Pixilang User Manual

2023.02.27

- Graphics
  - frame
  - vsync
  - set_pixel_size
  - get_pixel_size
  - set_screen
  - get_screen
  - set_zbuf
  - get_zbuf
  - clear_zbuf
  - get_color
  - get_red
  - get_green
  - get_blue
  - get_blend
  - transp
  - get_transp
  - clear
  - dot
  - dot3d
  - get_dot
  - get_dot3d
  - line
  - line3d
  - box
  - fbox
  - pixi
  - triangles3d
  - sort_triangles3d
  - set_key_color
  - get_key_color
  - set_alpha
  - get_alpha
  - print
  - get_text_xsize
  - get_text_ysize
  - get_text_xysize
  - set_font
  - get_font
  - effector
  - color_gradient
  - split_rgb
  - split_ycbcr
- OpenGL base
  - set_gl_callback
  - remove_gl_data
  - update_gl_data
  - gl_draw_arrays
  - gl_blend_func
  - gl_bind_framebuffer
  - gl_bind_texture
  - gl_get_int
  - gl_get_float
- OpenGL shaders
  - gl_new_prog
  - gl_use_prog

What is Pixilang

Pixilang is a pixel-oriented programming language for small graphics/sound applications and experiments: demos, games, synths, etc.

Pixilang programs are stored in text files (UTF-8 encoding) with extensions .txt or .pixi. So you can use your favorite text editor to create/edit these files. Pixilang has no built-in editor.

A program can be a simple list of instructions with conditional jumps, without declaring entry points and functions. A blank area (screen) inside the Pixilang window is provided for any program. The screen can be accessed as an array of pixels.

Command line options

```
pixilang [options] [filename] [arg]
Options:
 -?           show help
 clearall     reset all settings
 -cfg <config> apply additional configuration in the following format:
              "OPTION1=VALUE|OPTION2=VALUE|..."
 -c           generate bytecode; filename.pixicode file will be produced
Filename: source (UTF-8 text file) or bytecode (*.pixicode).
Additional arguments (arg): some options for the Pixilang program.
```

Basics

The basis of the Pixilang - containers and variables.

Container is a two-dimensional array of elements (numbers) of the same data type. Different containers may have different data types. Each container has its own unique ID number. ID is always a positive integer or zero: 0,1,2,3,4... Zero ID is ok, but it's the screen container number by default.

Container structure:

- two-dimensional table of container elements - it may be some image, text string, piece of audio, or other data;
- key color, which will be treated as transparent color (with alpha=0);
- reference to a container with alpha-channel;
- additional data:
    - properties (use get_prop(), set_prop(), remove_prop, remove_props(), or a . (dot) operator to access them);
    - animation (use Animation functions to access it).

The new container can be created with the new() function or returned by some other function. There are two ways to remove the container:

- by user through the remove() function;
- automatically after the end of the program.

There is no automatic garbage collector: each new container takes up memory until you delete it or until the program terminates.

Maximum number of containers is 8192. But you can increase it using the parameter pixi_containers_num in pixilang_config.ini. Also you can change this number in the app_info.cpp (if you compile Pixilang from source) through the following code:

```
app_option g_app_options[] = { { "pixi_containers_num", 16384 }, { NULL } };
```

Variable can contain 32-bit signed integer or 32-bit floating point number.

Numbers can be described in different formats. Examples:

- 33 - decimal;
- 33.55 - decimal floating point;
- 0xA8BC - hexadecimal;
- 0b100101011 - binary;
- #FF9080 - color, as in HTML; base format is #RRGGBB, where RR - red intensity (from 00 to FF), GG - green intensity; BB - blue intensity; the actual color format (bit arrangement in the value)

depends on the system, so always use form #RRGGBB or get_color(R,G,B) to get the correct color value.

Examples

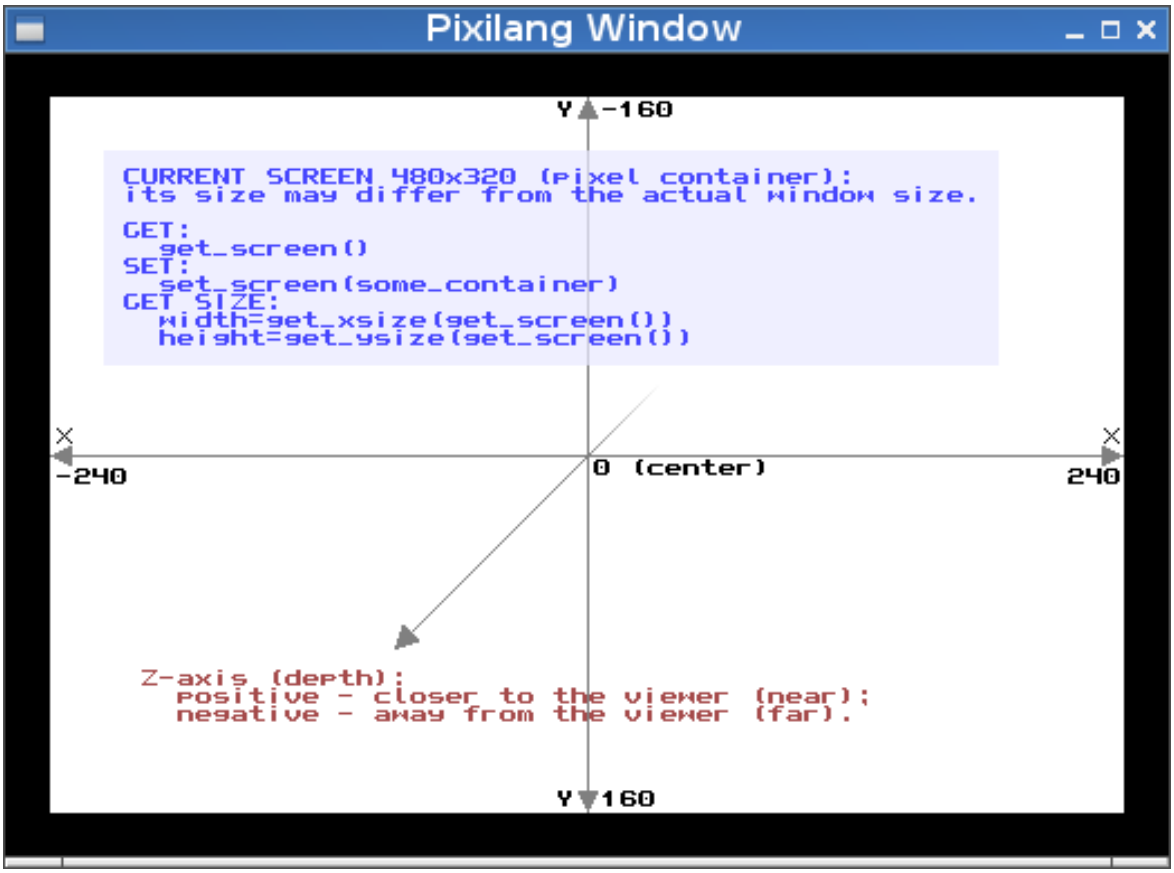
```
x = new( 4 ) //Create a new container with 4 pixels. Store its ID into the variable x.
/*
Container elements are addressed from zero:
  first element - x[ 0 ];
  second element - x[ 1 ];
    ...
  last element - x[ container_size - 1 ]
*/
x[ 0 ] = #0000FF //Assign blue color to the first pixel of container x.
x[ 2 ] = WHITE //Assign white color to pixel 2.
x[ 3 ] = RED //Assign red color to pixel 3 (last element).
x[ 4 ] = BLUE //Nothing will happen here because index 4 is out of range.
b = x[ 2 ] //Read an element with index 2.
//Here b contains the number corresponding to the white color.
c = x[ 8 ] //Try to read an element with index 8
//Here c contains a null value because index 8 is not available.
remove( x ) //Remove the container
```

```
c = new( 4, 4 ) //Create a new 2D 4x4 pixel container. Store its ID into the variable x.
c[ 2, 2 ] = WHITE //Assign the white color to a pixel with coordinates x=2 y=2.
remove( c ) //Remove this container
```

```
str = "Hello" //"Hello" is the string container with five 8-bit characters (UTF-8 encoding).
//This container is automatically created during the bytecode generation.
//You don't need to delete it manually.
str[ 0 ] = 'h' //Change the first character from 'H' to 'h'.
print( str, 0, 0 ) //will print the word "hello" on the screen
//The string can be terminated with a null character or the end of a container.
str[ 2 ] = 0 //Replace the char 'l' with a zero
print( str, 0, 8 ) //will print the word "he" on the screen
```

```
a = 4 //Global variable
fn function()
{
  $k = 2 //Local variable
  function2 = {
    //Body of another function
    $x = 899.334 //Local variable
    //$k is not accessible here
  }
  //$x is not accessible here
}
//$k and $x are not accessible here
```

Coordinate system of the Pixilang is described on the following image:



Paths and file names

```
//File is in the same folder with the pixi-program:
"filename"

//File is in the current working directory:
//  Linux/Windows/OSX: in the same folder with Pixilang;
//  iOS: documents;
//  WinCE: root of local filesystem (/);
"1:/filename"

//File is in the user directory for configs and caches :
//  Linux: /home/username/.config/Pixilang;
//  iOS: directory with application support files (NSApplicationSupportDirectory);
//  Android: internal files directory;
"2:/filename"

//File is in the temporary directory:
"3:/filename"
```

Operators

```
//if, else
if a == b
  { /*This code is executed if a equals b*/ }
else
  { /*This code is executed otherwise*/ }
if x == 4 && y == 2
  { /*This code is executed if x equals 4 and y equals 2*/ }

//while
a = 0
while( a < 3 )
{
  //This code is executed while a less than 3
  a + 3
}

//while + break
a = 0
while( a < 100 )
{
  if a == 10 { break } //If a = 10, break the loop
  //Use breakX operator to break X nested loops. Example: break2
  //Use breakall operator to break all current active loops
  a + 1
}

//while + continue
a = 0
b = 0
while( a < 100 )
{
  if a == 10 { a + 1 continue } //If a = 10, go to the next loop iteration
                               //           (ignore the next two lines of code)
  a + 1
  b + 1
}

//for( initialization ; condition ; increment ) { loop body }
for( i = 0; i < 4; i + 1 )
{
  printf( "i=%d\n", i )
}

//go, goto
m1:
a + 1
goto m1 //Go to the m1 label

//halt, stop
halt //Stop the program here

//include
include "prog2.txt" //Include code from prog2.txt

//fn
fn fff( $x, $y ) //Declare function fff with parameters $x and $y
{
  //Function fff body
  ret //Simple return from the function
  ret( 4 ) //Return with value 4
}
```

The following table lists the mathematical operators. Priority 0 - the highest, such operations will be executed first.

Priority	Operator	Description	Result	Example
0	%	Modulo (remainder)	Integer	a = b % 4

0	/	Division	Floating point	a = b / 4
0	div	Integer division	Integer	a = b div 4
0	*	Multiplication	Depends on the operands	a = b * 4
0	!	Logical NOT	Integer 1 or 0	if !condition { ... }
0	~	Bitwise NOT (inversion)	Integer	a = ~b
1	+	Addition	Depends on the operands	a = b + 4
1	-	Subtraction	Depends on the operands	a = b - 4
2	>>	Bitwise right shift	Integer	a = b >> 4
2	<<	Bitwise left shift	Integer	a = b << 4
3	==	Equal to	Integer 1 or 0	if a == b {}
3	!=	Not equal to	Integer 1 or 0	if a != b {}
3	<	Less than	Integer 1 or 0	if a < b {}
3	>	Greater than	Integer 1 or 0	if a > b {}
3	<=	Less than or equal to	Integer 1 or 0	if a <= b {}
3	>=	Greater than or equal to	Integer 1 or 0	if a >= b {}
4	|	Bitwise OR	Integer	a = b | 4
4	^	Bitwise XOR	Integer	a = b ^ 4
4	&	Bitwise AND	Integer	a = b & 4
5	||	Logical OR	Integer 1 or 0	if a || b {}
5	&&	Logical AND	Integer 1 or 0	if a && b {}

Built-in constants

Container types

64-bit types are not supported by default. But they can be enabled in pixilang.h for the custom build of Pixilang.

- INT - signed integer (size depends on the version of Pixilang);

- INT8 - signed integer (8 bit); range: -128 to 127;
- INT16 - signed integer (16 bit); range: -32768 to 32767;
- INT32 - signed integer (32 bit); range: -2147483648 to 2147483647;
- INT64 - signed integer (64 bit);
- FLOAT - floating point (size depends on the version of Pixilang);
- FLOAT32 - floating point (32 bit);
- FLOAT64 - floating point (64 bit);
- PIXEL - pixel in native color format; will be converted to type INTx, where x - number of bits per pixel.

Container flags

- CFLAG_INTERP - enable software interpolation.

For OpenGL acceleration:

- GL_MIN_LINEAR;
- GL_MAG_LINEAR;
- GL_NICEST - use 32bit color, when possible;
- GL_NO_XREPEAT;
- GL_NO_YREPEAT;
- GL_NO_ALPHA;
- GL_NPOT - set it if you want to use OpenGL texture (inside the container) whose dimensions are not limited to a power of two; on iOS this flag can only be enabled in conjunction with GL_NO_XREPEAT | GL_NO_YREPEAT;

Container resizing flags

- RESIZE_INTERP1 - rough resizing;
- RESIZE_INTERP2 - resize with linear interpolation;
- RESIZE_UNSIGNED_INTERP2;
- RESIZE_COLOR_INTERP1;
- RESIZE_COLOR_INTERP2.

Container copying flags

- COPY_NO_AUTOROTATE - don't rotate pixels from GL_SCREEN;
- COPY_CLIPPING - check and limit the values, if the type of the source container differs from the destination.

ZLib

Compression levels:

- Z_NO_COMPRESSION;
- Z_BEST_SPEED;
- Z_BEST_COMPRESSION;
- Z_DEFAULT_COMPRESSION.

File formats

R - reading is supported; W - writing is supported.

- FORMAT_RAW - raw data without a header (RW);
- FORMAT_WAVE (RW);
- FORMAT_AIFF (R);
- FORMAT_OGG;

- FORMAT_MP3;
- FORMAT_FLAC;
- FORMAT_MIDI (R; only with sv_load());
- FORMAT_SUNVOX (RW; only with sv_load() and sv_save());
- FORMAT_SUNVOXMODULE (R; only with sv_load_module());
- FORMAT_XM (R; only with sv_load());
- FORMAT_MOD (R; only with sv_load());
- FORMAT_JPEG (RW);
- FORMAT_PNG (RW);
- FORMAT_GIF (RW);
- FORMAT_AVI;
- FORMAT_MP4;
- FORMAT_ZIP;
- FORMAT_PIXICONTAINER - entire Pixilang container with properties and animation (RW).

Load/Save options

- LOAD_FIRST_FRAME - load first frame only.

GIF saving:

- GIF_GRAYSCALE;
- GIF_DITHER.

JPEG saving:

- JPEG_H1V1 - YCbCr, no subsampling (H1V1, YCbCr 1x1x1, 3 blocks per MCU);
- JPEG_H2V1 - YCbCr, H2V1 subsampling (YCbCr 2x1x1, 4 blocks per MCU);
- JPEG_H2V2 - YCbCr, H2V2 subsampling (YCbCr 4x1x1, 6 blocks per MCU); default;
- JPEG_TWOPASS - number of passes = 2.

Colors

- ORANGE;
- BLACK;
- WHITE;
- YELLOW;
- RED;
- GREEN;
- BLUE;

Alignment

- TOP;
- BOTTOM;
- LEFT;
- RIGHT.

Effects

Effect types for effector() function:

- EFF_NOISE - noise;
- EFF_SPREAD_LEFT - random pixel shift to the left;
- EFF_SPREAD_RIGHT - random pixel shift to the right;
- EFF_SPREAD_UP - random pixel shift to the top;
- EFF_SPREAD_DOWN - random pixel shift to the bottom;

- EFF_HBLUR - horizontal blur;
- EFF_VBLUR - vertical blur;
- EFF_COLOR - color fill.

OpenGL

gl_draw_arrays() (analog of the glDrawArrays()) modes:

- GL_POINTS;
- GL_LINE_STRIP;
- GL_LINE_LOOP;
- GL_LINES;
- GL_TRIANGLE_STRIP;
- GL_TRIANGLE_FAN;
- GL_TRIANGLES.

gl_blend_func() (analog of the glBlendFunc()) operations:

- GL_ZERO;
- GL_ONE;
- GL_SRC_COLOR;
- GL_ONE_MINUS_SRC_COLOR;
- GL_DST_COLOR;
- GL_ONE_MINUS_DST_COLOR;
- GL_SRC_ALPHA;
- GL_ONE_MINUS_SRC_ALPHA;
- GL_DST_ALPHA;
- GL_ONE_MINUS_DST_ALPHA;
- GL_SRC_ALPHA_SATURATE.

gl_bind_framebuffer() options:

- GL_BFB_IDENTITY_MATRIX - use the identity matrix (work in left-handed normalized device coordinates (NDC)).

State variables (for gl_get_int() and gl_get_float()):

- GL_MAX_TEXTURE_SIZE;
- GL_MAX_VERTEX_ATTRIBS;
- GL_MAX_VERTEX_UNIFORM_VECTORS;
- GL_MAX_VARYING_VECTORS;
- GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS;
- GL_MAX_TEXTURE_IMAGE_UNITS;
- GL_MAX_FRAGMENT_UNIFORM_VECTORS.

Default shader names for the gl_new_prog():

- GL_SHADER_SOLID - solid color;
- GL_SHADER_GRAD - gradient color;
- GL_SHADER_TEX_ALPHA_SOLID - solid color + one channel (opacity) texture;
- GL_SHADER_TEX_ALPHA_GRAD - gradient color + one channel (opacity) texture;
- GL_SHADER_TEX_RGB_SOLID - solid color + texture;
- GL_SHADER_TEX_RGB_GRAD - gradient color + texture.

Global OpenGL containers:

- GL_SCREEN; you can't read pixels from GL_SCREEN directly, but you can use copy(destination, GL_SCREEN) for this;

- GL_ZBUF.

Audio

Flags for set_audio_callback() function:

- AUDIO_FLAG_INTERP2 - linear interpolation (2 points).

MIDI

Flags for midi_get_device(), midi_open_port() functions:

- MIDI_PORT_READ;
- MIDI_PORT_WRITE.

Events

- EVT - ID of the container with event, which used by get_event() function.

EVT field numbers:

- EVT_TYPE - event type;
- EVT_FLAGS - event flags;
- EVT_TIME - event time;
- EVT_X (0 - center of the screen);
- EVT_Y (0 - center of the screen);
- EVT_KEY - ASCII code of pressed key, or one of the KEY_* constants;
- EVT_SCANCODE - key scancode, or touch number for multitouch devices;
- EVT_PRESSURE - touch pressure (normal - 1024).

Event types (for the EVT_TYPE field):

- EVT_MOUSEBUTTONDOWN - first touch;
- EVT_MOUSEBUTTONUP;
- EVT_MOUSEMOVE;
- EVT_TOUCHBEGIN - 2th, 3th etc. touch; only for multitouch devices;
- EVT_TOUCHEND;
- EVT_TOUCHMOVE;
- EVT_BUTTONDOWN;
- EVT_BUTTONUP;
- EVT_SCREENRESIZE;
- EVT_QUIT - Virtual Machine will be closed; if EVT[EVT_SCANCODE] = 0, this event can't be ignored, user code must be terminated as soon as possible; if EVT[EVT_SCANCODE] = 1, this event can be ignored by user.

Event flags (for the EVT_FLAGS field):

- EVT_FLAG_SHIFT;
- EVT_FLAG_CTRL;
- EVT_FLAG_ALT;
- EVT_FLAG_MODE;
- EVT_FLAG_MODS (mask with all modifiers like Shift, Ctrl, etc.);
- EVT_FLAG_DOUBLECLICK.

Event key codes (for the EVT_KEY field):

- KEY_MOUSE_LEFT;
- KEY_MOUSE_MIDDLE;

- KEY_MOUSE_RIGHT;
- KEY_MOUSE_SCROLLUP;
- KEY_MOUSE_SCROLLDOWN;
- KEY_BACKSPACE;
- KEY_TAB;
- KEY_ENTER;
- KEY_ESCAPE;
- KEY_SPACE;
- KEY_F1;
- KEY_F2;
- KEY_F3;
- KEY_F4;
- KEY_F5;
- KEY_F6;
- KEY_F7;
- KEY_F8;
- KEY_F9;
- KEY_F10;
- KEY_F11;
- KEY_F12;
- KEY_UP;
- KEY_DOWN;
- KEY_LEFT;
- KEY_RIGHT;
- KEY_INSERT;
- KEY_DELETE;
- KEY_HOME;
- KEY_END;
- KEY_PAGEUP;
- KEY_PAGEDOWN;
- KEY_CAPS;
- KEY_SHIFT;
- KEY_CTRL;
- KEY_ALT;
- KEY_MENU;
- KEY_UNKNOWN (system virtual key code = code - KEY_UNKNOWN).

Constants for the set_quit_action() function:

- QA_NONE;
- QA_CLOSE_VM.

Threads

Flags for thread_create():

- THREAD_FLAG_AUTO_DESTROY - thread with this flag will be destroyed automatically.

Mathematical constants

- M_E - e.
- M_LOG2E - $\log_2 e$.
- M_LOG10E - $\log_{10} e$.
- M_LN2 - $\log_e 2$.
- M_LN10 - $\log_e 10$.
- M_PI - PI.
- M_2_SQRTPI - 2 / sqrt(PI).

- M_SQRT2 - sqrt(2).
- M_SQRT1_2 - 1 / sqrt(2).

Data processing operations

For op_cn() function:

- OP_MIN - op_cn() return value = min(C1[i], C1[i + 1], ...);
- OP_MAX - op_cn() return value = max(C1[i], C1[i + 1], ...);
- OP_MAXABS - op_cn() return value = max(| C1[i] + N |, | C1[i + 1] + N |, ...);
- OP_SUM - op_cn() return value = C1[i] + C1[i + 1] + ... ;
- OP_LIMIT_TOP - if C1[i] > N { C1[i] = N } ;
- OP_LIMIT_BOTTOM - if C1[i] < N { C1[i] = N } ;
- OP_ABS - absolute value;
- OP_SUB2 - subtraction with reverse order of the operands (N - C1[i]);
- OP_COLOR_SUB2 - color subtraction with reverse order of the operands (N - C1[i]);
- OP_DIV2 - division with reverse order of the operands (N / C1[i]);
- OP_H_INTEGRAL - running sum (horizontal);
- OP_V_INTEGRAL - running sum (vertical);
- OP_H_DERIVATIVE - first difference (horizontal);
- OP_V_DERIVATIVE - first difference (vertical);
- OP_H_FLIP;
- OP_V_FLIP.

For op_cn(), op_cc() function:

- OP_ADD - addition;
- OP_SADD - addition with saturation;
- OP_COLOR_ADD - color addition;
- OP_SUB - subtraction;
- OP_SSUB - subtraction with saturation;
- OP_COLOR_SUB - color subtraction;
- OP_MUL - multiplication;
- OP_SMUL - multiplication with saturation;
- OP_MUL_RSHIFT15 - (C1[i] * N) >> 15;
- OP_COLOR_MUL - color multiplication;
- OP_DIV - division;
- OP_COLOR_DIV - color division;
- OP_AND - bitwise AND;
- OP_OR - bitwise OR;
- OP_XOR - bitwise XOR;
- OP_LSHIFT - bitwise left shift;
- OP_RSHIFT - bitwise right shift;
- OP_EQUAL;
- OP_LESS;
- OP_GREATER;
- OP_COPY - copy;
- OP_COPY_LESS - copy only if C1[i] < C2[i];
- OP_COPY_GREATER - copy only if C1[i] > C2[i].

For op_cc() function:

- OP_BMUL - if C2[i] == 0 { C1[i] = 0 };
- OP_EXCHANGE;
- OP_COMPARE - comparison; return value: 0 indicates that the contents are equal; 1 indicates that the first element that does not match in both containers has a greater value in C1 than in C2; -1 indicates the opposite.

For op_ccn() function:

- OP_MUL_DIV - C1[i] * C2[i] / N;
- OP_MUL_RSHIFT - (C1[i] * C2[i]) >> N (multiplication with bitwise right shift).

For generator() function:

- OP_SIN - sine;
- OP_SIN8 - fast, but rough sine (8bit table);
- OP_RAND - pseudo random (from -amp to +amp).

Sampler

- SMP_INFO_SIZE - size of the container with sample info.

Sample info field numbers:

- SMP_DEST - destination container;
- SMP_DEST_OFF - destination offset;
- SMP_DEST_LEN - destination length;
- SMP_SRC - container with sample;
- SMP_SRC_OFF_H - sample offset (left part of fixed point value);
- SMP_SRC_OFF_L - sample offset (right part of fixed point value from 0 to 65535);
- SMP_SRC_SIZE - sample size (or 0 for whole sample);
- SMP_LOOP - loop start;
- SMP_LOOP_LEN - loop length (or 0, if no loop);
- SMP_VOL1 - start volume (32768 = 1.0);
- SMP_VOL2 - end volume (32768 = 1.0);
- SMP_DELTA - delta (playing speed); fixed point (real_value * 65536);
- SMP_FLAGS - flags.

Sample info flags:

- SMP_FLAG_INTERP2 - linear interpolation (2 points);
- SMP_FLAG_INTERP4 - cubic spline interpolation (4 points);
- SMP_FLAG_PINGPONG - ping-pong loop;
- SMP_FLAG_REVERSE - reverse direction.

Convolution filter flags

- CONV_FILTER_COLOR;
- CONV_FILTER_BORDER_EXTEND;
- CONV_FILTER_BORDER_SKIP;
- CONV_FILTER_UNSIGNED.

File dialog options

- FDIALOG_FLAG_LOAD.

Native code

- CCONV_DEFAULT;
- CCONV_CDECL;
- CCONV_STDCALL;
- CCONV_UNIX_AMD64;
- CCONV_WIN64.

POSIX

- FOPEN_MAX;
- SEEK_CUR;
- SEEK_END;
- SEEK_SET;
- EOF;
- STDIN;
- STDOUT;
- STDERR.

Pixilang info flags

- PIXINFO_MULTITOUCH;
- PIXINFO_TOUCHCONTROL;
- PIXINFO_NOWINDOW;
- PIXINFO_MIDIIN;
- PIXINFO_MIDIOUT.

Misc

- PIXILANG_VERSION - Pixilang version ((major<<24) + (minor<<16) + (minor2<<16) + minor3);
  example: PIXILANG_VERSION = 0x03040700 for v3.4.7;
- OS_NAME - container with system name;
  examples: "ios", "win32", "linux";
- ARCH_NAME - container with architecture name;
  examples: "x86", "x86_64", "arm", "mips";
- LANG_NAME - container with the name of current system language (in POSIX format [language][_TERRITORY][.CODESET][@modifier]);
  examples: "en_US", "ru_RU.utf8";
- CURRENT_PATH;
- USER_PATH;
- TEMP_PATH;
- OPENGL - 1 if OpenGL available; 0 - otherwise;
- INT_SIZE - max size (in bytes) of the signed integer;
- FLOAT_SIZE - max size (in bytes) of the floating point number;
- INT_MAX - max positive integer;
- COLORBITS - number of bits per pixel.

Built-in global variables

- WINDOW_XSIZE - window width (in pixels);
- WINDOW_YSIZE - window height (in pixels);
- WINDOW_ZSIZE - window depth (for OpenGL); sets the range of Z values: from -WINDOW_ZSIZE (far) to WINDOW_ZSIZE (near); this is a constant in the current version of Pixilang;
- WINDOW_SAFE_AREA_X;
- WINDOW_SAFE_AREA_Y;
- WINDOW_SAFE_AREA_W;
- WINDOW_SAFE_AREA_H;
- FPS - frames per second;
- PPI - pixels per inch;
- UI_SCALE - UI scale (defined by user in the global preferences); use it to scale your UI elements; for example: button_size = PPI * UI_SCALE * 0.5;
- UI_FONT_SCALE - similar to UI_SCALE, but only for the text size;
- PIXILANG_INFO - information (set of flags PIXINFO_*) about the current Pixilang runtime

environment.

Reserved container properties

These properties can be used during playback, saving and loading of the containers:

- file_format;
- sample_rate;
- channels;
- loop_start - starting point (frame number) of the sample loop;
- loop_len - sample loop length (number of frames);
- loop_type - sample loop type: 0-none; 1-normal; 2-bidirectional;
- frames - number of frames;
- frame - current frame number;
- fps - frames per second;
- play - auto-play status (0/1);
- repeat - repeat count (-1 - infinitely);
- start_time - start time (in system ticks); automatically set by play();
- start_frame - start frame; automatically set by play().

Built-in functions

Containers (memory management)

new()

Create a new data container.

Note: immediately after its creation, the container may contain some random values. You should clean it up or fill it with useful data.

Parameters (xsize, ysize, type)

- xsize - width.
- ysize - height.
- type - type of the atomic element of the container. Valid values:
  - INT - signed integer (size depends on the version of Pixilang);
  - INT8 - signed integer (8 bit);
  - INT16 - signed integer (16 bit);
  - INT32 - signed integer (32 bit);
  - INT64 - signed integer (64 bit);
  - FLOAT - floating point (size depends on the version of Pixilang);
  - FLOAT32 - floating point (32 bit);
  - FLOAT64 - floating point (64 bit);
  - PIXEL - pixel; will be converted to type INTx, where x - number of bits per pixel;

Return value: ID of the container, or -1 (error).

Examples

```
p = new() //Create 1x1 container. Element type = pixel.
p = new( 4 ) //Create 4x1 container. Element type = pixel.
p = new( 4, 4 ) //Create 4x4 container. Element type = pixel.
p = new( 4, 4, INT32 ) //Create 4x4 container. Element type = INT32.
```

remove()

Remove a container.

Parameters (pixi)

- pixi - container ID.

Examples

```
p = new() //Create new container
remove( p ) //Remove it
```

remove_with_alpha()

Remove the container and its alpha channel (linked container).

Parameters (pixi)

- pixi - container ID.

resize()

Resize a container.

Parameters (pixi, xsize, ysize, type, flags)

- pixi - container ID;
- xsize - new width (or -1 if width not changed);
- ysize - height (or -1 if height not changed); optional;
- type - type of the atomic element of the container (or -1 if type not changed); optional; valid values:
    - INT8 - signed integer (8 bit);
    - INT16 - signed integer (16 bit);
    - INT32 - signed integer (32 bit);
    - INT64 - signed integer (64 bit);
    - FLOAT32 - floating point (32 bit);
    - FLOAT64 - floating point (64 bit);
    - PIXEL - pixel; will be converted to type INTx, where x - number of bits per pixel;
- flags - resizing flags (RESIZE_*); optional.

Return value: 0 - successful; 1 - error.

Examples

```
p = new( 4, 4 ) //Create new container
resize( p, 32, 32 ) //Resize it from 4x4 to 32x1
resize( p, -1, 64 ) //Resize it from 32x32 to 32x64
remove( p ) //Remove
```

rotate()

Rotate the container by angle*90 degrees (clockwise).

Параметры (pixi, angle)

convert_type()

Convert the type of a container.

Parameters (pixi, new_type)

clean()

Clean a container (fill with zeroes or with selected values).

Parameters (dest_cont, v, offset, count)

- dest_cont - container ID;
- v - value; optional; default - 0;
- offset - offset in dest_cont; optional; default - 0;
- count - number of elements to fill; optional; default - whole container.

Examples

```
p = new() //Create new container
clean( p ) //Clean with zero
clean( p, 3 )
clean( p, 3, 0, 24 ) //Fill 24 elements with value 3
remove( p ) //Remove
```

clone()

Make a duplicate of the container.

Parameters (pixi)

- pixi - container ID.

Return value: ID of the new container, or -1 (error).

copy()

Copy container elements from src to dest.

Parameters (dest, src, dest_offset, src_offset, count, dest_step, src_step, flags)

- dest - destination container;
- src - source container;
- dest_offset - offset in destination (first element to copy);
- src_offset - offset in source (first element to copy);
- count - number of elements to copy;
- dest_step - destination writing step (default - 1);
- src_step - source reading step (default - 1);
- flags - COPY_* flags; optional.

Examples

```
//Copy all elements from img1 to img2:
copy( img2, img1 )
//Copy elements 8...200 from img1 to img2:
copy( img2, img1, 8, 8, 200 )
//Copy elements 8...400 from img1 to img2 with step 2:
copy( img2, img1, 8, 8, 200, 2, 2 )
```

Return value: number of copied elements.

get_size()

Get size of a container (number of elements).

Parameters (pixi)

- pixi - container ID.

Examples

```
p = new( 8, 8 ) //Create a new container 8x8
size = get_size( p ) //Save its size to the "size" variable
remove( p )
```

get_xsize()

Get width of a container.

Parameters (pixi)

- pixi - container ID.

Examples

```
p = new( 8, 8 ) //Create a new container 8x8
xsize = get_xsize( p ) //Save its width to the "xsize" variable
remove( p )
```

get_ysize()

Get height of a container.

Parameters (pixi)

- pixi - container ID.

Examples

```
p = new( 8, 8 ) //Create a new container 8x8
ysize = get_xsize( p ) //Save its height to the "ysize" variable
remove( p )
```

get_esize()

Get the size of the element of a container (in bytes).

Parameters (pixi)

- pixi - container ID.

Examples

```
p = new( 8, 8, INT32 ) //Create a new container 8x8; element type = INT32
esize = get_esize( p ) //Save its element's size to the "esize" variable
//Now esize = 4.
remove( p )
```

get_type()

Get the type of the element of a container

Parameters (pixi)

- pixi - container ID.

Examples

```
p = new( 8, 8, FLOAT32 ) //Create a new container 8x8; element type = FLOAT32
type = get_type( p ) //Save its element's type to the "type" variable
//Now type = FLOAT32.
remove( p )
```

get_flags()

Get container flags.

Parameters (pixi)

Return value: container flags.

set_flags()

Set container flags.

Parameters (pixi, flags)

Examples

```
set_flags( img, GL_MIN_LINEAR | GL_MAG_LINEAR )
```

reset_flags()

Reset container flags.

Parameters (pixi, flags)

get_prop()

Get property value of the container. Each container can have unlimited number of properties.

Another way to get a property - use a . (dot) operator. Example: value = image.fps

Parameters (pixi, prop_name, def_value)

- pixi - container ID;
- prop_name - property name;
- def_vaule - default value, if the property is not exists; optional.

Return value: value of selected property.

set_prop()

Set property value of the container.

Another way to set a property - use a . (dot) operator. Example: image.fps = 30

Parameters (pixi, prop_name, value)

- pixi - container ID;
- prop_name - property name;
- value.

Examples

```
p = new( 8, 8, INT32 ) //Create a new container
set_prop( p, "speed", 777 ) //Add "speed" property to this container
v = get_prop( p, "speed" ) //Read the value of "speed" property
//Or more simple way:
p.speed = 777
v = p.speed
```

remove_prop()

Remove the container property.

Parameters (pixi, prop_name)

- pixi - container ID;
- prop_name - property name.

remove_props()

Remove all properties of the selected container.

Parameters (pixi)

get_proplist()

Get a list (array of strings) of properties for the specified container.

Parameters (pixi)

remove_proplist()

Remove the list of properties.

Parameters (list)

show_memory_debug_messages

Parameters (enable)

zlib_pack()

Parameters (source, level)

- source - container that will be compressed;
- level - Zlib compression level (Z_NO_COMPRESSION, Z_BEST_SPEED, Z_BEST_COMPRESSION, Z_DEFAULT_COMPRESSION); optional.

Return value: ID of the new compressed container (created from the source) or -1 in case of error.

zlib_unpack()

Parameters (source)

- source - container that will be decompressed.

Return value: ID of the new decompressed container (created from the source) or -1 in case of error.

Strings

num_to_str()

Aliases: num2str.

Convert number to string.

Parameters (str, num, radix, str_offset, no_null_term)

- str - container for the string;
- num - numeric value;
- radix; supported values: 10 and 16; any other value is treated as 10; optional;
- str_offset - offset within the string; optional;
- no_null_term - no NULL terminator at the end; optional.

Examples

```
v = 45.64
s = ""
num_to_str( s, v )
fputs( s ) fputs( "\n" )
```

str_to_num()

Aliases: str2num.

Convert string to number.

Parameters (str, str_offset, len)

- str - container with the string;
- str_offset - offset within the string; optional;
- len - length of substring with number; optional.

Return value: numeric value.

Examples

```
a = str_to_num( "-55.44" )
b = a + 4
```

strcat()

Appends a copy of the source string to the destination string. Both strings can be with terminating null character or without it (if the size of the container = number of characters in the string). Size of the source string can be changed after this function executes.

Parameters (destination, source)

Parameters (dest, dest_offset, source, source_offset)

strcmp()

Compares the string str1 to the string str2. Both strings can be with terminating null character or without it (if the size of the container = number of characters in the string).

Parameters (str1, str2)

Parameters (str1, str1_offset, str2, str2_offset)

Return value: a zero value indicates that both strings are equal; a value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; and a value less than zero indicates the opposite.

strlen()

Returns the length of string str. String can be with terminating null character or without it (if the size of the container = number of characters in the string).

Parameters (str)

Parameters (str, str_offset)

Return value: length of string str.

strstr()

Returns the offset of the first occurrence of str2 in str1, or a -1 if str2 is not part of str1.

Parameters (str1, str2)

Parameters (str1, str1_offset, str2, str2_offset)

Return value: offset of the first occurrence of str2 in str1, or a -1 if str2 is not part of str1.

sprintf()

Writes into the container str a string consisting on a sequence of data formatted as the format argument specifies. Detailed format description can be found here. The size of the str container will be increased automatically if there is not enough space.

Parameters (str, format, ...)

Return value: on success, the total number of characters written is returned; on failure, a negative number is returned.

Examples

```
sprintf( str, "Some text\n" ) //Just write some text to the str
sprintf( str, "Number: %d\n", 12 ) //Write signed decimal integer
```

sprintf2()

Same as sprintf() but with additional parameters.

Parameters (str, str_offset, no_null_term, format, ...)

- str - destination container;
- str_offset - offset relative to the start of container str;
- no_null_term - if not zero, then the null character will not be written at the end of the formatted string;
- format - string that contains the text to be written to str; it may also contain format specifiers.

Examples

```
str = "######"
val = 24
sprintf2( str, 2, 1, "%d", val )
//container str now contains the following:
//"##24##"
```

Return value: on success, the total number of characters written is returned; on failure, a negative number is returned.

printf()

Same as sprintf, but the destination is STDOUT.

Parameters (format, ...)

fprintf()

Print formatted output to the stream (opened by fopen() or fopen_mem()).

Parameters (stream, format, ...)

Log management

logf()

The same as printf, but the destination is the Pixilang log buffer.

Parameters (format, ...)

get_log()

Get the Pixilang log buffer.

Return value: the new container with the log buffer; (container must be removed manually).

get_system_log()

Get the Pixilang system log buffer.

Return value: the new container with the system log buffer; (container must be removed manually).

Working with files

load()

Load container from the file.

Parameters (filename, options)

Return value: the new loaded container, or -1 in case of error.

Examples

```
c = load( "smile.jpg" )
if c >= 0
{
  file_format = c.file_format //FORMAT_RAW / FORMAT_JPEG / ...
  if file_format == FORMAT_WAVE
  {
    printf( "This is audio file\n" )
  }
}
```

fload()

Load container from the stream (opened by fopen() or fopen_mem()).

Parameters (stream, options)

Return value: the new loaded container, or -1 in case of error.

save()

Save container to the file.

Parameters (pixi, filename, format, options)

- pixi;
- filename;
- format (FORMAT_RAW, FORMAT_JPEG, FORMAT_PNG, FORMAT_GIF, FORMAT_WAVE, FORMAT_PIXICONTAINER);
- options:
    - for JPEG: quality from 0 - worst to 100 - best;
    - for GIF: flags GIF_GRAYSCALE, GIF_DITHER.

Return value: 0 on success.

Examples

```
c = load( "smile.jpg" )
save( c, "smile.png", FORMAT_PNG )
save( c, "smile2.jpg", FORMAT_JPEG ) //Quality = 85 (default)
save( c, "smile3.jpg", FORMAT_JPEG, 100 ) //Quality = 100
```

fsave()

Save container to the stream (opened by fopen() or fopen_mem()).

Parameters (pixi, stream, format, options)

Return value: 0 on success.

get_real_path()

Convert Pixilang-style path (e.g. 1:/img.png) to the real filesystem path.

Parameters (path)

Return value: the container with real filesystem path; (container must be removed manually).

Examples

```
filename = "1:/some_file"
realpath = get_real_path( filename )
printf( "File name: %s; Real path: %s\n", filename, realpath )
remove( realpath )
```

new_flist()
remove_flist()
get_flist_name()
get_flist_type()
flist_next()

Functions for getting a list of files.

Examples

```
path = CURRENT_PATH
mask = -1 //Examples: "txt/doc", "avi"; or -1 for all files.
fl = new_flist( path, mask )
if fl >= 0
{
  printf( "Some files found in %s\n", path )
  while( 1 )
  {
    file_name = get_flist_name( fl )
    file_type = get_flist_type( fl ) //0 - file; 1 - directory;
    if file_type == 0
    {
      printf( "FILE %s%s\n", path, file_name )
    }
    else
    {
      printf( "DIR  %s%s\n", path, file_name )
    }
    if flist_next( fl ) == 0 //Go to the next file
    {
      //No more files
      break
    }
  }
  remove_flist( fl )
}
```

get_file_size()

Get the size of selected file.

Parameters (filename)

get_file_format()

Get file (or stream, if filename == -1) format (one of theFORMAT_* constants).

Parameters (filename, stream)

- filename;
- stream; optional.

Return value: file format (one of the FORMAT_* constants).

get_fformat_mime()

Get MIME type (string) for specified file format.

Parameters (fileformat)

- fileformat - one of the FORMAT_* constants.

Return value: MIME type (string container).

get_fformat_ext()

Get the extension (string) for specified file format.

Parameters (fileformat)

- fileformat - one of the FORMAT_* constants.

Return value: file extension (string container).

remove_file()

Parameters (filename)

rename_file()

Parameters (old_filename, new_filename)

copy_file()

Parameters (source_filename, destination_filename)

create_directory()

Parameters (directory_name, mode)

- directory_name;
- mode - system specific mode; optional.

set_disk0()

Use the _stream_ (opened by fopen() or fopen_mem()) as virtual disk 0:/ . _stream_ must point to the opened TAR archive.

Parameters (stream)

- stream - stream, or 0 if you want to disable disk 0:/

get_disk0()

fopen()

The fopen() function shall open the file whose pathname is the string pointed to by _filename_, and associates a stream with it.

Parameters (filename, mode)

- filename;
- mode:
    - r or rb - open file for reading;
    - w or wb - truncate to zero length or create file for writing;
    - a or ab - append; open or create file for writing at end-of-file;
    - r+ or rb+ or r+b - open file for update (reading and writing);

- w+ or wb+ or w+b - truncate to zero length or create file for update;
- a+ or ab+ or a+b - append; open or create file for update, writing at end-of-file.

Return value: upon successful completion, fopen() shall return ID of the object controlling the stream; otherwise, 0 shall be returned.

Examples

```
f = fopen( "/tmp/data.txt", "rb" ) //Open file data.txt for reading
fclose( f ) //...and close it.
```

fopen_mem()

Open the _data_ container as file.

Parameters (data)

Return value: upon successful completion, fopen_mem() shall return ID of the object controlling the stream; otherwise, 0 shall be returned.

fclose()

The fclose() function shall cause the stream to be flushed and the associated file to be closed.

Parameters (stream)

Return value: upon successful completion, fclose() shall return 0.

Examples

```
f = fopen( "/tmp/data.txt", "rb" ) //Open file data.txt for reading.
c = fgetc( f ) //Get a byte from this file.
fclose( f ) //Close the stream.
```

fputc()

Put a byte on a stream.

Parameters (c, stream)

Examples

```
f = fopen( "/tmp/data.txt", "wb" ) //Open file data.txt for writing.
fputc( 0x12, f ) //Put a byte 0x12 to this file.
fclose( f ) //Close the stream.
```

fputs()

Put a string on a stream.

Parameters (s, stream)

Examples

```
f = fopen( "/tmp/data.txt", "wb" ) //Open file data.txt for writing.
str = "Hello!"
fputc( str, f ) //Put a string "Hello!" to this file.
fclose( f ) //Close the stream.
```

fwrite()

The fwrite() function shall write, from the container **data**, up to **size** bytes, to the stream pointed to by **stream**.

Parameters (data, size, stream, data_offset_optional)

Return value: the number of bytes successfully written, which may be less than **size** if a write error is encountered.

Examples

```
f = fopen( "/tmp/data.txt", "wb" ) //Open file data.txt for writing.
str = "Hello!"
fwrite( str, 2, f ) //Put first two bytes from the string "Hello!" to this file.
fclose( f ) //Close the stream.
```

fgetc()

Get a byte from a stream.

Parameters (stream)

Return value: upon successful completion, fgetc() shall return the next byte from the input stream pointed to by **stream**.

fgets()

Get a string from a stream. Reading occurs until one of the following conditions is met:

- run out of space in container s;
- end of file reached;
- character '\n' (0xA) was received (line feed);
- character '\r' (0xD) was received (carriage return).

Parameters (s, n, stream, offset)

- s - container where the string will be stored;
- n - the maximum number of bytes that can be read; container size s must be at least (n-offset);
- stream - the stream previously opened with fopen();
- offset - starting position (byte number) in container s.

Return value:

- >= 0 - length of the received string;
- -1 - empty string, end of file reached; only for version 3.8.2b and above.

Examples

```
string = new( 256, 1, INT8 )
f = fopen( "/tmp/data.txt", "rb" ) //Open file data.txt for reading.
fgets( string, 256, f ) //Get a string from this file.
fclose( f ) //Close the stream.
```

fread()

The fread() function shall read into the container pointed to by **data** up to **size** bytes, from the stream pointed to by **stream**.

Parameters (data, size, stream, data_offset_optional)

Return value: the number of bytes successfully read which is less than **size** only if a read error or end-of-file is encountered.

feof()

Test end-of-file indicator on a stream.

Parameters (stream)

Return value: non-zero if the end-of-file indicator is set for **stream**.

fflush()

Flush a stream.

Parameters (stream)

fseek()

Reposition a file-position indicator in a stream.

Parameters (stream, offset, origin)

- stream;
- offset;
- origin:
    - SEEK_SET - beginning of file;
    - SEEK_CUR - current position of the file pointer;
    - SEEK_END - end of file.

ftell()

Return a file offset in a stream.

Parameters (stream)

Return value: the current value of the file-position indicator for the stream measured in bytes from the beginning of the file.

Examples

```
//One of the ways to get the file size:
f = fopen( "/tmp/data.txt", "rb" )
fseek( f, 0, SEEK_END )
size_of_file = ftell( f )
fclose( f )
```

setxattr()

Set an extended attribute value of the file.

Parameters (path, attr_name, data, data_size_in_bytes, flags)

Return value: 0 on success, or -1 in case of error.

Examples

```
//iOS: set "don't backup" file attribute:
if strstr( OS_NAME, "ios" ) >= 0
{
    $val = new( 1, 1, INT8 )
    $val[ 0 ] = 1
    setxattr( "myfile.txt", "com.apple.MobileBackup", $val, 1, 0 )
    remove( $val )
}
```

Graphics

frame()

Draw the current screen on the display and sleep for selected number of milliseconds.

Parameters (delay, x, y, xsize, ysize)

- delay - pause length in milliseconds;
- x, y, xsize, ysize - region of the screen; optional parameters.

Return value:

- 0 - successful;
- -1 - no screen;
- -2 - timeout (graphics engine is suspended?).

vsync()

Enable/disable vertical synchronization (when possible). vsync(1) - enable. vsync(0) - disable.

Parameters (enable)

set_pixel_size()

Change the size of the screen pixels. Default size = 1.

Parameters (size)

get_pixel_size()

Get the size of the screen pixel.

set_screen()

Set current screen - destination container (with image) for all graphic commands.ID of the default screen container is 0.

Parameters (pixi)

- pixi - ID of the container that will be used as the current screen.

get_screen()

Get current screen.

Return value: screen container ID.

set_zbuf()

Parameters (zbuf_container)

Set container (INT32 elements) with Z-buffer.

get_zbuf()

clear_zbuf()

get_color()

Get color by r,g,b (red,green,blue).

Parameters (red, green, blue)

- red - red intensity (0..255);
- green - green intensity (0..255);
- blue - blue intensity (0..255).

Return value: color value.

get_red()

Get red component intensity in a selected color.

Parameters (color)

Return value: red component intensity; from 0 to 255.

get_green()

Get green component intensity in a selected color.

Parameters (color)

Return value: green component intensity; from 0 to 255.

get_blue()

Get blue component intensity in a selected color.

Parameters (color)

Return value: blue component intensity; from 0 to 255.

get_blend()

Get an intermediate color value between two selected colors.

Parameters (c1, c2, v)

- c1 - first color;
- c2 - second color;
- v - position between c1 and c2:
    - 0 and lower - c1;
    - ...
    - 128 - (c1+c2)/2;
    - ...
    - 255 and higher - c2.

Return value: intermediate color value.

transp()

Set transparency.

Parameters (t)

- t - transparency from 0 to 255.

get_transp()

clear()

Clear current screen.

Parameters (color)

dot()

Draw a dot.

Parameters (x, y, color)

dot3d()

Draw a dot in 3D.

Parameters (x, y, z, color)

get_dot()

Get a dot's color.

Parameters (x, y)

Return value: color value.

get_dot3d()

Get a dot's color in 3D.

Parameters (x, y, z)

Return value: color value.

line()

Draw a line.

Parameters (x1, y1, x2, y2, color)

line3d()

Draw a line in 3D.

Parameters (x1, y1, z1, x2, y2, z2, color)

box()

Draw a rectangle.

Parameters (x, y, xsize, ysize, color)

fbox()

Draw a filled rectangle.

Parameters (x, y, xsize, ysize, color)

pixi()

Display the container with image.

Parameters (pixi_cont, x, y, color, xscale, yscale, src_x, src_y, src_xsize, src_ysize)

- pixi_cont - container ID (source);
- x;
- y;
- color - color of the filter; optional; default value is WHITE;
- xscale - scaling factor (x axis); optional; default value is 1;
- yscale - scaling factor (y axis); optional; default value is 1;
- src_x - x offset in pixi_cont; default - 0;
- src_y - y offset in pixi_cont; default - 0;
- src_xsize - width of drawable pixi_cont region; default - width of pixi_cont;
- src_ysize - height of drawable pixi_cont region; default - height of pixi_cont.

Examples

```
pixi( image )
pixi( image, 10, 20 )
pixi( image, 30, 40, GREEN )
pixi( image, 90, 20, GREEN, 0.5, 0.5 )
```

triangles3d()

Draw array of triangles.

Parameters (vertices, triangles, tnum)

- vertices - container of vertices; width = 8; height = number of vertices; vertex data format: X, Y, Z, TextureX (0..Width), TextureY (0..Height), Unused, Unused, Unused;
- triangles - container of triangles; width = 8; height = number of triangles; triangle data format: NumberOfVertex1, NumberOfVertex2, NumberOfVertex3, Color, Texture (or -1), Opacity (0..255), Unused, Order (triangles with lower Order value will be drawn first);
- tnum - number of triangles; optional.

sort_triangles3d()

Sort triangles by Z value.

Parameters (vertices, triangles, tnum)

- vertices - container of vertices; format is the same as in triangles3d();
- triangles - container of triangles; format is the same as in triangles3d();
- tnum - number of triangles; optional.

set_key_color()

Set / reset the color of transparency in the container.

Parameters (pixi, color)

- pixi - container ID;
- color - color to be transparent; ignore this parameter if you want to disable transparent color for this container.

get_key_color()

set_alpha()

Attach a container with alpha channel to another container. Alpha channel should be of type INT8.

Parameters (pixi, alpha)

- pixi - container ID;
- alpha - container with alpha channel; ignore this parameter if you want to disable alpha-channel for the pixi container.

get_alpha()

Get linked container with the alpha channel.

Parameters (cont)

Return value: container ID or -1 (if no alpha channel).

print()

Show text on the screen.

Parameters (text, x, y, color, align, max_xsize, str_offset, str_size)

- text - container with text;
- x, y - point of the alignment;
- color;
- align - alignment; optional;
- max_xsize; optional;
- str_offset - offset (number of the first byte) inside container text; optional;
- str_size - how many bytes to read from container text; optional.

Examples

```
print( "Hello Pixi!", 0, 0 ) //color = WHITE; centered;
print( "line1\nline2", 50, 50, RED ) //centered;
print( "line1\nline2", -50, 50, RED, TOP | LEFT ) //alignment = top left;
```

get_text_xsize()
get_text_ysize()
get_text_xysize()

Parameters (text, align, max_xsize, str_offset, str_size)

- text;
- align; optional;
- max_xsize; optional;
- str_offset - offset (number of the first byte) inside container text; optional;

- str_size - how many bytes to read from container text; optional.

get_text_xsize() return value: text width in pixels.
get_text_ysize() return value: text height in pixels.
get_text_xysize() return value: text size in pixels; width = retval & 0xFFFF; height = (retval >> 16) & 0xFFFF;

set_font()

Parameters (first_char_utf32, font_image, xchars, ychars, last_char, char_xsize, char_ysize, char_xsize2, char_ysize2, grid_xoffset, grid_yoffset, grid_cell_xsize, grid_cell_ysize)

- first_char_utf32 - first char code (UTF-32);
- font_image - container with the font texture;
- optional:
    - xchars - number of characters horizontally;
    - ychars - number of characters vertically;
    - last_char - last char code;
    - char_xsize - character width on the screen;
    - char_ysize - character height on the screen;
    - char_xsize2 - character width on the texture (font_image);
    - char_ysize2 - character height on the texture (font_image);
    - grid_xoffset - grid offset X;
    - grid_yoffset - grid offset Y;
    - grid_cell_xsize - grid cell width;
    - grid_cell_ysize - grid cell height.

get_font()

Parameters (char_utf32)

Return value: the font container font for the specified character.

effector()

Apply an effect to selected screen area. Coordinates of this function can't be changed by t_* transformation functions.

Parameters (type, power, color, x, y, xsize, ysize, x_step, y_step)

color_gradient()

Draw a smooth transition between the colors of 4 key points in the specified rectangle. Bilinear interpolation algorithm is used.
The coordinates of this function are not affected by the transformation.

Parameters (color1, opacity1, color2, opacity2, color3, opacity3, color4, opacity4, x, y, xsize, ysize, x_step, y_step)

- color1, opacity1, color2, opacity2, color3, opacity3, color4, opacity4 - color and opacity of 4 key points: top left, top right, bottom left, bottom right;
- x, y - coordinates of the upper left corner of the rectangle; optional;
- xsize, ysize - width and height of the rectangle; optional;
- x_step, y_step - horizontal and vertical step; for example x_step=4 means to skip every 2nd, 3rd and 4th horizontal pixels; optional.

split_rgb()

Split a container (with pixels) by color channels (red, green, blue) and vice versa.

Parameters (direction, image, red_channel, green_channel, blue_channel, image_offset, channel_offset, size)

- direction: 0 - from image to RGB; 1 - from RGB to image;
- image - source/destination container of type PIXEL;
- red_channel - container with red components of the image; optional; can be -1;
- green_channel - container with green components of the image; optional; can be -1;
- blue_channel - container with blue components of the image; optional; can be -1;
- image_offset - offset in the image container; optional;
- channel_offset - offset in the channel containers; optional;
- size - number of pixels to split; optional.

Examples

```
img = load( "some_image.jpg" )
xsize = get_xsize( img )
ysize = get_ysize( img )
r = new( xsize, ysize, INT16 )
g = new( xsize, ysize, INT16 )
b = new( xsize, ysize, INT16 )
split_rgb( 0, img, r, g, b ) //Convert image to RGB
//Get red value (from 0 to 255) of the first pixel:
value = r[ 0 ]
```

split_ycbcr()

Same as split_rgb() but for YCbCr conversion.

OpenGL base

OpenGL-version of Pixilang is based on the OpenGL ES 2.0 with OpenGL ES Shading Language 1.0 (GLSL ES).
Here is very good OpenGL ES 2.0 Quick Reference Card.

set_gl_callback()

Set OpenGL frame drawing callback.

Parameters (gl_callback, user_data)

- gl_callback - callback function with parameters ($user_data); almost all graphics functions will be redirected to OpenGL within this callback;
- user_data - some user-defined data that will be sent to gl_callback() during the frame redrawing.

Examples

```
fn gl_callback( $user_data )
{
  set_screen( GL_SCREEN ) //Enable OpenGL drawing mode
  clear( YELLOW )
  set_screen( 0 ) //Back to the default screen
}

set_gl_callback(
  gl_callback, //OpenGL frame drawing function
  0 ) //Some user-defined data

while( 1 )
{
  while( get_event() ) { if EVT[ EVT_TYPE ] == EVT_QUIT { break2 } }
  frame()
}

set_gl_callback( -1 ) //Remove OpenGL callback
```

remove_gl_data()

Remove all GL-specific data from the container. This data will be created again inside the OpenGL drawing callback.

Parameters (container)

update_gl_data()

Sends a request to update the OpenGL texture associated with container. Use this function if the contents (pixels) of the container have changed, but the size remains the same.

Parameters (container)

gl_draw_arrays()

Hybrid of the OpenGL functions glColor4ub(), glBindTexture(), glVertexPointer(), glColorPointer(), glTexCoordPointer(), glDrawArrays().
Can only be used inside the OpenGL drawing callback.

Parameters (mode, first, count, color_r, color_g, color_b, color_a, texture, vertex_array, color_array, texcoord_array)

- mode - what kind of primitives to render:
  - GL_POINTS;
  - GL_LINE_STRIP;
  - GL_LINE_LOOP;
  - GL_LINES;
  - GL_TRIANGLE_STRIP;
  - GL_TRIANGLE_FAN;
  - GL_TRIANGLES;
- first - starting vertex in the enabled arrays;
- count - number of vertices to be rendered;
- color_r, color_g, color_b, color_a - RGBA color (0..255 in each channel; ignored if color_array is specified);
- texture - container with texture, or -1;
- vertex_array - INT8, INT16 or FLOAT32 container with vertices; width = number of components per vertex; height = number of vertices;
- color_array - INT8 or FLOAT32 container with colors, or -1; width = number of components per color; height = number of vertices; optional;
- texcoord_array - INT8, INT16 or FLOAT32 container with texture coordinates; width = number of

texture dimensions; height = number of vertices; optional.

gl_blend_func()

Full analog of the OpenGL function glBlendFunc() (specify pixel arithmetic) or glBlendFuncSeparate (if sfactor_alpha and dfactor_alpha are specified).
Can only be used inside the OpenGL drawing callback.

Call this function without parameters, if you want to reset pixel arithmetic to the default values.

Parameters (sfactor, dfactor, sfactor_alpha, dfactor_alpha)

- sfactor - specifies how the red, green, and blue blending factors are computed;
- dfactor - specifies how the red, green, and blue destination blending factors are computed;
- sfactor_alpha - specified how the alpha source blending factor is computed; optional;
- dfactor_alpha - specified how the alpha destination blending factor is computed; optional.

gl_bind_framebuffer()

Convert specified pixel container (cnum) to the OpenGL framebuffer (with attached texture) and bind it. All rendering operations will be redirected to this framebuffer. To unbind - just call this function without parameters.
Pixel size does not affect the framebuffer.
The framebuffer is flipped along the Y-axis when shown with pixi(). (native OpenGL framebuffer coordinates are using)
Can only be used inside the OpenGL drawing callback.

Parameters (cnum, flags, x, y, width, height)

- cnum - pixel container;
- flags - optional flags (GL_BFB_*);
- x, y, width, height - optional viewport coordinates (relative to the bottom left corner of the framebuffer) and size.

gl_bind_texture()

Bind selected pixel container (cnum) to the specified texture image unit.
Can only be used inside the OpenGL drawing callback.

Parameters (cnum, texture_unit)

- cnum - pixel container;
- texture_unit - texture image unit number: 1, 2, 3, ... ; 0 is the main texture unit used by pixi().

Example

```
fn gl_callback( $userdata )
{
    set_screen( GL_SCREEN )
    gl_bind_texture( some_image2, 1 ) //bind some_image2 to texture unit 1
    gl_bind_texture( some_image3, 2 ) //bind some_image3 to texture unit 2
    gl_use_prog( gl_prog ) //Use user-defined GLSL program
    gl_uniform( gl_prog.g_texture2, 1 ) //set shader variable: g_texture2 = texture image unit 1
    gl_uniform( gl_prog.g_texture3, 2 ) //set shader variable: g_texture3 = texture image unit 2
    pixi( some_image )
    gl_use_prog() //Back to default GLSL program
    set_screen( 0 ) //Back to the default screen
}
gl_vshader = GL_SHADER_TEX_RGB_SOLID //Vertex shader = default shader for solid primitives drawing
gl_fshader = //Fragment shader
"uniform sampler2D g_texture; //main texture image unit 0 (set by pixi())
uniform sampler2D g_texture2; //texture image unit 1
uniform sampler2D g_texture3; //texture image unit 2
uniform vec4 g_color;
IN vec2 tex_coord_var;
void main()
{
    vec4 c1 = texture2D( g_texture, tex_coord_var );
    vec4 c2 = texture2D( g_texture2, tex_coord_var );
    vec4 c3 = texture2D( g_texture3, tex_coord_var );
    gl_FragColor = ( c1 + c2 + c3 ) * g_color;
}
"
gl_prog = gl_new_prog( gl_vshader, gl_fshader )
```

gl_get_int()

Get the value of the simple GL state variable. Full analog of the OpenGL function glGetIntegerv().
Can only be used inside the OpenGL drawing callback.

Parameters (pname)

- pname - symbolic constant indicating the state variable to be returned.

gl_get_float()

Get the value of the simple GL state variable. Full analog of the OpenGL function glGetFloatv().
Can only be used inside the OpenGL drawing callback.

Parameters (pname)

- pname - symbolic constant indicating the state variable to be returned.

OpenGL shaders

OpenGL-version of Pixilang uses OpenGL ES Shading Language 1.0 (GLSL ES) for the vertex and the fragment shaders.

Notes about the vertex shader syntax in Pixilang:

- use **IN** instead of **attribute** and **in** qualifiers;
- use **OUT** instead of **varying** and **out** qualifiers;
- use **LOWP**, **MEDIUMP** and **HIGHP** instead of the **lowp**, **mediump** and **highp** qualifiers;
- use **PRECISION(P, T)** instead of **precision P, T**.

Notes about the fragment shader syntax in Pixilang:

- use **IN** instead of **varying** and **in** qualifiers;
- use **LOWP**, **MEDIUMP** and **HIGHP** instead of the **lowp**, **mediump** and **highp** qualifiers;

- use **PRECISION(P, T)** instead of **precision P, T**.

gl_new_prog()

Create a new GLSL program - container with a vertex shader and a fragment shader. You can use this function anywhere in your code. Shader code will be compiled, and the program will be linked later - during the gl_use_prog() function call.

Parameters (vertex_shader, fragment_shader)

- vertex_shader - container with the source code of the vertex shader, or one of theGL_SHADER_* constants (predefined shaders);
- fragment_shader - container with the source code of the fragment shader, or one of the GL_SHADER_* constants (predefined shaders).

Return value: the new container with GLSL program, or negative value in case of some error; (container must be removed manually).

gl_use_prog()

Use the GLSL program previously created by the gl_new_prog() function.If you use this program first time - it will be compiled and linked.
Can only be used inside the OpenGL drawing callback.

Parameters (prog)

gl_uniform()

Change the value of the uniform variable within the GLSL program.
Can only be used inside the OpenGL drawing callback.

gl_uniform() can also change the contents of arrays if you use this function as follows: gl_uniform(var_location, src_container, vector_size, first_vector, count), where count is the number of vectors to write to the array.

Parameters (var_location, v0, v1, v2, v3)

- var_location - location of the uniform variable to be modified; you can get it from the GLSL program container easily: PROGRAM.UNIFORM_NAME;
- v0, v1, v2, v3 - new values to be used for the specified uniform variable; v1, v2 and v3 are optional; number of values depends the number of components in the data type of the specified uniform variable.

Examples

```
gl_use_prog( gl_prog ) //Use GLSL program gl_prog (vertex shader + fragment shader)
gl_uniform( gl_prog.g_time, get_timer( 0 ) ) //Set value of uniform variable g_time
```

gl_uniform_matrix()

Change the value of the uniform matrix within the GLSL program.
Can only be used inside the OpenGL drawing callback.

Parameters (size, matrix_location, transpose, matrix)

- size - dimensionality of the matrix: 2 = 2x2 matrix; 3 = 3x3 matrix; 4 = 4x4 matrix;
- var_location - location of the matrix to be modified; you can get it from the GLSL program container easily: PROGRAM.MATRIX_NAME;
- transpose - transpose the matrix (0 - no; 1 - yes);

- matrix - ID of the container with the matrix.

Examples

```
gl_use_prog( gl_prog ) //Use GLSL program gl_prog (vertex shader + fragment shader)
gl_uniform( 4, gl_prog.g_mat, 0, source_matrix ) //Set value of uniform matrix g_mat
```

Animation

pack_frame()

Pack current frame (from container data to hidden storage with frames). Frame number must be stored in the "frame" container property.

Parameters (pixi)

unpack_frame()

Unpack current frame (from hidden storage with frames to container data). Frame number must be stored in the "frame" container property.

Parameters (pixi)

create_anim()

Create the hidden storage with frames (animation) in the selected container.

Parameters (pixi)

remove_anim()

Remove the hidden storage with frames (animation) from the selected container.

Parameters (pixi)

clone_frame()

Clone current frame. Frame number must be stored in the "frame" container property.

Parameters (pixi)

remove_frame()

Remove current frame. Frame number must be stored in the "frame" container property.

Parameters (pixi)

play()

Enable auto-play mode. Frame will be changed automatically during the pixi() function call.

Parameters (pixi)

stop()

Disable auto-play mode.

Parameters (pixi)

Transformation

Coordinate transformation in Pixilang works similarly to other graphics APIs. It is based on a transformation matrix that affects the coordinates of the drawn objects. The matrix is a 4x4 array inside the Pixilang VM. It can be changed with t_* commands, and reset with t_reset(). Every time Pixilang draws something (including pixi()), it multiplies the object's vertex coordinates by the transformation matrix. The resulting coordinates are used for the final drawing of the figure.

The t_* commands do not affect any particular object, but the entire coordinate system, relative to previous transformations. The result (including both new and old changes) is stored in the matrix. This is convenient, because one small array can store an infinite number of t_* operations. Usually direct access to this array is not required.

The order of the values inside the matrix is as follows (different from the usual order in 2D containers):

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Example:
(draw a picture moved relative to the center of the screen, magnified by 2 times and rotated 45 degrees)

```
t_reset() //reset transformation
t_translate( 32, 0, 0 ) //#1: move 32 pixels horizontally
t_scale( 2, 2, 2 )      //#2: increase the size by 2 times
t_rotate( 45, 0, 0, 1 ) //#3: rotate 45 degrees around vector 0,0,1 (x,y,z)
pixi( img ) //draw the final picture with offset, scale and rotation
frame() //show it
```

Simplified, this example will result in the following transformation (for each vertex of the object being drawn):
v2 = wm * #1 * #2 * #3 * v;
wm - additional transformation for correct display in the window; not required in software rendering mode (non-OpenGL);
#1 - translation matrix;
#2 - scaling matrix;
#3 - rotation matrix;
v - initial vertex coordinates;
v2 - modified vertex coordinates;
The matrices and the final vertex v2 are 4D. Without going into details, this is required for translation and perspective projection (see Clip coordinates и Homogeneous coordinates). Before drawing, a perspective division operation is performed: v2 is converted from 4D to 3D.

In OpenGL mode, the coordinate transformation is performed by the following vertex shader:

```
// Window manager transformation matrix:
// (converts Pixilang coordinates
//   (X:-WINDOW_XSIZE/2...WINDOW_XSIZE/2; Y:-WINDOW_YSIZE/2...WINDOW_YSIZE/2; Z:-WINDOW_ZSIZE/2...WINDOW_ZSI
// to normalized device coordinates
//   (NDC; X:-1...1; Y:-1...1; Z:1...-1))
uniform mat4 g_wm_transform;
// Pixilang transformation matrix (with all t_* operations inside):
uniform mat4 g_pixi_transform;
// Initial vertex coordinates:
IN vec4 position;.
void main()
{
    gl_Position = g_wm_transform * g_pixi_transform * position;
    gl_PointSize = 1.0;
}
// After the shader finishes, perspective division is performed to obtain 3D coordinates in NDC
```

t_reset()

Reset transformation (set equal to identity matrix).

t_rotate()

Parameters (angle, x, y, z)

- angle - specifies the angle of rotation, in degrees;
- x, y, z - specify the x, y, and z coordinates of a vector, respectively.

t_translate()

Parameters (x, y, z)

t_scale()

Parameters (x, y, z)

t_push_matrix()

Save current transformation matrix to the stack.

t_pop_matrix()

Restore previous saved transformation matrix from the stack.

t_get_matrix()

Get transformation matrix (4x4 FLOAT).

Parameters (matrix_container)

t_set_matrix()

Set transformation matrix (4x4 FLOAT).

Parameters (matrix_container)

t_mul_matrix()

Multiply transformation matrix (4x4 FLOAT). Current matrix = current matrix * matrix_container.

Parameters (matrix_container)

t_point()

Transform point (container with 3 elements of type FLOAT).

Parameters (point_coordinates)

Audio

set_audio_callback()

Parameters (callback, userdata, sample_rate, format, channels, flags)

- callback - audio callback (function address);
- userdata - user-defined data for the callback;
- sample_rate - sample rate; set it to 0, if you want to use the sample rate from the global Pixilang preferences;
- format - sample format;
- channels - number of channels;
- flags - flags AUDIO_FLAG_*.

Examples

```
fn audio_callback( $stream, $userdata, $channels, $frames, $time )
{
  generator( OP_SIN, $channels[ 0 ], 0, 32767 / 2, 0.1, 0 ) //Left channel
  generator( OP_SIN, $channels[ 1 ], 0, 32767 / 2, 0.1, 0 ) //Right channel
  ret(1)
  //return values:
  // 0 - silence, output channels are not filled;
  // 1 - output channels are filled;
  // 2 - silence, output channels are filled with zeros (or values close to zero).
}
//Start audio:
set_audio_callback( audio_callback, 0, 22050, INT16, 2, AUDIO_FLAG_INTERP2 )
```

```
//Stop audio:
set_audio_callback( -1 )
```

get_audio_sample_rate()

Parameters (source)

- source: 0 - local sample rate; 1 - global (from Preferences) sample rate.

Return value: sample rate in Hz.

enable_audio_input()

Parameters (disable_enable)

get_note_freq()

Get the frequency of the specified note (using a fast, not very accurate algorithm).
Frequency of MIDI note = get_note_freq(midi_note, 0) / 64; (5 octaves lower)

Parameters (note, finetune)

- note - note number; 0 = C-0; 1 = C#0; 2 = D-0 ...

- finetune - from -64 (previous note) to 64 (next note).

Return value: note frequency in Hz.

MIDI

midi_open_client()

Parameters (client_name)

Return value: client ID or negative value in case of some error.

midi_close_client()

Parameters (client_id)

midi_get_device()

Parameters (client_id, device_num, flags)

Return value: selected device name, or -1 if not exists.

midi_open_port()

Parameters (client_id, port_name, device_name, flags)

Return value: port ID or negative value in case of some error.

midi_reopen_port()

Parameters (client_id, port_id)

midi_close_port()

Parameters (client_id, port_id)

midi_get_event()

Parameters (client_id, port_id, data_cont)

Return value: size of the current MIDI event (in bytes).

midi_get_event_time()

Parameters (client_id, port_id)

Return value: time of the current event (in system ticks).

midi_next_event()

Go to the next event.

Parameters (client_id, port_id)

midi_send_event()

Parameters (client_id, port_id, data_cont, data_size, t)

SunVox

SunVox modular synth engine is now part of Pixilang since v3.8.
See the SunVox Library Documentation for a detailed description of all functions.

Time

start_timer()

Start selected timer.

Parameters (timer_num)

get_timer()

Get value (in milliseconds) of selected timer.

Parameters (timer_num)

Return value: 32bit value of selected timer (in milliseconds).

get_year()

get_month()

get_day()

get_hours()

get_minutes()

get_seconds()

get_ticks()

Get current system tick counter (32bit).

get_tps()

Get number of system ticks per second.

sleep()

Parameters (delay)

- delay - delay in milliseconds.

Events

get_event()

Get a new event from the system.

Return value: 0 - no events; 1 - event is received and placed into the container EVT.

set_quit_action()

Set the program's behavior when receiving event EVT_QUIT.

Parameters (action)

- action - action number.

Possible values for the action parameter:

- QA_NONE - do nothing;
- QA_CLOSE_VM (default) - close the current virtual machine, but don't quit from Pixilang.

Threads

thread_create()

Parameters (thread_function, user_data, flags_optional)

Return value: thread ID or -1 if error occurred.

Examples

```
fn thread_body( $thread_id, $user_data )
{
  printf( "Thread code\n" )
}
thread_id = thread_create( thread_body, 0 )
err = thread_destroy( thread_id, 1000 ) //Wait for the thread to terminate
if err == 0 { printf( "Thread closed successful\n" ) }
if err == 1 { printf( "Time-out. Thread is not closed\n" ) }
if err == 2 { printf( "Some error occurred in thread_destroy() function\n" ) }
```

thread_destroy()

Parameters (thread_id, timeout_ms)

- thread_id;
- timeout_ms - time-out in milliseconds; negative values - don't try to kill the thread after timeout; INT_MAX - infinite.

Return value:

- 0 - thread closed successfully;
- 1 - time-out;
- 2 - some error.

mutex_create()

Examples

```
new_mutex = mutex_create()
mutex_lock( new_mutex )
mutex_unlock( new_mutex )
mutex_destroy( new_mutex )
```

mutex_destroy()

mutex_lock()

mutex_trylock()

mutex_unlock()

Mathematical

- acos(x) - arccosine;
- acosh(x) - hyperbolic arccosine;
- asin(x) - arcsine;
- asinh(x) - hyperbolic arcsine;
- atan(x) - arctangent;
- atan2(y, x) - angle (in radians) between the positive X axis and the vector (x,y);
- atanh(x) - hyperbolic arctangent;
- ceil(x) - smallest integer value not less than x;
- cos(x) - cosine;
- cosh(x) - hyperbolic cosine;
- exp(x) - e (Euler's number - 2.7182818) raised to the given power;
- exp2(x) - 2 raised to the given power;
- expm1(x) - e raised to the given power, minus one;
- abs(x) - absolute value of an integer value;
- floor(x) - nearest integer not greater than x;
- mod(x, y) - floating-point remainder of the division operation x/y;
- log(x) - natural logarithm (to base e);
- log2(x) - binary logarithm (to base 2);
- log10(x) - common logarithm (to base 10);
- pow(base, exp) - raise a number (base) to a given power (exp);
- sin(x) - sine;
- sinh(x) - hyperbolic sine;
- sqrt(x) - square root;
- tan(x) - tangent;
- tanh(x) - hyperbolic tangent;
- rand() - get random number from 0 to 32767;
- rand_seed(seed) - set random seed.

Type punning

Read more about type punning

reinterpret_type()

Parameters (value, mode, intermediate_value_bits)

- value;
- mode:
  - 0 - FLOAT -> X-bit FLOAT -> INT;
  - 1 - INT -> X-bit FLOAT -> FLOAT;
- intermediate_value_bits - 32 or 64.

Return value: converted value (INT or FLOAT depending on mode).

Data processing

op_cn()

Execute data processing operation.Operands:

1. container C1 (destination);
2. numerical value N.

Operation expression:

```
for each element of C1: C1[ i ] = C1[ i ] OP N,
Where i - element number; OP - selected operation.
```

Parameters (opcode, C1, N) - for whole container C1

Parameters (opcode, C1, N, x, xsize) - for 1D region

Parameters (opcode, C1, N, x, y, xsize, ysize) - for 2D region

- opcode - operation;
- C1 - destination container;
- N - numerical value;
- x,y,xsize,ysize - region.

Examples

```
//Add 32 to the whole container img:
op_cn( OP_ADD, img, 32 )

//Add 32 to 128..256th elements of the container img:
op_cn( OP_ADD, img, 32, 128, 128 )

//Add 32 to two-dimensional region (8,8,32,32) of elements:
op_cn( OP_ADD, img, 32, 8, 8, 32, 32 )
```

op_cc()

Execute data processing operation.Operands:

1. container C1 (destination);
2. container C2 (source).

Operation expression:

```
for each element of C1: C1[ i ] = C1[ i ] OP C2[ i ],
Where i - element number; OP - selected operation.
```

Parameters (opcode, C1, C2) - for whole container C1

Parameters (opcode, C1, C2, dest_x, src_x, xsize) - for 1D region

Parameters (opcode, C1, C2, dest_x, dest_y, src_x, src_y, xsize, ysize) - for 2D region

- opcode - operation;
- C1 - destination container;
- C2 - source container;
- x,y,xsize,ysize - region.

op_ccn()

Execute data processing operation.Operands:

1. container C1 (destination);
2. container C2 (source);
3. numerical value N.

Operation expression:

```
for each element of C1: C1[ i ] = C1[ i ] OP C2[ i ] OP2 N,
Where i - element number; OP - selected operation; OP2 - additional operation associated with OP.
```

Parameters (opcode, C1, C2, N) - for whole container C1

Parameters (opcode, C1, C2, N, dest_x, src_x, xsize) - for 1D region

Parameters (opcode, C1, C2, N, dest_x, dest_y, src_x, src_y, xsize, ysize) - for 2D region

- opcode - operation;
- C1 - destination container;
- C2 - source container;
- N - numerical value;
- x,y,xsize,ysize - region.

generator()

Generate a signal.

Parameters (opcode, pixi, phase, amplitude, delta_x, delta_y, x, y, xsize, ysize)

- opcode - operation;
- pixi - container;
- phase;
- amplitude;
- delta_x;
- delta_y;
- x,y,xsize,ysize - region.

Examples

```
//Generate a sine wave to the whole container img:
generator( OP_SIN, img, 0, 1, 0.1, 0.1 )

//Generate a rough sine wave to the whole container img:
generator( OP_SIN8, img, 0, 1, 0.1, 0.1 )

//Generate a sine wave to 8...128 elements of the container img:
generator( OP_SIN, img, 0, 1, 0.1, 0.1, 8, 120 )

//Generate a sine wave to region (8,8,32,32) of the container img:
generator( OP_SIN, img, 0, 1, 0.1, 0.1, 8, 8, 32, 32 )
```

wavetable_generator()

Very fast multichannel sampler, where the sample (table) is always looped and has a fixed size (32768).

Parameters (dest, dest_offset, dest_length, table, amp, amp_delta, pos, pos_delta, gen_offset, gen_step, gen_count)

- dest - audio destination (INT16 or FLOAT32 container);
- dest_offset;
- dest_length;
- table - table with waveform (supported formats: 32768 x INT16, 32768 x FLOAT32);
- amp - INT32 array of amplitudes (fixed point 16.16); after each sample, these values will increase by the values from amp_delta;
- amp_delta - INT32 array of amplitude delta values (fixed point 16.16);
- pos - INT32 array of wavetable positions (fixed point 16.16); after each sample, these values will increase by the values from pos_delta;
- pos_delta - INT32 array of wavetable position delta values (fixed point 16.16);
- gen_offset - number of the first generator;
- gen_step - play every gen_step generator;
- gen_count - total number of generators to play.

Return value: 0 if success.

sampler()

Parameters (sample_info)

Examples

```
sample_data = new( 256, 1, INT16 ) //16bit sample
sample_info = new( SMP_INFO_SIZE, 1, INT32 )
clean( sample_info )
sample_info[ SMP_DEST ] = buffer //Destination container
sample_info[ SMP_DEST_OFF ] = 0 //Destination offset
sample_info[ SMP_DEST_LEN ] = 256 //Destination length
sample_info[ SMP_SRC ] = sample_data
sample_info[ SMP_SRC_OFF_H ] = 0 //Sample offset (left part of fixed point value)
sample_info[ SMP_SRC_OFF_L ] = 0 //Sample offset (right part of fixed point value from 0 to 65535)
sample_info[ SMP_SRC_SIZE ] = 0 //Sample size (0 - whole sample)
sample_info[ SMP_LOOP ] = 0 //Loop start
sample_info[ SMP_LOOP_LEN ] = 128 //Loop length (or 0, if no loop)
sample_info[ SMP_VOL1 ] = 0 //Start volume
sample_info[ SMP_VOL2 ] = 32768 //End volume (32768 = 1.0)
sample_info[ SMP_DELTA ] = ( 1 << 16 ) //Delta; fixed point (real_value * 65536)
sample_info[ SMP_FLAGS ] = SMP_FLAG_INTERP4 | SMP_FLAG_PINGPONG //Cubic spline interpolation and ping-pong loop
sampler( sample_info ) //Go!
```

envelope2p()

Apply gain and DC-offset two-points envelope to selected container area. Without clipping.

Parameters (data_cont, v1, v2, offset, size, dc_offset1, dc_offset2)

- data_cont - container with data (audio waveform, for example);
- v1 - initial gain (0 - no signal; 32768 - original unchanged amplitude);
- v2 - final gain (0 - no signal; 32768 - original unchanged amplitude);
- offset - offset in the data_cont; optional; default - 0;
- size - size of area for envelope; optional; default - whole container;
- dc_offset1 - initial DC offset; optional; default - 0;
- dc_offset1 - final DC offset; optional; default - 0.

gradient()

Fill intermediate values between the key points of the rectangle in the specified container. The bilinear interpolation algorithm is used.

Parameters (container, val1, val2, val3, val4, x, y, xsize, ysize, x_step, y_step)

- container;
- val1, val2, val3, val4 - values of 4 key points: top left, top right, bottom left, bottom right;
- x, y - coordinates of the upper left corner of the rectangle; optional;
- xsize, ysize - width and height of the rectangle; optional;
- x_step, y_step - horizontal and vertical step; for example, x_step=4 means skip every 2nd, 3rd and 4th value horizontally; optional.

fft()

Perform a fast fourier transform.

Parameters (inverse, im, re, size)

new_filter()

Create a new filter with the following function:

```
output[ n ] = ( a[ 0 ] * input[ n ] + a[ 1 ] * input[ n - 1 ] + ... + a[ a_count - 1 ] * input[ n - a_count - 1 ]
                + b[ 0 ] * output[ n - 1 ] + ... + b[ b_count - 1 ] * output[ n - b_count - 1 ] ) >> rshift;
```

Parameters (flags_for_future_use)

Return value: ID of the new container with the filter.

remove_filter()

Parameters (filter)

reset_filter()

Parameters (filter)

init_filter()

Parameters (filter, a, b, rshift, flags)

- filter;
- a - container with the feedforward filter coefficients;
- b - container with the feedback filter coefficients; optional; can be -1 (for FIR filters);
- rshift - bitwise right shift for fixed point computations; optional;
- flags - for future use; optional.

Return value: 0 if success.

apply_filter()

Parameters (filter, output, input, flags, offset, size)

- filter;
- output - output container;
- input - input container;
- flags - for future use; optional;
- offset - output and input offset; optional;
- size - size of the processed block; optional.

Return value: 0 if success.

replace_values()

For each element of dest container: dest[i] = values[(unsigned)src[i]].The dest container must have the same type as the values.

Parameters (dest, src, values, dest_offset, src_offset, size)

- dest - destination;
- src - source;
- values - substitution values;
- dest_offset - offset in the destination container; optional;
- src_offset - offset in the source container; optional;
- size - number of elements to replace; optional.

Examples

```
//Convert 8-bit image with palette to the screen pixel format:
replace_values( scr, img8, palette )
```

copy_and_resize()

Parameters (dest, src, flags, dest_x, dest_y, dest_rect_xsize, dest_rect_ysize, src_x, src_y, src_rect_xsize, src_rect_ysize)

- dest - destination container;
- src - source container;
- flags - flags RESIZE_*; optional; default = RESIZE_INTERP1;
- dest_x, dest_y, dest_rect_xsize, dest_rect_ysize, src_x, src_y, src_rect_xsize, src_rect_ysize - optional parameters.

conv_filter()

Apply the convolution filter (convolution matrix).

Parameters (dest, src, kernel, div, offset, flags, kernel_xcenter, kernel_ycenter, dest_x, dest_y, src_x, src_y, xsize, ysize, xstep, ystep)

- dest - destination container;
- src - source container;
- kernel - kernel container (convolution matrix);
- div - division of the result value (default is 1); optional;
- offset - offset of the result value (default is 0); optional;
- flags - flags CONV_FILTER_*; optional;
- kernel_xcenter - central element of the matrix horizontally (from 0; default is kernel width / 2); optional;
- kernel_ycenter - central element of the matrix vertically (from 0; default is kernel height / 2); optional;
- dest_x, dest_y, src_x, src_y, xsize, ysize, xstep, ystep - optional parameters.

Dialogs

file_dialog()

Open file dialog.

Parameters (dialog_name, mask, id, default_name, flags)

- dialog_name;
- mask - file type mask (examples: "gif/jpg" for .gif and .jpg files; "" for all files);
- id - name of the file for saving the current dialog state;
- default_name - default file name; optional;
- flags - flags FDIALOG_FLAG_*; optional.

Return value: string container with the name of the selected file; or -1 if file not selected; (container must be removed manually).

prefs_dialog()

Open window with global Pixilang preferences.

textinput_dialog()

Open SunDog-based text input dialog and return the entered string. Only Latin letters are supported now.

Parameters (default_text, dialog_name)

- default_text; optional;

- dialog_name; optional.

Return value: string container with the entered text; or -1 in case of cancellation; (container must be removed manually).

Network

open_url()

Open web browser window with selected URL.

Parameters (url_string)

Native code

dlopen()

Open dynamic library (for example - .DLL file in Windows, or .SO file in Linux).

Parameters (lib_file_name)

Return value: library ID or -1 if error occured.

Examples

```
//For example we have some C function int show_info( int x, int y, void* data )
//in mylib.dll library.
//Let's call it from Pixilang:
dl = dlopen( "mylib.dll" ) //Open the library
if dl >= 0
{
  f = dlsym( dl, "show_info", "iip" ) //Get the function show_info() from dynamic library
  // "iip" - int int pointer
  if f >= 0
  {
    retval = dlcall( dl, f, 1, 2, "blahblah" ) //Call the function show_info() with parameters 1, 2, "blahblah"
  }
  dlclose( dl ) //Close the library
}
```

dlclose()

Close dynamic library.

Parameters (lib_id)

dlsym()

Get symbol (function or variable) from dynamic library.

Parameters (lib_id, symbol_name, format, calling_convention)

- lib_id;
- symbol_name - function/variable name;
- format - function format; optional; text string with the following structure: "R(P)", where the R - return value type (one ascii character), P - parameter types (multiple ascii characters or empty); below is a list of characters that are used for the construction of this string:
    - v - void;
    - c - signed int8;
    - C - unsigned int8

    - s - signed int16;
    - S - unsigned int16;
    - i - signed int32;
    - I - unsigned int32;
    - l - signed int64;
    - L - unsigned int64;
    - f - float32;
    - d - double64;
    - p - pointer;
- calling_convention - one of the CCONV_* constants; optional; default - CCONV_DEFAULT.

Return value: symbol ID or -1 if error occured.

dlcall()

Call the function from dynamic library.

Parameters (lib_id, symbol_id, optional_function_parameters)

System functions

system()

Issue a OS command.

Parameters (command)

Return value: termination status of the command.

Examples

```
//Remove some file:
system( "rm /tmp/data.txt" )
```

argc()

Returns the number of arguments.

argv()

Returns the container with selected argument.

Parameters (n)

Examples

```
if argc() >= 4
{
  a = argv( 3 )
  remove( a )
}
```

exit()

Quit from Pixilang.

Parameters (exit_code)

Examples

```
exit( 4 ) //Exit with code 4
```