

Musubi v 1.0

August 28, 2008

Mark K. Akita - mkakita@juno.com

Table of Contents

- 1 Introduction
- 2 Programs and Files
- 3 Using Musubi
- 4 The Musbi Library Routines

- 4.1 M_init()
- 4.2 graphics_mode()
- 4.3 clear_screen()
- 4.4 pixel()
- 4.5 draw_line()
- 4.6 polygon()
- 4.7 ellipse()
- 4.8 save_screen()
- 4.9 palette()
- 4.10 all_palette()
- 4.11 get_all_palette()
- 4.12 M_hline()
- 4.13 M_vline()
- 4.14 M_rectangle()
- 4.15 M_box()
- 4.16 M_bezier()
- 4.17 M_string()
- 4.18 M_get_update_mode()
- 4.19 M_set_update_mode()
- 4.20 M_get_screen_mode()
- 4.21 M_set_screen_mode()
- 4.22 M_get_screen_info()
- 4.23 M_load_screen()
- 4.24 M_load_bitmap()
- 4.25 M_set_bitmap_trans()
- 4.26 M_blit()
- 4.27 M_copy_screen()
- 4.28 M_set_frames()
- 4.29 M_load_font()
- 4.30 M_font_style()
- 4.31 M_render_text()
- 4.32 M_read_keys()
- 4.33 M_read_mouse()
- 4.34 M_mouse_area()
- 4.35 M_key_debounce()
- 4.36 M_hide_cursor()
- 4.37 M_show_cursor()
- 4.38 M_number_joysticks()

- 4.39 M_get_joystick()
- 4.40 M_load_wave()
- 4.41 M_play_wave()
- 4.42 M_load_music()
- 4.43 M_set_music_volume()
- 4.44 M_play_music()
- 4.45 M_halt_music()
- 4.46 M_wait()
- 4.47 M_ticks()
- 4.48 M_set_window_caption()
- 4.49 M_get_events()
- 4.50 M_update()
- 4.51 M_cleanup()

5 Epilogue and Coda

1. Introduction

Welcome to the first version of Musubi, my attempt at creating an easier way to use SDL and the various SDL extension libraries from Euphoria. Musubi is essentially a graphics/games library for Euphoria built using SDL. The SDL_Wrap header for Euphoria requires a number of machine level peeks and pokes for the many basic operations. That kind of complexity was at odds with the essential Euphoria philosophy of simplicity and elegance. The project to remedy those shortcomings arose from the need to make the SDL library more like Euphoria. Musubi also replaces a number of the Euphoria DOS pixel graphics routines with SDL versions, providing a way to create simple graphics programs for Windows. Musubi doesn't wrap all of the SDL functions, but I've tried to include a set of routines that are comprehensive enough to make a game program or other application. The name Musubi comes from a fondly remembered childhood comfort food. A musubi (or omusubi) is a simple ball of rice with a bit of salt, and in the more traditional Japanese style wrapped in a piece of nori (dried seaweed).

Musubi uses the SDL, SDL_TTF, SDL_img, SDL_Mixer and SDL_GFX libraries

SDL was written by Sam Lantinga and is distributed under the GNU Lesser Public License.
SDL_TTF, SDL_img, SDL_Mixer and SDL_GFX are distributed under the GNU Lesser Public License.

The official SDL website <http://www.libsdl.org> contains more information both general and specific.

2. Programs and Files

The following files are included with the Musubi package:

Musubi.ew – the Musubi library

Musubi_demo_nn.exw – the Euphoria source code for the Musubi demo programs

Musubi_demo_nn.exe – the Musubi demo programs compiled executables

Example_Game_nn.exw – the Euphoria source code for the game example programs

Example_Game_nn.exe – the game examples compiled executables

Audio – a folder containing the music and sound files used by the demo programs

bitmaps – a folder containing the images used by the demo programs

fonts – a folder containing the images used by the demo programs

Docs – a folder containing the Musubi manual in PDF, HTML and RTF formats

includes – a folder containing the Euphoria header files used by Musubi

SDL.dll – the SDL library

SDL_image.dll – the SDL image library

SDL_mixer.dll – the SDL mixer sound and audio library

SDL_ttf.dll – the SDL true type font library

SDL_gfx.dll – the SDL graphics library

jpeg.dll

libpng.dll

ogg.dll

smpeg.dll

vorbis.dll

vorbisfile.dll

zlib.dll – libraries for graphics and sound formats used by SDL_image and SDL_mixer.

3. Using Musubi

To use Musubi you need to know a the basics of Euphoria programming. The easiest way to learn is to examine the source code for the demo the game programs. The included programs have been commented extensively and cover each of the Musubi library functions. In general, a Euphoria/Musubi program should start by calling the `M_init` function and end with by calling `M_cleanup`. If you have a Euphoria program that uses DOS graphics functions, in many cases you can turn it into a Musubi program that runs under Windows by adding the `M_init` and `M_cleanup` calls to the program's beginning and end. I would recommend using `M_set_screen_mode(windowed)` to run your program in a window until it is completely debugged. You can delete the line later to run your program in full screen mode.

4. The Musubi Library Routines

4.1 M_init()

Description: Initialize the Musubi Library

Syntax: M_init()

Comment: You need to call the M_init() procedure before using any of the other Musubi routines.

4.2 graphics_mode()

Description: Change the graphics resolution and bit depth.

Syntax: i=graphics_mode(o)

Comment: If o an integer, it works like the Euphoria DOS graphics_mode function. This method is included for compatibility with the old DOS graphics functions.

o can also be a sequence of 3 integers of the form {x,y,z} where x is the width in pixels, y is the height in pixels and z is the the bit depth. For example the line

```
test=graphics_mode({800,600,16})
```

will set a graphics buffer size of 800x600 pixels with 16 bits of color.

Use the graphics_mode function carefully. If you start by specifying a DOS compatible graphics mode, then you will need to use only comparable integer values for color selections. Likewise if you use a sequence, the color value in subsequent Musubi graphics calls will need to be a sequence in RGB format.

4.3 clear_screen()

Description: Clear the screen.

Syntax: clear_screen()

Comment: The clear_screen function will always clear the current screen to black, since the background color feature is not supported in Musubi.

4.4 pixel()

Description: Plot a pixel or series of pixels to the screen.

Syntax: pixel(o,s)

Comment: This function works similarly to the Euphoria DOS graphics function. S is a sequence in the form {x,y} where x and y are the horizontal and vertical coordinates of the pixel to be plotted. If o is a single integer or color sequence then one point will be plotted. If o is a sequence of either integers or color sequences then a series of pixels will be drawn starting at (x,y).

4.5 draw_line()

Description: Draw a line or series of lines to the screen.

Syntax: draw_line(o,s)

Comment: This function works similarly to the Euphoria DOS graphics function. s is a sequence of coordinates in the form {x,y} where x and y are the horizontal and vertical coordinates of the vertexes to be plotted. o is a single integer or color sequence in RGBA format.

4.6 polygon()

Description: Draw a polygon to the screen.

Syntax: polygon(o,i,s)

Comment: This function works similarly to the Euphoria DOS graphics function. s is a sequence of coordinates in the form {x,y} where x and y are the horizontal and vertical coordinates of the vertexes to be plotted. o is a single integer or color sequence in RGBA format. i is a an integer flag value. If i is 0 then the polygon will not be filled, and if i is non-zero then the polygon will be filled.

4.7 ellipse()

Description: Draw an ellipse to the screen.

Syntax: ellipse(o,i,s1,s2)

Comment: This function works similarly to the Euphoria DOS graphics function. s1 and s2 are sequences of coordinates in the form {x,y} where x and y are the horizontal and vertical coordinates of the upper left and lower right boundaries of the ellipse. o is a single integer or color sequence in RGBA format. i is a an integer flag value. If i is 0 then the ellipse will not be filled, and if i is non-zero then the ellipse will be filled.

4.8 save_screen()

Description: Save the screen or a portion of the screen in a BMP format file.

Syntax: i=save_screen(o,s)

Comment: This function works similarly to the Euphoria DOS graphics function. If o is an integer than the entire screen will be saved. If o is a sequence the 2 coordinate sequences in the form {x,y} where x and y are the horizontal and vertical coordinates of the upper left and lower right boundaries of the screen area to be saved. s the path/file name of the the BMP file to be saved. The function will return a non-zero if the save failed.

4.9 palette()

Description: Change a single color palette.

Syntax: `i1=palette(i2,s)`

Comment: This function works similarly to the Euphoria DOS graphics function. `i2` is index of the color number to be changed. `s` of the form `{r,g,b}` where `r`, `g`, and `b` are the red, green, and blue values of the color. `r`, `g`, and `b` must be between 0 and 63. Musubi does not actually use a 16 or 256 color mode, so programs that use the palette to swap colors may not work the way they would under DOS. If you change colors in the palette, previously drawn colors on screen will not be changed under Musubi. In general the palette routines have been included only for the sake of compatibility, and should be used only sparingly.

4.10 all_palette()

Description: Change a color palette.

Syntax: `all_palette(s)`

Comment: This function works similarly to the Euphoria DOS graphics function. `s` is a sequence of color sequences of the form `{r,g,b}` where `r`, `g`, and `b` are the red, green, and blue values of the color. `r`, `g`, and `b` must be between 0 and 63.

4.11 get_all_palette()

Description: Get the entire palette as a sequence.

Syntax: `s=get_all_palette()`

Comment: This function works similarly to the Euphoria DOS graphics function. The function returns the sequence `s`. `s` is a sequence of color sequences of the form `{r,g,b}` where `r`, `g`, and `b` are the red, green, and blue values of the color. `r`, `g`, and `b` must be between 0 and 63.

4.12 M_hline()

Description: Draw a horizontal line to the screen.

Syntax: `M_hline(o,s)`

Comment: `s` is a sequence of coordinates in the form `{x1,x2,y}` where `x1` and `x2` are the horizontal coordinates and `y` is vertical coordinate of the line to be plotted. `o` is a single integer or color sequence in RGBA format. Using the `M_hline()` procedure is slightly faster than the equivalent `draw_line()`.

4.13 M_vline()

Description: Draw a vertical line to the screen.

Syntax: M_vline(o,s)

Comment: s is a sequence of coordinates in the form {x,y1,y2} where x is the horizontal coordinate and y1 and y2 are the vertical coordinates of the line to be plotted. o is a single integer or color sequence in RGBA format. Using the M_vline() procedure is slightly faster than the equivalent draw_line().

4.14 M_rectangle()

Description: Draw a rectangle to the screen.

Syntax: M_rectangle(o,s)

Comment: s is a sequence of the form {{x1,y1},{x2, y2}} where x1 and x2 are the horizontal coordinates and y1 and y2 are the vertical coordinates of the upper left and lower right corners of the rectangle. o is a single integer or color sequence in RGBA format.

4.15 M_box()

Description: Draw a box (filled rectangle) to the screen.

Syntax: M_box(o,s)

Comment: s is a sequence of the form {{x1,y1},{x2, y2}} where x1 and x2 are the horizontal coordinates and y1 and y2 are the vertical coordinates of the upper left and lower right corners of the box. o is a single integer or color sequence in RGBA format.

4.16 M_bezier()

Description: Draw a bezier curve to the screen.

Syntax: M_bezier(o,i,s)

Comment: s is a sequence of sequences of the form {x,y} where x and y the horizontal and vertical coordinates of points that define the curve. o is a single integer or color sequence in RGBA format. i is an integer that determines how the curve is drawn. In general higher values of i are smoother but take longer to draw.

4.17 M_string()

Description: Draw a text sting to the screen using 8x8 characters.

Syntax: M_string(o,s1,s2)

Comment: s1 is a sequence in the form {x,y} where x and y the horizontal and vertical coordinates of upper left point of where the string will be drawn. o is a single integer or color sequence in RGBA format. s2 is sequence containing the text string to be output.

4.18 M_get_update_mode()

Description: Get the current update mode.

Syntax: i=M_get_update_mode()

Comment: The update mode determines whether Musubi will automatically update and redisplay the screen after performing a graphics routine, or will require manually calling m_update(). The update mode returned in integer i will be either 0 for automatic, or 1 for manual. The auto mode was provided for compatibility with Euphoria DOS graphics. Manual mode allows you to only update and redraw the screen at will, and is much faster (albiet less obvious) than auto mode.

4.19 M_set_update_mode()

Description: Set the current update mode.

Syntax: M_set_update_mode(i)

Comment: Set the current update mode. Integer i can be either 0 for automatic mode, or 1 for manual mode. The procedure also accepts AUTO or MANUAL constants for i.

4.20 M_get_screen_mode()

Description: Get the current screen mode.

Syntax: i=M_get_screen_mode()

Comment: The screen mode determines whether a Musubi program will run in a window or full screen. The mode returned in integer i will be either 16 (hex 00000010) for windowed or 2147483648 (hex 80000000) for full screen. You can also use the WINDOWED or FULLSCREEN constants for comparisons.

4.21 M_set_screen_mode()

Description: Set the current screen mode.

Syntax: M_set_screen_mode(i)

Comment: Set the current screen mode. Integer i can be either 16 (hex 00000010) for windowed mode, or 2147483648 (hex 80000000) for full screen mode. The procedure also accepts WINDOWED or FULLSCREEN constants for i.

4.22 M_get_screen_info()

Description: Get the current screen info.

Syntax: s=M_get_screen_info()

Comment: This function returns the current screen information in sequence s. Sequence s will contain: the SDL_surface, SDL_format, start of screen memory, screen width, and screen height. The SDL pointers were provided for advanced users.

4.23 M_load_screen()

Description: Load a screen from a graphics file.

Syntax: M_load_screen(s)

Comment: Sequence s is the filename/path of the graphics file to be loaded. The loaded image will be displayed at the upper left corner and will be clipped if the image is larger than the screen. The image can be in BMP, JPG, GIF, PCX, or PNG format.

4.24 M_load_bitmap()

Description: Load a bitmap from a graphics file.

Syntax: M_load_bitmap(i,s)

Comment: Integer i is a number between 1 and 512. This number will be used to reference the bitmap in the program until the bitmap is reassigned or cleared. The Sequence s is the filename/path of the graphics file to be loaded. The image can be in BMP, JPG, GIF, PCX, or PNG format.

4.25 M_set_bitmap_trans()

Description: Set a transparent color for a bitmap.

Syntax: M_set_bitmap_trans(i1,i2,i3,i4)

Comment: Integer i1 is the bitmap number to assign a transparency to. Integers i1, i2, and i3 are the red, blue, and green components of the transparency color and are between 0 and 255. For example, to set the transparent color for bitmap number 2 to black you would enter the line:

```
M_set_bitmap_trans(2,0,0,0)
```

4.26 M_blit()

Description: Copy a bitmap or a portion of a bitmap to the screen

Syntax: M_blit(i1,o,i2,i3)

Comment: M_blit() is one the more complex procedures in Musubi. It does a number of different types of blit from bitmap to screen operations. You must have loaded a bitmap and assigned it a number using M_load_bitmap() before you can call M_blit. Integer i1 is the bitmap number to copy from. If o is an integer, then the entire bitmap will be copied. If o is a sequence containing a single integer then the frame specified by the integer will be displayed. If o is a sequence containing 2 sequences, of the form {x,y}, then the sequences define the upper left and lower right corners of the area to be copied. Integers i2, and i3 are the horizontal and vertical screen coordinates to copy the bitmap to. For example:

To copy all of bitmap 3 to the screen at position (100,80) use

```
M_blit(3,0,100,80)
```

To copy bitmap 5, frame 8 to the screen at position (45,128) use

```
M_blit(5,{8},45,128)
```

To copy the area of bitmap 6 bounded by (2,4) and (67,30) to the screen at position (65,43) use

```
M_blit(6,{{2,4},{67,30}},65,43)
```

Frames are defined using the M_set_frames() routine, and are explained in further depth there.

4.27 M_copy_screen()

Description: Copy the screen or a portion of the screen to a bitmap

Syntax: M_copy_screen(i,o)

Comment: Integer i is the bitmap number to copy to. If o is an integer then the entire screen will be copied to the bitmap. If o is a sequence containing 2 sequences of the form {x,y} then the sequences define the upper left and lower right corners of the area to be copied. For example:

To copy all of the screen to bitmap 5 use

```
M_copy_screen(5,0)
```

To copy the area of the screen bounded by (25,34) and (88,90) to bitmap 27 use

```
M_copy(27,{{25,34},{88,90}})
```

4.28 M_set_frames()

Description: Create a series of frames from a bitmap

Syntax: M_set_frames(i,s)

Comment: Creating frames means to subdivide a bitmap into smaller, equally sized regions that you can copy to the screen in the same manner as a bitmap. Integer i is the bitmap number to create frames from. Sequence s is of the form {h,v} where h is the number of horizontal divisions and v is the number of vertical divisions. For example:

Suppose previously defined bitmap 4 contains 8 images in 2 rows. You could create 16 frames from bitmap 4 using:

```
M_set_frames(4,{8,2})
```

To display frame 6 of bitmap 4 at screen position (55,32) you would use

```
M_blit(4,{6},55,32)
```

You can create a simple animation by looping through a series of frames. Frames are also useful for creating map elements that can be indexed by the frame number.

4.29 M_load_font()

Description: Load a true type font

Syntax: M_load_font(i1,s,i2)

Comment: Integer i1 is the font number for the font you are loading. The font number is used to identify the font and must be between 1 and 64 until the font is reassigned or deleted. Sequence s is the filename/path of the font to load. Integer i2 is point size of the font.

4.30 M_font_style()

Description: Change attributes for a font

Syntax: M_font_style(i,s)

Comment: Integer i is the font number that you wish to change attributes. Sequence s contains the constants normal, bold, underline or italic. You can assign a combination of attributes using a '+' sign. For example.

To set font 1 to italic style use:

```
M_font_style(1,{italic})
```

To set font 3 to bold and underline style use:

```
M_font_style(3,{bold+underline})
```

4.31 M_render_text()

Description: Draw a text string using a true type font to the screen

Syntax: M_render_text(i,s1,s2,o)

Comment: Integer i is the number of the previously loaded font to use. Sequence s1 is the text string to be printed. Sequence s2 is in the form {x,y} where x and y are the horizontal and vertical coordinates of the screen position to print the text. Object o is the text color either in integer or constant for DOS compatibility mode or an RGB sequence for user mode.

4.32 M_read_keys()

Description: Read the keyboard

Syntax: s=M_read_keys()

Comment: The M_read_keys() function returns a sequence s that contains the code for every key that is pressed at the time of the call. If no keys are pressed, then M_read_keys() returns an empty sequence. You can search the returned sequence using the Euphoria find function to see if a specific key has been pressed..

4.33 M_read_mouse()

Description: Read the mouse

Syntax: s=M_read_mouse()

Comment: The M_read_mouse() function returns a sequence s of the form {b,x,y} where b is the button status, x is the horizontal position and y is the vertical position. The integer b represents 3 mouse buttons as values 1, 2, and 4. If more than one mouse button is pressed b will contain the sum of all the buttons.

4.34 M_mouse_area()

Description: Test if the mouse is in a specified area

Syntax: i=M_mouse_area(s)

Comment: Sequence s is of the form {x1,y1,x2,y2} where (x1,y1) are the screen coordinates of the upper left corner and (x2,y2) are the coordinates of the lower right corner of the area to be tested. The M_mouse_area() function returns a 1 if the mouse is in the area specified, and 0 otherwise.

4.35 M_key_debounce()

Description: Wait until a specific key has been lifted

Syntax: M_key_debounce(i)

Comment: Integer i is the code for the key to check. This procedure is useful for single key input, in which you want to wait until the key is up before continuing.

4.36 M_hide_cursor()

Description: Hide the mouse cursor

Syntax: M_hide_cursor()

Comment: The M_hide_cursor() routine makes the mouse cursor invisible. Use the M_show_cursor() procedure to display the cursor again.

4.37 M_show_cursor()

Description: Show the mouse cursor

Syntax: M_show_cursor()

Comment: Use the M_show_cursor() makes the mouse cursor visible. You will typically use M_show_cursor() after a previous call to M_hide_cursor().

4.38 M_number_joysticks()

Description: Returns the number of joysticks available on your system

Syntax: i=M_number_joysticks()

Comment: Use the M_number_joysticks() function to find out how many joysticks/game-pad controllers are connected.

4.39 M_get_joystick(integer j)

Description: Get information for a joystick/game-pad controller

Syntax: s=M_get_joystick(i)

Comment: Integer i is the number of the joystick to read. The returned sequence s is of the form {{x,y},{b1,b2,b3,b4,b5,b6,b7,b8}} where x and y are the horizontal and vertical joystick values and b1-b8 are represent 8 possible buttons. A button value will be up if 0 and pressed if 1. For example:

The line

```
jstick=M_get_joystick(1)
```

will return the information for joystick 1

if jstick looks contains {{45,-112},{0,1,0,1,0,0,0,0}} then the x value is 45, the y value is -112 and buttons 2 and 4 are pressed.

4.40 M_load_wave()

Description: Load a sound file in WAV, AIFF, RIFF, or VOC format

Syntax: M_load_wave(i1,s,i2)

Comment: The M_load_wave() routine is used to load sound files. Integer i1 is the sound number and must be between 1 and 256. The sound number is used by the M_play_wave() procedure to reference which sound to play back. String s is the path/filename of the sound file to load. Integer i2 is the sound volume and must be between 0 and 127. For example:

To load a sound file named "boom.wav", using sound number 5, at volume 100 you would use
`M_load_wave(5,"boom.wav",100)`

4.41 M_play_wave()

Description: Play a sound

Syntax: M_play_wave(i)

Comment: The M_load_wave() procedure will play back a previously loaded sound sample. Integer i is the sound number of the sound to play and must be between 1 and 256. For example:

To play a sound number 2 you would use the line
`M_play_wave(2)`

4.42 M_load_music()

Description: Load a music clip in MOD, WAV, MID, OGG, or MP3 format

Syntax: M_load_music(i,s)

Comment: The M_load_music() procedure is used to load music files. Integer i is the the music clip number and must be between 1 and 64. Sequence s is the path/filename of the music clip to be loaded.

4.43 M_set_music_volume()

Description: Set the volume for a music clip

Syntax: M_set_music_volume(i)

Comment: The M_set_music_volume() routine sets the volume for music playback. Integer i is the volume and must be between 0 and 127.

4.44 M_play_music(integer n, integer r)

Description: Play a music clip

Syntax: `M_play_music(i1,i2)`

Comment: The `M_play_music()` routine will play back music clip. Integer `i1` is a previously assigned music clip number. Integer `i2` is number of times to play the clip. If `i2` is -1 then the music clip will repeat indefinitely. For example:

The line

```
M_play_music(2,15)
```

will play back music clip number 2, 15 times.

4.45 M_halt_music()

Description: Halt music clip playback

Syntax: `M_halt_music()`

Comment: The `M_halt_music()` routine will immediately stop any currently playing music clips.

4.46 M_wait()

Description: Pause the program for a given number of milliseconds

Syntax: `M_wait(a)`

Comment: The atom `a` is the number of milliseconds (1/1000 of second) to wait. For example:

To pause the program for 2.5 seconds use the line

```
M_wait(2500)
```

4.47 M_ticks()

Description: Returns the SDL millisecond counter

Syntax: `a=M_ticks()`

Comment: The `M_ticks()` function returns a counter value that is incremented every millisecond. This function is useful for program timing without pausing.

4.48 M_set_window_caption()

Description: Change the window caption

Syntax: M_set_window_caption(s)

Comment: The M_set_window_caption() routine will change the caption text for the program window. Sequence s contains the text string to use as the caption. M_set_window_caption() works only if the screen mode has been set to WINDOWED using M_set_screen_mode().

4.49 M_get_events()

Description: Check and return the window events

Syntax: s=M_get_events()

Comment: The M_get_events() function works in WINDOWED screen mode. The function returns sequence s, which is of the form {q,x,y}. Integer q is 0 if no events took place, 1 if the Quit window button was pressed or 2 if the window was resized. If the window was resized, x and y are the new window dimensions in pixels.

4.50 M_update()

Description: Manually update the screen

Syntax: M_update()

Comment: The M_update() procedure immediately updates the screen. To use M_update you must first change the update mode to MANUAL using M_set_update_mode(). M_update allows you to update the screen only after you've made any changes to you that you want.

4.51 M_cleanup()

Description: Cleanup everything

Syntax: M_cleanup()

Comment: Call the M_cleanup() routine before ending a Musubi program.

5. Epilogue and Coda

Musubi took a bit longer for me to finalize than I would have liked, due to distractions from the other (non-programming) parts of my life. For the next version, I'd like to include additional functionality, and possibly functions to call the SDL_net networking library.

Send comments & feedback to mkakita@juno.com