maskdmap; /* Transparency mask drawmap */
    u_int16 num_frames; /* Number of frames */
    ANM_FRAME *frames; /* Pointer to array of frames */
    void *user_data; /* User defined data */
} ANM_SPRITE;

Description

This data structure defines a sprite. A sprite is a collection of frames. Each frame defines an area on the source drawmap.

srcdmap specifies the source drawmap. This drawmap must contain all the frames for the sprite.

trans_pixel specifies the transparent pixel value.

maskdmap specifies the transparency mask. If it is not NULL, then it should point to the drawmap to be used as a transparency mask. The coding_method, width, height, and line_size for this drawmap must be the same as for the source drawmap srcdmap.

num_frames indicates how many frames are included in the sprite. An array of ANM_FRAME objects is pointed to by frames.

user_data is user defined data. You may use this for any purpose, but it is usually used to extend the sprite information with application specific information.
See Also
- `anm_create_sprite()`
- `anm_set_object_pos()`
- `ANM_FRAME`
- `GFX_DMAP`
- `GFX_PIXEL`
Chapter 14: Bit-BLT Data Types
**BLT_CONTEXT_ID**

Bit-BLT Context ID

**Syntax**

typedef void * BLT_CONTEXT_ID;

**Description**

This data type defines a Bit-BLT context ID. This ID is returned by `blt_create_context()` and is used in subsequent calls to functions that require a Bit-BLT context identifier.

**See Also**

`blt_create_context()`
Bit-BLT Data Types

### BLT_CONTEXT_PARAMS

**Bit-BLT Context Parameters**

#### Syntax

```c
typedef struct _BLT_CONTEXT_PARAMS {
    GFX_DEV_ID gfxdev;           /* Graphics device ID */
    BLT_MIX drwmix;              /* Mixing mode for draw ops */
    BLT_MIX cpymix;              /* Mixing mode for copy ops */
    BLT_MIX expmix;              /* Mixing mode for expand ops */
    GFX_PIXEL drwpixel;          /* Drawing pixel value */
    const GFX_DMAP *srcdmap;     /* Source drawmap */
    u_int8 exptbl_entries;       /* Number of entries in expand table */
    const GFX_PIXEL *exptbl;     /* Pixel expansion table */
    GFX_PIXEL transpixel;        /* Transparent pixel value */
    const GFX_DMAP *mask_dmap;   /* Mask drawmap */
    GFX_PIXEL ofspixel;          /* Offset pixel value */
    const GFX_DMAP *dstdmap;     /* Destination drawmap */
} BLT_CONTEXT_PARAMS;
```

#### Description

This data structure is used by `blt_get_context()` to return the current settings in a Bit-BLT context object.

- `gfxdev` is the graphics device ID.
- `drwmix, cpymix, and expmix` specify the mixing mode for draw, copy, and expand operations, respectively.
- `drwpixel` is the pixel value for drawing operations.
- `srcdmap` is the source drawmap.
- `exptbl_entries` specifies the number of entries in the table.
- `exptbl` specifies the table to use for expand operations.
- `transpixel` specifies the transparent pixel value for copy operations when `cpymix` is `BLT_MIX_RWT`. 
mask_dmap is a mask drawmap used for copy operations when 
cpymix is BLT_MIX_RWM. The mask drawmap must have the same 
width, height, and pixel depth as the source drawmap.

ofspixel specifies the offset value for expand operations. This 
value is added to each pixel after the pixel is expanded.

dstdmap is the destination drawmap.

See Also

blt_get_context()
BLT_MIX
GFX_DEV_ID
GFX_DMAP
GFX_PIXEL
Bit-BLT Data Types

**BLT_MIX**

Mixing Mode

**Syntax**

```c
typedef enum {
    BLT_MIX_REPLACE,            /* Source */
    BLT_MIX_SXD,                 /* Source XOR destination */
    BLT_MIX_N_SXD,               /* NOT (source XOR dest) */
    BLT_MIX_SOD,                 /* Source OR destination */
    BLT_MIX_N_SOD,               /* NOT (source OR destination) */
    BLT_MIX_NS_AD,               /* (NOT source) AND dest */
    BLT_MIX_SO_ND,               /* Source OR (NOT destination) */
    BLT_MIX_SPD,                 /* Src plus (arithmetic) dest */
    BLT_MIX_DMS,                 /* Dest minus (arithmetic) src */
    BLT_MIX_SPO,                 /* Source plus offset */
    BLT_MIX_RWT,                 /* Replace with transparency */
    BLT_MIX_RWM                   /* Replace with trans mask */
} BLT_MIX;
```

**Description**

This enumerated type defines the mixing mode used for draw, copy, and expand operations. This controls how pixels in the source are combined with those in the destination before they are written to the destination.

When `BLT_MIX_REPLACE` is used, the source pixels are transferred directly to the destination.

When the boolean modes (`BLT_MIX_SXD` through `BLT_MIX_SO_ND`) are used, the source pixels and destination pixels are mixed using the respective boolean operation.

When the arithmetic modes (`BLT_MIX_SPD`, `BLT_MIX_DMS`, and `BLT_MIX_SPO`) are used, the source and destination pixels are mixed using the respective arithmetic operation. Be careful using the arithmetic modes. If the operation exceeds the minimum or maximum value for the pixel, the arithmetic carry or borrow may affect the adjacent pixel.

When `BLT_MIX_RWT` is used, only non-transparent pixels are transferred to the destination. If the source pixel has the transparent pixel value (see `blt_set_context_trans()`), then it is skipped.
When `BLT_MIX_RWM` is used, the source pixels are transferred to the destination through a mask. The mask drawmap must have the same dimensions as the source drawmap, so the mask bits are obtained using the same index into the mask drawmap as that used to obtain the source pixel from the source drawmap. For each bit in the mask, if the value is 0, the destination bit is unaffected. If the bit is 1, then the source bit is transferred to the destination.

Not all mixing modes are allowed (or make sense) for all types of Bit-BLT operations. The following table details the currently supported mixing modes. In this table, the column labeled Draw refers to all drawing operations, Copy refers to the copy operations, and Expand refers to all expansion operations. A bullet-mark indicates that the mixing mode is allowed for that type of operation.
Not all Bit-BLT context parameters are used in all of the operations shown in the table above. The parameters used depend on the mixing mode and the type of Bit-BLT operation being performed.
The following tables show the parameters that are used (required) for each mixing mode. The parameter names used in these tables match the name used for the function that sets it. For example, `dst` is the destination drawmap, and its value is set by calling `blt_set_context_dst()`.

### Table 14-2 Required Parameters for Draw Operations

<table>
<thead>
<tr>
<th>Mixing Mode For Draw Operations</th>
<th>pix(^b)</th>
<th>ofs(^c)</th>
<th>dst(^d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLT_MIX_REPLACE</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_SXD</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_N_SXD</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_SOD</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_N_SOD</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_NS_AD</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_SO_ND</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_SPD</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_DMS</td>
<td>⚫</td>
<td></td>
<td>⚫</td>
</tr>
<tr>
<td>BLT_MIX_SPO</td>
<td>⚫</td>
<td>⚫</td>
<td>⚫</td>
</tr>
</tbody>
</table>

\(^a\) Use `blt_set_context_drwmix()`  
\(^b\) Use `blt_set_context_pix()`  
\(^c\) Use `blt_set_context_ofs()`  
\(^d\) Use `blt_set_context_dst()`
### Table 14-3 Required Parameters for Copy Operations

<table>
<thead>
<tr>
<th>Mixing Mode For Copy Operations</th>
<th>src</th>
<th>trans</th>
<th>mask</th>
<th>ofs</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLT_MIX_REPLACE</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_SXD</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_N_SXD</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_SOD</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_N_SOD</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_NS_AD</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_SO_ND</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_SPD</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_DMS</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>BLT_MIX_SPO</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>BLT_MIX_RWT</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>BLT_MIX_RWM</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

- a. Use `blt_set_context_cpymix()`
- b. Use `blt_set_context_src()`
- c. Use `blt_set_context_trans()`
- d. Use `blt_set_context_mask()`
- e. Use `blt_set_context_ofs()`
- f. Use `blt_set_context_dst()`
### Table 14-4 Required Parameters for Expand Operations

<table>
<thead>
<tr>
<th>Mixing Mode For Expand Operations $^a$</th>
<th>exptbl$^b$</th>
<th>src$^c$</th>
<th>ofsd$^d$</th>
<th>dst$^e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLT_MIX_REPLACE</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_SXD</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_N_SXD</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_SOD</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_N_SOD</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_N_SOD</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_NS_AD</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_SO_ND</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_SPD</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_DMS</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>BLT_MIX_SPO</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>

$^a$ Use `blt_set_context_expmix()`  
$^b$ Use `blt_set_context_exptbl()`  
$^c$ Use `blt_set_context_src()`  
$^d$ Use `blt_set_context_ofs()`  
$^e$ Use `blt_set_context_dst()`

**See Also**

- `blt_set_context_cymix()`  
- `blt_set_context_drwmix()`  
- `blt_set_context_dst()`  
- `blt_set_context_expmix()`  
- `blt_set_context_exptbl()`
Bit-BLT Data Types

blt_set_context_mask()
blt_set_context_ofs()
blt_set_context_pix()
blt_set_context_src()
blt_set_context_trans()
BLT_CONTEXT_PARAMS
14 Bit-BLT Data Types
Chapter 15: CDB Data Types
CDB_MAX_DNAME

Maximum Length of a Device Name

Syntax

CDB_MAX_DNAME

Description

This constant defines the maximum length of a device name.

See Also

cdb_get_ddr()
CDB DATA TYPES

CDB_MAX_PARAM

Maximum Length of Parameter String

Syntax
CDB_MAX_PARAM

Description
This constant defines the maximum length of a parameter string.

See Also
cdb_get_ddr()
CDB_DATA_TYPES

Device Type Names

Syntax

typedef enum
{
  CDB_TYPE_SYSTEM =0, /* System Description */
  CDB_TYPE_CDC=1, /* CD-Control Unit */
  CDB_TYPE_SOUND=2, /* Sound Processor */
  CDB_TYPE_GRAPHIC=3, /* Video Output Processor */
  CDB_TYPE_NVRAM=4, /* Non-volatile RAM Device */
  CDB_TYPE_REMOTE=5, /* Remote, Pointing, Key Devices */
  CDB_TYPE_IROUT=6, /* IR Output Blaster */
  CDB_TYPE_PIPEDEV=9, /* Pipe Device */
  CDB_TYPE_SER=20, /* SCF Serial Device */
  CDB_TYPE_PRNT=21, /* SCF Parallel Printer */
  CDB_TYPE_MIDI=25, /* SCF MIDI Device */
  CDB_TYPE_LED=26, /* LED Device */
  CDB_TYPE_RAM=30, /* RAM Extensions */
  CDB_TYPE_FLASH=31, /* FLASH Memory */
  CDB_TYPE_PD=40, /* RBF (Universal Format) Floppy Disk */
  CDB_TYPE_HC=50, /* RBF Hard disk */
  CDB_TYPE_PCFF=60, /* PCF MS-DOS Formatted Floppy Disk */
  CDB_TYPE_PCHD=70, /* PCF MS-DOS Formatted Hard Disk*/
  CDB_TYPE_TAPE=80, /* SBF Format Magnetic Tape */
  CDB_TYPE_MPV=90, /* MPEG Video Decoder */
  CDB_TYPE_MPA=91, /* MPEG Audio Decoder */
  CDB_TYPE_ANET=100, /* NFM (Arcnet) LAN */
  CDB_TYPE_ENET=101, /* FMAN (Ethernet) LAN */
  CDB_TYPE_ISDN=110, /* ISM (ISDN) WAN */
  CDB_TYPE_RTNFM=111, /* RTNFM Real-Time WAN */
  CDB_TYPE_SPC=112, /* SPF Device */
  CDB_TYPE_CTRLCHAN=113, /* Control Channel Device */
  CDB_TYPE_DATA_CHAN=114, /* Data Channel Device */
  CDB_TYPE_MACFD=120, /* MACFM Floppy Disk */
  CDB_TYPE_MACHD=130, /* MACFM hard disk */
  CDB_TYPE_WIN=1000 /* MAUI Win Pseudo-Device */
} CDB_TYPE;
**Description**

This enumerated type defines the *names* of device types known by this API. For a complete description of the CDB device types, [See CDB Device Types Description](#) on page 690.

Microware reserves the values below 30,000.

**See Also**

cdb_get_ddr()
The configuration description block contains descriptions of each device in your system. Each device is assigned a numeric value as shown in the `CDB_TYPE` data type description. The device descriptions that may be included in a CDB are described on the following pages, in alphabetical order of device name.
CDB Data Types

CDB_TYPE_ANET
NFM (Arcnet) LAN Device

Device Type

100:name:

Parameters

There are no parameters associated with this device. The inclusion of this device in the CDB indicates an NFM (Arcnet) LAN is connected to this system.
CDB_TYPE_CDC
CD Control Unit

Device Type
1:/name:SP#:n:DV#:n:

Parameters
SP#:n Sector processing delay expressed in milliseconds. Default value of \( n=27 \).
DV#:n Device number, if more than one CD control unit is present in this system. Default value is 0.

Note
This is typically a CDFM device such as the device found in a CD-i player.
CDB Data Types

CDB_TYPE_CTRLCHAN
Control Channel Device

Device Type
113:/name:SH="string":SS="string":AP="string":

Parameters
The inclusion of this device in the CDB indicates the name of the control channel device.

/enet Parameters
If /name is "/enet" the following mandatory parameters should exist:

SH="string" (S)erver (H)ost name. This is the host name to look up using gethostbyname() to get the IP address of the server for this device. An example parameter value might be: "uplink_server".
SS="string"  (S)erver (S)ervice name. This is the service name and protocol to look up using `getservbyname()` to get the port number that the server will listen on for a control channel connection from this device. This string is composed of two comma separated strings. The first is the service name and the second is the protocol. An example parameter value might be: “uplink_service_cc,tcp”.

**Note**
If the protocol is omitted, the first service entry found will be used.

If `/name` is `/enet` the following optional parameter may exist:

AP="string"  Additional Protocol. Any SPF protocol module(s) that should be pushed onto the path returned by `socket()` using `ite_path_push()` before calling `connect()`. Example parameter values might be: “/spmyprotocol0“ or “/spprot1/spprot0“.
CDB Data Types

CDB_TYPE_DATACHAN
Data Channel Device

Device Type
114:/name:HD:SH="string":SS="string":AP="string":

Parameters

HD
If this parameter is present, it indicates that a Hardware-Direct channel from the demultiplexor to the MPEG hardware is available. If this parameter is not present, no hardware-direct channel to the MPEG hardware is available.

/net Parameters
If /name is "/net" the following mandatory parameters should exist:

SH="string"
Server Host name. This is the host name to look up using gethostbyname() to get the IP address of the server for this device. An example parameter value might be: "uplink_server". 
SS="string"

Server Service name. This is the service name and protocol to look up using getservbyname() to get the port number that the server will listen on for a data channel connection from this device. This string is composed of two comma separated strings. The first is the service name and the second is the protocol. An example parameter value might be: “uplink_service_dc,tcp”.

---

**Note**

If the protocol is omitted, the first service entry found will be used.

---

If /name is "/enet" the following optional parameter may exist:

AP="string"

Additional Protocol. Any SPF protocol module(s) that should be pushed onto the path returned by socket() using ite_path_push() before calling connect(). Example parameter values might be: “/smyprotocol0” or “/spprot1/spprot0”.
CDB Data Types

CDB_TYPE_ENET
IFMAN (Ethernet) LAN Device

Device Type
101:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an IFMAN local area network device (Ethernet) is connected to the system.
CDB_TYPE_FD
RBF (Universal Format) Floppy Disk

Device Type
40: /name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an RFB (universal format) floppy disk drive is present in the system.
Device Type

31: \textit{name}:

Parameter

There are no parameters associated with this device. The inclusion of this device in the CDB indicates Flash RAM is present in the system.
Device Type
3:/name:DE#n:LI#n:AI="MAUI":RE#n:GR#sz,co:
  PR#sz,co:PL#n:

Parameters

DE#n  Indicates maximum pixel depth (not color depth). The default value is 8.

LI#n  Indicates the maximum number of sequential scan lines in display. The default value is 240. Example parameter values are:
  240 (NTSC) 280 (PAL)

AI="MAUI"  Required field. Indicates MAUI API.

RE#n  Indicates the number of columns in the maximum resolution. The default value is 720.

GR#sz,co  Bank of graphic RAM of size (sz) kilobytes and type color (co). Graphics RAM in this DDR is accessible by the graphics processor and the CPU, but it is allocated by the graphics driver rather than the kernel. There are no default values.

PR#sz,co  Bank of pseudo RAM of size (sz) kilobytes and type color (co). Pseudo RAM is only accessible by the graphics processor. Memory out of this bank is allocated to the application by the graphics driver rather than the kernel. There are no default values.
Indicates the maximum depth of the viewport stack in planes. The default value is 1.

**Note**
For a more detailed description of a graphics device use the `gfx_get_dev_cap()` call.

**For More Information**
For more information on how the RAM is allocated and managed by the system and graphics device, see Chapter 6: Graphics Functions.
CDB_TYPE_HD
RBF (Universal Format) Hard Disk

Device Type
50 : name :

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates a hard disk drive is present in the system.
CDB Data Types

CDB_TYPE_IROUT
IR Output Blaster

Device Type
6 : /name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an IR Output Blaster is present in the system.
CDB_TYPE_ISDN
ISM (ISDN) WAN Device

Device Type
110:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an ISM (ISDN) wide area network (WAN) is connected to the system.
CDB Data Types

CDB_TYPE_LED

LED Device

Device Type
26:/name:DW#n:TY="string":

Parameters

DW#n  n=display width
TY="string"  Type of LED defined by:
            string="alpha"
            string="numeric"
            string="dot"
CDB_DATA_TYPES

CDB_TYPE_MACFD
Mac FM Floppy Disk

Device Type
120:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates a Macintosh formatted floppy disk drive is present in the system.
CDB Data Types

CDB_TYPE_MACHD
Mac FM Hard Disk

Device Type
130:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates a Macintosh formatted hard disk drive is present in the system.
CDB_TYPE_MIDI
SCF Midi Device

Device Type
25:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an SCF MIDI device is present in the system.
MPEG Audio Device

Device Type
91:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates the device name of the MPEG audio decoder.
CDB_TYPE_MPV
Motion Video Device

Device Type
90:\name: MV="string" :

Parameters
MV="string"
This parameter indicates the type of MPEG that is supported by the MPEG video decoder. The default value is "MPEG1". Valid values for this parameter are:

  string="MPEG1"
  string="MPEG2"
**CDB_TYPE_NVRAM**

Non-Volatile Ram

**Device Type**

4:\name:SZ#n:

**Parameters**

SZ#n Size of the NVRAM in kilobytes. The default size is 8.
CDB_TYPE_PCFD
PCF (MS-DOS Format) Floppy Disk

Device Type
60:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates a PCF (MS-DOS format) floppy disk drive is present in the system.
CDB Data Types

CDB_TYPE_PCHD
PCF (MS-DOS Format) Hard Disk

Device Type
70: /name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates a PCF (MS-DOS format) hard disk drive is present in the system.
**CDB_TYPE_PIPEDEV**

Pipe Device

**Device Type**

9 : *name :

**Parameters**

There are no parameters associated with this device. The inclusion of this device in the CDB indicates a pipe device is present in the system.
CDB Data Types

CDB_TYPE_PRNT
SCF Parallel Printer Device

Device Type
21:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an SCF parallel printer device is present in the system.
CDB_TYPE_RAM

RAM Extensions

Device Type
30:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates RAM extensions are present in the system.
Device Type
5 : /name : TY = "string" :

Parameters
TY = "string"

Indicates the type of Input device. The default type is "key". Valid types are:
string = "key"  key device
string = "ptr"  pointer device
string = "cmb"  combination device
CDB_TYPE_RTNFM
RTNFM Real-Time WAN Device

Device Type
111:/name:SD:

Parameters
SD
If this parameter is present, it indicates that a direct DMA channel to the MPEG hardware is available.
CDB Data Types

CDB_TYPE_SER
SCF Serial Device

Device Type
20:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an SCF Serial device is present in the system.
CDB TYPE SOUND
MFM Sound Processor

Device Type
2:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an MFM sound processor is present in the system.
CDB Data Types

CDB_TYPE_SPF
SPF Device

Device Type
112:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an SPF device is present in the system.
CDB_TYPE_SYSTEM

System Description

Device Type
0:sys:CP="string";LE:OS="os";RV="n.n";DV="n.n": 
   MM="model";SR#sz,co;GR#sz,co;ED#ed:

Parameters
CP="string" There is no default. CPU type is defined by:

Motorola Family Targets
"68000"  "68010"  "68020"  "68030"  "68040"
"68060"  "68070"  "68301"  "68302"  "68303"
"68306"  "68307"  "68322"  "68328"  "68330"
"68331"  "68332"  "68334"  "68340"  "68341"
"68349"  "68360"  "68EC030"  "68EC040"
"68F333"  "68LC040"  "CPU32"

Power PC Family Targets
"PPC403"  "PPC505"  "PPC601"  "PPC602"  "PPC603"
"PPC604"  "PPC821"  "PPC860"

Intel 80x86 Family Targets
"80386"  "80486"  "P5"

MIPS Family Targets
"MIPS"  "MIPS3000"  "IDT3081"  "MIPS4000"
"MIPS4FPP"  "IDT4650"  "MIPS4DFP"  "IDT4700"

ARM Family Targets
"ARM"  "ARMV3"  "ARM710A"
"ARMV4"  "ARM7TDMI"
**SH Family Targets**

"SH"    "SH7700"    "SH3"    "SH3E"

This list is constantly expanding. If your processor is not listed, please contact Microware Support for assistance.

**LE**

Default is MSB first (big-endian), indicated by the absence of this parameter. If this parameter is present, it indicates that the target processor uses an LSB first byte ordering system (little-endian).

**OS=“string”** Defines the operating system. There are no default values. Valid string values are:

"OS9"    "OS9000"

**RV="n.n"** Indicates the operating system revision number, such as RV="3.0". There are no default values.

**DV="n.n"** Indicates the DAVID revision level, such as DV="2.1". There are no default values.

**MM="string"** Provides the manufacturer's model. There are no default values.

**SR#sz,co** Bank of system RAM of size (sz) kilobytes and color (co). System RAM is allocated by the kernel and is not accessible by the graphics processor. There are no default values.

**GR#sz,co** Bank of graphic RAM of size (sz) kilobytes and color (co). Graphics RAM in this DDR is allocated by the kernel and is directly accessible by both the CPU and graphics processor. There are no default values.

**ED#ed** Indicates the release and edition level. Default edition level is 0.
Device Type
80: /name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates an SBF format magnetic tape drive is present in the system.
CDB Data Types

CDB_TYPE_WIN
MAUI WIN Pseudo-Device

Device Type
1000:/name:

Parameters
There are no parameters associated with this device. The inclusion of this device in the CDB indicates a MAUI windowing pseudo-device is present in the system.
Chapter 16: Drawing Data Types
**DRW_CONTEXT_ID**

Drawing Context ID

**Syntax**

typedef void * DRW_CONTEXT_ID;

**Description**

This data type defines a drawing context ID. This ID is returned by `drw_create_context()` and is used in subsequent calls to functions that require a drawing context identifier.

**See Also**

`drw_create_context()`
Drawing Context Parameters

Syntax

typedef struct _DRW_CONTEXT_PARAMS {
    GFX_DEV_ID gfxdev;   /* Graphics device */
    DRW_FM fill_mode;   /* Fill mode */
    DRW_LS line_style;  /* Line style */
    u_int32 dash_pattern; /* Dash pattern */
    u_int16 dash_magnify; /* Magnification for dash ptn */
    BLT_MIX mixmode;    /* Mixing mode */
    GFX_PIXEL drwpixel; /* Pixel value for drawing */
    GFX_PIXEL transpixel; /* Transparent pixel value */
    GFX_PIXEL ofspixel; /* Offset pixel value */
    const GFX_DMAP *dstdmap; /* Destination drawmap */
    GFX_POINT origin;   /* Drawing origin */
    GFX_RECT draw_area; /* Drawing area */
    u_int32 num_clip_areas; /* Number of clipping areas */
    const GFX_RECT *clip_areas; /* Array of clipping areas */
    BLT_MIX cpymix;     /* Mixing mode for copying */
    BLT_MIX expmix;     /* Mixing mode for expanding */
    const GFX_DMAP *srcdmap; /* Source dmap (NULL if none) */
    u_int8 exptbl_entries; /* Number of entries in extbl */
    const GFX_PIXEL *exptbl; /* Pixel expansion table */
    const GFX_DMAP *mask_dmap; /* Mask drawmap (NULL if none */
} DRW_CONTEXT_PARAMS;

Description

This data structure is used by drw_get_context() to return the current values in a text context object. gfxdev is set when the context object is created with drw_create_context(). All other members are set with their respective set-context function listed in the SEE ALSO section.
See Also

drw_get_context()
drw_set_context_dst()
drw_set_context_clip()
drw_set_context_dash()
drw_set_context_draw()
drw_set_context_dst()
drw_set_context_expmix()
drw_set_context_exptbl()
drw_set_context_fm()
drw_set_context_ls()
drw_set_context_mask()
drw_set_context_mix()
drw_set_context_ofs()
drw_set_context_origin()
drw_set_context_pix()
drw_set_context_src()
drw_set_context_trans()
BLT_MIX
DRW_FM
DRW_LS
GFX_DMAP
GFX_PIXEL
GFX_POINT
GFX_RECT
### Syntax

typedef enum {
    DRW_FM_OUTLINE, /* Draw an outlined shape */
    DRW_FM_SOLID    /* Draw a solid shape */
} DRW_FM;

### Description

This enumerated type defines how closed shapes are drawn. If DRW_FM_SOLID is used, then solid shapes are drawn. If DRW_FM_OUTLINE is used, then only the outline of shapes are drawn.

This attribute only has an effect on closed shapes, such as rectangles and circles. Open shapes, such as lines and polylines are not affected.

### Syntax

drw_set_context_fm()
**DRW_LS**

**Line Style**

**Syntax**

```c
typedef enum {
    DRW_LS_SOLID, /* Draw a solid outline */
    DRW_LS_DASHED /* Draw a dashed outline */
} DRW_LS;
```

**Description**

This enumerated type defines how outlined shapes are drawn. If `DRW_LS_SOLID` is used, then shapes are drawn with a solid outline. If `DRW_LS_DASHED` is used, then shapes are drawn with a dashed outline.

This attribute only has an effect on outlined shapes. Solid shapes are not affected.

**See Also**

`drw_set_context_ls()`
Chapter 17: Graphics Data Types
GFX_A1_RGB
RGB Color with Alpha Flag

Syntax
typedef u_int32 GFX_A1_RGB;

Description
This data type defines a color by specifying an alpha flag and its red, green, and blue components. The most significant bit is the alpha flag. The remaining 7 bits of the top byte (bits 24 through 30) are unused and must be zero. If the alpha flag is 0, then the color is transparent. If it is 1, then the color is opaque.

The remaining three bytes (from most significant to least significant) specify the red, green, and blue values. The following table shows the format and range for each component.

<table>
<thead>
<tr>
<th>Component</th>
<th>Format</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Green</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Blue</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
</tbody>
</table>
See Also

gfx_set_dev_attribute()
gfx_set_display_transcol()
gfx_set_vport_colors()
GFX_COLOR
GFX_COLOR_TYPE
GFX_PALETTE
GFX_ANGLE
Angle in 64ths of a Degree

Syntax
typedef int32 GFX_ANGLE;

Description
This data type defines an angle value. The unit of measurement is one 64th (1/64) of a degree.

See Also
None
Graphics Data Types

**GFX_ATTR_MODE**
Graphics Device Attribute Mode

**Syntax**

```c
typedef enum {
    GFX_ATTR_RESET, /* Reset attribute to the default */
    GFX_ATTR_ABSOLUTE, /* Set to specific value */
    GFX_ATTR_RELATIVE /* Set relative to the current value */
} GFX_ATTR_MODE;
```

**Description**

This enumerated type is used to specify the update mode when calling `gfx_set_dev_attribute()`. Valid attribute modes are:

- **GFX_ATTR_RESET** causes the attribute value to reset to the default value.
- **GFX_ATTR_ABSOLUTE** set the attribute value to specific value.
- **GFX_ATTR_RELATIVE** set the attribute value relative to the current value.

**See Also**

- `GFX_ATTR_TYPE`
- `gfx_set_dev_attribute()`
# GFX_ATTR_TYPE

Graphics Device Attribute Types

## Syntax

```c
typedef enum {
    GFX_ATTR_BRIGHTNESS, /* Brightness control */
    GFX_ATTR_CONTRAST,   /* Contrast control */
    GFX_ATTR_HUE,        /* Hue control */
    GFX_ATTR_SATURATION, /* Saturation control */
    GFX_ATTR_SHARPNESS,  /* Sharpness control */
    GFX_ATTR_GAMMA,      /* Gamma control */
    GFX_ATTR_WHITEBALANCE, /* White Balance control */
    GFX_ATTR_DEVSPECIFIC /* Device/OEM specific */
} GFX_ATTR_TYPE;
```

## Description

This enumerated type defines an attribute type when calling `gfx_get_dev_attribute()` or `gfx_set_dev_attribute()`. Valid attributes types include:

- `GFX_ATTR_BRIGHTNESS` specifies brightness control.
- `GFX_ATTR_CONTRAST` specified contrast control.
- `GFX_ATTR_HUE` specifies hue control.
- `GFX_ATTR_SATURATION` specifies saturation control.
- `GFX_ATTR_SHARPNESS` specifies sharpness control.
- `GFX_ATTR_GAMMA` specifies gamma control.
- `GFX_ATTR_WHITEBALANCE` specifies white balance control.

`GFX_ATTR_TYPES` between `GFX_ATTR_DEVSPECIFIC` and `MAX_INT32` are device/OEM specific and should be defined contiguously to allow searching. A description of the attribute is pointed to by the `description` field of the `GFX_DEV_ATTR` returned by `gfx_get_dev_attribute()`.
See Also

GFX_ATTR_MODE
GFX_DEV_ATTR
gfx_get_dev_attribute()
gfx_set_dev_attribute()
GFX_CM
Pixel Coding Method

**Syntax**

typedef u_int32 GFX_CM;

**Description**

This enumerated type defines the coding method for a drawmap (GFX_DMAP). The coding method for a drawmap defines the method used to encode the pixel data pointed to by the pixmem member of the drawmap data structure.

Be aware that not all coding methods can be displayed by all hardware. The coding methods supported by a graphics device may be obtained from its device capabilities. Use gfx_get_devcap() to retrieve the device capabilities.

The following diagram shows the bit-fields contained in the coding method.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| BE | NE | DEPTH | ALIGN | RESERVED | NAME |

"True Color Swap"

There are macros for getting and setting each of these bit-fields within the GFX_CM. The set macros take the bit-field value to set and return this value shifted and masked correctly for its placement in GFX_CM. The get macros take a GFX_CM value and return the specified bit-field. As an example, the following two lines of code sets and gets the depth field.

```c
   cm |= gfx_set_cm_depth(2);
   depth = gfx_get_cm_depth(cm);
```
**BE**

Bit 31 contains the Byte Endianess value. It defines the byte-endianess of the pixel data. If this value is 0, the data is big-endian. If this value is 1, the data is little-endian. You should use the `gfx_set_cm_byte_order()` macro to set this field and the `gfx_get_cm_byte_order()` macro to read this field.

**bE**

Bit 30 contains the Bit Endianess value. It defines the bit-endianess of the pixel data. If this value is 0, the data is big-endian. If this value is 1, the data is little-endian. You should use the `gfx_set_cm_bit_order()` macro to set this field and the `gfx_get_cm_bit_order()` macro to read this field.

**Depth**

This value defines the depth of each pixel. If this value is 0, you must infer the depth from the NAME field. This is for backwards compatibility with previous versions of MAUI. The following table defines the possible values for this field.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pixel depth must be inferred from NAME field</td>
</tr>
<tr>
<td>1</td>
<td>Pixel depth is 1 bit-per-pixel</td>
</tr>
<tr>
<td>2</td>
<td>Pixel depth is 2 bits-per-pixel</td>
</tr>
<tr>
<td>3</td>
<td>Pixel depth is 4 bits-per-pixel</td>
</tr>
<tr>
<td>4</td>
<td>Pixel depth is 8 bits-per-pixel</td>
</tr>
</tbody>
</table>
Use the `gfx_set_cm_depth()` macro to set this field and the `gfx_get_cm_depth()` macro to read this field.

**Align**

This value defines the alignment required for each line of pixels. For example, the default (0) indicates that each line must start on a 4-byte boundary. The following table defines the possible values for this field.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description (Each line must start on the specified boundary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A 4-byte boundary</td>
</tr>
<tr>
<td>1</td>
<td>An 8-byte boundary</td>
</tr>
<tr>
<td>2</td>
<td>A 16-byte boundary</td>
</tr>
<tr>
<td>3</td>
<td>A 32-byte boundary</td>
</tr>
<tr>
<td>4</td>
<td>A 64-byte boundary</td>
</tr>
<tr>
<td>5</td>
<td>A 128-byte boundary</td>
</tr>
<tr>
<td>6</td>
<td>A 256-byte boundary</td>
</tr>
<tr>
<td>7</td>
<td>A 512-byte boundary</td>
</tr>
</tbody>
</table>
The bit arrangement in the align field is non-standard as it was expanded to support hardware that requires large line requirements while providing backward compatibility with earlier versions of MAUI. Bit 23 of the GFX_CM is the most significant bit of the align field. Bit 24 is the least significant bit of the align field, with bits 25 and 26 increasing in significance in a normal fashion.

<table>
<thead>
<tr>
<th>ALIGN 23 26 25 24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined Format</td>
</tr>
<tr>
<td>Interpreted Format</td>
</tr>
</tbody>
</table>

The macros gfx_set_cm_align() and gfx_get_cm_align() interpret this organization correctly. The application programmer does not need to work directly within the align field.

Use the gfx_set_cm_align() macro to set this field and gfx_get_cm_align() macro to read this field. In general, the alignment may be calculated as:

```
padsize = 1 << (gfx_get_cm_align(coding_method) + 2);
```
True Color Swap

The true color swap bit is an indication to the application program that pixel values (GFX_PIXEL) must be swapped so colors appear correctly for color modes greater than 16 bits.

Use the `gfx_set_cm_tc_swap()` macro to set this field and the `gfx_get_cm_tc_swap()` macro to read this field.

Name

This value defines the name of the coding method. This field has been segmented into the following numeric ranges that indicate the class of coding method.

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 255</td>
<td>Standard Microware-defined coding methods.</td>
</tr>
<tr>
<td>256-511</td>
<td>Standard Microware-defined coding methods that require the driver to perform Bit-BLT.</td>
</tr>
<tr>
<td>512-767</td>
<td>Reserved.</td>
</tr>
<tr>
<td>768-1023</td>
<td>Defined by OEMs. If the DEPTH is 0, the driver must perform Bit-BLT operations. If it is non-zero, the Bit-BLT can be handled directly by the MAUI API code.</td>
</tr>
</tbody>
</table>

Use the `gfx_set_cm_name()` macro to set this field and the `gfx_get_cm_name()` macro to read this field.
The following table summarizes the currently supported coding method names and indicates how Bit-BLT operations are supported.

Table 17-5  Supported Coding Methods in GFX_CM

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Depth</th>
<th>API BLT</th>
<th>Driver BLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GFX_CM_UNKNOWN</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>GFX_CM_1BIT</td>
<td>1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>GFX_CM_2BIT</td>
<td>2</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>GFX_CM_3BIT</td>
<td>4</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>GFX_CM_4BIT</td>
<td>4</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GFX_CM_5BIT</td>
<td>8</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>GFX_CM_6BIT</td>
<td>8</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>GFX_CM_7BIT</td>
<td>8</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>GFX_CM_8BIT</td>
<td>8</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>GFX_CM_RGB555</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>GFX_CM_RGB888</td>
<td>32</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>GFX_CM_CDI_RL3</td>
<td>4</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>GFX_CM_CDI_RL7</td>
<td>8</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>GFX_CM_CDI_DYUV</td>
<td>8</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>GFX_CM_1A7_8BIT</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
The following paragraphs explain each coding method name. These explanations assume a big-endian encoding.

**GFX_CM_1BIT**
This is a packed-pixel format containing 1 bit for each pixel. Each 32-bit `GFX_PIXEL` represents 32 pixels. The most significant bit represents the left-most pixel.

**GFX_CM_2BIT**
This is a packed-pixel format containing 2 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 16 pixels. The most significant 2 bits represent the left-most pixel.

### Table 17-5  Supported Coding Methods in GFX_CM (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Depth</th>
<th>API</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>GFX_CM_A1_RGB555</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>GFX_CM_YCBCR422</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>GFX_CM_YCRCB422</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>GFX_CM_RGB565</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>GFX_CM_RGB555</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>GFX_CM_RGB556</td>
<td>16</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>GFX_CM_A8_RGB888</td>
<td>32</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>GFX_CM_YCBCR420</td>
<td>12</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>257</td>
<td>GFX_CM_YCRCB420</td>
<td>12</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
**GFX_CM_3BIT**

This is a packed-pixel format containing 4 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 8 pixels. The most significant 4 bits represent the left-most pixel. The most significant bit of each pixel is ignored.

**GFX_CM_4BIT**

This is a packed-pixel format containing 4 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 8 pixels. The most significant 4 bits represent the left-most pixel.

**GFX_CM_5BIT**

This is a packed-pixel format containing 8 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 4 pixels. The most significant 8 bits represent the left-most pixel. The most significant 3 bits of each pixel is ignored.

**GFX_CM_6BIT**

This is a packed-pixel format containing 8 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 4 pixels. The most significant 8 bits represent the left-most pixel. The most significant 2 bits of each pixel is ignored.

**GFX_CM_7BIT**

This is a packed-pixel format containing 8 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 4 pixels. The most significant 8 bits represent the left-most pixel. The most significant bit of each pixel is ignored.
17 Graphics Data Types

**GFX_CM_8BIT**
This is a packed-pixel format containing 8 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 4 pixels. The most significant 8 bits represent the left-most pixel.

**GFX_CM_RGB555**
This is a packed-pixel format containing 16 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 2 pixels. The most significant 16 bits represent the left-most pixel. Each 16-bit pixel is formatted as follows. The most significant bit is ignored. The next 5 bits represent the red component. The next 5 bits represent the green component. The least significant 5 bits represent the blue component.

**GFX_CM_RGB888**
This is a packed-pixel format containing 32 bits for each pixel. Therefore, each 32-bit `GFX_PIXEL` represents 1 pixel.

Each 32-bit pixel is formatted as follows. The most significant 8 bits are ignored. The next 8 bits represent the red component. The next 8 bits represent the green component. The least significant 8 bits represent the blue component.

**GFX_CM_CDI_RL3**
This is a packed-pixel format containing two types of encoding: an 8-bit value representing a pixel pair, and a 16-bit value representing a run of pixel pairs. Without runs, each 32-bit `GFX_PIXEL` represents 8 pixels. In this case, the most significant 8 bits represent the left-most pixel pair.
A pixel pair is an 8-bit value encoded as shown in the following diagram. The most significant bit is 0. The next 3 bits represent the 1st (left-most) pixel of the pixel pair. The next bit is 1. The least significant 3 bits represent the 2nd pixel of the pixel pair.

![Pixel Pair Diagram]

A run of pixel pairs is a 16-bit value encoded as shown in the following diagram. The most significant bit is 1. The next 3 bits represent the 1st (left-most) pixel of the pixel pair. The next bit is 0. The next 3 bits represent the 2nd pixel of the pixel pair. The least significant 8 bits represent the length of the run in pixel pairs.

![Run of Pixel Pairs Diagram]

The length of the run must not be 1. A length of 0 indicates a run to the end of the line. Each line of a drawmap using this coding method must end with this 0 length run.

See the Motorola Video Decoder and System Controller (VDSC) technical specification for more information about this coding method.

**GFX_CM_CDI_RL7**

This is a packed-pixel format containing two types of encoding: an 8-bit value representing a single pixel, and a 16-bit value representing a run of pixels. Without runs, each 32-bit `GFX_PIXEL` represents 4 pixels. In this case, the most significant 8 bits represent the left-most pixel.

A single pixel is an 8-bit value encoded as follows. The most significant bit is 0. The remaining 7 bits represent the pixel value.

A run of pixels is a 16-bit value encoded as follows. The most significant bit is 1. The next 7 bits represent the pixel value. The least significant 8 bits represent the length of the run in pixel pairs.
The length of the run must not be 1. A length of 0 indicates a run to the end of the line. Each line of a drawmap using this coding method must end with this 0 length run.

See the Motorola Video Decoder and System Controller (VDSC) technical specification for more information about this coding method.

**GFX_CM_CDI_DYUV**

This is a packed-pixel format containing 8 bits for each pixel. However, the pixels are grouped as 16-bit pairs because the YUV components are interleaved. Each 32-bit GFX_PIXEL represents 4 pixels (2 pixel pairs). The most significant 16 bits represent the left-most pixel pair.

Each 16-bit pixel pair is formatted as follows. The most significant 4 bits are the U component. The next 4 bits are the Y component for the 1st (left-most) pixel. The next 4 bits are the V component. The least significant 4 bits are the Y component for the 2nd pixel.

See the Motorola Video Decoder and System Controller (VDSC) technical specification for more information about this coding method.

**GFX_CM_1A7_8BIT**

This is a packed-pixel format containing 16 bits for each pixel. Each 32-bit GFX_PIXEL represents 2 pixels. The most significant 16 bits represent the left-most pixel.

Each pixel is formatted as follows. The most significant bit is an alpha flag. The next 7 bits are the alpha (translucency) value. The least significant 8 bits are the pixel value.

If the alpha flag is 0, then the pixel is opaque. If the alpha flag is 1, then the alpha value controls the translucency of the pixel. An alpha value of 0 means that the pixel is transparent. A value of 0x7f indicates that the pixel is opaque.
**GFX_CM_A1_RGB555**

This is a packed-pixel format containing 16 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 2 pixels. The most significant 16 bits represent the left-most pixel.

Each 16-bit pixel is formatted as follows. The most significant bit is the alpha flag. The next 5 bits represent the red component. The next 5 bits represent the green component. The least significant 5 bits represent the blue component.

If the alpha flag is 0, then the pixel is completely transparent. If the alpha flag is 1, then the pixel is completely opaque.

**GFX_CM_YCBCR422**

This is a packed-pixel format containing 16 bits for each pixel. The pixels are grouped as 32-bit pairs because the Cb and Cr components are interleaved. Each 32-bit `GFX_PIXEL` represents 2 pixels (1 pixel pair) and is encoded as follows.

<table>
<thead>
<tr>
<th>Y of 1st (left-most) pixel</th>
<th>Cb</th>
<th>Y of 2nd pixel</th>
<th>Cr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y of 1st (left-most) pixel</td>
<td>Cb</td>
<td>Y of 2nd pixel</td>
<td>Cr</td>
</tr>
</tbody>
</table>

**GFX_CM_YCRCB422**

This mode is identical to `GFX_CM_YCBCR422` except that the position of the Cr and Cb components are reversed.

**GFX_CM_RGB565**

This is a packed-pixel format containing 16 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 2 pixels. Each 16-bit pixel is formatted as follows. The most significant 5 bits represent the red component. The next 6 bits represent the green component. The least significant 5 bits represent the blue component.
**GFX_CM_RGB655**

This is a packed-pixel format containing 16 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 2 pixels. Each 16-bit pixel is formatted as follows. The most significant 6 bits represent the red component. The next 5 bits represent the green component. The least significant 5 bits represent the blue component.

**GFX_CM_RGB556**

This is a packed-pixel format containing 16 bits for each pixel. Each 32-bit `GFX_PIXEL` represents 2 pixels. Each 16-bit pixel is formatted as follows. The most significant 5 bits represent the red component. The next 5 bits represent the green component. The least significant 6 bits represent the blue component.

**GFX_CM_A8_RGB888**

This is a packed-pixel format containing 32 bits for each pixel. Therefore, each 32-bit `GFX_PIXEL` represents 1 pixel. Each pixel is formatted as follows. The most significant 8 bits is the alpha (translucency) value where 0xFF is opaque and 0x00 is transparent. The next 8 bits represent the red component. The next 8 bits represent the green component. The least significant 8 bits represent the blue component.

**GFX_CM_YCBCR420**

This is a format containing separate luma and chroma blocks. The pixel memory is formatted in two sections so that the luma data for all pixels is in the first section immediately followed by the chroma data for all pixels in the second section.
The first section contains luma data encoded as 8 bits for each pixel and appears within longwords formatted as follows:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Y of 1st (left-most) pixel</td>
<td>Y of 2nd pixel</td>
<td>Y of 3rd pixel</td>
<td>Y of 4th pixel</td>
</tr>
</tbody>
</table>

The second section contains chroma data encoded as 16 bits for each pixel and appears within longwords formatted as follows:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cb of 1st (top-left-most quad)</td>
<td>Cr of 1st (top-left-most quad)</td>
<td>Cb of 2nd (right of first quad)</td>
<td>Cr of 2nd (right of first quad)</td>
</tr>
</tbody>
</table>

There are two lines of luma data (first section) for each line of chroma data (second section). All three lines have the same length because the luma data is twice as tall, but only half as wide as the chroma data. This line size is reflected in the line_size member of the drawmap and is computed as 1 byte per pixel.

The triplet consisting of two luma lines and one chroma line should be thought of as an indivisible unit. The pixel pairs within a line are also indivisible units. Therefore, both the width and height in pixels should be of multiples of 2. The total size of the pixel memory can be computed as 12 bits per pixel (plus the necessary line padding). Use gfx_calc_pixmem_size() to compute the memory requirements.

For functions that require it (such as BLT), a single pixel of this coding method should be encoded as a 32-bit value with the following format:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Y</td>
<td>Cb</td>
</tr>
</tbody>
</table>
17 Graphics Data Types

**GFX_CM_YCRCB420**

This mode is identical to GFX_CM_YCBCR420 except the position of the Cr and Cb are reversed.

**See Also**

gfx_calc_pixmem_size()
gfx_create_dmap()
GFX_DMAP
GFX_PIXEL
**Syntax**

typedef struct _GFX_COLOR {
    GFX_COLOR_TYPE color_type; /* Type of color */
    GFX_COLOR_VALUE color; /* Color value */
} GFX_COLOR;

**Description**

This data structure defines a color. The element `color_type` specifies the type of color coding used in `color`. `color` is a union of each color type defined by this API.

See GFX_COLOR_VALUE for details specific to each color type.

**See Also**

gfx_set_dev_attribute()
gfx_set_display_transcol()
GFX_COLOR_TYPE
GFX_COLOR_VALUE
Syntax

typedef enum {
    GFX_COLOR_NONE, /* No color encoding */
    GFX_COLOR_RGB, /* RGB color(s) */
    GFX_COLOR_YUV, /* YUV color(s) */
    GFX_COLOR_A1_RGB, /* RGB color(s) with alpha flag */
    GFX_COLOR_YCBCR /*YCbCr color(s) */
} GFX_COLOR_TYPE;

Description

This enumerated type defines a color type. GFX_COLOR_NONE may be used to indicate that no color is specified. See GFX_RGB, GFX_YUV, GFX_A1_RGB, and GFX_YCBCR for information about the format in each color type.

See Also

GFX_COLOR
GFX_PALETTE
GFX_RGB
GFX_YCBCR
GFX_YUV
GFX_A1_RGB
GFX_COLOR_VALUE
Untyped Color Value

Syntax

typedef union _GFX_COLOR_VALUE {
    GFX_RGB rgb;    /* RGB color */
    GFX_YUV yuv;    /* YUV color */
    GFX_A1_RGB a1_rgb /* RGB color with alpha flag */
    GFX_YCBCR ycbcr; /* YCbCr color */
    u_int8 reserved[8]    /* Force max size to 8 bytes */
} GFX_COLOR_VALUE;

Description

This union defines an untyped color value. This is normally used in conjunction with a separate piece of data that specifies the actual color type.

If the color type is GFX_COLOR_NONE then no color is specified.
If the color type is GFX_COLOR_RGB then the color is specified by color.rgb.
If the color type is GFX_COLOR_YUV then the color is specified by color.yuv.
If the color type is GFX_COLOR_A1_RGB then the color is specified by color.a1 rgb.
If the color type is GFX_COLOR_YCBCR then the color is specified by color.ycbcr.

color.reserved should not be used. It is only present to force the maximum size of a color entry to 8 bytes. This is done in consideration of future color encoding types that may require more than 4 bytes.

See Also

gfx_set_dev_attribute()
gfx_set_display_transcol()
GFX_A1_RGB
GFX_COLOR
Graphics Data Types

GFX_COLOR_TYPE
GFX_RGB
GFX_YCBCR
GFX_YUV
Graphics Data Types

GFX_CURSOR_CAP
Hardware Cursor Capabilities

Syntax

typedef struct _GFX_CURSOR_CAP {
    u_int8 num_info; /* Number of cursor formats supported */
    GFX_CURSOR_INFO *info; /* Pointer to an array of cursor formats */
} GFX_CURSOR_CAP;

Description

This data structure is used to get information about the existence and capabilities of the graphic hardware cursor.

Use gfx_get_cursor_cap() to retrieve this information.

num_info
Indicates the number of GFX_CURSOR_INFO structures that are pointed to by info.

info
A pointer to an array of GFX_CURSOR_INFO structures. Each of the GFX_CURSOR_INFO structures indicates a supported cursor format.

See Also

GFX_CURSOR_INFO
gfx_get_cursor_cap()
**GFX_CURSOR_ID**

**Hardware Cursor ID**

**Syntax**

typedef void * GFX_CURSOR_ID;

**Description**

GFX_CURSOR_ID defines a hardware cursor ID. This ID is returned by gfx_create_cursor() and is used in subsequent calls to functions that require a cursor identifier.

**See Also**

gfx_create_cursor()
gfx_destroy_cursor()
GFX_CURSOR_INFO
Describes a Hardware Cursor Format

Syntax

typedef struct _GFX_CURSOR_INFO {
    GFX_DIMEN width; /* Width in pixels */
    GFX_DIMEN height; /* Height in pixels */
    u_int16 num_colors; /* Number of colors */
    GFX_CM bitmap_cm; /* Bitmap coding method */
    GFX_CM mask_cm; /* Mask coding method */
    u_int16 num_cursors; /* Max supported cursors */
} GFX_CURSOR_INFO;

Description

This data structure describes a supported cursor format of the graphic device. This structure is referenced by GFX_CURSOR_CAP. If a graphic device supports more than one cursor format, a GFX_CURSOR_INFO structure is defined for each format.

width and height Specifies the dimensions of the cursor in pixels.

num_colors Specifies the number of allowed colors in the cursor. If num_colors is 0, then colors cannot be set. If num_colors is greater than 256, then the cursor is not CLUT based.

bitmap_cm The required coding method for the actual cursor image.

mask_cm The required coding method for the cursor mask. Drivers will usually either want a 1 bit mask or a mask equal to the bit depth of the bitmap.

See Also

GFX_CURSOR_CAP
gfx_get_cursor_cap()
GFX_CURSOR_SPEC
Hardware Cursor Graphic Information

**Syntax**

typedef struct _GFX_CURSOR_SPEC {
    GFX_POINT hit_point; /* Hit point */
    GFX_DMAP *bitmap;    /* Cursor bitmap */
    GFX_DMAP *mask;      /* Mask for bitmap */
} GFX_CURSOR_SPEC;

**Description**

This data structure contains information about a hardware graphics cursor. It is used with `gfx_create_cursor()` to specify a hardware cursor.

**hit_point**

A data structure that contains the x and y coordinates of the hit point. `hit_point.x` and `hit_point.y` are used to determine the hit point. The hit point is used when positioning the cursor via `gfx_set_cursor_pos()`.

**bitmap**

A pointer to a drawmap structure that contains the cursor image.

**mask**

A pointer to a drawmap structure containing a logical AND mask used to limit the size and shape of the visible cursor.

`mask` must not point to a drawmap structure that has a palette associated with it. The palette entry in the drawmap pointed to by `mask` must be NULL.

**See Also**

`gfx_create_cursor()`
GFX_DELTA
Delta Between Two Positions

Syntax
typedef struct _GFX_DELTA {
    GFX_OFFSET x;    /* X offset */
    GFX_OFFSET y;    /* Y offset */
} GFX_DELTA;

Description
This data structure defines an offset from one position to another position. Each position is specified by its X and Y coordinate.

See Also
GFX_OFFSET
GFX_POINT
GFX_DEV_ATTR
Graphics Device Attribute Description

Syntax
typedef struct _GFX_DEV_ATTR {
    int32 curvalue;     /* Current value */
    int32 defvalue;     /* Default value */
    int32 minvalue;     /* Minimum value */
    int32 maxvalue;     /* Maximum value */
    int32 stepsize;     /* Step size */
    const char *description; /* Pointer to description */
} GFX_DEV_ATTR;

Description
This data structure is used by the API to give information to the
application about a graphics device attribute. Use
gfx_get_dev_attribute() to retrieve this information.

The default resolution for the device is always returned as the first entry
in res_info. The default coding method is always returned as the first
entry in cm_info.

See Also
GFX_ATTR_TYPE
gfx_get_dev_attribute()
Syntax

typedef struct _GFX_DEV_CAP {
   char *hw_type; /* Hardware type */
   char *hw_subtype; /* Hardware subtype */
   BOOLEAN sup_vpmix; /* Supports viewport mixing */
   BOOLEAN sup_extvid; /* Supports external video */
   BOOLEAN sup_bkcol; /* Supports backdrop color */
   BOOLEAN sup_vptrans; /* Supports vport transparency */
   BOOLEAN sup_vpinten; /* Supports vport intensity */
   BOOLEAN sup_sync; /* Supports retrace sync */
   u_int8 num_res; /* Num of display resolutions */
   GFX_DEV_RES *res_info; /* Array of dpy resolutions */
   u_int8 dac_depth; /* Depth of the DAC in bits */
   u_int8 num_cm; /* Num of coding methods */
   GFX_DEV_CM *cm_info; /* Array of coding methods */
   BOOLEAN sup_decode; /* Supports video decoding */
} GFX_DEV_CAP;

Description

This data structure is used by the API to give information to the application about the capabilities of a graphics device. Use gfx_get_dev_cap() to retrieve this information.

The default resolution for the device is always returned as the first entry in res_info. The default coding method is always returned as the first entry in cm_info.

See Also

gfx_get_dev_cap()
BOOLEAN GFX_DEV_CAPEXTEN
GFX_DEV_CM
GFX_DEV_RES
**GFX_DEV_CAPEXTEN**

Graphics Device Extended Capabilities

**Syntax**

```c
typedef struct _GFX_DEV_CAPEXTEN {
    /* MAUI 3.1 fields */
    u_int16 exten_ver;   /* Struct version/size */
    u_int16 num_modes;   /* Number of modes in mode_info */
    GFX_DEV_MODES *mode_info; /* Array of supported modes */
    GFX_VPC vp_complexity; /* Viewport complexity hint */
    GFX_VPDMC vpdm_complexity; /* Drawmap/vp complexity hint */
    /* add future fields here */
} GFX_DEV_CAPEXTEN;
```

**Description**

This data structure is used by the API to give additional information to the application about the capabilities of a graphics device beyond that provided by `GFX_DEV_CAP`. Use `gfx_get_dev_capexten()` to retrieve the `GFX_DEV_CAPEXTEN` information from the driver.

`exten_ver` is used to determine the revision of this structure. This field is initialized by the driver to `sizeof(GFX_DEV_CAPEXTEN)`. This allows applications to programmatically determine if their definition of `GFX_DEV_CAPEXTEN` matches that of the driver's.

`num_modes` defines the number of entries in `mode_info`. `mode_info` is a pointer to an array of resolution coding method pairs. If `num_modes` is zero, `mode_info` should be `NULL`.

`vp_complexity` indicates the class of viewport complexity supported by the graphics driver and hardware. Given the range of graphics hardware supported by MAUI, viewport complexity is too complex to be fully represented by `enum`. This field is only a brief overview of what the hardware and driver supports. The only way to determine if the driver and hardware support a specific configuration is to try it. See `GFX_VPC` for a description of each viewport complexity value.

`vpdm_complexity` indicates the types of drawmaps that can be assigned to a viewport. Given the range of graphics hardware supported by MAUI, viewport drawmap complexity is too complex to be fully represented by `enum`. This field is only a brief overview of what the
hardware and driver supports. The only way to determine if the driver and hardware support a specific configuration is to try it. See GFX_VPDMC for a description of each viewport drawmap complexity value.

This structure was created in MAUI Version 3.1. Future versions of MAUI may extend this structure by adding members to the end and changing the value in exten_ver. To determine if the version of the structure returned by the driver contains a specific member, use the GFX_DEV_CAPEXTEN_VALIDATE(ptr, mem_desig) macro. For example:

```c
if (GFX_DEV_CAPEXTEN_VALIDATE(edcap, vpdm_complexity))
    printf("vpdp = %d\n", edcap->vpdm_complexity);
```

See Also
gfx_get_dev_capexten()
GFX_DEV_CAP
GFX_DEV_MODES
GFX_VPC
GFX_VPDMC
**Syntax**

typedef struct _GFX_DEV_CM {
    GFX_CM coding_method;  /* Coding method */
    BOOLEAN clut_based;    /* TRUE if CLUT-based */
    u_int16 dm2dp_xmul;    /* Multiplier to cvt X coords */
    u_int16 dm2dp_ymul;    /* Multiplier to cvt Y coords */
    u_int8 num_color_types; /* Number of color types */
    GFX_COLOR_TYPE *color_types; /* Array of color types */
} GFX_DEV_CM;

**Description**

This data structure defines the coding methods supported by a graphics device.

coding_method is the coding method.

dm2dp_xmul and dm2dp_ymul define the multipliers used to convert values in the drawmap coordinate system to equivalent values in the display coordinate system.

num_color_types defines the number of entries in color_types. color_types is an array of the color types supported in the palette for this coding method. If num_color_types is zero, color_types will be NULL.

**See Also**
gfx_get_dev_cap()  
BOOLEAN  
GFX_CM  
GFX_COLOR_TYPE  
GFX_DEV_CAP
Graphics Data Types

GFX_DEV_ID
Graphics Device ID

Syntax

typedef void * GFX_DEV_ID;

Description

This data type defines a graphics device ID. This ID is returned by gfx_open_dev() and is used in subsequent calls to functions that require a device identifier.

See Also

gfx_clone_dev()
gfx_close_dev()
gfx_open_dev()
GFX_DEV_MODES

Graphics Device Display Modes

Syntax

typedef struct _GFX_DEV_MODES {
    u_int16 res_idx;    /* res_info index index */
    u_int16 cm_idx;     /* cm_info index */
    char *desc;         /* Description of mode */
} GFX_DEV_MODES;

Description

This data structure is used by GFX_DEV_CAPEXTEN to indicate compatible device resolutions and coding methods supported by the graphics device. This may only be a subset of the support modes.

res_idx is an array index within res_info in the device’s GFX_DEV_CAP.

cm_idx is an array index within cm_info in the device’s GFX_DEV_CAP.

desc is a pointer to an optional text description of the mode.

See Also

gfx_get_dev_cap()
gfx_get_dev_capexten()
GFX_DEV_CAP
GFX_DEV_CAPEXTEN
**GFX_DEV_PLACEMENT**

Device Placement

**Syntax**

typedef enum {
    GFX_DEV_FRONT, /* In front of all devices */
    GFX_DEV_BACK, /* In back of all devices */
    GFX_DEV_FRONT_OF, /* In front of another device */
    GFX_DEV_BACK_OF /* In back of another device */
} GFX_DEV_PLACEMENT;

**Description**

This enumerated type defines how a graphics device should be inserted into the stack of graphics devices. Devices are stacked front to back with the front-most device being the one actually seen by the user.

GFX_DEV_FRONT places the device in front of all other devices.
GFX_DEV_BACK places the device in back of all other devices.
GFX_DEV_FRONT_OF places the device in front of the specified device.
GFX_DEV_BACK_OF places the device in back of the specified device.

**See Also**

gfx_open_dev()
gfx_restack_dev()
Syntax

typedef struct _GFX_DEV_RES {
    GFX_DIMEN disp_width; /* Display width */
    GFX_DIMEN disp_height; /* Display height */
    u_int16 refresh_rate; /* Display refresh rate */
    GFX_INTL_MODE intl_mode; /* Interlace mode */
    u_int16 aspect_x; /* X dimension of aspect ratio */
    u_int16 aspect_y; /* Y dimension of aspect ratio */
} GFX_DEV_RES;

Description

This data structure defines the resolution of a graphics device.

width and height define the size of the display in pixels.

refresh_rate defines the refresh rate for the display.

intl_mode defines the interlace mode for the display. See GFX_INTL_MODE for information about this value.

aspect_x and aspect_y define the aspect ratio for this resolution.

See Also

gfx_get_dev_cap()
gfx_set_display_size()
GFX_DEV_CAP
GFX_DIMEN
GFX_INTL_MODE
GFX_DEV_STATUS
Graphics Device Status

Syntax

typedef struct _GFX_DEV_STATUS {
    GFX_DEV_RES devres;       /* Device resolution */
    BOOLEAN vpmix;            /* Viewport mixing on/off */
    BOOLEAN extvid;           /* External video on/off */
    GFX_COLOR bkcol;          /* Backdrop color */
    GFX_COLOR transcol;       /* Transparent color */
    GFX_DMAP *decode_dmap;    /* Dest dmap for decoding */
} GFX_DEV_STATUS;

Description

This data structure is used by gfx_get_dev_status() to return the status of a graphics device.

See Also

gfx_get_dev_status()
BOOLEAN
GFX_COLOR
GFX_DEV_RES
**GFX_DIMEN**  
Dimension In Pixels

**Syntax**  
```c
typedef u_int32 GFX_DIMEN;
```

**Description**  
This data type defines a dimension such as a width or height. The unit of measurement is the pixel. The maximum value is defined by `GFX_DIMEN_MAX`.

**See Also**  
- `GFX_DIMEN_MAX`
- `GFX_DIMEN_MIN`
GFX_DIMEN_MAX
Maximum Value For GFX_DIMEN

Syntax
GFX_DIMEN_MAX

Description
This constant defines the maximum value for the data type GFX_DIMEN.

See Also
GFX_DIMEN
GFX_DIMEN_MIN
Minimum Value for GFX_DIMEN

Syntax
GFX_DIMEN_MIN

Description
This constant defines the minimum value for the data type GFX_DIMEN.

See Also
GFX_DIMEN
Syntax

typedef struct _GFX_DMAP {
    GFX_CM coding_method; /* Coding method */
    GFX_DIMEN width; /* Width in pixels */
    GFX_DIMEN height; /* Height in pixels */
    size_t line_size; /* Line size in bytes */
    GFX_PIXEL *pixmem; /* Pointer to pixel memory */
    u_int32 pixmem_shade; /* Shade used for pixmem */
    size_t pixmem_size; /* Size of pixmem in bytes */
    GFX_PALETTE *palette; /* Pointer to color palette */
} GFX_DMAP;

Description

This data structure defines a rectangular area of pixel memory. This pixel memory is pointed to by pixmem. The shade that was used to allocate pixmem is specified by pixmem_shade and its size by pixmem_size.

The coding method for the drawmap is given by coding_method. If it is a CLUT based coding method, then palette should point to the color palette. Otherwise, the palette member should be NULL.

If the coding method is GFX_CM_CDI_DYUV, then the palette should contain one YUV encoded value. This value is used as the start value for each line in the drawmap. The value for palette may also be NULL. In this case the default value of 0x108080 is used.

The dimensions of the drawmap are given by width and height. These values are in pixels. The line_size is given in bytes, and must be rounded up to a multiple of GFX_LINE_PAD.
17 Graphics Data Types

Syntax

```
gfx_create_dmap()
GFX_CM
GFX_DIMEN
GFX_LINE_PAD
GFX_PALETTE
GFX_PIXEL
```
**GFX_INTL_MODE**

Interlace Mode

**Syntax**

```c
typedef enum {
    GFX_INTL_OFF, /* Interlace off */
    GFX_INTL_ON,  /* Interlace on */
    GFX_INTL_REPEAT /* Interlace field repeat */
} GFX_INTL_MODE;
```

**Description**

This enumerated type defines the interlace mode used for a display.

**GFX_INTL_OFF** turns interlace mode off.

**GFX_INTL_ON** turns interlace mode on and doubles the number of lines on the display. Odd lines from the drawmap are displayed during the odd field of each frame. Even lines from the drawmap are displayed during the even field of each frame. The first line is 0 (even).

**GFX_INTL_REPEAT** turns interlace mode on but does not change the number of lines displayed. Instead, the same image is displayed on both the odd and even fields of each frame.

When **GFX_INTL_OFF** is used, the hardware generates a progressive scan signal. When **GFX_INTL_ON** or **GFX_INTL_REPEAT** is used, the hardware generates an interlaced signal.

**See Also**

- `gfx_get_dev_cap()`
- `gfx_get_dev_status()`
- `gfx_set_display_size()`
- `GFX_DEV_RES`
GFX_LINE_PAD
Line Padding

Syntax
GFX_LINE_PAD

Description
This constant defines the line padding for pixel memory in a drawmap. Each line of the pixel memory must be a multiple of GFX_LINE_PAD bytes.

See Also
gfx_calc_pixmem_size()
gfx_set_dmap_size()
GFX_DMAP
GFX_MAX_DEV_NAME
Maximum Length of Device Name

Syntax
GFX_MAX_DEV_NAME

Description
This constant defines the maximum length of a device name. This includes the NULL byte used to terminate the string.

See Also
gfx_open_dev()
GFX_OFFSET
Offset in Pixels

Syntax
typedef int32 GFX_OFFSET;

Description
This data type defines an offset. The unit of measurement is the pixel. The minimum value is defined by GFX_OFFSET_MIN and the maximum value is GFX_OFFSET_MAX.

See Also
GFX_OFFSET_MAX
GFX_OFFSET_MIN
Graphics Data Types

GFX_OFFSET_MAX
Maximum Value For GFX_OFFSET

Syntax
GFX_OFFSET_MAX

Description
This constant defines the maximum value for the data type GFX_OFFSET.

See Also
GFX_OFFSET
17 Graphics Data Types

GFX_OFFSET_MIN
Minimum Value For GFX_OFFSET

Syntax
GFX_OFFSET_MIN

Description
This constant defines the minimum value for the data type GFX_OFFSET.

See Also
GFX_OFFSET
# GFX_PALETTE

## Color Palette

### Syntax

typedef struct _GFX_PALETTE {
    u_int16 start_entry; /* Starting entry */
    u_int16 num_colors; /* Number of colors */
    GFX_COLOR_TYPE color_type; /* Type of color table */
    union {
        GFX_RGB *rgb; /* Array of "num_colors" RGB colors */
        GFX_YUV *yuv; /* Array of "num_colors" YUV colors */
        GFX_A1_RGB *a1_rgb; /* Array of "num_colors" A1_RGB colors */
        GFX_YCBCR *ycbcr; /* Array of "num_colors" YCbCr colors */
    } colors;
} GFX_PALETTE;

### Description

This data structure defines a palette of colors. The entries in the palette correspond to pixel values in the pixel memory.

The `start_entry` defines the first pixel value that maps to an entry in the palette. Pixels with this value correspond to the first entry in the palette.

`num_colors` specifies the number of colors in the palette, and `color_type` specifies the type of color coding used in colors.

If `color_type` is `GFX_COLOR_RGB` then the array of colors is specified by `colors.rgb`.

If `color_type` is `GFX_COLOR_YUV` then the array of colors is specified by `colors.yuv`.

If `color_type` is `GFX_COLOR_A1_RGB` then the array of colors is specified by `colors.a1_rgb`.

If `color_type` is `GFX_COLOR_YCBCR` then the array of colors is specified by `colors.ycbcr`. 
See Also
gfx_set_vport_dmap()
gfx_set_vport_colors()
GFX_COLOR_TYPE
GFX_RGB
GFX_YCBCR
GFX_YUV
GFX_A1_RGB
Graphics Data Types

GFX_PIXEL

Pixel Value

Syntax

typedef u_int32 GFX_PIXEL;

Description

This data type is used to represent an individual pixel value. It may be used (in its pointer form) to point to an array of pixels. This is how the pixel memory is defined in drawmaps (see GFX_DMAP).

See Also

GFX_DMAP
GFX_POINT
Position of a Point Given by X and Y

Syntax
typedef struct _GFX_POINT {
    GFX_POS x;  /* X position */
    GFX_POS y;  /* Y position */
} GFX_POINT;

Description
This data structure is used to define a point by its coordinates. x and y determine the X and Y coordinate of the pixel.

See Also
GFX_POS
**GFX_POS**

Pixel Position

**Syntax**

typedef int32 GFX_POS;

**Description**

This data type is used to indicate an X or Y position. The unit of measurement is a pixel. The minimum value is defined by GFX_POS_MIN, and the maximum value is GFX_POS_MAX.

**See Also**

GFX_POS_MAX
GFX_POS_MIN
17 Graphics Data Types

GFX_POS_MAX
Maximum Value For GFX_POS

**Syntax**

GFX_POS_MAX

**Description**

This constant defines the maximum value for the data type GFX_POS.

**See Also**

GFX_POS
GFX_POS_MIN
Minimum Value For GFX_POS

Syntax
GFX_POS_MIN

Description
This constant defines the minimum value for the data type GFX_POS.

See Also
GFX_POS
### GFX_RECT
Rectangle Defined by X, Y, Width, and Height

**Syntax**

```c
typedef struct _GFX_RECT {
    GFX_POS x;  /* Top-left X coordinate */
    GFX_POS y;  /* Top-left Y coordinate */
    GFX_DIMEN width; /* Width of the rectangle */
    GFX_DIMEN height; /* Height of the rectangle */
} GFX_RECT;
```

**Description**

This data structure is used to define a rectangular array of pixels. The position of the upper-left corner of the rectangular area is defined by `x` and `y`. The size of the area is defined by `width` and `height`.

**See Also**

- `GFX_DIMEN`
- `GFX_POS`
**GFX_RGB**

**RGB Color**

**Syntax**

typedef u_int32 GFX_RGB;

**Description**

This data type defines a color by specifying its red, green, and blue components. The top byte is unused and must be zero. The remaining three bytes (from most significant to least significant) specify the red, green, and blue values. The following table shows the format and range for each component.

<table>
<thead>
<tr>
<th>Component</th>
<th>Format</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Green</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Blue</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
</tbody>
</table>

**See Also**

gfx_set_dev_attribute()
gfx_set_display_transcol()  
GFX_COLOR  
GFX_COLOR_TYPE  
GFX_PALETTE
## GFX_VPC

### Viewport Complexity

**Syntax**

```c
typedef enum {
    GFX_VPC_UNKNOWN,    /* Unknown, undetermined */
    GFX_VPC_OTHER,      /* Other, undefined */
    GFX_VPC_ONE_EXACT,  /* One viewport, the exact size */
    GFX_VPC_ONE_ANY,    /* One viewport, any size */
    GFX_VPC_ANY_FULL,   /* Multiple full width viewports */
    GFX_VPC_ANY_NOSHARE, /* Multiple vps on separate lines */
    GFX_VPC_ANY_SEPARATE, /* Multiple non-overlapping vps */
    GFX_VPC_ANY_COMPLEX, /* Multiple overlapping viewports */
    GFX_VPC_DEVSPECIFIC /* Device/OEM specific range */
} GFX_VPC;
```

**Description**

This enumerated type is used by `GFX_DEV_CAPEXTEN` to indicate a viewport complexity class. Viewport complexity is the number and relative position of viewports that may be simultaneously active on the display.

- **GFX_VPC_UNKNOWN** indicates that the viewport complexity is unknown or has not yet been determined.
- **GFX_VPC_OTHER** indicates a viewport complexity that does not fix an existing category or is explicitly undefined.
- **GFX_VPC_ONE_EXACT** indicates that the graphic device supports only one viewport that must be equal to the size of the display.
- **GFX_VPC_ONE_ANY** indicates that the graphic device supports only one viewport, but that it may be any size.
- **GFX_VPC_ANY_FULL** indicates that the graphic device supports multiple full display width viewports.
GFX_VPC_ANY_NOSHARE indicates that the graphic device supports multiple non-overlapping viewports which also do not share lines of the display.

GFX_VPC_ANY_SEPARATE indicates that the graphic device supports multiple non-overlapping viewports.

GFX_VPC_ANY_COMPLEX indicates that the graphic device supports multiple overlapping viewports.

GFX_VPC_DEVSPECIFIC to MAX_INT32 is a range of values for use by OEMs for device specific viewport complexity classes. All other values are reserved.

See Also
gfx_get_dev_capexten()
GFX_DEV_CAPEXTEN
GFX_VPDMC

Viewport Drawmap Complexity

Syntax

typedef enum {
    GFX_VPDMC_UNKNOWN,    /* Unknown, undetermined */
    GFX_VPDMC_OTHER,      /* Other, undefined */
    GFX_VPDMC_EXACT,      /* Drawmaps same size as viewport */
    GFX_VPDMC_TALLER,     /* Drawmaps taller than viewport */
    GFX_VPDMC_WIDER,      /* Drawmaps wider than viewport */
    GFX_VPDMC_LARGER,     /* Drawmaps larger than viewport */
    GFX_VPDMC_DEVSPECIFIC /* Device/OEM specific range */
} GFX_VPDMC;

Description

This enumerated type is used by GFX_DEV_CAPEXTEN to indicate a viewport drawmap complexity class. Viewport drawmap complexity refers to the relative size of a drawmap that can be associated with a viewport.

GFX_VPDMC_UNKNOWN indicates that the viewport drawmap complexity is unknown or has not yet been determined.

GFX_VPDMC_OTHER indicates a viewport viewport drawmap that does not fix an existing category or is explicitly undefined.

GFX_VPDMC_EXACT indicates that the graphic device supports only drawmaps the same relative size as the viewport. The size of the drawmap may have to be adjusted according to the device’s GFX_DEV_CM dm2dp_xmul and dm2dp_xmul fields. These devices do not support hardware scrolling.

GFX_VPDMC_TALLER indicates that the graphic device supports drawmaps taller than viewport. These devices support vertical hardware scrolling.

GFX_VPDMC_WIDER indicates that the graphic device supports drawmaps wider than viewport. These devices support horizontal hardware scrolling.
GFX_VPDMC_LARGER indicates that the graphic device supports drawmaps larger than viewport. These devices support both horizontal and vertical hardware scrolling.

GFX_VPDMC_DEVSPECIFIC to MAX_INT32 is a range of values for use by OEMs for device specific viewport drawmap complexity classes. All other values are reserved.

See Also
gfx_get_dev_capexten()
GFX_DEV_CAPEXTEN
**Syntax**

typedef void * GFX_VPORT_ID;

**Description**

This data type defines a viewport ID. This ID is returned by `gfx_create_vport()` and is used in subsequent calls to functions that require a viewport identifier.

**See Also**

`gfx_create_vport()`
`gfx_destroy_vport()`
`gfx_clone_vport()`
Graphics Data Types

17

GFX_VPORT_PLACEMENT
Viewport Placement

Syntax

typedef enum {
    GFX_VPORT_FRONT, /* In front of all viewports */
    GFX_VPORT_BACK, /* In back of all viewports */
    GFX_VPORT_FRONT_OF, /* In front of another viewport */
    GFX_VPORT_BACK_OF /* In back of another viewport */
} GFX_VPORT_PLACEMENT;

Description

This enumerated type defines how a viewport should be inserted into the viewport stack. Viewports are stacked front to back with the front-most viewport being closer to the user.

GFX_VPORT_FRONT places the viewport in front of all other viewports.
GFX_VPORT_BACK places the viewport in back of all other viewports.
GFX_VPORT_FRONT_OF places the viewport in front of the specified viewport.
GFX_VPORT_BACK_OF places the viewport in back of the specified viewport.

See Also

gfx_create_vport()
gfx_restack_vport()
**GFX_VPORT_STATUS**

Viewport Status

**Syntax**

typedef struct _GFX_VPORT_STATUS {
    GFX_DEV_ID gfxdev;                /* Device ID */
    u_int8 intensity;                 /* Viewport intensity */
    GFX_RECT area;                    /* Viewport area on display */
    BOOLEAN active;                   /* Viewport is active if TRUE */
    const GFX_DMAP *dmap;             /* Drawmap in the viewport */
    GFX_POS dmapx;                     /* X position in drawmap */
    GFX_POS dmapy;                     /* Y position in drawmap */
} GFX_VPORT_STATUS;

**Description**

This data structure is used by `gfx_get_vport_status()` to return the status of a viewport.

**See Also**

`gfx_get_vport_status()`

BOOLEAN

GFX_DEV_ID

GFX_DMAP

GFX_POS

GFX_RECT
GFX_YCBCR

YCbCr Color

Syntax

typedef u_int32 GFX_YCBCR;

Description

This data type defines a color by specifying its intensity (Y), and color (Cb and Cr) components. The top byte is unused and must be zero. The remaining three bytes (from most significant to least significant) specify the Y, Cb, and Cr values. The following table shows the format and range for each component.

Table 17-7 Format and Range for YCbCr Components in GFX_YCBCR

<table>
<thead>
<tr>
<th>Component</th>
<th>Format</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Cb</td>
<td>8-bit signed</td>
<td>128</td>
<td>127</td>
</tr>
<tr>
<td>Cr</td>
<td>8-bit signed</td>
<td>128</td>
<td>127</td>
</tr>
</tbody>
</table>

See Also

gfx_set_dev_attribute()
gfx_set_display_transcol()
GFX_COLOR
GFX_COLOR_TYPE
GFX_PALETTE
Syntax

typedef u_int32 GFX_YUV;

Description

This data type defines a color by specifying its intensity (Y), and color (U and V) components. The top byte is unused and must be zero. The remaining three bytes (from most significant to least significant) specify the Y, U, and V values. The following table shows the format and range for each component.

<table>
<thead>
<tr>
<th>Component</th>
<th>Format</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>8-bit unsigned</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>U</td>
<td>8-bit signed</td>
<td>-112</td>
<td>112</td>
</tr>
<tr>
<td>V</td>
<td>8-bit signed</td>
<td>-157</td>
<td>157</td>
</tr>
</tbody>
</table>

See Also

gfx_set_dev_attribute()
gfx_set_display_transcol()
GFX_COLOR
GFX_COLOR_TYPE
GFX_PALETTE
Chapter 18: Input Data Types
INP_DEV_CAP
Input Device Capabilities

Syntax

typedef struct _INP_DEV_CAP {
    INP_DEV_CLASS ptr_type; /* Device classification */
    u_int8 ptr_buttons;     /* Number of pointer buttons */
    u_int8 func_keys;       /* Number of function keys */
    INP_KEYMOD key_modifiers; /* Key modifiers supported */
    INP_KEYS key_cap;       /* Key symbol capabilities */
} INP_DEV_CAP;

Description

This data structure is used by the API to give information to the application about the capabilities of an input device. Use inp_get_dev_cap() to retrieve this information.

The MAUI Input Hardware Specification gives detailed information about the capabilities of each specific MAUI input device.

See Also

inp_get_dev_cap()
INP_DEV_CLASS
INP_KEYS
INP_KEYMOD
Input Data Types

INP_DEV_CLASS
Input Device Classification

Syntax

typedef enum {
    INP_CLASS_MOUSE,     /* Mouse */
    INP_CLASS_TRACKBALL, /* Trackball */
    INP_CLASS_TOUCHSCR,  /* Touch-screen */
    INP_CLASS_TABLET,    /* Tablet */
    INP_CLASS_JOYSTICK,  /* Joystick */
    INP_CLASS_GAMEPAD,   /* Game pad */
    INP_CLASS_KEYBOARD,  /* Alphanumeric keyboard */
    INP_CLASS_REMOTE     /* Remote control */
} INP_DEV_CLASS;

Description

This enumerated type is used to classify the characteristics of the input device.

The values in this enumerated type are powers of two. Therefore, you may combine them safely. For example:

    INP_CLASS_REMOTE | INP_CLASS_TRACKBALL

may be used to specify an input device that has the combined characteristics of a remote control and a trackball.

See Also

inp_get_dev_cap()
INP_DEV_ID
Input Device ID

Syntax
typedef void * INP_DEV_ID;

Description
This data type defines an input device ID. This ID is returned by
inp_open_dev() and is used in subsequent calls to functions that
require a device identifier.

See Also
inp_open_dev()
INP_DEV_PLACEMENT
Device Placement

Syntax

typedef enum {
    INP_DEV_FRONT,    /*In front of all devices*/
    INP_DEV_BACK,     /*In back of all devices*/
    INP_DEV_FRONT_OF, /*In front of another device*/
    INP_DEV_BACK_OF   /*In back of another device*/
} INP_DEV_PLACEMENT;

Description

This enumerated type defines how an input device should be inserted into the stack of input devices. Devices are stacked front to back with the front-most device being the one that receives input from the physical device.

INP_DEV_FRONT places the device in front of all other devices.
INP_DEV_BACK places the device in back of all other devices.
INP_DEV_FRONT_OF places the device in front of the specified device.
INP_DEV_BACK_OF places the device in back of the specified device.

See Also

inp_open_dev()
inp_restack_dev()
Syntax

typedef struct _INP_DEV_STATUS {
    GFX_POINT ptr_min;      /* Min position for pointer */
    GFX_POINT ptr_max;      /* Max position for pointer */
    GFX_POINT ptr_cur;      /* Current pos of pointer */
    u_int8 button_state;    /* Current button state */
    INP_KEYMOD key_modifiers;  /* Current key modifiers */
    INP_SIM_METH sim_meth;  /* Simulation method */
    GFX_DELTA speed;        /* X/Y speed for simulation */
    const wchar_t button_map[INP_MAX_BUTTONS]; /* Button to/from key symbol */
                            /* associations*/
    u_int32 write_mask;     /* Write mask for messages */
    void (*callback)(const void *); /* Callback for messages */
} INP_DEV_STATUS;

Description

This data structure is used by inp_get_dev_status() to return the status of an input device.

See Also

inp_get_dev_status()
inp_set_callback()
GFX_DELTA
GFX_POINT
INP_SIM_METH
INP_KEYMOD
INP_MAX_BUTTONS
Syntax

INP_KEY_ *

This key symbol map is defined in such a way that each key symbol has a unique value. These values are independent of the device from which they are generated. For example, the letter “A” is always reported as key symbol 0x41, and the “PLAY” button is symbol 0xfe80 regardless of the type of input device.

Description

INP_KEY_ is a prefix used to define a set of constants. Constants that start with INP_KEY_ are used to define the names of key symbols known by this API. Following is a table of key symbols currently defined.

<table>
<thead>
<tr>
<th>Key Symbol Name</th>
<th>Key Symbol Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_KEY_NONE</td>
<td>0x0000</td>
<td>No key</td>
</tr>
<tr>
<td>INP_KEY_NULL</td>
<td>0x0000</td>
<td>Null key</td>
</tr>
<tr>
<td>N/A</td>
<td>0x0001-0x0007</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_BS</td>
<td>0x0008</td>
<td>Backspace</td>
</tr>
<tr>
<td>INP_KEY_HT</td>
<td>0x0009</td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>INP_KEY_LF</td>
<td>0x000a</td>
<td>Line feed</td>
</tr>
<tr>
<td>N/A</td>
<td>0x000b-0x000c</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_CR</td>
<td>0x000d</td>
<td>Carriage return</td>
</tr>
</tbody>
</table>
### Table 18-1  Key Symbols in INP_KEY_* (continued)

<table>
<thead>
<tr>
<th>Key Symbol Name</th>
<th>Key Symbol Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_KEY_ENTER</td>
<td>0x000d</td>
<td>Enter</td>
</tr>
<tr>
<td>N/A</td>
<td>0x000e-0x001a</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_ESC</td>
<td>0x001b</td>
<td>Escape</td>
</tr>
<tr>
<td>N/A</td>
<td>0x001c-0x001f</td>
<td>Reserved</td>
</tr>
<tr>
<td>N/A</td>
<td>0x0020-0x007e</td>
<td>ASCII character set</td>
</tr>
<tr>
<td>INP_KEY_DEL</td>
<td>0x007f</td>
<td>Delete</td>
</tr>
<tr>
<td>N/A</td>
<td>0x0080-0xfe7f</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_PLAY</td>
<td>0xfe80</td>
<td>Play</td>
</tr>
<tr>
<td>INP_KEY_STOP</td>
<td>0xfe81</td>
<td>Stop</td>
</tr>
<tr>
<td>INP_KEY_PAUSE</td>
<td>0xfe82</td>
<td>Pause</td>
</tr>
<tr>
<td>INP_KEY_NEXT</td>
<td>0xfe83</td>
<td>Next</td>
</tr>
<tr>
<td>INP_KEY_PREV</td>
<td>0xfe84</td>
<td>Previous</td>
</tr>
<tr>
<td>INP_KEY_REWIND</td>
<td>0xfe85</td>
<td>Rewind</td>
</tr>
<tr>
<td>INP_KEY_FASTFWD</td>
<td>0xfe86</td>
<td>Fast forward</td>
</tr>
<tr>
<td>INP_KEY_RECORD</td>
<td>0xfe87</td>
<td>Record</td>
</tr>
<tr>
<td>N/A</td>
<td>0xfe88-0xfe8b</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_CUR_UL</td>
<td>0xfe8c</td>
<td>Cursor up left</td>
</tr>
</tbody>
</table>
### Table 18-1 Key Symbols in INP_KEY_* (continued)

<table>
<thead>
<tr>
<th>Key Symbol Name</th>
<th>Key Symbol Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_KEY_CUR_UR</td>
<td>0xfe8d</td>
<td>Cursor up right</td>
</tr>
<tr>
<td>INP_KEY_CUR_DL</td>
<td>0xfe8e</td>
<td>Cursor down left</td>
</tr>
<tr>
<td>INP_KEY_CUR_DR</td>
<td>0xfe8f</td>
<td>Cursor down right</td>
</tr>
<tr>
<td>N/A</td>
<td>0xfe90</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_GOTO</td>
<td>0xfe91</td>
<td>Goto</td>
</tr>
<tr>
<td>INP_KEY_EXIT</td>
<td>0xfe92</td>
<td>Exit</td>
</tr>
<tr>
<td>INP_KEY_DISPLAY</td>
<td>0xfe93</td>
<td>Display</td>
</tr>
<tr>
<td>INP_KEY_STORE</td>
<td>0xfe94</td>
<td>Store</td>
</tr>
<tr>
<td>INP_KEY_RECALL</td>
<td>0xfe95</td>
<td>Recall</td>
</tr>
<tr>
<td>N/A</td>
<td>0xfe96-0xfe98</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_CHAN_U</td>
<td>0xfe99</td>
<td>Channel up</td>
</tr>
<tr>
<td>INP_KEY_CHAN_D</td>
<td>0xfe9a</td>
<td>Channel down</td>
</tr>
<tr>
<td>INP_KEY_LASTCHAN</td>
<td>0xfe9b</td>
<td>Last channel</td>
</tr>
<tr>
<td>N/A</td>
<td>0xfe9c</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_VIP</td>
<td>0xfe9d</td>
<td>VIP</td>
</tr>
<tr>
<td>INP_KEY_VDT</td>
<td>0xfe9e</td>
<td>VDT</td>
</tr>
<tr>
<td>INP_KEY_POWER</td>
<td>0xfe9f</td>
<td>Power</td>
</tr>
<tr>
<td>Key Symbol Name</td>
<td>Key Symbol Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>INP_KEY_POWER_ON</td>
<td>0xfea0</td>
<td>Power on</td>
</tr>
<tr>
<td>INP_KEY_POWER_OFF</td>
<td>0xfea1</td>
<td>Power off</td>
</tr>
<tr>
<td>INP_KEY_BYPASS</td>
<td>0xfea2</td>
<td>Bypass</td>
</tr>
<tr>
<td>INP_KEY_BYPASS_ON</td>
<td>0xfea3</td>
<td>Bypass on</td>
</tr>
<tr>
<td>INP_KEY_BYPASS_OFF</td>
<td>0xfea4</td>
<td>Bypass off</td>
</tr>
<tr>
<td>INP_KEY_GUIDE</td>
<td>0xea5</td>
<td>Guide</td>
</tr>
<tr>
<td>INP_KEY_TUNE</td>
<td>0xfea6</td>
<td>Tune</td>
</tr>
<tr>
<td>INP_KEY_THEME</td>
<td>0xfea7</td>
<td>Theme</td>
</tr>
<tr>
<td>INP_KEY_LIST</td>
<td>0xfea8</td>
<td>List</td>
</tr>
<tr>
<td>INP_KEY_MOVE</td>
<td>0xfea9</td>
<td>Move</td>
</tr>
<tr>
<td>INP_KEY_PAGE_U</td>
<td>0xfeaa</td>
<td>Page up</td>
</tr>
<tr>
<td>INP_KEY_PAGE_D</td>
<td>0xfeab</td>
<td>Page down</td>
</tr>
<tr>
<td>INP_KEY_VOL_U</td>
<td>0xfeac</td>
<td>Volume up</td>
</tr>
<tr>
<td>INP_KEY_VOL_D</td>
<td>0xfead</td>
<td>Volume down</td>
</tr>
<tr>
<td>INP_KEY_MUTE</td>
<td>0xfeae</td>
<td>Mute</td>
</tr>
<tr>
<td>INP_KEY_SAP</td>
<td>0xfeaf</td>
<td>SAP</td>
</tr>
<tr>
<td>INP_KEY_PROG</td>
<td>0xfceb0</td>
<td>Program</td>
</tr>
</tbody>
</table>
### Table 18-1 Key Symbols in INP_KEY_* (continued)

<table>
<thead>
<tr>
<th>Key Symbol Name</th>
<th>Key Symbol Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_KEY_PPV</td>
<td>0xfeb1</td>
<td>Pay per view</td>
</tr>
<tr>
<td>INP_KEY_FAV</td>
<td>0xfeb2</td>
<td>Favorite</td>
</tr>
<tr>
<td>INP_KEY_DAY_U</td>
<td>0xfeb3</td>
<td>Day up</td>
</tr>
<tr>
<td>INP_KEY_DAY_D</td>
<td>0xfeb4</td>
<td>Day down</td>
</tr>
<tr>
<td>INP_KEY_INFO</td>
<td>0xfeb5</td>
<td>Information</td>
</tr>
<tr>
<td>INP_KEY_OPTIONS</td>
<td>0xfeb6</td>
<td>Options</td>
</tr>
<tr>
<td>INP_KEY_DEGAUSS</td>
<td>0xfeb7</td>
<td>Degauss</td>
</tr>
<tr>
<td>INP_KEY_ZOOM</td>
<td>0xfeb8</td>
<td>Zoom</td>
</tr>
<tr>
<td>INP_KEY_ZOOM_IN</td>
<td>0xfeb9</td>
<td>Zoom in</td>
</tr>
<tr>
<td>INP_KEY_ZOOM_OUT</td>
<td>0xfeba</td>
<td>Zoom out</td>
</tr>
<tr>
<td>INP_KEY_FORWARD</td>
<td>0xfebb</td>
<td>Forward</td>
</tr>
<tr>
<td>INP_KEY_BACK</td>
<td>0xfebc</td>
<td>Back</td>
</tr>
<tr>
<td>INP_KEY_RELOAD</td>
<td>0xfebd</td>
<td>Reload</td>
</tr>
<tr>
<td>INP_KEY_PIP</td>
<td>0xfebe</td>
<td>Picture-in-picture</td>
</tr>
<tr>
<td>INP_KEY_PIP_EXCHNG</td>
<td>0xfefb</td>
<td>PIP exchange</td>
</tr>
<tr>
<td>INP_KEY_PIP_CHAN_U</td>
<td>0xfec0</td>
<td>PIP channel up</td>
</tr>
<tr>
<td>INP_KEY_PIP_CHAN_D</td>
<td>0xfec1</td>
<td>PIP channel down</td>
</tr>
<tr>
<td>Key Symbol Name</td>
<td>Key Symbol Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>INP_KEY_PIP_LASTCH</td>
<td>0xfee2</td>
<td>PIP last channel</td>
</tr>
<tr>
<td>N/A</td>
<td>0xfec3-0xfeef</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_TEL0</td>
<td>0xfee0</td>
<td>Telephone key 0</td>
</tr>
<tr>
<td>INP_KEY_TEL1</td>
<td>0xfee1</td>
<td>Telephone key 1</td>
</tr>
<tr>
<td>INP_KEY_TEL2</td>
<td>0xfee2</td>
<td>Telephone key 2</td>
</tr>
<tr>
<td>INP_KEY_TEL3</td>
<td>0xfee3</td>
<td>Telephone key 3</td>
</tr>
<tr>
<td>INP_KEY_TEL4</td>
<td>0xfee4</td>
<td>Telephone key 4</td>
</tr>
<tr>
<td>INP_KEY_TEL5</td>
<td>0xfee5</td>
<td>Telephone key 5</td>
</tr>
<tr>
<td>INP_KEY_TEL6</td>
<td>0xfee6</td>
<td>Telephone key 6</td>
</tr>
<tr>
<td>INP_KEY_TEL7</td>
<td>0xfee7</td>
<td>Telephone key 7</td>
</tr>
<tr>
<td>INP_KEY_TEL8</td>
<td>0xfee8</td>
<td>Telephone key 8</td>
</tr>
<tr>
<td>INP_KEY_TEL9</td>
<td>0xfee9</td>
<td>Telephone key 9</td>
</tr>
<tr>
<td>INP_KEY_TEL_STAR</td>
<td>0xfef0a</td>
<td>Telephone star key</td>
</tr>
<tr>
<td>INP_KEY_TEL_POUND</td>
<td>0xfee0b</td>
<td>Telephone pound key</td>
</tr>
<tr>
<td>INP_KEY_SPEAKER</td>
<td>0xfee0c</td>
<td>Speaker</td>
</tr>
<tr>
<td>INP_KEY_REDIAL</td>
<td>0xfee0d</td>
<td>Redial</td>
</tr>
<tr>
<td>INP_KEY_FLASH</td>
<td>0xfee0e</td>
<td>Flash</td>
</tr>
<tr>
<td>Key Symbol Name</td>
<td>Key Symbol Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>INP_KEY_HANGUP</td>
<td>0xfeff</td>
<td>Hang up</td>
</tr>
<tr>
<td>N/A</td>
<td>0xff00-0xff0a</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_CLEAR</td>
<td>0xff0b</td>
<td>Clear</td>
</tr>
<tr>
<td>N/A</td>
<td>0xff0c-0xff12</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_HOLD</td>
<td>0xff13</td>
<td>Hold</td>
</tr>
<tr>
<td>N/A</td>
<td>0xff14-0xff4f</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_HOME</td>
<td>0xff50</td>
<td>Home</td>
</tr>
<tr>
<td>INP_KEY_CUR_L</td>
<td>0xff51</td>
<td>Cursor left</td>
</tr>
<tr>
<td>INP_KEY_CUR_U</td>
<td>0xff52</td>
<td>Cursor up</td>
</tr>
<tr>
<td>INP_KEY_CUR_R</td>
<td>0xff53</td>
<td>Cursor right</td>
</tr>
<tr>
<td>INP_KEY_CUR_D</td>
<td>0xff54</td>
<td>Cursor down</td>
</tr>
<tr>
<td>N/A</td>
<td>0xff55-0xff56</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_END</td>
<td>0xff57</td>
<td>End-of-line</td>
</tr>
<tr>
<td>INP_KEY_BEGIN</td>
<td>0xff58</td>
<td>Beginning-of-line</td>
</tr>
<tr>
<td>N/A</td>
<td>0xff59-0xff5f</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_SELECT</td>
<td>0xff60</td>
<td>Select</td>
</tr>
<tr>
<td>INP_KEY_PRINT</td>
<td>0xff61</td>
<td>Print</td>
</tr>
</tbody>
</table>
### Table 18-1 Key Symbols in INP_KEY_* (continued)

<table>
<thead>
<tr>
<th>Key Symbol Name</th>
<th>Key Symbol Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_KEY_EXECUTE</td>
<td>0xff62</td>
<td>Execute or run</td>
</tr>
<tr>
<td>INP_KEY_INSERT</td>
<td>0xff63</td>
<td>Insert</td>
</tr>
<tr>
<td>N/A</td>
<td>0xff64</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_UNDO</td>
<td>0xff65</td>
<td>Undo</td>
</tr>
<tr>
<td>INP_KEY_REDO</td>
<td>0xff66</td>
<td>Redo</td>
</tr>
<tr>
<td>INP_KEY_MENU</td>
<td>0xff67</td>
<td>Menu</td>
</tr>
<tr>
<td>INP_KEY_FIND</td>
<td>0xff68</td>
<td>Find</td>
</tr>
<tr>
<td>INP_KEY_CANCEL</td>
<td>0xff69</td>
<td>Cancel</td>
</tr>
<tr>
<td>INP_KEY_HELP</td>
<td>0xff6a</td>
<td>Help</td>
</tr>
<tr>
<td>INP_KEY_BREAK</td>
<td>0xff6b</td>
<td>Break</td>
</tr>
<tr>
<td>N/A</td>
<td>0xff6c-0xffbd</td>
<td>Reserved</td>
</tr>
<tr>
<td>INP_KEY_F1 to F35</td>
<td>0xffbe-0xffe0</td>
<td>Function keys F1-F35</td>
</tr>
<tr>
<td>INP_KEY_L1 to L10</td>
<td>0xffc8-0xffd1</td>
<td>Function keys L1-L10</td>
</tr>
<tr>
<td>INP_KEY_R1 to R15</td>
<td>0xffd2-0xffe0</td>
<td>Function keys R1-R15</td>
</tr>
<tr>
<td>N/A</td>
<td>0xffe1-0xfffff</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
This key symbol map is defined in such a way that each key symbol has a unique value. These values are independent of the device from which they are generated. For example, the letter “A” is always reported as key symbol 0x41, and the “PLAY” button is symbol 0xfe80 regardless of the type of input device.

For a discussion of how modifier keys effect the key symbols see the MSG_KEY data type.

**See Also**

None
**INP_KEY_SUBTYPE**
Key Symbol Message Subtype

**Syntax**

```c
typedef enum {
    INP_KEY_DOWN   = 1 /* Key down (pressed) */,
    INP_KEY_UP     = 2 /* Key up (released) */,
    INP_KEYMOD_DOWN= 4 /* Key modifier down (pressed) */,
    INP_KEYMOD_UP  = 8 /* Key modifier up (released) */
} INP_KEY_SUBTYPE;
```

**Description**

This enumerated type defines the subtype for key symbol messages. The values in this enumerated type are powers of two. Therefore, they may be combined safely.

- **INP_KEY_DOWN** indicates that a key was pressed.
- **INP_KEY_UP** indicates that a key was released.
- **INP_KEYMOD_DOWN** indicates that a key modifier was pressed.
- **INP_KEYMOD_UP** indicates that a key modifier was released.

**See Also**

MSG_KEY
INP_KEYMOD
Key Modifiers

Syntax

typedef enum {
  INP_KEY_NOMOD, /* No modifier keys */
  INP_KEY_SHIFT_L, /* Left shift key */
  INP_KEY_SHIFT_R, /* Right shift key */
  INP_KEY_CAPLCK_L, /* Left caps-lock key */
  INP_KEY_CAPLCK_R, /* Right caps-lock key */
  INP_KEY_CNTL_L, /* Left control key */
  INP_KEY_CNTL_R, /* Right control key */
  INP_KEY_ALT_L, /* Left alt key */
  INP_KEY_ALT_R, /* Right alt key */
  INP_KEY_META_L, /* Left meta key */
  INP_KEY_META_R, /* Right meta key */
  INP_KEY_NUMLCK, /* Num-lock key */
  INP_NUMPAD /* Recognize NUMPAD keys */
  INP_KEY_SCRLCK /* Scroll-lock key */
} INP_KEYMOD;

Description

This enumerated type defines the key symbol modifiers known by this API. These modifiers may effect the key symbol input you receive.

The values in this enumerated type are powers of two. Therefore, you may combine them safely. For example:

```
INP_KEY_SHIFT_L | INP_KEY_SHIFT_R
```

may be used to specify either the left or right shift key.

See Also

`inp_get_dev_cap()`
## Syntax

typedef enum {
    INP_KEYS_NONE, /* No keys */
    INP_KEYS_ASCII, /* ASCII character set */
    INP_KEYS_UCASE, /* Upper case alpha keys (A-Z) */
    INP_KEYS_LCASE, /* Lower case alpha keys (a-z) */
    INP_KEYS_NUM, /* Numeric keys (0-9) */
    INP_KEYS_4WAY, /* Four-way cursor movement */
    INP_KEYS_8WAY, /* Eight-way cursor movement */
    INP_KEYS_PLAYER, /* Player device keys */
    INP_KEYS_PHONE /* Telephone keypad */
} INP_KEYS;

## Description

This enumerated type is used to classify the key symbol groups defined by the Input Device API.

The values in this enumerated type are powers of two. Therefore, you may combine them safely. For example:

```
INP_KEYS_4WAY | INP_KEYS_NUM
```

may be used to specify an input device that has both 4-way direction keys and numeric keys.

The following table defines the set of key symbols that must be present for each of the groups specified in the syntax.

### Table 18-2 Key Symbols in INP_KEYS

<table>
<thead>
<tr>
<th>Key Symbol Group</th>
<th>Keys that Must be Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP_KEYS_ASCII</td>
<td>0x20 through 0x7e</td>
</tr>
<tr>
<td>INP_KEYS_UCASE</td>
<td>0x41 through 0x5a</td>
</tr>
<tr>
<td>INP_KEYS_LCASE</td>
<td>0x61 through 0x7a</td>
</tr>
</tbody>
</table>
Table 18-2  Key Symbols in INP_KEYS (continued)

<table>
<thead>
<tr>
<th>Key Symbol Group</th>
<th>Keys that Must be Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP KEYS_NUM</td>
<td>0x30 through 0x39</td>
</tr>
<tr>
<td>INP KEYS_4WAY</td>
<td>INP_KEY_CUR_L</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_CUR_U</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_CUR_R</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_CUR_D</td>
</tr>
<tr>
<td>INP KEYS_8WAY</td>
<td>INP_KEY_4WAY plus:</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_CUR_UL</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_CUR UR</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_CUR_DL</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_CUR_DR</td>
</tr>
<tr>
<td>INP KEYS_PLAYER</td>
<td>INP_KEY_PLAY</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_STOP</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_PAUSE</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_FASTFWD</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_REWIND</td>
</tr>
<tr>
<td>INP KEYS_PHONE</td>
<td>INP_KEY_TEL_1 through</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_TEL_9</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_TEL_STAR</td>
</tr>
<tr>
<td></td>
<td>INP_KEY_TEL_POUND</td>
</tr>
</tbody>
</table>

See Also

inp_get_dev_cap()
INP_KEY
INP_MAX_BUTTONS
Maximum Buttons for Pointer Device

Syntax
INP_MAX_BUTTONS

Description
This constant defines the maximum number of buttons that may be present on a pointer device. If simulation is used, this indicates the maximum number of virtual pointer buttons.

See Also
inp_set_sim_meth()
INP_MAX_DEV_NAME
Maximum Length of Device Name

Syntax

INP_MAX_DEV_NAME

Description
This constant defines the maximum length of a device name. This includes the NULL byte used to terminate the string.

See Also
inp_open_dev()
Syntax

```c
typedef union _INP_MSG {
    MSG_PTR ptr;    /* Pointer message */
    MSG_KEY key;    /* Key symbol message */
    MSG_COMMON any; /* Common for all messages */
} INP_MSG;
```

Description

This data structure defines a union of all input message structures. This currently includes pointer and key symbol messages. If these are the only types of messages being written to a mailbox, then the minimum mailbox message size is:

```
sizeof(INP_MSG)
```

See Also

- `msg_create_mbox()`
- `msg_open_mbox()`
- `msg_read()`
- `MSG_COMMON`
- `MSG_KEY`
- `MSG_PTR`
**INP_PTR_SUBTYPE**

**Pointer Message Subtype**

### Syntax

typedef enum {
    INP_PTR_DOWN /* Pointer button down */
    INP_PTR_UP /* Pointer button up */
    INP_PTR_MOVE /* Pointer move */
} INP_PTR_SUBTYPE;

### Description

This enumerated type defines the subtype for pointer messages. The values in this enumerated type are powers of two. Therefore, you may combine them safely.

- **INP_PTR_DOWN** Indicates that a pointer button was pressed.
- **INP_PTR_UP** Indicates that a pointer button was released.
- **INP_PTR_MOVE** Indicates that the pointer moved to a new X,Y position.

### See Also

MSG_PTR
INP_SIM_METH
Simulation Method

Syntax

typedef enum {
    INP_SIM_NATIVE, /* Use native mode */
    INP_SIM_PTR,   /* Simulate pointer input */
    INP_SIM_KEY    /* Simulate key symbol input */
} INP_SIM_METH;

Description

This enumerated type defines the type of input messages expected for a device. See inp_set_sim_meth() for information about how these values are used.

If you specify INP_SIM_NATIVE, then all messages are delivered in their native state, without translation. In this mode, messages from a pointer device are delivered as pointer messages, and those from a key symbol device are delivered as key symbol messages.

If you specify INP_SIM_PTR, then all cursor movement is reported as pointer messages regardless of whether they are generated by a pointer device or a key symbol device. All key symbols that are associated with pointer buttons are reported as a sequence of a pointer button down message followed by a pointer button up message. All other keys generate key symbol messages.

If you specify INP_SIM_KEY, then all cursor movement is reported as key symbol messages regardless of whether they are generated by a pointer device or a key symbol device. All pointer buttons that have been associated with key symbols are reported as key symbol messages. All other pointer buttons generate pointer button messages.

See Also

inp_set_sim_meth()
Chapter 19: MAUI System Data Types
**BOOLEAN**

Boolean Type

**Syntax**

typedef enum {
    FALSE = 0, /*False*/
    TRUE = !0, /*True*/
    OFF = 0,   /*Off*/
    ON = !0    /*On*/
} BOOLEAN;

**Description**

This enumerated type defines logical TRUE/FALSE and ON/OFF. This type is used throughout the MAUI APIs to represent the state of an object or the status of a flag.

The names used in this enumerated type are common enough that they may also be defined by other non-MAUI header files. If you would prefer that MAUI not define these, you should include the following in your source file before you include any MAUI header files:

```
#define _MAUI_DISABLE_BOOLEAN
```

You must properly define all the names in this enumerated type for MAUI to compile and function correctly.

**See Also**

None
**LSBFIRST**

**Little-Endian**

**Syntax**

`LSBFIRST`

**Description**

This constant is used to indicate that something is formatted as little-endian data. This is also referred to as least significant byte/bit first.

**See Also**

`MSBFIRST`
MAUI_COMPAT_LEVEL
Compatibility Level

Syntax
MAUI_COMPAT_LEVEL

Description
This constant defines the MAUI compatibility level. The initial value is 1, and it is increased by 1 each time the interface is changed in a way that breaks compatibility.

This constant is used internally by the components of MAUI to ensure they are compatible with each other, or to cope with each other when they are not compatible.

See Also
None
MAUI System Data Types

MAUI_ERR_LEVEL

Error Level

Syntax

typedef enum {
    MAUI_ERR_AS_IS,  /*Don’t change current value*/
    MAUI_ERR_ANY,    /*Any error */
    MAUI_ERR_NOTICE, /*Notice */
    MAUI_ERR_WARNING,  /*Warning */
    MAUI_ERR_NON_FATAL, /*Non-Fatal */
    MAUI_ERR_FATAL, /*Fatal */
    MAUI_ERR_NONE,  /*No error */
    MAUI_ERR_DEFAULT /*Restore default value */
} MAUI_ERR_LEVEL;

Description

This enumerated type defines an error level. These values are used when calling `maui_set_error_action()`.

See Also

`maui_set_error_action()`
Union of All MAUI Message Types

Syntax

typedef union _MAUI_MSG {
    MSG_PTR ptr; /* Pointer message */
    MSG_KEY key; /* Key symbol message */
    MSG_WIN_BORDER border; /* Border enter/leave message */
    MSG_WIN_BUTTON button; /* Button up/down message */
    MSG_WIN_CREATE create; /* Create child window message */
    MSG_WIN_DESTROY destroy; /* Destroy window message */
    MSG_WIN_EXPOSE expose; /* Expose message */
    MSG_WIN_FOCUS focus; /* Focus in/out message */
    MSG_WIN_KEY winkey; /* Key up/down message */
    MSG_WIN_MOVE move; /* Window move message */
    MSG_WIN_PTR winptr; /* Pointer move message */
    MSG_WIN_REPARENT reparent; /* Re-parent window message */
    MSG_WIN_RESIZE resize; /* Window resized message */
    MSG_WIN_RESTACK restack; /* Window restacked message */
    MSG_WIN_STATE state; /* Window state change message */
    MSG_WIN_INK_OFF inkoff; /* Inking turned off message */
    MSG_WIN_COMMON any_win; /* Common for all window messages */
    MSG_COMMON any; /* Common for all messages */
} MAUI_MSG;

Description

This data structure defines a union of all MAUI message types. This currently includes pointer and key symbol messages. If these are the only types of messages being written to a mailbox, then the minimum mailbox message size is:

    sizeof(MAUI_MSG)

Syntax

msg_create_mbox()
msg_open_mbox()
msg_read()
INP_MSG
MSG_COMMON
WIN_MSG
Syntax
MSBFIRST

Description
This constant indicates that something is formatted as big-endian data. This is also referred to as most-significant byte/bit first.

See Also
LSBFIRST
19 MAUI System Data Types
Chapter 20: Shaded Memory Data Types
MEM_DEF_SHADE

Default Shade

Syntax
MEM_DEF_SHADE

Description
This constant defines the shade ID for the default shade. This shade is automatically created when `mem_init()` is called.

See Also
`mem_init()`
Shaded Memory Data Types

MEM_GROW
Grow Method

Syntax

typedef enum {
   MEM_GROW_LARGER,  /* Use larger size */
   MEM_GROW_MULTIPLE /* Multiple of grow size */
} MEM_GROW;

Description

This enumerated type defines how shades grow when they need additional memory from the system.

If the grow method for a shade is set to MEM_GROW_LARGER, then the size of the block requested from the system is the larger of the grow size for the shade and the size being requested by the application.

If the grow method for a shade is set to MEM_GROW_MULTIPLE, the size of the block requested from the system is a multiple of the grow size for the shade.

See Also

mem_set_grow_method()
MEM_MIN_ALLOC
Minimum Allocation Size

Syntax
MEM_MIN_ALLOC

Description
This constant defines the minimum size of an allocation. Memory
segments allocated by this API are always multiples of
MEM_MIN_ALLOC. A larger boundary size may be set by calling
mem_set_alloc_bndry().

See Also
mem_calloc()
mem_malloc()
mem_realloc()
mem_set_alloc_bndry()
MEM_OVTYPE
Overhead Type

Syntax

typedef enum {
    MEM_OV_ATTACHED, /* Attached to the segment */
    MEM_OV_SEPARATE  /* Separate from the segments */
} MEM_OVTYPE;

Description

This enumerated type defines whether the overhead for the allocated memory segments are attached to the segment or kept in a separate area of memory.

MEM_OV_ATTACHED indicates that the overhead for each segment is attached to the segment.

MEM_OV_SEPARATE indicates that the overhead for each segment is in memory that is not attached to the segment. This is the method used for pseudo shades since the API assumes it is not possible to directly access pseudo memory.

See Also

mem_create_shade()
Syntax

typedef struct _MEM_SHADE_STATUS {
    MEM_SHADE_TYPE type; /* Shade type */
    MEM_OVTYPE ovtype; /* Overhead type */
    BOOLEAN overflow_detect; /* Overflow detection ON/OFF */
    u_int32 color; /* Color */
    size_t initial_size; /* Initial size */
    MEM_GROW grow_method; /* Grow method */
    size_t grow_size; /* Grow size */
    error_code (*alloc_func)(void *, size_t *, void **, u_int32);
    /* Function to allocate memory */
    void *alloc_data; /* Passed to alloc_func */
    error_code (*dealloc_func)(void *, size_t, void *, u_int32);
    /* Function to de-allocate mem */
    void *dealloc_data; /* Passed to dealloc_func*/
} MEM_SHADE_STATUS;

Description

This data structure is used by mem_get_shade_status() to return the status of a shade.

Syntax

mem_create_shade()
mem_get_shade_status()
BOOLEAN
MEM_GROW
MEM_OVTYPE
MEM_SHADE_TYPE
### Syntax

typedef enum {
    MEM_SHADE_PSEUDO,  /* Pseudo shade */
    MEM_SHADE_NORMAL   /* Normal shade */
} MEM_SHADE_TYPE;

### Description

This enumerated type defines the shade type. There are two types of shades: normal and pseudo.

Normal shades contain memory that the API assumes can be written to. Applications may also write directly to memory allocated from this type of shade.

Pseudo shades contain memory that the API assumes cannot be directly accessed by the CPU. Applications cannot write directly to this type of memory.

### See Also

mem_create_shade()
mem_get_shade_status()
20 Shaded Memory Data Types
Chapter 21: Messaging Data Types
MSG_BLOCK_TYPE
I/O Blocking Type

Syntax
typedef enum {
    MSG_BLOCK, /* Block until a msg is ready */
    MSG_NOBLOCK /* Do not block */
} MSG_BLOCK_TYPE;

Description
This enumerated type defines the blocking mechanism to use when reading messages from a mailbox.

If MSG_BLOCK is used, then the I/O function blocks until a message is available in the mailbox.

If MSG_NOBLOCK is used, then the I/O function returns immediately.

See Also
msg_peek()
msg_peekn()
msg_read()
msg_readn()
Syntax
typedef struct _MSG_COMMON {
    u_int32 type;         /* Message type */
    u_int32 time_queued;  /* Time the message was queued */
    process_id pid;       /* Process ID of writer */
    void (*callback)(const void *msg); /* Message callback */
} MSG_COMMON;

Description
This data structure defines the common header at the beginning of all messages. A message must have this header to be valid.

type defines the message type.

time_queued is the time that the message was queued (written to the mailbox). The time is represented as the number of ticks since the system was started. This value rolls over when it reaches its maximum value.

callback is a pointer to the callback function for the message. This is the function called by msg_dispatch().

See Also
msg_dispatch()}
MSG_MAX_MBOX_NAME
Maximum Length of Mailbox Name

Syntax
MSG_MAX_MBOX_NAME

Description
This constant defines the maximum length for the name of a mailbox. This includes the NULL byte used to terminate the string.

See Also
msg_create_mbox()
msg_get_mbox_status()
msg_open_mbox()
MSG_MBOX_STATUS
### Syntax

typedef struct _MSG_KEY {
    MSG_COMMON com; /* Common section */
    INP_KEY_SUBTYPE subtype; /* Type of key symbol message */
    INP_DEV_ID device_id; /* Device ID */
    wchar_t keysym; /* Key symbol */
    INP_KEYMOD key_modifiers; /* Key modifiers */
} MSG_KEY;

### Description

This data structure defines a key symbol message. com is the common header required for all messages and com.type is MSG_TYPE_KEY. device_id is the device that caused the message.

subtype indicates the type of key symbol message. INP_KEY_DOWN indicates that the key was pressed. INP_KEY_UP indicates that it was released. If both values are set, then a key-press (down then up) is indicated.

dkeysym specifies the key symbol and key_modifiers specifies the state of all key modifiers at the time the key input was generated.

Here are some examples of the types of messages returned when the device is a scan-code device.

Consider the following sequence of actions. The key “a” is pressed and then released. Two messages are issued, the first with a subtype of INP_KEY_DOWN and a keysym with a value of 0x61 for ‘a’. The second message would have a subtype of INP_KEY_UP and a keysym with a value of 0x61 for ‘a’.

Now, consider this sequence of actions. The left “SHIFT” key is pressed and then the key “a” is pressed and then the “a” key is released and finally the “SHIFT” key is released. Four messages are issued in the following order. The first message would have a subtype of INP_KEYMOD_DOWN with a keysym value of 0x0 and a key_modifiers value of INP_KEY_SHIFT_L. The second message
would have a subtype of INP_KEY_DOWN with a keysym value of 0x41 for ‘A’ and a key_modifiers value of INP_KEY_SHIFT_L. The third message would have a subtype of INP_KEY_UP with a keysym value of 0x41 for ‘A’ and a key_modifiers value of INP_KEY_SHIFT_L. Finally, The fourth message would have a subtype of INP_KEYMOD_UP with a keysym value of 0x0 and a key_modifiers value of INP_KEY_NOMOD.

Finally, consider this sequence of actions. The key “CTRL” is pressed and then the key “a” is pressed and then the “a” key is released and finally the “CTRL” key is released. Four messages are issued in the following order. The first message would have a subtype of INP_KEYMOD_DOWN with a keysym value of 0x0 and a key_modifiers value of INP_KEY_CNTL_L. The second message would have a subtype of INP_KEY.DOWN with a keysym value of 0x1 and a key_modifiers value of INP_KEY_CNTL_L. The third message would have a subtype of INP_KEY_UP with a keysym value of 0x1 and a key_modifiers value of INP_KEY_CNTL_L. Finally, The fourth message would have a subtype of INP_KEYMOD_UP with a keysym value of 0x0 and a key_modifiers value of INP_KEY_NOMOD.

On single stroke devices only the keysym is filled in. For example, on a dumb terminal if the letter “A” is pressed then the value of keysym is 0x41 and the subtype and key_modifiers are not changed. On a remote control device if the “PLAY” button is pressed then the value of keysym is 0xfe80.

See Also
INP_DEV_ID
INP_KEY_SUBTYPE
INP_KEYMOD
MSG_COMMON
MSG_TYPE
**MSG_MBOX_ID**

**Mailbox ID**

### Syntax

typedef void * MSG_MBOX_ID;

### Description

This data type defines a mailbox ID. This ID is returned by `msg_create_mbox()` and is used in subsequent calls to functions that require a mailbox identifier.

### See Also

`msg_create_mbox()`
MSG_MBOX_STATUS
Mailbox Status

Syntax

typedef struct _MSG_MBOX_STATUS {
    char name[MSG_MAX_MBOX_NAME]; /* Mailbox name */
    u_int32 num_entries; /* Maximum entries for mailbox */
    u_int32 free_entries; /* Free entries in the mailbox */
    size_t entry_size; /* Size of each entry in bytes */
    u_int16 link_count; /* Number of links to the mbox */
    u_int32 write_mask; /* Mask for writing to mbox */
    error_code (*filter)(BOOLEAN *, const void *, void *); /* Filter function */
    void *filter_data; /* Appl data for (*filter)() */
} MSG_MBOX_STATUS;

Description

This data structure is used by msg_get_mbox_status() to return the current status of a mailbox. The maximum length of the mailbox name is defined by MSG_MAX_MBOX_NAME.

name is the name specified when the mailbox is created with msg_create_mbox().

link_count is the current number of users of the mailbox. After the initial msg_create_mbox(), the link count is 1. Each subsequent msg_open_mbox() increments the link count by 1. Each msg_close_mbox() decrements the count by 1.

num_entries is the maximum number of message entries in the queue and free_entries is the current number of free slots in the queue.

entry_size is the size of each entry.

write_mask prevents unwanted messages from being written to the mailbox. See msg_set_mask().

filter is a pointer to a callback function used to filter messages.

filter_data is passed to the callback function whenever it is called. See msg_set_filter().
See Also

msg_close_mbox()
msg_create_mbox()
msg_get_mbox_status()
msg_open_mbox()
msg_set_filter()
msg_set_mask()
BOOLEAN
MSG_MAX_MBOX_NAME
MSG_PLACEMENT
Message Placement

Syntax

typedef enum {
    MSG_AT_HEAD,  /* Place at head of the queue */
    MSG_AT_TAIL   /* Place at tail of the queue */
} MSG_PLACEMENT;

Description

This enumerated type defines how a message is inserted into a mailbox.

MSG_AT_HEAD specifies that the message is inserted at the head of the queue and is the first one read by `msg_read()`.

MSG_AT_TAIL (typical) specifies that the message is inserted at the tail of the queue and is read after all other messages.

See Also

`msg_read()`
`msg_readn()`
`msg_unread()`
`msg_unreadn()`
`msg_write()`
`msg_writen()`
**Syntax**

typedef struct _MSG_PTR {
    MSG_COMMON com;            /* Common section */
    INP_PTR_SUBTYPE subtype;   /* Type of pointer message */
    INP_DEV_ID device_id;      /* Device ID */
    u_int8 button;             /* Button that changed (1-8) */
    u_int8 button_state;       /* State of all ptr buttons */
    GFX_POINT position;        /* New X,Y position */
    wchar_t keysym;            /* Original key symbol */
} MSG_PTR;

**Description**

This data structure defines a pointer message. `com` is the common header and `com.type` is MSG_TYPE_PTR.

device_id is the device that caused the message.

If this is a button message (subtype is INP_PTR_DOWN or INP_PTR_UP), then `button` indicates which button changed state. If more than one button changed state, then you receive a button message for each change.

`button_state` indicates the state of all buttons at the time the message was sent. The state includes the latest change indicated by `button`.

`position` gives the X and Y position of the pointer at the time the message was sent.

If simulation caused this pointer message, then `keysym` is the key symbol of the key that caused the pointer message to be generated. Otherwise it is 0.

Pointer movement messages are always reported separately from button up or down messages.
See Also
GFX_POINT
INP_DEV_ID
INP_PTR_SUBTYPE
MSG_COMMON
MSG_TYPE
Syntax
MSG_TYPE

Description
MSG_TYPE is a prefix used to define a set of constants. Constants that start with MSG_TYPE are used to define the message types registered with this API. Following is a table of message types currently registered.

Table 21-1 Message Types

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Bit(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TYPE_NONE</td>
<td>No bits set</td>
<td>No message</td>
</tr>
<tr>
<td>MSG_TYPE_PTR</td>
<td>Bit 0 set</td>
<td>Pointer message</td>
</tr>
<tr>
<td>MSG_TYPE_KEY</td>
<td>Bit 1 set</td>
<td>Key symbol message</td>
</tr>
<tr>
<td>Reserved for MAUI use</td>
<td>Bits 2 - 23</td>
<td>Reserved</td>
</tr>
<tr>
<td>Reserved for application use</td>
<td>Bits 24 - 31</td>
<td>Applications may use</td>
</tr>
<tr>
<td>MSG_TYPE_ANY</td>
<td>Bits 0 - 31 set</td>
<td>Any message</td>
</tr>
</tbody>
</table>

Each message type is represented by a bit in an unsigned 32-bit integer. Therefore, a maximum of 32 message types may be defined. Messages may be combined by using the logical OR operation. For example, the following expressions may be used to create a mask for pointer and key symbol messages.

\[
\text{u_int32 mask} = \text{MSG_TYPE_PTR} \mid \text{MSG_TYPE_KEY}
\]
See Also

msg_flush()
msg.peek()
msg.peekn()
msg_read()
msg_readn()
msg_send_sig()
msg_set_mask()
Chapter 22: Text Data Types
Text Context ID

Syntax

typedef void * TXT_CONTEXT_ID;

Description

This data type defines a text context ID. This ID is returned by `txt_create_context()` and is used in subsequent calls to functions that require a text context identifier.

See Also

`txt_create_context()`
Syntax

typedef struct _TXT_CONTEXT_PARAMS {
    GFX_DEV_ID gfxdev;       /* Graphics device ID */
    const TXT_FONT *font;    /* Current font */
    int16 cpad;              /* Character padding */
    BLT_MIX mixmode;         /* Mixing mode */
    u_int8 exptbl_entries;   /* Number of entries in exptbl */
    const GFX_PIXEL *exptbl; /* Pixel expansion table */
    GFX_PIXEL transpixel;   /* Transparent pixel value */
    GFX_PIXEL ofspixel;     /* Offset pixel value */
    const GFX_DMAP *dstdmap; /* Destination drawmap */
    GFX_POINT origin;       /* Drawing origin */
    GFX_RECT draw_area;     /* Drawing area */
    u_int32 num_clip_areas; /* Number of clipping areas */
    const GFX_RECT *clip_areas; /* Array of clipping areas */
} TXT_CONTEXT_PARAMS;

Description

This data structure is used by `txt_get_context()` to return the current values in a text context object.

See Also

txt_get_context()
BLT_MIX
GFX_DEV_ID
GFX_DMAP
GFX_PIXEL
GFX_POINT
GFX_RECT
TXT_FONT
Syntax

typedef struct _TXT_FONT {
    TXT_FONTYPE font_type; /* Font type */
    GFX_DIMEN width;    /* Maximum cell width */
    GFX_DIMEN height;   /* Cell height */
    GFX_DIMEN ascent;   /* Ascent above baseline */
    GFX_DIMEN descent;  /* Descent below baseline */
    wchar_t default_char; /* Default character */
    u_int8 num_ranges;   /* Number of ranges */
    TXT_RANGTBL *range_tbl; /* Pointer to an array of */
    /* range tables; "num_ranges" long */
} TXT_FONT;

Description

The font object is the object that defines all the glyphs within a font. Font objects may be created by calling `txt_create_font()` function, or they may be created directly by the application—such as using initialized data. The structure of a font object is shown below.

The `font_type` is `TXT_FONTYPE_MONO` for mono-spaced fonts and `TXT_FONTYPE_PROP` for proportional spaced fonts.

The `ascent` and `descent` are the number of pixels in the character cell above and below the baseline, respectively. The baseline is an imaginary line between the pixels that make up the ascent and the descent. Therefore the `height` is always equal to the `ascent` plus the `descent`.

The `default_char` specifies the character value to use for characters that are not present in the font.

The `range_tbl` points to an array of glyph range definition structures. The number of entries in this array is specified by `num_ranges`. Each entry in the array defines a specific glyph range for the font. These ranges should not overlap, but if they do, glyphs are selected from the first range that covers that character code.
Text Data Types

See Also

txt_create_font()
GFX_DIMEN
GFX_OFFSET
TXT_FONTYPE
TXT_RAGTBL
Syntax
typedef enum {
    TXT_FONTYPE_MONO, /* Mono-spaced font */
    TXT_FONTYPE_PROP  /* Proportional spaced font */
} TXT_FONTYPE;

Description
This enumerated type defines the font type. The type may be mono-spaced or proportional.

See Also
TXT_FONT
**Syntax**

TXT_NOGLYPH

**Description**

This constant marks an entry in the offset table as unused. This indicates that there is no corresponding glyph defined for the font.

**See Also**

TXT_RANGTBL
Syntax

typedef struct _TXT_RANGTBL {
    wchar_t first_char; /* First character in font */
    wchar_t last_char; /* Last character in font */
    GFX_OFFSET *offset_tbl; /* Pointer to offset table */
    GFX_DIMEN *width_tbl; /* Pointer to width table */
    GFX_DMAP *bitmap;    /* Pointer to bitmap */
} TXT_RANGTBL;

Description

This data structure defines a range of glyphs for a font object.

The offset_tbl is a pointer to the glyph offset table. This table is used to determine the horizontal position where each glyph starts in bitmap. The value is in pixels and there is one entry for each character in the range first_char to last_char inclusive. The value TXT_NOGLYPH should be used for characters that have no glyph defined.

The width_tbl is a pointer to the glyph width table. This table is used to determine the width of each character. It is ignored for mono-spaced fonts (should be NULL), but must be present for proportional spaced fonts. If the table is present, it must contain one entry for each character in the range first_char to last_char inclusive.

See Also

txt_create_font()
GFX_DIMEN
GFX_DMAP
GFX_OFFSET
TXT_FONT
TXT_NOGLYPH
Chapter 23: Windowing Data Types
MSG_WIN_BORDER
Border Enter/Leave Message

Syntax

typedef struct _MSG_WIN_BORDER {
    MSG_COMMON com;    /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev;  /* Windowing device ID */
    WIN_ID win;         /* Window message is for */
    GFX_POINT root_pos; /* Border position in root */
    GFX_POINT position; /* Border position in parent */
} MSG_WIN_BORDER;

Description

This data structure defines a border message. This message is sent by
maui_win when the pointer crosses the boundary between two
windows.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with
win_set_callback(). windev is the windowing device that
generated the message.

If wtype is WIN_MSG_BORDER_ENTER then win is the window that the
pointer moved into. In this case, root_pos and position reflect the
first pointer position in the window being entered.

If wtype is WIN_MSG_BORDER_LEAVE then win is the window that the
pointer moved out of. In this case, root_pos and position reflect the
last pointer position in the window being left.

See Also

win_destroy_win()
win_move_win()
win_resize_win()
win_restack_win()
win_set_callback()
win_set_state()
Windowing Data Types

- GFX_POINT
- MSG_COMMON
- MSG_TYPE
- WIN_DEV_ID
- WIN_ID
- WIN_MSG_TYPE
Syntax

typedef struct _MSG_WIN_BUTTON {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for */
    u_int8 button; /* Button that changed (1-8) */
    u_int8 button_state; /* State of all ptr buttons */
    GFX_POINT root_pos; /* Button position in root */
    GFX_POINT position; /* Button position in parent */
    wchar_t keysym; /* Original key symbol */
} MSG_WIN_BUTTON;

Description

This data structure defines a button message. This message is sent by maui_win when a pointer button is pressed or released in the specified win.

If the message mask for win does not include this message type, then the message propagates up to the first parent whose mask includes this type. If no parent includes this type, then the message is not queued by maui_win.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with win_set_callback().
windev is the windowing device that generated the message.

The window message type is specified by wtype and is either WIN_MSG_BUTTON_DOWN or WIN_MSG_BUTTON_UP. While the button is de-pressed, maui_win sends all pointer messages to this same window win. This grabbing of pointer input ceases when the pointer button is released.

button_state indicates the state of all buttons at the time the message was sent. The state includes the latest change indicated by button.
root_pos and position gives the position of the pointer at the time the message was sent.

If simulation caused this pointer message, then keysym is the key symbol of the key that caused the pointer message to be generated. Otherwise it is 0.

See Also
win_set_callback()
GFX_POINT
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
**MSG_WIN_COMMON**

Common Part of Window Messages

**Syntax**

typedef struct _MSG_WIN_COMMON {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for */
} MSG_WIN_COMMON;

**Description**

This data structure defines the common entries in all window messages. This is used in cases where you do not yet know the message type, but you know it is a windowing message.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with win_set_callback(). windev is the windowing device that generated the message.

**See Also**

win_set_callback()
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
Windowing Data Types

**MSG_WIN_CREATE**

Window Created Message

**Syntax**

```c
typedef struct _MSG_WIN_CREATE {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for */
    WIN_ID create_win; /* Window created */
    GFX_POINT position; /*Position in parent*/
    GFX_DIMEN width; /*Width of window*/
    GFX_DIMEN height; /*Height of window*/
} MSG_WIN_CREATE;
```

**Description**

This data structure defines a window create message. This message is sent by `maui_win` to give notice that a new child window has been created on the window `win`.

- `com.type` is the message type and is always `MSG_TYPE_WIN`.
- `com.callback` is the callback function set with `win_set_callback()`.
- `windev` is the windowing device that generated the message.

The window that was created is specified by `create_win` and the parent window is `win`. The window message type is specified by `wtype` and is always `WIN_MSG_CREATE`.

The position of the upper-left corner of the window is `position` and its size is specified by `width` and `height`.

**Syntax**

```c
win_create_win()
win_set_callback()
```
GFX_DIMEN
GFX_POINT
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
**Windowing Data Types**

**MSG_WIN_DESTROY**

Window Destroyed Message

**Syntax**

typedef struct _MSG_WIN_DESTROY {
    MSG_COMMON com;    /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev;  /* Windowing device ID */
    WIN_ID win;         /* Window message is for */
    WIN_ID destroy_win; /* Window destroyed */
} MSG_WIN_DESTROY;

**Description**

This data structure defines a window destroy message. This message is sent by maui_win to give notice that the window destroy_win has been destroyed.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with

win_set_callback().windev is the windowing device that generated the message.

The window that was destroyed is specified by destroy_win. The window message type is specified by wtype and is always

WIN_MSG_DESTROY.

**Syntax**

win_destroy_win()
win_set_callback()
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG>Type
**MSG_WIN_EXPOSE**

**Expose Message**

**Syntax**

typedef struct _MSG_WIN_EXPOSE {
    MSG_COMMON com;       /* Common section */
    WIN_MSG_TYPE wtype;   /* Window message type */
    WIN_DEV_ID windev;    /* Windowing device ID */
    WIN_ID win;           /* Window message is for */
    GFX_RECT exposed;     /* Exposed rectangle in parent */
} MSG_WIN_EXPOSE;

**Description**

Defines an expose message. The message is sent by maui_win when part or all of a window that was previously hidden is exposed.

- **com.type** is the message type and is always MSG_TYPE_WIN.
- **com.callback** is the callback function set with **win_set_callback()**.
- **windev** is the windowing device that generated the message.
- The window being exposed is specified by **win**. The window message type is specified by **wtype** and is always WIN_MSG_EXPOSE.
- **exposed** is the rectangular area of the window that was exposed.

**See Also**

- win_destroy_win()
- win_move_win()
- win_resize_win()
- win_restack_win()
- win_set_callback()
- win_set_state()
- GFX_RECT
- MSG_COMMON
- MSG_TYPE
- WIN_DEV_ID
- WIN_ID
- WIN_MSG_TYPE
Windowing Data Types

MSG_WIN_FOCUS
Focus In/Out Message

Syntax
typedef struct _MSG_WIN_FOCUS {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for */
} MSG_WIN_FOCUS;

Description
This data structure defines a keyboard focus message. This message is sent by maui_win when the keyboard focus moves out of or into a window.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with win_set_callback().
windev is the windowing device that generated the message.

If wtype is WIN_MSG_FOCUS_IN then win is window that is receiving the keyboard focus.

If wtype is WIN_MSG_FOCUS_OUT then win is window that is losing the keyboard focus.

See Also
win_set_callback()
win_set_focus()
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
MSG_WIN_INK_OFF
Inking Disabled Message

Syntax

typedef struct _MSG_WIN_INK_OFF {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for */
} MSG_WIN_INK_OFF;

Description

This data structure defines an inking disabled message. This message is sent by `maui_win` when it attempts to ink but is not able to because the inking window is obscured by another window.

- `com.type` is the message type and is always MSG_TYPE_WIN.
- `com.callback` is the callback function set with `win_set_callback()`.
- `windev` is the windowing device that generated the message.
- The window message type is specified by `wtype` and is always WIN_MSG_INK_OFF.
- `win` is the window that attempted to draw ink but could not because it was obscured.

See Also

- `win_set_callback()`
- `win_set_ink_method()`
- MSG_COMMON
- WIN_DEV_ID
- WIN_ID
- WIN_MSG_TYPE
Windowing Data Types

MSG_WIN_KEY
Key Down/Up Message

Syntax

typedef struct _MSG_WIN_KEY {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for*/
    wchar_t keysym; /* Key symbol */
    INP_KEYMOD key_modifiers; /* key modifiers */
} MSG_WIN_KEY;

Description

This data structure defines a key symbol message. This message is sent by maui_win when a key on the keyboard (or other key symbol device such as a remote control) is pressed or released.

This message is sent to the window win that currently has the keyboard focus. If the message mask for win does not include this message type, then the message propagates up to the first parent whose mask includes this type. If no parent includes this type, then the message is not queued by maui_win.

com.type is the message type and is always MSG_TYPE_WIN. com.callback is the callback function set with win_set_callback(). windev is the windowing device that generated the message.

The window message type is specified by wtype and is either WIN_MSG_KEY_DOWN or WIN_MSG_KEY_UP. The down and up are never combined in the same message; you will always get two separate messages.

keysym specifies the key symbol and key_modifiers specifies the state of all key modifiers at the time the key input was generated.
See Also

win_set_callback()
win_set_focus()
INP_KEYMOD
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
Windowing Data Types

MSG_WIN_MOVE
Window Move Message

Syntax

typedef struct _MSG_WIN_MOVE {
    MSG_COMMON com;       /* Common section */
    WIN_MSG_TYPE wtype;   /* Window message type */
    WIN_DEV_ID windev;    /* Windowing device ID */
    WIN_ID win;           /* Window message is for */
    WIN_ID move_win;      /* Window moved or requested to be moved */
    GFX_POINT position;  /* New position in parent */
} MSG_WIN_MOVE;

Description

This data structure defines a window move message. This message is sent by maui_win to give notice that the window move_win has been moved, or to request that win be moved.

com.type is the message type and is always MSG_TYPE_WIN. com.callback is the callback function set with win_set_callback(). windev is the windowing device that generated the message.

The window message type is specified by wtype. If wtype is WIN_MSG_MOVE then the message is notification that the window moved. If wtype is WIN_MSG_MOVE_REQ then the message is a request that the window be moved.

position is the new position of the window.

See Also

win_move_win()
win_set_callback()
GFX_POINT
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
MSG_WIN_PTR
Pointer Move Message

Syntax

typedef struct _MSG_WIN_PTR {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for */
    GFX_POINT root_pos; /* New position in root */
    GFX_POINT position; /* New position in parent */
    wchar_t keysym; /* Original key symbol */
} MSG_WIN_PTR;

Description

This data structure defines a pointer move message. This message is sent by maui_win when the pointer is moved into, or within win.

If the message mask for win does not include this message type, then the message propagates up to the first parent whose mask includes this type. If no parent includes this type, then the message is not queued by maui_win.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with win_set_callback(). windev is the windowing device that generated the message.

The window message type is specified by wtype and is always WIN_MSG_PTR.

root_pos and position give the new position of the pointer at the time the message was sent.

If simulation caused this pointer message, then keysym is the key symbol of the key that caused the pointer message to be generated. Otherwise it is 0.
See Also

win_set_callback()
GFX_POINT
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
typedef struct __MSG_WIN_REPARENT {
    MSG_COMMON com;  /* Common section */
    WIN_MSG_TYPE wtype;  /* Window message type */
    WIN_DEV_ID windev;  /* Windowing device ID */
    WIN_ID win;  /* Window message is for */
    WIN_ID new_parent;  /* New parent window */
    WIN_PLACEMENT placement;  /* New placement */
    WIN_ID ref_win;  /* Reference window for new placement */
    GFX_POINT position;  /* New position in parent */
} MSG_WIN_REPARENT;

Description

This data structure defines a window re-parent message. This message is sent by maui_win to give notice that the window win has been re-parented.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with win_set_callback(). windev is the windowing device that generated the message.

The window message type is specified by wtype and is always WIN_MSG_REPARENT.

new_parent is the new parent window. placement specifies the new placement in the stack of siblings and ref_win is the reference sibling for this placement. position is the new position of the upper-left corner of win within the new parent.
See Also

- `win_reparent_win()`
- `win_set_callback()`
- `MSG_COMMON`
- `MSG_TYPE`
- `WIN_DEV_ID`
- `WIN_ID`
- `WIN_MSG_TYPE`
Window Re-sized Message

Syntax

typedef struct _MSG_WIN_RESIZE {
    MSG_COMMON com; /* Common section */
    WIN_MSG_TYPE wtype; /* Window message type */
    WIN_DEV_ID windev; /* Windowing device ID */
    WIN_ID win; /* Window message is for */
    WIN_ID resize_win; /* Window resized or */
                     /* requested to be resized */
    GFX_DIMEN width; /* New width */
    GFX_DIMEN height; /* New height */
} MSG_WIN_RESIZE;

Description

This data structure defines a window resize message. This message is sent by maui_win to give notice that the window resize_win has been resized, or to request that resize_win be resized.

com.type is the message type and is always MSG_TYPE_WIN.
com.callback is the callback function set with win_set_callback(). windev is the windowing device that generated the message.

The window message type is specified by wtype. If wtype is WIN_MSG_RESIZE then the message is notification that the window size has changed. If wtype is WIN_MSG_RESIZE_REQ then the message is a request that the window size be changed.

width and height specify the new size of the window.

See Also

win_resize_win()
win_set_callback()
GFX_DIMEN
MSG_COMMON
MSG_TYPE
Windowing Data Types

WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
MSG_WIN_RESTACK
Window Re-stacked Message

Syntax

typedef struct _MSG_WIN_RESTACK {
    MSG_COMMON com;        /* Common section */
    WIN_MSG_TYPE wtype;    /* Window message type */
    WIN_DEV_ID windev;     /* Windowing device ID */
    WIN_ID win;            /* Window message is for */
    WIN_ID restack_win;    /* Window re-stacked or */
                              /* requested to be re-stacked */
    WIN_PLACEMENT placement; /* New placement */
    WIN_ID ref_win;        /* Reference window for new placement */
} MSG_WIN_RESTACK;

Description

This data structure defines a window re-stack message. This message is sent by maui_win to give notice that the window restack_win has been re-stacked, or to request that restack_win be re-stacked.

com.type is the message type and is always MSG_TYPE_WIN. com.callback is the callback function set with win_set_callback(). windev is the windowing device that generated the message.

The window message type is specified by wtype. If wtype is WIN_MSG_RESTACK then the message is notification that the window has been re-stacked. If wtype is WIN_MSG_RESTACK_REQ then the message is a request that the window be re-stacked.

placement specifies the new placement in the stack of siblings and ref_win is the reference sibling for this placement. position is the new position of the upper-left corner of win within the new parent.
See Also
win_restack_win()
win_set_callback()
MSG_COMMON
MSG_TYPE
WIN_DEV_ID
WIN_ID
WIN_MSG_TYPE
**Syntax**

```c
typedef struct _MSG_WIN_STATE {
    MSG_COMMON com;       /* Common section */
    WIN_MSG_TYPE wtype;   /* Window message type */
    WIN_DEV_ID windev;    /* Windowing device ID */
    WIN_ID win;           /* Window message is for */
    WIN_ID state_win;     /* Window whose state was changed */
    BOOLEAN new_state;    /* New state */
} MSG_WIN_STATE;
```

**Description**

This data structure defines a window state change message. This message is sent by `maui_win` to give notice that the window `state_win` state has changed, or to request that the state of `state_win` be changed.

- `com.type` is the message type and is always `MSG_TYPE_WIN`.
- `com.callback` is the callback function set with `win_set_callback()`.
- `windev` is the windowing device that generated the message.

The window message type is specified by `wtype`. If `wtype` is `WIN_MSG_STATE` then the message is notification that the window state has changed. If `wtype` is `WIN_MSG_STATE_REQ` then the message is a request that the window state be changed.

- `new_state` specifies the new state of the window. If set to `TRUE`, then the window is now active. If `FALSE`, then the window is now inactive.
See Also

- `win_set_callback()`
- `win_set_state()`
- `BOOLEAN`  
- `MSG_COMMON`  
- `MSG_TYPE`  
- `WIN_DEV_ID`  
- `WIN_ID`  
- `WIN_MSG_TYPE`
WIN_CALLBACK
Callback Function

Syntax
typedef void (*WIN_CALLBACK); (const WIN_MSG *msg,
void *user_data)

Description
This data type defines a callback function. Callback functions are
functions in the application code that are called by msg_dispatch() when messages are dispatched. See win_set_callback() for
information on setting the callback function for a window.

See Also
msg_dispatch()
win_set_callback()
WIN_CELL_PARAMS
Colormap cell parameters

Syntax
typedef struct _WIN_CELL_PARAMS
{
    BOOLEAN allocated;         /* TRUE if a cell is allocated */
    BOOLEAN shareable;         /* TRUE if a cell is sharable */
    u_int32 link_count;         /* Number of allocations made */
                               /* for this cell; always 1 for */
                               /* private cells */
    GFX_COLOR_VALUE color_value;  /* Color value assigned to */
                                  /* the cell */
} WIN_CELL_PARAMS;

Description
This data structure defines returnable parameters for a colormap cell.
allocated indicates whether a cell was allocated.
shareable indicates that a cell is accessible by non-owner processes,
unlike a private cell.
link_count is the number of allocations done on this sharable cell.
Link count for private cells is always 1.
color_value is the color value assigned to this cell.

See Also
win_get_cells_params()
GFX_COLOR_VALUE
WIN_CMAP_ID
Colormap ID

Syntax
typedef void * WIN_CMAP_ID;

Description
This data type defines a colormap ID. This ID is returned by
win_create_cmap() and is used in subsequent calls to functions that
require a colormap identifier.

See Also
win_alloc_cmap_cells()
win_alloc_cmap_color()
win_create_cmap()
win_free_cmap_cells()
win_destroy_cmap()
win_get_cmap_cells()
win_get_cmap_free()
win_set_cmap()
win_set_cmap_cells()
Windowing Data Types

WIN_CMAP_Params
Colormap parameters

Syntax

typedef struct _WIN_CMAP_PARAMS
{
    u_int32 num_colors;        /* number of colors in colormap */
    WIN_CMATCH colormatch_method; /* Method currently used to */
        /* match colors on color */
        /* allocation */
    GFX_COLOR_TYPE color_type;   /* Colormap color type */
} WIN_CMAP_PARAMS;

Description

This data structure defines returnable parameters for a colormap.
num_colors indicates the number of colors available in the colormap.
colormatch_method is the current method used to match colors on
color allocation. color_type is the colormap color type (such as
GFX_RGB, etc).

See Also

win_get_cmap_params()

WIN_CMATCH

GFX_COLOR_TYPE
WIN_CMATCH
Color match method

Syntax

typedef enum
{
    WIN_CMATCH_NONE, /* always allocate a new cell */
    WIN_CMATCH_CLOSEST, /* never allocate a new cell */
    WIN_CMATCH_CLOSE, /* allocate cell only if no */
        /* match is close enough*/
    WIN_CMATCH_EXACT /* allocate cell only if no */
        /* exact match*/
} WIN_CMATCH;

Description

This enumerator defines implemented color matching methods. The methods are used to determine whether the existing color should be reused (its link count incremented) or a new cell should be allocated.

WIN_CMATCH_NONE provides no color matching attempts to be made. A new cell is always allocated.

WIN_CMATCH_CLOSEST traverses existing colors and picks one which is closest to the specified color. Allocation is made only when the first colormap cell is allocated.

WIN_CMATCH_CLOSE looks for the close-enough color to reuse. If such is not found, a new cell is allocated.

WIN_CMATCH_EXACT looks for the exact match. If not found, a new cell is allocated.

See Also

win_alloc_cmap_color()
win_alloc_cmap_colors()
WIN_CURSOR
Graphics Cursor

Syntax

typedef struct _WIN_CURSOR {
    GFX_POINT hit_point;    /* Hit point */
    GFX_DMAP *bitmap;       /* Cursor bitmap */
    GFX_DMAP *mask;         /* Source mask for Bit-BLT */
    GFX_PIXEL trans_pixel;  /* Transparent pixel value */
} WIN_CURSOR;

Description

This data structure defines a graphics cursor. It is set up by the application and passed to win_set_cursor() to set the graphics cursor for a window.

The hit_point defines the hit point for the cursor. This is the position within the cursor that corresponds to the pointer position on the windowing device.

bitmap defines the bitmap for the cursor. mask defines the source mask used by blt_copy_block() when the cursor is drawn. If it is NULL, then the transparent pixel value trans_pixel is used instead of a source mask.

See Also

win_set_cursor()
GFX_DMAP
GFX_PIXEL
GFX_POINT
WIN_DEV_ID

Windowing Device ID

Syntax

typedef void * WIN_DEV_ID;

Description

This data type defines a windowing device ID. This ID is returned by
\texttt{win\_create\_dev()} and \texttt{win\_open\_dev()} and is used in
subsequent calls to functions that require a device identifier.

See Also

\texttt{win\_close\_dev()}
\texttt{win\_close\_inpdev()}
\texttt{win\_create\_dev()}
\texttt{win\_destroy\_dev()}
\texttt{win\_get\_dev\_status()}
\texttt{win\_open\_dev()}
\texttt{win\_open\_inpdev()}
Windowing Data Types

WIN_DEV_STATUS

Windowing Device Status

Syntax

typedef struct _WIN_DEV_STATUS {
    GFX_DEV_ID gfxdev;  /* Graphics device ID */
    WIN_ID root_win;    /* Root window */
    GFX_DIMEN width;    /* Width of root window */
    GFX_DIMEN height;   /* Height of root window */
    GFX_CM coding_method; /* Coding method of device */
    WIN_ID focus_win;   /* Window with keyboard focus */
    process_id owner_pid; /* Process ID of owner */
} WIN_DEV_STATUS;

Description

This data structure is used by win_get_dev_status() to return the status of a windowing device.
The graphics device ID is gfxdev. This ID is cloned from the original ID owned by maui_win using gfx_clone_dev when the windowing device is created or opened.
The root window for the device is root_win and its dimensions are width and height.
The coding method being used by the root window is coding_method.
The window that currently has the keyboard focus is focus_win.
The owner of the windowing device is owner_pid. This is the process that originally called win_create_dev() to create the device.

See Also

gfx_clone_dev()
win_create_dev()
win_get_dev_status()
GFX_CM
GFX_DEV_ID
GFX_DIMEN
WIN_ID
WIN_ID
Window ID

Syntax

typedef void * WIN_ID;

Description

This data type defines a window ID. This ID is returned by
win_create_win() and is used in subsequent calls to functions that
require a window identifier.

See Also

win_close_dev()
win_create_dev()
win_create_win()
win_destroy_win()
Window Inking Method

Syntax

typedef enum {
    WIN_INK_OFF, /* No inking method */
    WIN_INK_REPLACE /* Replace method */
} WIN_INK_METHOD;

Description

This enumerated type defines the inking method for a window.

If method is WIN_INK_OFF, then no inking is performed in the window.
If method is WIN_INK_REPLACE, then ink applied to this window will replace any contents in this window.

See Also

win_erase_ink()
win_set_ink_method()
win_set_ink_pix()
WIN_MAX_DEV_NAME
Maximum Length of Device Name

Syntax
WIN_MAX_DEV_NAME

Description
This constant defines the maximum length of a device name. This includes the NULL byte used to terminate the string.

See Also
win_create_dev()
win_open_dev()
Windowing Data Types

WIN_Msg
Union of All Window Message Structures

Syntax

typedef union _WIN_MSG {
    MSG_WIN_BORDER border; /* Border enter/leave message */
    MSG_WIN_BUTTON button; /* Button down/up message */
    MSG_WIN_CREATE create; /* Create child window message */
    MSG_WIN_DESTROY destroy; /* Destroy window message */
    MSG_WIN_EXPOSE expose; /* Expose message */
    MSG_WIN_FOCUS focus; /* Focus in/out message */
    MSG_WIN_KEY winkey; /* Key down/up message */
    MSG_WIN_MOVE move; /* Window move message */
    MSG_WIN_PTR winptr; /* Pointer move message */
    MSG_WIN_REPARENT reparent; /* Re-parent window message */
    MSG_WIN_RESIZE resize; /* Window re-sized message */
    MSG_WIN_RESTACK restack; /* Window Re-stacked message */
    MSG_WIN_STATE state; /* Window state change message */
    MSG_WIN_INK_OFF /* Inking turned off message */
    MSG_WIN_COMMON any_win; /* Any window message */
    MSG_COMMON any; /* Common for all messages */
} WIN_MSG;

Description

This data structure defines a union of all window message structures. If these are the only types of messages being written to a mailbox, then the minimum mailbox message size is:

sizeof(WIN_MSG)
See Also
MSG_COMMON
MSG_WIN_BORDER
MSG_WIN_BUTTON
MSG_WIN_COMMON
MSG_WIN_CREATE
MSG_WIN_DESTROY
MSG_WIN_EXPOSE
MSG_WIN_FOCUS
MSG_WIN_INK_OFF
MSG_WIN_KEY
MSG_WIN_MOVE
MSG_WIN_PTR
MSG_WIN_REPARENT
MSG_WIN_RESIZE
MSG_WIN_RESTACK
MSG_WIN_STATE
Windowing Data Types

WIN_MSG_MASK

Window Message Mask

Syntax

typedef enum {
    WIN_MASK_NONE, /* No messages */
    WIN_MASK_BORDER, /* Border messages */
    WIN_MASK_BUTTON, /* Button messages */
    WIN_MASK_CHILD_CONFIG, /* Child configuration msgs */
    WIN_MASK_CHILD_CREATE, /* Child create window message */
    WIN_MASK_CHILD_DESTROY, /* Child destroy window msg */
    WIN_MASK_CONFIG, /* Configuration messages */
    WIN_MASK_DESTROY, /* Destroy window message */
    WIN_MASK_EXPOSE, /* Expose message */
    WIN_MASK_FOCUS, /* Focus messages */
    WIN_MASK_KEY, /* Key messages */
    WIN_MASK_REPARENT, /* Re-parent window message */
    WIN_MASK_PTR, /* Pointer move message */
    WIN_MASK_INK_OFF, /* Inking turned off message*/
    WIN_MASK_ANY /* Any message */
} WIN_MSG_MASK;
Description

This enumerated type defines the mask values that may be used to control which messages are sent by `maui_win`. Use `win_set_msg_mask()` to set which messages you are interested in for each window. The following table shows the relationship between message mask, message type, and message structure.

**Table 23-1  Relationship Between Message Mask, Message Type, and Message Structure**

<table>
<thead>
<tr>
<th>Message Mask</th>
<th>Message Type</th>
<th>Message Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN_MASK_BORDER</td>
<td>WIN_MSG_BORDER_ENTER</td>
<td>MSG_WIN_BORDER_ENTER</td>
</tr>
<tr>
<td></td>
<td>WIN_MSG_BORDER_LEAVE</td>
<td>MSG_WIN_BORDER_LEAVE</td>
</tr>
<tr>
<td>WIN_MASK_BUTTON</td>
<td>WIN_MSG_BUTTON_DOWN</td>
<td>MSG_WIN_BUTTON_DOWN</td>
</tr>
<tr>
<td></td>
<td>WIN_MSG_BUTTON_UP</td>
<td>MSG_WIN_BUTTON_UP</td>
</tr>
<tr>
<td>WIN_MASK_CHILD_CONFIG</td>
<td>WIN_MSG_MOVE_REQ</td>
<td>MSG_WIN_MOVE_REQ</td>
</tr>
<tr>
<td></td>
<td>WIN_MSG_RESIZE_REQ</td>
<td>MSG_WIN_RESIZE_REQ</td>
</tr>
<tr>
<td></td>
<td>WIN_MSG_RESTACK_REQ</td>
<td>MSG_WIN_RESTACK_REQ</td>
</tr>
<tr>
<td></td>
<td>WIN_MSG_STATE_REQ</td>
<td>MSG_WIN_STATE_REQ</td>
</tr>
<tr>
<td>Message Mask</td>
<td>Message Type</td>
<td>Message Structure</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>WIN_MASK_CONFIG WIN_MSG_MOVE</td>
<td>MSG_WIN_MOVE</td>
<td></td>
</tr>
<tr>
<td>WIN_MSG_RESIZE</td>
<td>MSG_WIN_RESIZE</td>
<td></td>
</tr>
<tr>
<td>WIN_MSG_RESTACK</td>
<td>MSG_WIN_RESTACK</td>
<td></td>
</tr>
<tr>
<td>WIN_MSG_STATE</td>
<td>MSG_WIN_STATE</td>
<td></td>
</tr>
<tr>
<td>WIN_MASK_CHILD_CREATE</td>
<td>WIN_MSG_CREATE</td>
<td>MSG_WIN_CREATE</td>
</tr>
<tr>
<td>WIN_MASK_CHILD_DESTROY</td>
<td>WIN_MSG_DESTROY</td>
<td>MSG_WIN_DESTROY</td>
</tr>
<tr>
<td>WIN_MASK_DESTROY WIN_MASK_DESTROY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WIN_MASK_EXPOSE</td>
<td>WIN_MSG_EXPOSE</td>
<td>MSG_WIN_EXPOSE</td>
</tr>
<tr>
<td>WIN_MASK_FOCUS WIN_MSG_FOCUS_IN</td>
<td>MSG_WIN_FOCUS</td>
<td></td>
</tr>
<tr>
<td>WIN_MSG_FOCUS_OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WIN_MASK_KEY</td>
<td>WIN_MSG_KEY_DOWN</td>
<td>MSG_WIN_KEY_DOWN</td>
</tr>
<tr>
<td>WIN_MSG_KEY_UP</td>
<td>MSG_WIN_KEY_UP</td>
<td></td>
</tr>
<tr>
<td>WIN_MASK_REPARENT WIN_MSG_REPARENT</td>
<td>MSG_WIN_REPARENT</td>
<td></td>
</tr>
<tr>
<td>WIN_MSG_REPARENT</td>
<td>MSG_WIN_REPARENT</td>
<td></td>
</tr>
</tbody>
</table>
### Table 23-1 Relationship Between Message Mask, Message Type, and Message Structure (continued)

<table>
<thead>
<tr>
<th>Message Mask</th>
<th>Message Type</th>
<th>Message Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN_MASK_PTR</td>
<td>WIN_MSG_PTR</td>
<td>MSG_WIN_PTR</td>
</tr>
<tr>
<td>WIN_MASK_INK_OFF</td>
<td>WIN_MSG_INK_OFF</td>
<td>MSG_WIN_INK_OFF</td>
</tr>
</tbody>
</table>

**See Also**
- `win_set_msg_mask()`
- `MSG_WIN_BORDER`
- `MSG_WIN_BUTTON`
- `MSG_WIN_CREATE`
- `MSG_WIN_DESTROY`
- `MSG_WIN_EXPOSE`
- `MSG_WIN_FOCUS`
- `MSG_WIN_INK_OFF`
- `MSG_WIN_KEY`
- `MSG_WIN_MOVE`
- `MSG_WIN_PTR`
- `MSG_WIN_REPARENT`
- `MSG_WIN_RESIZE`
- `MSG_WIN_RESTACK`
- `MSG_WIN_STATE`
- `WIN_MSG_TYPE`
**WIN_MSG_TYPE**

Window Message Type

### Syntax

typedef enum {
    WIN_MSG_BUTTON_DOWN, /* Pointer button pressed */
    WIN_MSG_BUTTON_UP, /* Pointer button released */
    WIN_MSG_BORDER_ENTER, /* Pointer entered window */
    WIN_MSG_BORDER_LEAVE, /* Pointer left window */
    WIN_MSG_CREATE, /* Child window was created */
    WIN_MSG_DESTROY, /* Window was destroyed */
    WIN_MSG_EXPOSE, /* Part of window exposed */
    WIN_MSG_FOCUS_IN, /* Received keyboard focus */
    WIN_MSG_FOCUS_OUT, /* Lost keyboard focus */
    WIN_MSG_KEY_DOWN, /* Key pressed */
    WIN_MSG_KEY_UP, /* Key released */
    WIN_MSG_MOVE, /* Window moved */
    WIN_MSG_MOVE_REQ, /* Window move request */
    WIN_MSG_PTR, /* Pointer moved */
    WIN_MSG_REPARENT, /* Window was re-parented */
    WIN_MSG_RESIZE, /* Window was re-sized */
    WIN_MSG_RESIZE_REQ, /* Window re-size request */
    WIN_MSG_RESTACK, /* Window was re-stack */
    WIN_MSG_RESTACK_REQ, /* Window re-stack request */
    WIN_MSG_STATE, /* Window state changed */
    WIN_MSG_STATE_REQ, /* Window state change request */
    WIN_MSG_INK_OFF /* Inking turned off */
} WIN_MSG_TYPE;

### Description

This enumerated type defines the window message types generated by maui_win. You may use `win_set_msg_mask()` to control which messages are sent for each window.

See `WIN_MSG_MASK` for a table that shows the relationship between message types, masks, and structures.
See Also

win_set_msg_mask()
MSG_WIN_BORDER
MSG_WIN_BUTTON
MSG_WIN_CREATE
MSG_WIN_DESTROY
MSG_WIN_EXPOSE
MSG_WIN_FOCUS
MSG_WIN_INK_OFF
MSG_WIN_KEY
MSG_WIN_MOVE
MSG_WIN_PTR
MSG_WIN_REPARENT
MSG_WIN_RESIZE
MSG_WIN_RESTACK
MSG_WIN_STATE
WIN_MSG_MASK
Windowing Data Types

WIN_PLACEMENT
Window Placement

Syntax

typedef enum {
    WIN_FRONT, /* In front of all siblings */
    WIN_BACK, /* In back of all siblings */
    WIN_FRONT_OF, /* In front of a sibling */
    WIN_BACK_OF /* In back of a sibling */
} WIN_PLACEMENT;

Description

This enumerated type defines how a window should be inserted into the list of sibling windows. Sibling windows are stacked front to back with the front-most window being closer to the user.

WIN_FRONT places the window in front of all other sibling windows.
WIN_BACK places the window in back of all other sibling windows.
WIN_FRONT_OF places the window in front of the specified sibling window.
WIN_BACK_OF places the window in back of the specified sibling window.

See Also

win_create_win()
win_reparent_win()
win_restack_win()
### Syntax

```c
typedef struct _WIN_STATUS {
    WIN_DEV_ID windev;  /* Windowing device ID */
    WIN_ID parent;      /* Parent window */
    WIN_ID child_front; /* Front-most child window */
    WIN_ID child_back;  /* Back-most child window */
    WIN_ID sib_front_of; /* Sibling in front of window */
    WIN_ID sib_back_of; /* Sibling in back of window */
    BOOLEAN state;      /* Window state */
    GFX_POINT root_pos; /* Window position in root */
    GFX_POINT position; /* Window position in parent */
    GFX_DIMEN width;    /* Window width */
    GFX_DIMEN height;   /* Window height */
    u_int32 cursor_id;  /* Cursor ID */
    WIN_INK_METHOD ink_method; /* Inking method */
    GFX_PIXEL ink_pixel; /* Inking pixel value */
    u_int32 msg_mask;   /* Message mask */
    WIN_CALLBACK callback; /* Callback for messages */
    void *user_data;    /* User data for callback */
    DRW_CONTEXT_ID drwctx; /* Drawing context */
    TXT_CONTEXT_ID txtctx; /* Text context */
    WIN_CMAP_ID cmap;   /* Colormap ID (NULL if none) */
    process_id owner_pid; /* Process ID of owner */
} WIN_STATUS;
```

### Description

This data structure is used by `win_get_win_status()` to return the status of a window.

- `windev` is the windowing device that contains the window. `parent`, `child_front`, `child_back`, `sib_front_of`, and `sib_back_of` specify the IDs of all windows directly related to this one.

- `state` indicates the current state of the window. If `TRUE`, then the window is active (visible). If `FALSE`, then the window is inactive (not visible).

The position of the windows is specified by `root_pos` and `position`. The size is `width` and `height`.
The cursor for the window is defined by `cursor_id`. The colormap is defined by `cmap`.

Ink for the window is controlled by `ink_method` and `ink_pixel`.

Only messages present in `msg_mask` are sent on behalf of this window. When sent, `callback` is used to set the callback member of the message. `user_data` is passed to the callback function when the message is dispatched.

The drawing context `drwctx` and the text context `txtctx` are automatically updated by the API as changes are made to this window.

**See Also**

`win_get_win_status()`

`BOOLEAN`  
`DRW_CONTEXT_ID`  
`GFX_DEV_ID`  
`GFX_DIMEN`  
`GFX_DMAP`  
`GFX_PIXEL`  
`GFX_POINT`  
`TXT_CONTEXT_ID`  
`WIN_DEV_ID`  
`WIN_ID`  
`WIN_INK_METHOD`
Appendix A: MAUI Error Codes

This appendix describes the error codes currently defined by MAUI. These error codes are defined in the `errno.h` header file.

Following is a numerically ordered list of error codes. For each error code, the typical causes and remedies are listed.

010:001 EOS_MAUI_BADACK
A bad acknowledgment was received from the MAUI process. The MAUI process may be confused or the protocol module may not be able to support the request being made. You may need to re-start the MAUI process or confirm that the protocol module supports the request being made. See *MAUI Porting Guide* for information about a specific protocol module.

010:002 EOS_MAUI_BADCODEMETH
A bad coding method was specified. The coding method specified is not defined by `GFX_CM` or is not valid for the attempted operation.

010:003 EOS_MAUI_BADCOLORTYPE
The color type specified is not a valid type, or is not supported for the attempted operation.

010:004 EOS_MAUI_BADCOMPATLEVEL
The compatibility level reported by a MAUI component is not legal. This indicates an implementation error in the component being asked to report its compatibility level. See the supplier of this component (driver or protocol module) to remedy the problem.
Appendix A: MAUI Error Codes

010:005  EOS_MAUI_BADDEFCHAR
The glyph corresponding to the default character for a font is not present in the font.

010:006  EOS_MAUI_BADDIMEN
The dimension (see GFX_DIMEN) specified is 0 or is out of range for the operation attempted.

010:007  EOS_MAUI_BADFRAME
The frame specified for an animation object is not valid for the sprite currently assigned to the animation object.

010:008  EOS_MAUI_BADID
The ID specified is 0, or is not a valid ID for the attempted operation.

010:009  EOS_MAUI_BADLINESIZE
The line size specified for a drawmap (see GFX_DMAP) is not a legal value.

010:010  EOS_MAUI_BADMBC
An error was received trying to parse a multi-byte string. See the ANSI C Specification for information on proper formatting of multi-byte strings.

010:011  EOS_MAUI_BADPOS
The position (see GFX_POS) specified is out of range for the attempted operation.

010:012  EOS_MAUI_BADPTR
The pointer value specified is NULL, or is not valid for the attempted operation.
Appendix A: MAUI Error Codes

010:013  **EOS_MAUI_BADRANGE**
The range specified is not legal. For example, there may have been a minimum value specified that was larger than the maximum value.

010:014  **EOS_MAUI_BADSHADE**
The shade specified has not been created yet. Use `mem_create_shade()` to create the shade. The shade `MEM_DEF_SHADE` is automatically created when you initialize the Shaded Memory API.

010:015  **EOS_MAUI_BADSIZE**
The size specified is 0, or is not valid for the attempted operation.

010:016  **EOS_MAUI_BADVALUE**
The value for an enumerated type is not legal. Use names only defined for the enumerated type.

010:017  **EOS_MAUI_BUSY**
The resource attempted is already in use and is not sharable.

010:018  **EOS_MAUI_CANTDISPLAY**
The drawmap specified cannot be displayed by the specified graphics device. This is usually caused by trying to display a drawmap whose coding method is not supported by the hardware. See `gfx_get_dev_cap()` for information about supported coding methods.

010:019  **EOS_MAUI_DAMAGE**
MAUI has detected that its data structures are damaged. This problem is usually caused by the use of un-initialized or improperly initialized pointers.
Appendix A: MAUI Error Codes

010:020  **EOS_MAUI_DEFINED**  
There has been an attempt to define something that has already defined. For example, there may have been an attempt to create a shade that already exists.

010:021  **EOS_MAUI_DMAPTOOSMALL**  
The drawmap specified is too small for the viewport in which it is being placed. Normally, the drawmap must be large enough to fill the viewport.

010:022  **EOS_MAUI_INCOMPATCM**  
The coding methods specified for the operation are not compatible. For example, `blt_copy_block()` requires source and destination drawmaps that have the same pixel depth.

010:023  **EOS_MAUI_INTERNAL**  
MAUI has detected an internal error. Please report the incident to Microware Customer Service. See the Preface of this manual for contact information.

010:024  **EOS_MAUI_INUSE**  
The resource upon which there has been an attempt to destroy is still being used. You must stop using it before it can be destroyed. This error code may also be returned in cases where an attempt to modify a resource fails because the resource is already in use.

010:025  **EOS_MAUI_ISINIT**  
The `maui_init()` function has already been called. This function should only be called once.

010:026  **EOS_MAUI_ISRESERVED**  
A reserve attempt on a resource failed because the resource is already reserved.
Appendix A: MAUI Error Codes

010:027   EOS_MAUI_MASKED
  There has been an attempt to write a message to a mailbox, but the
  mailbox is currently configured to reject messages of this type. See
  msg_set_mask() for information on setting the message mask for the
  mailbox.

010:028   EOS_MAUI_MBOXFULL
  The mailbox that has been attempted to write to is currently full. Wait
  (using _os_sleep()) to allow the reader time to read a message from
  the mailbox, then attempt the write operation again.

010:029   EOS_MAUI_MISSINGFEP
  The driver is missing the fast-entry-point necessary to perform the
  attempted operation. This is usually due to an old driver. See your
  supplier about upgrading your driver.

010:030   EOS_MAUI_NOCALLBACK
  msg_dispatch() has been called with a message that did not contain a
  callback function pointer. See MSG_COMMON for the layout of the
  common section of all messages.

010:031   EOS_MAUI_NODMAP
  A drawmap has not been assigned to the viewport. You must assign a
  drawmap to this viewport before attempting this operation.

010:032   EOS_MAUI_NODSTDMAP
  No destination drawmap has been specified for the context object to
  which you are trying to draw or copy.

010:033   EOS_MAUI_NOEXPTABLE
  No expansion table has been specified for the context object, but you
  are using a mixing mode that requires it. See BLT_MIX for information
  about settings used by different mixing modes.
Appendix A: MAUI Error Codes

010:034  **EOS_MAUI_NOFONT**
No font has been assigned to the text context object for which you are trying to draw text. You must first assign a font to the text object.

010:035  **EOS_MAUI_NOHWSUPPORT**
The hardware does not support the attempted operation.

010:036  **EOS_MAUI_NOINIT**
The API function called requires the API to be initialized first. Either call `maui_init()` to initialize all MAUI APIs or call the initialization function for the API.

010:037  **EOS_MAUI_NOMASKDMAP**
No mask drawmap has been specified for the context object for which you are trying to draw, and the drawing operation requested requires a mask drawmap.

010:038  **EOS_MAUI_NOMAUIP**
The **MAUI Input Process** was not found. This is detected during `inp_init()` when it tries to link to the MAUI process command mailbox (`mp_mbox`). You must start `mauip` before running MAUI applications that require pointer or key symbol input. This is usually done at system start-up.

010:039  **EOS_MAUI_NOPIXMEM**
No pixel memory has been assigned to the drawmap, but the operation attempted requires it. Assign pixel memory to the drawmap (call `gfx_set_dmap_pixmem()`) and attempt the operation again.

010:040  **EOS_MAUI_NOPMOD**
The specified protocol module could not be found. The protocol module must already be loaded in memory.
Appendix A: MAUI Error Codes

010:041  EOS_MAUI_NOSPRITE
No sprite has been assigned to the animation object, but the object has been attempted to be displayed. You must first assign a sprite to the animation object.

010:042  EOS_MAUI_NOSRCDMAP
No source drawmap has been specified for the context object being drawn with, and the drawing operation requested requires a source drawmap.

010:043  EOS_MAUI_NOTALIGNED
The lines of the pixel memory within a drawmap are not properly aligned. They should be padded such that the line size is a multiple of GFX_LINE_PAD bytes.

010:044  EOS_MAUI_NOTFOUND
The object specified was not found. The object may not exist at all, or may exist in a different list.

010:045  EOS_MAUI_NOTIMPLEMENTED
The feature attempted has not been implemented yet. All sources of this error have been noted in the respective function reference pages.

010:046  EOS_MAUI_NOTPENDING
The asynchronous operation that is trying to be released is not pending. The asynchronous operation was either never requested, or it already occurred.

010:047  EOS_MAUI_NOTRESERVED
The resource attempted has not been reserved.

010:048  EOS_MAUI_SIGNAL
A signal caused the operation being performed to abort.
Appendix A: MAUI Error Codes

010:049  EOS_MAUI_TOOCOMPLEX
The operation being requested is too complex for the hardware to support. For example, you will receive this error if you try to create a viewport stack that is too complex for the graphics device.

010:050  EOS_MAUI_TOOLONG
The specified string is too long. See the respective reference page for information about the maximum length allowed.

010:051  EOS_MAUI_CANTRESIZE
The size of the specified object cannot be changed at this time.

010:052  EOS_MAUI_NOPALETTE
A palette has not been specified, but one is required by the specified operation.

010:053  EOS_MAUI_BADNUMCHAN
The number of channels specified is not valid.

010:054  EOS_MAUI_NOTBUSY
The specified device is not busy.

010:055  EOS_MAUI_NOTPAUSED
The specified device is not paused.

010:056  EOS_MAUI_ABORT
The operation was aborted prematurely.

010:057  EOS_MAUI_TOOOLD
The feature requested is not present because an old MAUI component is being used. You need to update the necessary MAUI component to obtain this feature.
Appendix A: MAUI Error Codes

010:058  EOS_MAUI_INCOMPATVER
An incompatibility exists between two MAUI components. This incompatibility cannot be overcome. You must update the necessary MAUI component.

010:059  EOS_MAUI_NOTALLOWED
The operation attempted is not allowed.

010:060  EOS_MAUI_NOTOWNER
This process is not the owner of the object being modified. Only the owner can make this change.

010:061  EOS_MAUI_DISABLED
The feature attempted has been disabled in this version of MAUI. You need a more full-featured version of MAUI to obtain this function.

010:062  EOS_MAUI_NOTRAP
The MAUI trap module was not found. You need to load the MAUI trap module or link with the library version.

010:063  EOS_MAUI_DEVNOTFOUND
The specified device was not found.

010:064  EOS_MAUI_CMAPFULL
The colormap is full. The color (or cells) requested can not be allocated because there are no more free entries in the colormap.

010:065  EOS_MAUI_PAUSED
There has been an operation pause attempt while the operation is already paused.
Appendix A: MAUI Error Codes

010:066   **EOS_MAUI_BADRATE**
           The data rate specified is not valid.

010:067   **EOS_MAUI_NODVSUPPORT**
           The operation requested requires support by the device driver and the
           driver (and/or hardware) you are using does not support it.

010:068   **EOS_MAUI_NOTVISIBLE**
           The object referenced is not currently visible. This operation is only
           allowed if the object is visible.
Index

Symbols

Value of Context Settings 114
Initialize
   CDB API 170
Convert
   Display to Drawmap Position 285

Align 742
Allocate
   and Clear Shaded Memory Segment 408
   Color 549
   Graphics Memory 264
   Group of Color Cells 546
   Shaded Memory 425
Allocate and Clear Shaded Memory Segment 408
Allocating and de-allocating memory segments 48
Angle in 64ths of a Degree 736
Animation
   API 12
      Dependencies 12
   Data Types 14, 661
   Draw Group 71
   Functions 13, 59
   Group 13
      group 13
   Group functions 13
   Group Parameters 73
   Initialization and termination 13
   Object 12
      object functions 14
   Object Parameters 75
Sprite functions  13
Sprites  12
Animation Group
  Create  60
  Destroy  67
  Parameters  664
Animation group  13
Animation Group ID  663
Animation Group Parameters  664
Animation Object
  Create  62
  Destroy  69
  ID  667
  Parameters  668
  Placement  669
  Re-stack  80
Animation object  12,  14
AnimationGroup ID  663
anm_create_group()  60
anm_create_object()  62
anm_create_sprite()  65
anm_destroy_group()  67
anm_destroy_object()  69
anm_destroy_sprite()  70
anm_draw_group()  71
ANM_FRAME  662
anm_get_group()  73
anm_get_object()  75
ANM_GROUP_ID  663
ANM_GROUP_PARAMS  664
anm_init()  77
ANM_METHOD  665
ANM_OBJECT_ID  667
ANM_OBJECT_PARAMS  668
ANM_OBJECT_PLACEMENT  669
anm_process_group()  78
anm_restack_object()  80
anm_set_error_action()  82
anm_set_group_bkg()  85
anm_set_group_dst()  87
anm_set_object_bhv() 90
anm_set_object_frame() 92
anm_set_object_meth() 94
anm_set_object_pos() 96
anm_set_object_sprite 98
anm_set_object_sprite() 98
anm_set_object_state() 100
ANM_SPRITE 670
anm_term() 102
Appendix A
  MAUI Error Codes 913
Application messages 58

Big Endian 833
Bit-BLT
  API 15
  Context ID 674
  Context Parameters 675
  Data reference 18
  Data Types 673
  Features 15
  Function reference 17
  Functions 103
Bit-BLT API 15
Bit-BLT context 16, 17
Bit-BLT Context ID 674
Bit-BLT Context Object
  Destroy 117
Bit-BLT Context Object, Create 114
Bit-Bit Context Object, Create 114
Bit-BLT Context Parameters 675
Bit-BLT Data Types 673
Bit-BLT Functions 103
Block of Pixels 118
Block of Pixels, Draw 118
Block transfer operations 18
BLT_CONTEXT_ID 674
BLT_CONTEXT_PARAMS 675
blt_copy_block() 104
A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y   Z

blt_copy_next_block() 107
blt_copy_next_block(), Context Object Parameters 107
blt_copy_oblock() 110
blt_create_context() 114
blt_destroy_context() 117
blt_draw_block() 118
blt_draw_hline() 120
blt_draw_pixel() 122
blt_draw_vline() 124
blt_expd_block() 126
blt_expd_next_block() 129
blt_get_context() 132
blt_get_pixel() 134
blt_init() 136
BLT_MIX 677
blt_set_context_cpymix() 137
blt_set_context_drwmix() 139
blt_set_context_dst() 141
blt_set_context_expmix() 143
blt_set_context_exptbl() 145
blt_set_context_mask() 147
blt_set_context_ofs() 149
blt_set_context_pix() 151
blt_set_context_src() 153
blt_set_context_trans() 155
blt_set_error_action() 157
blt_term() 160
BOOLEAN 828
Boolean Type 828
Border Enter/Leave Message 866
Button Down/Up Message 868

Calculate Size of Pixel Memory 266
Callback Function 890
CD Control Unit 692
CDB
  API 19
  Data Types 685
  Device Types Description 690
Functions 161
functions 22
modules 22
cdb_get_copy() 162
cdb_get_ddr() 164
cdb_get_ncopy() 166
cdb_get_size() 168
cdb_init() 170
CDB_MAX_DNAME 686
CDB_MAX_PARAM 687
cdb_set_error_action() 171
cdb_term() 173
CDB_TYPE 688
CDB_TYPE_ANET 691
CDB_TYPE_CDC 692
CDB_TYPE_CTRLCHAN 693
CDB_TYPE_DATACHAN 695
CDB_TYPE_ENET 697
CDB_TYPE_FD 698
CDB_TYPE_FLASH 699
CDB_TYPE_GRAPHIC 700
CDB_TYPE_HDR 702
CDB_TYPE_IROUT 703
CDB_TYPE_ISDN 704
CDB_TYPE_LED 705
CDB_TYPE_MACFD 706
CDB_TYPE_MACHD 707
CDB_TYPE_MIDI 708
CDB_TYPE_MPA 709
CDB_TYPE_MPV 710
CDB_TYPE_NVRAM 711
CDB_TYPE_PCFD 712
CDB_TYPE_PCHD 713
CDB_TYPE_PIPEDEV 714
CDB_TYPE_PRNT 715
CDB_TYPE_RAM 716
CDB_TYPE_REMOTE 717
CDB_TYPE_RTNFM 718
CDB_TYPE_SER 719
CDB_TYPE_SOUND 720
CDB_TYPE_SPF 721
CDB_TYPE_SYSTEM 722
CDB_TYPE_TAPE 724
CDB_TYPE_WIN 725
Check a Range of Key Symbols 364
Circle 179
Circular Arc 176, 194, 197, 209, 212
Clone
  Graphics Device 268
  Viewport 270
Close
  Graphics Device 272
  Input Device 366, 555
  Mailbox 446
  Windowing Device 553
Coding Methods Supported 745
Color
  Allocation 549
  Palette 785
  Type 756
  Untyped 757
Color Type
  YCbCr 801
  YUV 802
Colormap
  Destroy 573
  ID 892
Colormap, Create 559
Colormaps 56
Common Part of Window Messages 870
Compatibility Level 830
Concepts of Windowing 54
Context
  Drawing, Create 188
Context Object Parameters
  blt_copy_block() 104, 110
  blt_draw_block() 118
  blt_draw_hline() 120
  blt_draw_pixel() 122
  blt_draw_vline() 124
  blt_expd_block() 126
  Set in blt_expd_block() 126
Used in blt_copy_block() 104, 110
Used in blt_copy_next_block() 107
Used in blt_draw_block() 118
Used in blt_draw_hline() 120
Used in blt_draw_pixel() 122
Used in blt_draw_vline() 124
Used in drw_arc() 176, 194, 197, 209, 212
Used in drw_earc() 194
Used in drw_ellipse() 197
Used in drw_expd_block() 200
Used in drw_oval() 209
Used in drw_oval_arc() 212
Context Parameters
Used in drw_circle() 179
Used in drw_copy_block() 182, 185
Used in drw_line() 206
Used in drw_point() 215
Used in drw_polygon() 217
Used in drw_polyline() 220
Used in drw_rectangle() 223
Context Settings 114
Context Settings, Value 114
Control Channel Device 693
Convert
Display to Drawmap Position 285
Drawmap to Display Position 283
Copy
Block of Pixels 182, 185
Next Rectangular Block of Pixels 107
Rectangular Block of Pixels 104, 557, 598
Rectangular Overlapping Blocks of Pixels 110
Copy operations 26
Create
Animation Group 60
Animation Object 62
Bit-BLT Context Object 114
Colormap 559
Cursor 561
Drawing Context 188
Drawmap Object 277
Font Object 503
Mailbox 448
Shade 410
Sprite 65
Text Context Object 500
Viewport 279
Window 568
Windowing Device 564
Creating and destroying shades 48
Cursor
  Destroy 575
Cursor, Create 561

Data Channel Device 695
Data reference 18, 23, 26, 31, 36, 38, 42, 49
Data structures 14, 18, 26, 32, 36, 38, 43, 49, 53, 58
Data type 57
Data Type reference 14, 57
Data type reference 53
Data types 14, 18, 26, 31, 36, 43, 53
DDR Parts 20
Dealing with drawmaps 29
De-allocate Graphics Memory 287
Default Shade 836
Default Value and Modifications
  Drawing Context 188
Default Value and Modifications for Drawing Context 188
Default Values
  anm_create_group() 60
  anm_create_object() 62
  Text Context Parameters 500
Default Values for Parameters in anm_create_group() 60
Default Values for Parameters in anm_create_object() 62
Default Values of Text Context Parameters 500
Defined constants 23, 31, 36, 38, 42, 49, 53, 57
Delta Between Two Positions 763
Dependencies 12, 15, 19, 24, 27, 34, 37, 39, 44, 50, 54
Depth 741
Destroy
  Animation Group 67
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animation Object</td>
<td>69</td>
</tr>
<tr>
<td>Bit-BLT Context Object</td>
<td>117</td>
</tr>
<tr>
<td>Colormap</td>
<td>573</td>
</tr>
<tr>
<td>Cursor</td>
<td>575</td>
</tr>
<tr>
<td>Drawing Context</td>
<td>192</td>
</tr>
<tr>
<td>Drawmap Object</td>
<td>291</td>
</tr>
<tr>
<td>Font Object</td>
<td>506</td>
</tr>
<tr>
<td>Shade of Memory</td>
<td>413</td>
</tr>
<tr>
<td>Sprite</td>
<td>70</td>
</tr>
<tr>
<td>Text Context Object</td>
<td>505</td>
</tr>
<tr>
<td>Viewport</td>
<td>292</td>
</tr>
<tr>
<td>Window</td>
<td>579</td>
</tr>
<tr>
<td>Windowing Device</td>
<td>577</td>
</tr>
<tr>
<td>Device name</td>
<td>21</td>
</tr>
<tr>
<td>parameter</td>
<td>21</td>
</tr>
<tr>
<td>Placement</td>
<td>771, 807</td>
</tr>
<tr>
<td>Re-stack</td>
<td>311, 617</td>
</tr>
<tr>
<td>Status</td>
<td>808</td>
</tr>
<tr>
<td>Device description record (DDR)</td>
<td>20</td>
</tr>
<tr>
<td>Device type</td>
<td>20</td>
</tr>
<tr>
<td>Device Type Names</td>
<td>688</td>
</tr>
<tr>
<td>Dimension In Pixels</td>
<td>774</td>
</tr>
<tr>
<td>Dispatch Message</td>
<td>451</td>
</tr>
<tr>
<td>Display and drawmap coordinate systems</td>
<td>28</td>
</tr>
<tr>
<td>Draw Block of Pixels</td>
<td>118</td>
</tr>
<tr>
<td>Circle</td>
<td>179</td>
</tr>
<tr>
<td>Circular Arc</td>
<td>176, 194, 197, 209, 212</td>
</tr>
<tr>
<td>Ellipse</td>
<td>197</td>
</tr>
<tr>
<td>Elliptical Arc</td>
<td>194</td>
</tr>
<tr>
<td>Horizontal Line of Pixels</td>
<td>120</td>
</tr>
<tr>
<td>Line</td>
<td>206</td>
</tr>
<tr>
<td>Multi-Byte String</td>
<td>507</td>
</tr>
<tr>
<td>Objects in a Group</td>
<td>71</td>
</tr>
<tr>
<td>Oval</td>
<td>209</td>
</tr>
<tr>
<td>Oval Arc</td>
<td>212</td>
</tr>
<tr>
<td>Point</td>
<td>215</td>
</tr>
<tr>
<td>Polygon</td>
<td>217</td>
</tr>
<tr>
<td>Polyline</td>
<td>220</td>
</tr>
<tr>
<td>Rectangle</td>
<td>223</td>
</tr>
</tbody>
</table>
Single Pixel 122
Vertical Line of Pixels 124
Wide Character String 510
Drawing API 24
Context ID 728
Context Parameters 729
Data Types 727
Functions 175
Method for an Object 665
Drawing API 24
Drawing Context
Create 188
Destroy 192
ID 728
Parameters 729
Drawing context 25
Drawing Data Types 727
Drawing Functions 175
Drawing Method for an Object 665
Drawing operations 25
Drawmap 777
Drawmap Object
Destroy 291
Drawmap Object, Create 277
Drawmaps 27
drw_arc() 176, 200
drw_circle() 179
DRW_CONTEXT_ID 728
DRW_CONTEXT_PARAMS 729
drw_copy_block() 182, 185
drw_create_context() 188
drw_destroy_context() 192
drw_earc() 194
drw_ellipse() 197
drw_expd_block() 200
DRW_FM 731
drw_get_context() 203
drw_init() 205
drw_line() 206
DRW_LS 732
drw_oval() 209
drw_oval_arc() 212
drw_point() 215
drw_polygon() 217
drw_polyline() 220
drw_rectangle() 223
drw_set_context_clip() 226
drw_set_context_cpymix() 228
drw_set_context_dash() 230
drw_set_context_draw() 232
drw_set_context_dst() 234
drw_set_context_expmix() 236
drw_set_context_exptbl() 238
drw_set_context_fm() 240
drw_set_context_ls() 242
drw_set_context_mask() 244
drw_set_context_mix() 246
drw_set_context_ofs() 248
drw_set_context_origin() 250
drw_set_context_pix() 252
drw_set_context_src() 256
drw_set_context_trans() 254
drw_set_error_action() 258
drw_term() 261

E

E1  741
E2  741
Enumerated types  14, 18, 26, 31, 36, 38, 42, 49, 53, 57
EOS error codes  913
Erase Ink from a Window  581
Error codes  913
Error Level  831
Error Level for drw_set_error_action()  258
Error Levels
  anm_set_error_action()  82
  blt_set_error_action()  157
  cdb_set_error_action  171
  drw_set_error_action()  258
  gfx_set_error_action()  340
inp_set_error_action() 385
  txt_set_error_action() 539
  win_set_error_action() 640
Error Levels in blt_set_error_action() 157
Error Levels in txt_set_error_action() 539
Error Levels 402
Example CDB 19
Expand
  Block of Pixels 126, 200
  Next Block of Pixels 129
Expose Message 874

Features 15
Fill Mode 731
Find
  Viewport at the Specified Position 296
Find the Viewport at the Specified Position 296
Flash RAM 699
Flush Messages 452
Focus In/Out Message 875
Font
  Structure 860
  Type 862
Font Object
  Destroy 506
Font Object, Create 503
Font Structure 860
Font Type 862
Fonts 52
Format and Range
  of RGB Components 734
  of YUV Values in GFX_YUV 802
  RGB Components for GFX_RGB 793
  YCbCr Components in GFX_YCBCR 801
Free
  a Segment from a Normal Shade 415
  a Segment from the Specified Shade 438
  All Segments from the Specified Shade 440
  Range of Color Cells 583
G

Get

Animation Group Parameters  73
Animation Object Parameters  75
Bit-BlT Context Parameters  132
Colors From a Colormap  588
Copy of the CDB  162,  166
Device Description By Type and Number  164
Drawing Context Parameters  203
Free Space in a Colormap  590
Graphics Device Attribute  298
Graphics Device Capabilities  300,  302
Graphics Device Status  304
Input Device Capabilities  368
Input Device Status  370
Mailbox Status  454
N byte Copy of the CDB  166
Pixel  134
Shade Status  417
Size of the CDB  168
Text Context Parameters  513
Viewport Status  306
Width of a Multi-Byte String  515
Width of a Wide Character String  517
Window Status  596
Windowing Device Status  594
GFX_A1_RGB  734
gfx_alloc_mem()  264
GFX_ANGLE  736
GFX_ATTR_MODE  737
GFX_ATTR_TYPE  738
gfx_calc_pixmem_size()  266
gfx_clone_dev()  268
gfx_clone_vport()  270
gfx_close_dev()  272
GFX_CM  740,  745
GFX_CM_1A7_8BIT  750
GFX_CM_1BIT 746
GFX_CM_2BIT 746
GFX_CM_3BIT 747
GFX_CM_4BIT 747
GFX_CM_5BIT 747
GFX_CM_6BIT 747
GFX_CM_7BIT 747
GFX_CM_8BIT 748
GFX_CM_A1_RGB555 751
GFX_CM_CDI_DYUV 750
GFX_CM_CDI_RL3 748
GFX_CM_CDI_RL7 749
GFX_CM_RGB555 748
GFX_CM_RGB556 752
GFX_CM_RGB565 751
GFX_CM_RGB655 752
GFX_CM_RGB888 748
GFX_CM_YCBCR420 752
GFX_CM_YCBCR422 751
GFX_CM_YCRCB420 754
GFX_CM_YCRCB422 751
GFX_COLOR 755
GFX_COLOR_TYPE 756
GFX_COLOR_VALUE 757
gfx_create_cursor() 274
gfx_create_dmap() 277
gfx_create_vport() 279
GFX_CURSOR_CAP 759
GFX_CURSOR_ID 760
GFX_CURSOR_INFO 761
GFX_CURSOR_SPEC 762
gfx_cvt_dmpos_dppos() 283
gfx_cvt_dppos_dmpos() 285
gfx_dealloc_mem() 287
GFX_DELTA 763
gfx_destroy_cursor() 289
gfx_destroy_dmap() 291
gfx_destroy_vport() 292
GFX_DEV_ATTR 764
GFX_DEV_CAP 765
GFX_DEV_CAPEXTEN 766
GFX_DEV_CAPEXTEN_VALIDATE() 767
GFX_DEV_CM 768
GFX_DEV_ID 769
GFX_DEV_MODES 770
GFX_DEV_PLACEMENT 771
GFX_DEV_RES 772
GFX_DEV_STATUS 773
GFX_DIMEN 774
GFX_DIMEN_MAX 775
GFX_DIMEN_MIN 776
GFX_DMAP 777
GFX_DMAP Initialization Values 277
gfx_find_vport() 296
gfx_get_cm_align() 743
gfx_get_cm_bit_order() 741
gfx_get_cm_byte_order() 741
gfx_get_cm_depth() 742
gfx_get_cm_name() 744
gfx_get_cm_tc_swap() 744
gfx_get_cursor_cap() 294
gfx_get_dev_attribute() 298
gfx_get_dev_status() 304
gfx_get_vport_status() 306
gfx_init() 308
GFX_INTL Mode 779
GFX_LINE_PAD 780
GFX_MAX_DEV_NAME 781
GFX_OFFSET 782
GFX_OFFSET_MAX 783
GFX_OFFSET_MIN 784
gfx_open_dev() 309
GFX_PALETTE 785
GFX_PIXEL 787
GFX_POINT 788
GFX_POS 789
GFX_POS_MAX 790
GFX_POS_MIN 791
GFX_RECT 792
gfx_restack_dev() 311
gfx_restack_vport() 314
GFX_RGB 793
gfx_set_cm_align()  743
gfx_set_cm_bit_order()  741
gfx_set_cm_byte_order()  741
gfx_set_cm_depth()  742
gfx_set_cm_name()  744
gfx_set_cm_tc_swap()  744
gfx_set_cursor()  317
gfx_set_cursor_pos()  319
gfx_set_decode_dst()  321
gfx_set_dev_attribute()  323
gfx_set_display_bkcol()  326
gfx_set_display_extvid()  328
gfx_set_display_size()  330
gfx_set_display_transcol()  332
gfx_set_display_vpmix()  334
gfx_set_dmap_pixmem()  336
gfx_set_dmap_size()  338
gfx_set_error_action()  340
gfx_set_vport_colors()  343
gfx_set_vport_dmap()  345
gfx_set_vport_dmpos()  348
gfx_set_vport_intensity()  350
gfx_set_vport_position()  352
gfx_set_vport_size()  354
gfx_set_vport_state()  356
gfx_sync_retrace()  358
gfx_term()  360
gfx_update_display()  361
GFX_VPC  794
GFX_VPDMC  796
GFX_VPORT_ID  798
GFX_VPORT_PLACEMENT  799
GFX_VPORT_STATUS  800
GFX_YCBCR  801
GFX_YUV  802
gfxdev Settings  279
Glyph
  Unused  863
Glyph Range Table  864
Graphic Device Attribute  323
Graphics
<table>
<thead>
<tr>
<th>Category</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cursor</td>
<td>30, 56, 895</td>
</tr>
<tr>
<td>Data Types</td>
<td>733</td>
</tr>
<tr>
<td>Device</td>
<td>700</td>
</tr>
<tr>
<td>Device API</td>
<td>27</td>
</tr>
<tr>
<td>Device Attribute Description</td>
<td>764</td>
</tr>
<tr>
<td>Device Capabilities</td>
<td>765</td>
</tr>
<tr>
<td>Device Coding Methods</td>
<td>768</td>
</tr>
<tr>
<td>Device Extended Capabilities</td>
<td>766</td>
</tr>
<tr>
<td>Device ID</td>
<td>769</td>
</tr>
<tr>
<td>Device Modes</td>
<td>770</td>
</tr>
<tr>
<td>Device Resolution</td>
<td>772</td>
</tr>
<tr>
<td>Device Status</td>
<td>773</td>
</tr>
<tr>
<td>Display Modes</td>
<td>770</td>
</tr>
<tr>
<td>Functions</td>
<td>263</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>De-allocate</td>
<td>287</td>
</tr>
<tr>
<td>Graphics device</td>
<td>28</td>
</tr>
<tr>
<td>Graphics Device Attribute Mode</td>
<td>737</td>
</tr>
<tr>
<td>Graphics Device Attribute Types</td>
<td>738</td>
</tr>
<tr>
<td>Graphics Memory</td>
<td>29</td>
</tr>
<tr>
<td>Grow Method</td>
<td>837</td>
</tr>
</tbody>
</table>

**H**

<table>
<thead>
<tr>
<th>Category</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal line</td>
<td>120</td>
</tr>
<tr>
<td>How mode effects value in gfx_set_dev_attribute()</td>
<td>323</td>
</tr>
</tbody>
</table>

**I**

<table>
<thead>
<tr>
<th>Category</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Blocking Type</td>
<td>844</td>
</tr>
<tr>
<td>IFMAN (Ethernet) LAN Device</td>
<td>697</td>
</tr>
<tr>
<td>Initialization and termination</td>
<td>13, 17, 22, 24, 28, 35, 41, 47, 52, 54</td>
</tr>
<tr>
<td>Initialize</td>
<td></td>
</tr>
<tr>
<td>Animation API</td>
<td>77</td>
</tr>
<tr>
<td>Bit-BLT API</td>
<td>136</td>
</tr>
<tr>
<td>CDB API</td>
<td>170</td>
</tr>
<tr>
<td>Drawing API</td>
<td>205</td>
</tr>
<tr>
<td>Graphics Device API</td>
<td>308</td>
</tr>
<tr>
<td>Input Device API</td>
<td>372</td>
</tr>
</tbody>
</table>
MAUI APIs 400
Messaging API 456
Shaded Memory API 418
Text API 519
Windowing API 600
Initialization and termination 38
Inking Disabled Message 876
inp_check_keys() 364
inp_close_dev() 366
INP_DEV_CAP 804
INP_DEV_CLASS 805
INP_DEV_ID 806
INP_DEV_PLACEMENT 807
INP_DEV_STATUS 808
inp_get_dev_cap() 368
inp_get_dev_status() 370
inp_init() 372
INP_KEY_* 809
INP_KEY_SUBTYPE 818
INP_KEYMOD 819
INP_KEYS 820
INP_MAX_BUTTONS 822
INP_MAX_DEV_NAME 823
INP_MSG 824
inp_open_dev() 374
INP_PTR_SUBTYPE 825
inp_release_key() 377
inp_reserve_key() 379
inp_restack_dev() 381
inp_set_callback() 383
inp_set_error_action() 385
inp_set_msg_mask() 388
inp_set_ptr_limit() 390
inp_set_ptr_pos() 392
inp_set_sim_meth() 394
INP_SIM_METH 826
inp_term() 397
Input
Data Types 803
Device API 34
Device Capabilities 804
Device Classification  805
Device ID  806
Functions  363
Input Data Types  803
Input Device
  Re-stack  381
Input device  35
Input Device API  34
Input Device Capabilities  804
Input Device Classification  805
Input Device ID  806
Input Functions  363
Interlace Mode  779
Introduction  12, 15, 19, 24, 27, 34, 37, 39, 44, 50, 54
IR Output Blaster  703
ISM (ISDN) WAN Device  704

Key
  Down/Up Message  877
  Modifiers  819
Key reservation and simulation  35
Key Symbol
  Groups  820
  Message  847
  Message Subtype  818
  Names  809
Key Symbol Groups  820
Key Symbol Message  847
Key Symbol Message Subtype  818
Key Symbol Names  809
Key Symbols
  INP_KEY_*  809
  INP_KEYS  820
Key Symbols in INP_KEY_*  809
Key Symbols in INP_KEYS  820
LED Device  705
Line  206
   Horizontal  120
   Padding  780
   Style  732
   Vertical  124
Line Padding  780
Line Style  732
Little Endian  829
Lock a Region of a Window  601
LSBFIRST  829

Mac FM Floppy Disk  706
Mac FM Hard Disk  707
Mailbox ID  849
Mailbox Status  850
Mailbox, Create  448
MAUI
   Error Codes  913
MAUI System
   API  37
   Data Types  827
   Functions  399
MAUI WIN Pseudo-Device  725
MAUI_COMPAT_LEVEL  830
MAUI_ERR_LEVEL  831
maui_init()  400
MAUI_MSG  832
maui_set_error_action()  402
maui_term()  405
Maximum
   Buttons for Pointer Device  822
   Length of a Device Name  686
   Length of Device Name  781,  823,  900
   Length of Mailbox Name  846
   Length of Parameter String  687
   Message
Value
  GFX_POS  790
Value
  GFX_DIMEN  775
  GFX_OFFSET  783
Value For GFX_DIMEN  775
Value For GFX_OFFSET  783
Value For GFX_POS  790
mem_calloc()  408
mem_create_shade()  410
MEM_DEF_SHADE  836
mem_destroy_shade()  413
mem_free()  415
mem_get_shade_status()  417
MEM_GROW  837
mem_init()  418
mem_list_overflows()  419
mem_list_segments()  421
mem_list_tables()  423
mem_malloc()  425
MEM_MIN_ALLOC  838
MEM_OVTYPE  839
mem_realloc()  427
mem_set_alloc()  429
mem_set_alloc_bndry()  431
mem_set_dealloc()  433
mem_set_error_action()  435
mem_set_grow_method()  437
mem_sfree()  438
mem_sfree_all()  440
MEM_SHADE_STATUS  840
MEM_SHADE_TYPE  841
mem_term()  442
Memory Shade, Destroy  413
Message
  Dispatch  451
  Header  845
  Inking disabled  876
  Placement  852
  Type  855
  Types  855
types  40
Write to Mailbox  492, 495
Message mailbox  41
Messages  41
Messaging
  Data Types  843
  Functions  445
Messaging API  39
Messaging Data Types  843
Messaging Functions  445
Method
  ANM_METHOD  665
    anm_set_object_meth()  94
    inp_set_sim_meth()  395
MFM Sound Processor  720
Midi  708
Minimum
  Allocation Size  838
  Value for GFX_DIMEN  776
  Value For GFX_OFFSET  784
  Value For GFX_POS  791
Minimum Allocation Size  838
Minimum Value for GFX_DIMEN  776
Minimum Value For GFX_OFFSET  784
Minimum Value For GFX_POS  791
Miscellaneous  30
Mixing Mode  677
Mixing Modes for BLT_MIX  679
Motion Video Device  710
Move Window  604
MPEG Audio Device  709
MSBFIRST  833
MSG_BLOCK_TYPE  844
msg_close_mbox()  446
MSG_COMMON  845
msg_create_mbox()  448
msg_dispatch()  451
msg_flush()  452
msg_get_mbox_status()  454
msg_init()  456
MSG_KEY  847
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_MAX_MBOX_NAME</td>
<td>846</td>
</tr>
<tr>
<td>MSG_MBOX_ID</td>
<td>849</td>
</tr>
<tr>
<td>MSG_MBOX_STATUS</td>
<td>850</td>
</tr>
<tr>
<td>msg_open_mbox()</td>
<td>457</td>
</tr>
<tr>
<td>msg.peek()</td>
<td>459</td>
</tr>
<tr>
<td>msg.peekn()</td>
<td>461</td>
</tr>
<tr>
<td>MSG_PLACEMENT</td>
<td>852</td>
</tr>
<tr>
<td>MSG_PTR</td>
<td>853</td>
</tr>
<tr>
<td>msg_read()</td>
<td>463</td>
</tr>
<tr>
<td>msg_readn()</td>
<td>466</td>
</tr>
<tr>
<td>msg_release_sig()</td>
<td>469</td>
</tr>
<tr>
<td>msg_release_watch()</td>
<td>471</td>
</tr>
<tr>
<td>msg_send_sig()</td>
<td>473</td>
</tr>
<tr>
<td>msg_send_watch()</td>
<td>475</td>
</tr>
<tr>
<td>msg_set_error_action()</td>
<td>477</td>
</tr>
<tr>
<td>msg_set_error_action() Error Levels</td>
<td>477</td>
</tr>
<tr>
<td>msg_set_filter()</td>
<td>480</td>
</tr>
<tr>
<td>msg_set_mask()</td>
<td>483</td>
</tr>
<tr>
<td>msg_term()</td>
<td>485</td>
</tr>
<tr>
<td>MSG_TYPE</td>
<td>855</td>
</tr>
<tr>
<td>msg_unread()</td>
<td>487</td>
</tr>
<tr>
<td>msg_unreadn()</td>
<td>490</td>
</tr>
<tr>
<td>MSG_WIN_BORDER</td>
<td>866</td>
</tr>
<tr>
<td>MSG_WIN_BUTTON</td>
<td>868</td>
</tr>
<tr>
<td>MSG_WIN_COMMON</td>
<td>870</td>
</tr>
<tr>
<td>MSG_WIN_CREATE</td>
<td>871</td>
</tr>
<tr>
<td>MSG_WIN_DESTROY</td>
<td>873</td>
</tr>
<tr>
<td>MSG_WIN_EXPOSE</td>
<td>874</td>
</tr>
<tr>
<td>MSG_WIN_FOCUS</td>
<td>875</td>
</tr>
<tr>
<td>MSG_WIN_INK_OFF</td>
<td>876</td>
</tr>
<tr>
<td>MSG_WIN_KEY</td>
<td>877</td>
</tr>
<tr>
<td>MSG_WIN_MOVE</td>
<td>879</td>
</tr>
<tr>
<td>MSG_WIN_PTR</td>
<td>880</td>
</tr>
<tr>
<td>MSG_WIN_REPARENT</td>
<td>882</td>
</tr>
<tr>
<td>MSG_WIN_RESIZE</td>
<td>884</td>
</tr>
<tr>
<td>MSG_WIN_RESTACK</td>
<td>886</td>
</tr>
<tr>
<td>MSG_WIN_STATE</td>
<td>888</td>
</tr>
<tr>
<td>msg_write()</td>
<td>492</td>
</tr>
<tr>
<td>msg_writen()</td>
<td>495</td>
</tr>
<tr>
<td>Multi-Byte String</td>
<td>507</td>
</tr>
</tbody>
</table>
N
Name 744
Name Field Ranges in GFX_CM 744
Names
   Device Type 688
   NFM (Arcnet) LAN Device 691
nm_set_group_bkg() 85
Non-Volatile Ram 711

O
Objects in a Group 71
Offset in Pixels 782
Open
   Graphics Device 309
   Input Device 374, 609
   Mailbox 457
   Windowing Device 606
Overhead Type 839
Overview of MAUI 11

P
Parallel Printer 715
Parameters
   Copy Operations 681
   Draw Operations 680
   Expand Operations 682
PCF (MS-DOS Format) Floppy Disk 712
PCF (MS-DOS Format) Hard Disk 713
Peek at
   Next Message in Mailbox 459, 461
Pen and Ink 56
Pipe Device 714
Pixel
   Coding Method 740
   Draw 122
   Position 789
   Value 787
Pixel Coding Method  740
Pixel Position    789
Pixel Value       787
Pixel, Single     122
Placement
anm_create_object   63
anm_restack_object ()  80
Device             771, 807
gfx_restack_dev()  311, 617
gfx_restack_vport() 314
inp_restack_dev()  381
Specify New Position 280
win_create_win()  570
win_reparent_win() 611
win_restack_win()  619
Point             215
Pointer
    Message   853
    Message Subtype 825
    Move Message  880
Polygon           217
Polyline          220
Position of a Point Given by X and Y  788
Possible Values of Align Field in GFX_CM  742
Possible Values of Depth Field in GFX_CM  741
Print
    Listing of Allocated Segments  421
    Listing of Memory Tables     423
    Listing of Overflows        419
Printer
    Parallel   715
Process Objects in a Group  78
Processing messages  41

R

RAM Extensions  716
Range of Bits    40
RBF (Universal Format) Floppy Disk  698
RBF (Universal Format) Hard Disk  702
Read
Next Message from Mailbox  
Reallocate Shaded Memory  
Rectangle  
Rectangle Defined by X, Y, Width, and Height  
Registered Message Types  
Relationship between colored memory and pseudo memory.  
Relationship Between Message Mask, Message Type, and Message Structure  
Release Key  
Release Signal on Message Ready  
Release Watch Signal  
Remote, Pointing, Key Device  
Re-parent a Window  
Required Parameters  
Copy Operations  
Copy Operations  
Draw Operations  
Expand Operations  
Expand Operations  
Reserve Key  
Resize Window  
Re-stack  
Animation Object  
Device  
Input Device  
Viewport  
Window  
Restack  
Animation Object  
Device  
RGB  
Color  
Color with Alpha Flag  
Format and Range  
RGB Color  
RGB Color with Alpha Flag  
RTNFM Real-Time WAN Device  
SBF Tape Device  

SCF Midi Device  708  
SCF Parallel Printer Device  715  
SCF Serial Device  719  
Send Signal on Message Ready  473  
Send Signal on msg_term()  475  
SendSig  473,  475  
Serial Device  719  
Set  
Action to Take in Error Handler  82,  157,  258,  340,  385,  402,  435,  477,  539,  640  
Action to take in Error Handler  171  
Allocator Function for a Shade  429  
Backdrop Color  326  
Behavior for an Object  90  
Callback for Queuing Messages  383,  622  
Character Padding  522  
Clipping Area  226,  520  
Coding Method and Size of Drawmap  338  
Coding Method and Size of Drawmap  338  
Colormap for a Window  624  
Colors for a Viewport  343  
Colors in Group of Cells  626  
Cursor For a Window  630,  632  
Dash Pattern  230  
De-allocator Function for a Shade  433  
Destination Drawmap  87,  141,  234,  525  
Destination for Video Decoding  321  
Display Size  330  
Drawing Area  232,  523,  636  
Drawing Context  638  
Drawing Method for an Object  94  
Drawing Origin  250,  535  
Drawmap Position In a Viewport  348  
Drawmap To Use In a Viewport  345  
External Video On/Off  328  
Fill Mode  240  
Filter for Searching a Mailbox  480  
Font To Use  529  
Frame for an Object  92  
Group Background  85  
Grow Method For a Shade  437
Inking Method  645
Line Style  242
Mask Drawmap  147, 244
Mask for Queuing Messages  388, 483, 649
Memory Allocation Boundary  431
Mixing Mode  531
Mixing Mode for Copying  137, 228
Mixing Mode for Drawing  139, 246
Mixing Mode for Expanding  143, 236
Offset Pixel Value  149, 248, 533
Pixel Expansion Table  145, 238, 527
Pixel Memory Pointer in Drawmap  336
Pixel Memory Pointer in Drawmap  336
Pixel Value For Drawing  151, 252
Pixel Value for Ink  647
Pointer Limit  390
Pointer Position  392
Position for an Object  96
Position of a Viewport  352
Simulation Method  394
Size of a Viewport  354
Source Drawmap  153, 256
Sprite for an Object  98
State for an Object  100
State of a Viewport  356
Text Context  653
Transparent Color  332
Transparent Pixel Value  254, 537
Transparent Pixel Value  155
Viewport Intensity  350
Viewport Mixing On/Off  334
Window State  651
Window That Has Keyboard Focus  643
Set Graphic Device Attribute  323
Shade
  Default  836
  Status  840
  Type  841
Shade of Memory
  Destroy  413
Shade Status  840
Shade Type  841
Shade, Create  410
Shaded Memory
  API  44
  Data Types  835
  Functions  407
Shaded Memory API  44
Shaded Memory Data Types  835
Shaded Memory Functions  407
sim_meth
  inp_set_sim_meth()  395
Simulation Method  826
Single Pixel  122
Size
  Dimension In Pixels  774
SPF Device  721
Sprite  12
  Create  65
  Destroy  70
  Frame Structure  662
  Structure  670
Sprites  13
Status
  Device  808
Status and debugging functions  48
Supported Coding Methods in GFX_CM  745
Supported Coding Methods in GFX_CM  745
Synchronize with Vertical Retrace  358
System Description  722

T

Tape  724
Terminate
  Animation API  102
  Bit-BLT API  160
  CDB API  173
  Drawing API  261
  Graphics Device API  360
  Input Device API  397
  MAUI APIs  405
Messaging API 485
Shaded Memory API 442
Text API 542
use of the Animation API 102
use of the Drawing API 261
use of the Input Device API 397
use of the Messaging API 485
Windowing API 655
Terminate Shaded Memory API 442
Terminates the CDB API 173
Text
API 50
Context ID 858
Context Parameters 859
Data Types 857
Functions 499
Text API 50
Text context 52
Text Context ID 858
Text Context Object
Create 500
Destroy 505
Text Context Parameters 859
Text Data Types 857
Text drawing operations 53
Text Functions 499
True Color Swap 744
TXT_CONTEXT_ID 858
TXT_CONTEXT_PARAMS 859
txt_create_context() 500
txt_create_font() 503
txt_destroy_context() 505
txt_destroy_font() 506
txt_draw_mbs() 507
txt_draw_wcs() 510
TXT_FONT 860
TXT_FONTYPE 862
txt_get_context() 513
txt_get_mbs_width() 515
txt_get_wcs_width() 517
txt_init() 519
TXT_NOGLYPH  863
TXT_RANGTBL  864
txt_set_context_clip()  520
txt_set_context_cpad()  522
txt_set_context_draw()  523
txt_set_context_dst()  525
txt_set_context_exptbl()  527
txt_set_context_font()  529
txt_set_context_mix()  531
txt_set_context_ofs()  533
txt_set_context_origin()  535
txt_set_context_trans()  537
txt_set_error_action()  539
txt_term()  542
Typed Color Value  755

Union
  All Input Message Structures  824
  All MAUI Message Types  832
  All Window Message Structures  901
Unlock a Region of a Window  659
Unread
  Message to a Mailbox  487, 490
Untyped Color Value  757
Unused Glyph  863
Update
  Display  361
Use of Placement in gfx_restack_dev()  311
Use of Placement in gfx_restack_vport()  314
Use of Placement to Specify New Position  280
Using normal shades  45
Using pseudo shades  47

Value of Context Settings  114
Value of Method for anm_set_object_meth()  94
Value of Method in ANM_METHOD  665
Value of Placement in anm_create_object 63
Value of Placement in anm_restack_object () 80
Value of Placement in inp_restack_dev() 381
Value of sim_meth for inp_set_sim_meth() 395
Vertical Line 124
Vertical line 124
Viewport
  Complexity 794
  Create 279
  Destroy 292
  Drawmap Complexity 796
  Find 296
  ID 798
  Placement 799
  Re-stack 314
  Status 800
  Viewports 27, 29

WAN 718
What this API does not do 16
Wide Character String 510
win_alloc_cmap_cell() 544
win_alloc_cmap_cells() 546
win_alloc_cmap_color() 549
win_alloc_cmap_colors() 551
WIN_CALLBACK 890
WIN_CELL_PARAMS 891
win_close_dev() 553
win_close_inpdev() 555
WIN_CMAP_ID 892
WIN_CMAP_PARAMS 893
WIN_CMATCH 894
win_create_cmap() 559
win_create_cursor() 561
win_create_dev() 564
win_create_win() 568
win_create_win() Parameter Default Values 568
WIN_CURSOR 895
win_destroy_cmap() 573
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>win_destroy_cursor()</td>
<td>575</td>
</tr>
<tr>
<td>win_destroy_dev()</td>
<td>577</td>
</tr>
<tr>
<td>win_destroy_win()</td>
<td>579</td>
</tr>
<tr>
<td>WIN_DEV_ID</td>
<td>896</td>
</tr>
<tr>
<td>WIN_DEV_STATUS</td>
<td>897</td>
</tr>
<tr>
<td>win_erase_ink()</td>
<td>581</td>
</tr>
<tr>
<td>win_free_cmap_cells()</td>
<td>583</td>
</tr>
<tr>
<td>win_get_cells_params()</td>
<td>586</td>
</tr>
<tr>
<td>win_get_cmap_cells()</td>
<td>588</td>
</tr>
<tr>
<td>win_get_cmap_free()</td>
<td>590</td>
</tr>
<tr>
<td>win_get_cmap_params()</td>
<td>592</td>
</tr>
<tr>
<td>win_get_dev_status()</td>
<td>594</td>
</tr>
<tr>
<td>win_get_win_status()</td>
<td>596</td>
</tr>
<tr>
<td>WIN_ID</td>
<td>898</td>
</tr>
<tr>
<td>win_init()</td>
<td>600</td>
</tr>
<tr>
<td>WIN_INK_METHOD</td>
<td>899</td>
</tr>
<tr>
<td>win_lock_region()</td>
<td>601</td>
</tr>
<tr>
<td>WIN_MAX_DEV_NAME</td>
<td>900</td>
</tr>
<tr>
<td>win_move_win()</td>
<td>604</td>
</tr>
<tr>
<td>WIN_MSG</td>
<td>901</td>
</tr>
<tr>
<td>WIN_MSG_MASK</td>
<td>903</td>
</tr>
<tr>
<td>WIN_MSG_TYPE</td>
<td>907</td>
</tr>
<tr>
<td>win_open_dev()</td>
<td>606</td>
</tr>
<tr>
<td>win_open_inpdev()</td>
<td>609</td>
</tr>
<tr>
<td>WIN_PLACEMENT</td>
<td>909</td>
</tr>
<tr>
<td>win_reparent_win()</td>
<td>611</td>
</tr>
<tr>
<td>win_resize_win()</td>
<td>614</td>
</tr>
<tr>
<td>win_restack_win()</td>
<td>619</td>
</tr>
<tr>
<td>win_set_callback()</td>
<td>622</td>
</tr>
<tr>
<td>win_set_cmap()</td>
<td>624</td>
</tr>
<tr>
<td>win_set_cmap_cells()</td>
<td>626</td>
</tr>
<tr>
<td>win_set_color_match()</td>
<td>628</td>
</tr>
<tr>
<td>win_set_cursor()</td>
<td>630, 632</td>
</tr>
<tr>
<td>win_set_drw_area()</td>
<td>636</td>
</tr>
<tr>
<td>win_set_drw_context()</td>
<td>638</td>
</tr>
<tr>
<td>win_set_error_action()</td>
<td>640</td>
</tr>
<tr>
<td>win_set_focus()</td>
<td>643</td>
</tr>
<tr>
<td>win_set_ink_method()</td>
<td>645</td>
</tr>
<tr>
<td>win_set_ink_pix()</td>
<td>647</td>
</tr>
<tr>
<td>win_set_msg_mask()</td>
<td>649</td>
</tr>
<tr>
<td>win_set_state()</td>
<td>651</td>
</tr>
</tbody>
</table>
win_set_txt_context() 653
WIN_STATUS 910
win_term() 655
win_unlock_region() 659
Window
  Create 568
  Created Message 871
  Destroy 579
  Destroyed Message 873
  ID 898
  Inking Method 899
  Message Mask 903
  Message Type 907
  Messages
    Common 870
  Move Message 879
  Placement 909
  Re-parented Message 882
  Re-sized Message 884
  Re-stack 619
  Re-stacked Message 886
  State Change Message 888
  Status 910
  Unlock Region 659
Windowing 54
  API 54
  Concepts 54
  Data Types 865
  Device ID 896
  Device Status 897
  Functions 543
  System 54
Windowing Device 55
  Create 564
  Destroy 577
Windows 55
Write
  Message to a Mailbox 492, 495
YCbCr
  Format and Range  801
YCbCr Color  801
YUV
  Format and Range  802
YUV Color  802