PDF версия

Что такое Pixilang

Pixilang - кросс-платформенный язык программирования для небольших графических и звуковых приложений. Примеры применения: арт-эксперименты, демки (demoscene), синтезаторы, игры. Концепция языка разработана в 2006 году Александром Золотовым (NightRadio) и Михаилом Разуваевым (Goglus).

Pixilang-программы хранятся в текстовых файлах (UTF8) с расширением .txt или .pixi. Поэтому вы можете использовать любой текстовый редактор для создания таких программ. Pixilang не имеет встроенного редактора. После старта появляется файловый диалог, в котором нужно указать, где лежит запускаемая pixi-программа.

Ключевые особенности:

- простые правила, низкий порог вхождения;
- поддержка графических и звуковых форматов файлов;
- программу можно писать без объявления функций, просто списком инструкций с условными переходами;
- сразу после старта программе выделяется чистая область (экран) внутри окна, к которой можно обращаться как к массиву пикселей и использовать готовые графические примитивы.

Запуск из командной строки

При запуске из командной строки Pixilang принимает дополнительные опции в приведенном ниже формате.

```
pixilang [options] [source_file_name]
[options]
  -c    генерация байт-кода; будет создан файл source_file_name.pixicode.
        Примечание: файлы в формате *.pixicode привязаны к архитектуре и
могут неправильно работать на других устройствах.
```

Основы

В основе Pixilang - контейнеры (или pixi-контейнеры, как их иногда называют) и переменные.

Что такое контейнер? Если в двух словах, то это двумерный массив, таблица из X колонок и Y строк. Каждая ячейка этой таблицы - число определенного формата. Формат задан один на весь контейнер. Например, ячейки могут хранить цвета пикселей, тогда контейнер превращается в картинку. Контейнер с таким же успехом может быть строкой текста, куском звука и т.д. Если вы знакомы с другими языками программирования, то считайте контейнер массивом, состоящим из (X*Y) ячеек. Каждый контейнер после создания имеет свой ID

(порядковый номер).

Структура контейнера:

- двумерный массив (таблица) элементов контейнера;
- key color - цвет, который будет отображаться, как прозрачный;
- ссылка на контейнер (должен быть типа INT8) с альфа-каналом - для изображений с плавными изменениями прозрачности;
- дополнительные данные:
  - свойства (используйте функции get_prop(), set_prop(), remove_props() или оператор . (точка) для доступа к ним);
  - анимация (используйте функции для анимации контейнеров).

Новый контейнер можно получить используя функцию new() или какую-то другую специальную функцию, возвращающую блок данных. Контейнер удаляется двумя способами:

- пользователем при помощи функции remove();
- автоматически после завершения программы.

В Pixilang отсутствует автоматический сборщик мусора, поэтому будьте внимательны - каждый новый контейнер отнимает память до тех пор, пока вы не удалите этот контейнер.

Переменная - имя ячейки памяти, в которой хранится одно знаковое целое 32-битное число (например, 25) или 32-битное число с плавающей запятой (например, 33.44). Локальные переменные (с символом $ перед именем) доступны только в рамках одной функции, в которой эти переменные определены. Глобальные переменные (без символа $) доступны в любом месте программы.

Числа можно описывать в различных форматах. Примеры:

- 33 - обычное целое десятичное;
- 33.55 - десятичное с плавающей запятой;
- 0xA8BC - шестнадцатеричное;
- 0b100101011 - двоичное;
- #FF9080 - цвет, как в HTML; общий формат такой: #RRGGBB, где RR - яркость красного, GG - яркость зеленого; BB - яркость голубого.

В каком бы формате вы не описали число, внутри Pixilang оно все равно будет 32-битным целым или 32-битным с плавающей запятой.

Простейшие примеры применения контейнеров и переменных:

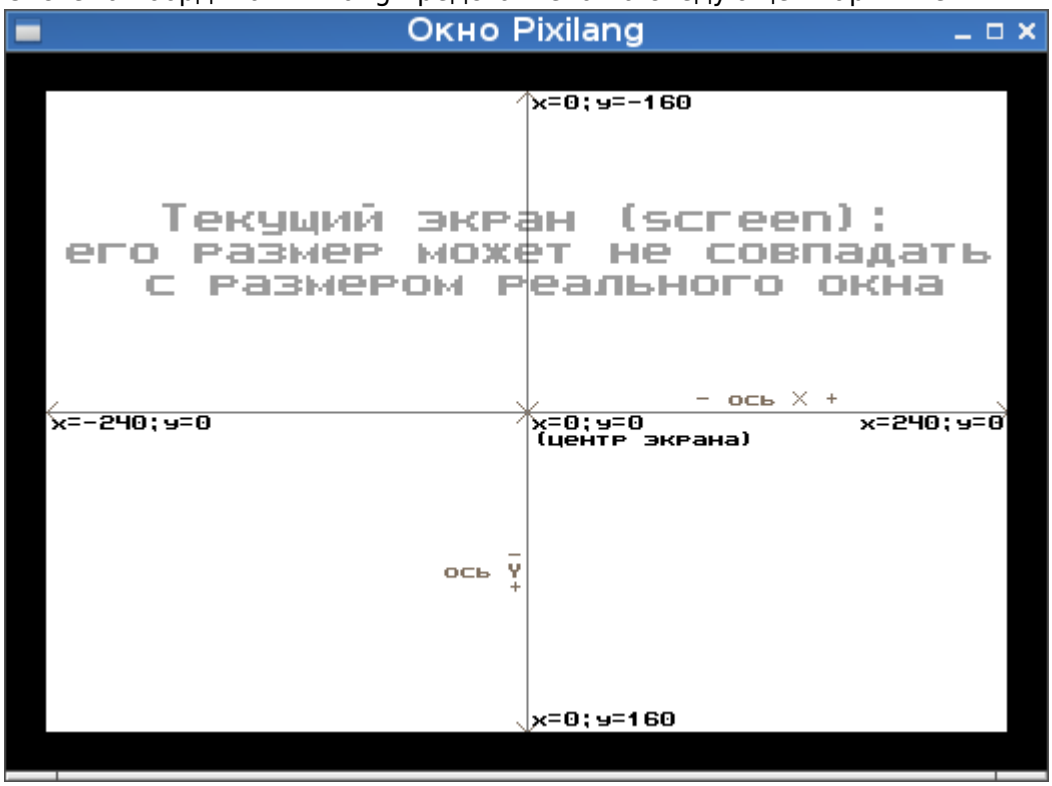
```
x = new( 4 ) //Создаем контейнер из 4х пикселей. Сохраняем ID контейнера в переменную x.
x[ 2 ] = WHITE //Присваиваем пикселю под номером 2 белый цвет.
remove( x ) //Удаляем контейнер
```

```
c = new( 4, 4 ) //Создаем 2D контейнер 4 на 4 пикселя. Сохраняем ID контейнера в переменную c.
c[ 2, 2 ] = WHITE //Присваиваем пикселю с координатами 2,2 (относительно левого верхнего угла) белый цвет.
remove( c ) //Удаляем контейнер
```

```
str = "Hello" //"Hello" - это строковый контейнер, состоящий из пяти 8-битных символов (кодировка UTF-8).
//Подобные контейнеры-строки создаются автоматически на этапе компиляции программы.
//Удалять их вручную так, как это сделано в предыдущем примере, не надо.
//Контейнеру со строкой "Hello" автоматически присвоится ID (порядковый номер).
//Например, ID будет равен 4.
//Тогда код str = "Hello" будет равноценен коду str = 4.
str[ 0 ] = 'h' //Меняем самую первую букву в строке. Было - 'H'. Станет - 'h'.
```

```
a = 4 //Глобальная переменная
fn function()
{
  $k = 2 //Локальная переменная
  function2 = {
    //Определяем еще одну функцию
    $x = 899.334 //Локальная переменная
    //В этом месте $k недоступна, т.к. находится в другой функции
  }
  //В этом месте $x недоступна
}
//В этом месте $k и $x недоступны
```

Система координат Pixilang представлена на следующей картинке:



Имена файлов и директорий

Ниже приведены примеры, показывающие основные правила оформления имен файлов и директорий.

```
//Файл спрятан в нескольких директориях относительно текущего местоположения пикси-программы:
"folder1/folder2/folder3/filename"

//Файл лежит в текущей рабочей папке Pixilang
//  iOS: documents;
//  WinCE: корень файловой системы (/);
//  для остальных систем: в той же самой папке, в которой лежит Pixilang;
"1:/filename"

//Файл лежит в директории для настроек
//  Linux: /home/username/.config/Pixilang;
//  iOS: папка для кэшируемых данных (NSCachesDirectory);
//  Android: папка с настройками на внутренней карте памяти устройства;
"2:/filename"

//Файл лежит во временной директории:
"3:/filename"
```

Встроенные операторы

Рассмотрим операторы на конкретных примерах.

```
//Условные операторы if, else
if a == b
  { /*Код в этом месте выполняется, если a равно b*/ }
else
  { /*Код в этом месте выполняется в противном случае (a не равно b)*/ }
if x == 4 && y == 2
  { /*Код в этом месте выполняется, если x равно 4 и y равно 2*/ }

//Оператор цикла: while
a = 0
while( a < 3 )
{
  //Код в этом месте выполняется, если a меньше 3
  a + 3
}

//Операторы цикла: while, break
a = 0
```

```
while( a < 100 )
{
  //Код в этом месте выполняется, если a меньше 100
  if a == 10 { break } //Если a = 10, то разрываем цикл оператором break
  //Для остановки нескольких вложенных циклов сразу можно использовать
оператор breakX,
  //где X - глубина. Например break2 остановит два цикла.
  //А при помощи оператора breakall можно остановить все циклы,
  //которые активны в данный момент для текущего потока выполнения.
  a + 1
}

//Операторы цикла: while, continue
a = 0
b = 0
while( a < 100 )
{
  //Код в этом месте выполняется, если a меньше 100
  if a == 10 { a + 1 continue } //Если a = 10, то переходим к следующей
итерации цикла
                                //             (игнорируем следующие две
строчки кода)
  a + 1
  b + 1
}

//Операторы перехода: go, goto
m1:
a + 1
goto m1 //Переход на метку m1

//Операторы остановки: halt, stop
halt //В этом месте программа останавливается

//Оператор подключения: include
include "prog2.txt" //В этом месте подключаем код из файла prog2.txt

//Оператор определения функции: fn
fn fff( $x, $y ) //Определяем функцию fff с параметрами $x и $y
{
  //Код функции fff
  ret //Простой выход из функции
  ret( 4 ) //Выход из функции с возвращением значения 4
}
```

Ниже приведена таблица математических операторов. Приоритет 0 - наивысший, такие операции будут выполняться в первую очередь.

Приоритет	Оператор	Описание	Результат	Пример
0	%	Деление по модулю	Целое число	a = b % 4

Приоритет	Оператор	Описание	Результат	Пример
0	/	Деление	Число с плавающей запятой	a = b / 4
0	div	Целочисленное деление	Целое число	a = b div 4
0	*	Умножение	Зависит от операндов	a = b * 4
1	+	Сложение	Зависит от операндов	a = b + 4
1	-	Вычитание	Зависит от операндов	a = b - 4
2	>>	Битовый сдвиг вправо	Целое число	a = b >> 4
2	<<	Битовый сдвиг влево	Целое число	a = b << 4
3	==	Равно	Целое число 1 или 0	if a == b {}
3	!=	Не равно	Целое число 1 или 0	if a != b {}
3	<	Меньше	Целое число 1 или 0	if a < b {}
3	>	Больше	Целое число 1 или 0	if a > b {}
3	<=	Меньше или равно	Целое число 1 или 0	if a <= b {}
3	>=	Больше или равно	Целое число 1 или 0	if a >= b {}
4	|	Побитовая операция ИЛИ (OR)	Целое число	a = b | 4
4	^	Побитовая операция исключающего ИЛИ (XOR)	Целое число	a = b ^ 4
4	&	Побитовая операция И (AND)	Целое число	a = b & 4
5	||	Логическая операция ИЛИ (OR)	Целое число 1 или 0	if a || b {}
5	&&	Логическая операция И (AND)	Целое число 1 или 0	if a && b {}

Встроенные константы

Типы контейнеров

Контейнер может содержать элементы одного из нижеперечисленных типов.

64-битные типы в текущей версии не поддерживается, но поддержка может быть включена при самостоятельной сборке Pixilang из исходников. Для включения необходимо внести небольшие правки в файле pixilang.h в разделе Configuration.

- INT - знаковое целое (размер зависит от версии Pixilang);
- INT8 - знаковое целое (8 бит); диапазон значений: от -128 до 127;
- INT16 - знаковое целое (16 бит); диапазон значений: от -32768 до 32767;
- INT32 - знаковое целое (32 бита); диапазон значений: от -2147483648 до 2147483647;
- INT64 - знаковое целое (64 бита);
- FLOAT - плавающая запятая (размер зависит от версии Pixilang);
- FLOAT32 - плавающая запятая (32 бита);
- FLOAT64 - плавающая запятая (64 бита);
- PIXEL - цвет пикселя (формат зависит от версии Pixilang); после создания контейнера это значение превращается в INTx, где x - кол-во бит на пиксель.

Флаги (опции) контейнера

- CFLAG_INTERP - включить программную интерполяцию.

Для OpenGL:

- GL_MIN_LINEAR;
- GL_MAG_LINEAR;
- GL_NICEST - использовать наилучшее отображение и 32-битный цвет, когда возможно;
- GL_NO_XREPEAT;
- GL_NO_YREPEAT;
- GL_NO_ALPHA.

Флаги (опции) для функции resize()

- RESIZE_INTERP1 - грубое масштабирование;
- RESIZE_INTERP2 - масштабирование с линейной интерполяцией;
- RESIZE_UNSIGNED_INTERP2 - аналогично RESIZE_INTERP2, но все числа контейнера рассматриваются как беззнаковые;
- RESIZE_COLOR_INTERP1 - аналогично RESIZE_INTERP1, но для цветов пикселей;
- RESIZE_COLOR_INTERP2 - аналогично RESIZE_INTERP2, но для цветов пикселей.

Флаги (опции) для функции copy()

- COPY_NO_AUTOROTATE - не переворачивать данные после копирования из GL_SCREEN;
- COPY_CLIPPING - проверять и ограничивать копируемые значения, если типы контейнеров не совпадают (разная разрядность типов данных).

Размеры

- INT_SIZE - максимальный размер (в байтах) знакового целого числа, с которым может работать Pixilang.
- FLOAT_SIZE - максимальный размер (в байтах) числа с плавающей запятой, с которым может работать Pixilang.
- INT_MAX - максимальное положительное целое;
- COLORBITS - количество бит на пиксель.

ZLib

Уровни компрессии:

- Z_NO_COMPRESSION;
- Z_BEST_SPEED;
- Z_BEST_COMPRESSION;

- Z_DEFAULT_COMPRESSION.

Типы файлов

- FORMAT_RAW;
- FORMAT_JPEG;
- FORMAT_PNG;
- FORMAT_GIF;
- FORMAT_WAVE;
- FORMAT_AIFF (поддерживается только загрузка AIFF файлов);
- FORMAT_PIXICONTAINER - весь контейнер целиком (со свойствами и анимацией).

Опции для load() и save()

- LOAD_FIRST_FRAME - загрузить только первый кадр.

Сохранение GIF:

- GIF_GRAYSCALE;
- GIF_DITHER.

Сохранение JPEG:

- 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 - удвоенное количество проходов.

Цвета

- ORANGE - ораньжевый;
- BLACK - черный;
- WHITE - белый;
- YELLOW - желтый;
- RED - красный;
- GREEN - зеленый;
- BLUE - синий.

Выравнивание

Эти константы используются, например, при выводе текста. Их можно комбинировать при помощи побитовой операции ИЛИ (|). Отсутствие TOP и BOTTOM обозначает вертикальное выравнивание по центру. Отсутствие LEFT и RIGHT обозначает горизонтальное выравнивание по центру.

- TOP - по верхнему краю;

- BOTTOM - по нижнему краю;
- LEFT - по левому краю;
- RIGHT - по правому краю.

Типы эффектов

Типы эффектов для функции effector():

- EFF_NOISE - шум;
- EFF_SPREAD_LEFT - случайные сдвиги пикселей влево;
- EFF_SPREAD_RIGHT - случайные сдвиги пикселей вправо;
- EFF_SPREAD_UP - случайные сдвиги пикселей вверх;
- EFF_SPREAD_DOWN - случайные сдвиги пикселей вниз;
- EFF_HBLUR - горизонтальное размывание;
- EFF_VBLUR - вертикальное размывание;
- EFF_COLOR - сплошная заливка одним цветом.

OpenGL

Режимы для gl_draw_arrays() (аналог функции glDrawArrays()):

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

Константы для gl_blend_func() (аналог функции glBlendFunc()):

- 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.

Параметры состояния (для gl_get_int() и 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.

Встроенные шейдеры для функции gl_new_prog():

- GL_SHADER_SOLID - заливка одним цветом;
- GL_SHADER_GRAD - градиентная заливка;
- GL_SHADER_TEX_ALPHA_SOLID - заливка одним цветом + одноканальная текстура (только канал прозрачности);
- GL_SHADER_TEX_ALPHA_GRAD - градиентная заливка + одноканальная текстура (только канал прозрачности);
- GL_SHADER_TEX_RGB_SOLID - заливка одним цветом + текстура;
- GL_SHADER_TEX_RGB_GRAD - градиентная заливка + текстура.

Глобальные OpenGL контейнеры:

- GL_SCREEN; вы не можете обращаться к экрану GL_SCREEN, как к массиву пикселей напрямую, но вы можете использовать для этих целей функцию copy(dest, GL_SCREEN);
- GL_ZBUF.

Звук

Флаги для функции set_audio_callback():

- AUDIO_FLAG_INTERP2 - линейная интерполяция (по двум точкам).

MIDI

Флаги для функций midi_get_device(), midi_open_port():

- MIDI_PORT_READ;
- MIDI_PORT_WRITE.

События

- EVT - ID контейнера, в который помещается событие из функции get_event().

Номера ячеек (полей) в контейнере EVT:

- EVT_TYPE - тип;
- EVT_FLAGS - флаги;
- EVT_TIME - время;
- EVT_X - координата X (0 - центр экрана);
- EVT_Y - координата Y (0 - центр экрана);
- EVT_KEY - ASCII код нажатой клавиши или одна из констант KEY_xxx;

- EVT_SCANCODE - сканкод (если имеется) или номер нажатия на экран (только для multitouch устройств; от нуля);
- EVT_PRESSURE - давление нажатия (норма = 1024);
- EVT_UNICODE - юникод (если имеется).

Типы событий (для поля EVT_TYPE):

- EVT_MOUSEBUTTONDOWN - клик мышкой, первое нажатие на экран;
- EVT_MOUSEBUTTONUP;
- EVT_MOUSEMOVE;
- EVT_TOUCHBEGIN - второе, третье и т.д. нажатие на экран (только для multitouch устройств);
- EVT_TOUCHEND;
- EVT_TOUCHMOVE;
- EVT_BUTTONDOWN;
- EVT_BUTTONUP;
- EVT_SCREENRESIZE;
- EVT_QUIT - виртуальная машина будет закрыта; если при этом EVT[EVT_SCANCODE] = 0, то данное событие нельзя игнорировать, виртуальная машина может подождать пользовательский код максимум 0.5 секунды; а если EVT[EVT_SCANCODE] = 1, то событие можно игнорировать, например выдать диалоговое окно “Вы действительно хотите выйти?”.

Флаги события (для поля EVT_FLAGS):

- EVT_FLAG_SHIFT;
- EVT_FLAG_CTRL;
- EVT_FLAG_ALT;
- EVT_FLAG_MODE;
- EVT_FLAG_MODS (маска модификаторов со всеми флагами типа Shift, Ctrl и т.д.);
- EVT_FLAG_DOUBLECLICK.

Коды клавиш (для поля EVT_KEY):

- 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).

Константы для функции set_quit_action():

- QA_NONE;
- QA_CLOSE_VM.

Многопоточность

Флаги (опции) для thread_create():

- THREAD_FLAG_AUTO_DESTROY - поток с этим флагом будет удален автоматически после завершения.

Математические константы

- 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);

Операции обработки данных

Для функции op_cn():

- 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 - сумма элементов; 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 - абсолютное значение;
- OP_SUB2 - вычитание с измененным порядком операндов (N - C1[i]);
- OP_COLOR_SUB2 - то же, что OP_COLOR_SUB, но с измененным порядком операндов (N - C1[i]);
- OP_DIV2 - то же, что OP_DIV, но с измененным порядком операндов (N / C1[i]);
- OP_H_INTEGRAL - дискретный интеграл (по горизонтали);
- OP_V_INTEGRAL - дискретный интеграл (по вертикали);
- OP_H_DERIVATIVE - дискретная производная (по горизонтали);
- OP_V_DERIVATIVE - дискретная производная (по вертикали);
- OP_H_FLIP;
- OP_V_FLIP.

Для функций op_cn(), op_cc():

- OP_ADD - сложение;
- OP_SADD - сложение с ограничением (защита от переполнения);
- OP_COLOR_ADD - сложение цветов (отдельно складываются значения яркости красного, зеленого и синего) с ограничением;
- OP_SUB - вычитание;
- OP_SSUB - вычитание с ограничением (защита от переполнения);
- OP_COLOR_SUB - вычитание цветов с ограничением;
- OP_MUL - умножение;
- OP_SMUL - умножение с ограничением (защита от переполнения);
- OP_MUL_RSHIFT15 - умножение с последующим сдвигом вправо на 15 бит;
- OP_COLOR_MUL - умножение цветов;
- OP_DIV - деление;
- OP_COLOR_DIV - деление цветов;
- OP_AND - побитовая операция И;
- OP_OR - побитовая операция ИЛИ;
- OP_XOR - побитовая операция исключающего ИЛИ;
- OP_LSHIFT - битовый сдвиг влево;
- OP_RSHIFT - битовый сдвиг вправо;
- OP_EQUAL;
- OP_LESS;
- OP_GREATER;
- OP_COPY - копирование;
- OP_COPY_LESS - копировать только те элементы, для которых справедливо C1[i] < C2[i];
- OP_COPY_GREATER - копировать только те элементы, для которых справедливо C1[i] > C2[i].

Для функции op_cc():

- OP_BMUL - if C2[i] == 0 { C1[i] = 0 };
- OP_EXCHANGE;
- OP_COMPARE - сравнение; если сравниваемые участки равны, операция возвращает 0; если первый отличный элемент C1 больше элемента в C2, возвращается 1, если меньше -

возвращается -1.

Для функции op_ccn():

- OP_MUL_DIV - перемножить значения из контейнеров и поделить результат на число;
- OP_MUL_RSHIFT - перемножить значения из контейнеров и сдвинуть результат на указанное кол-во бит вправо.

Для функции generator():

- OP_SIN - синус;
- OP_SIN8 - быстрый, но менее точный синус (вычисляется по таблице 8-битных значений);
- OP_RAND - псевдо-случайные числа (в диапазоне от -amp до +amp).

Sampler

- SMP_INFO_SIZE - размер контейнера с информацией о сэмпле.

Номера ячеек (полей) в контейнере с информацией о сэмпле:

- SMP_DEST - ID контейнера, в который будет производиться запись;
- SMP_DEST_OFF - смещение в контейнере для записи;
- SMP_DEST_LEN - длина участка для записи;
- SMP_SRC - ID контейнера с сэмплом;
- SMP_SRC_OFF_H - смещение сэмпла (левая часть числа с фиксированной точкой);
- SMP_SRC_OFF_L - смещение сэмпла (правая часть числа с фиксированной точкой, от 0 до 65535);
- SMP_SRC_SIZE - размер сэмпла (0 - весь сэмпл);
- SMP_LOOP - начало лупа (повторяемого участка сэмпла);
- SMP_LOOP_LEN - длина лупа (или 0, если луп отключен);
- SMP_VOL1 - начальная громкость (32768 = 1.0);
- SMP_VOL2 - конечная громкость (32768 = 1.0);
- SMP_DELTA - дельта (скорость проигрывания); fixed point (real_value * 65536);
- SMP_FLAGS - флаги.

Флаги:

- SMP_FLAG_INTERP2 - линейная интерполяция (по двум точкам);
- SMP_FLAG_INTERP4 - кубическая сплайновая интерполяция (по четырем точкам);
- SMP_FLAG_PINGPONG - режим ping-pong для лупа;
- SMP_FLAG_REVERSE - обратное направление проигрывания.

Константы для нативных вызовов

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

Константы для совместимости с POSIX

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

Разное

- PIXILANG_VERSION - версия Pixilang ((major<<24) + (minor<<16) + (minor2<<16) + minor3); например, если версия языка = 3.4.7, то PIXILANG_VERSION будет 0x03040700;
- OS_NAME - контейнер с названием операционной системы, в которой запущен Pixilang; примеры: “ios”, “win32”, “linux”;
- ARCH_NAME - контейнер с названием архитектуры процессора; примеры: “x86”, “x86_64”, “arm”, “mips”;
- LANG_NAME - контейнер с именем системного языка в POSIX формате [language][_TERRITORY][.CODESET][@modifier]; примеры: en_US, ru_RU.utf8;
- CURRENT_PATH;
- USER_PATH;
- TEMP_PATH;
- OPENGL - 1, если OpenGL доступен; 0 - в противном случае.

Встроенные глобальные переменные

- WINDOW_XSIZE - ширина пользовательского окна (в пикселях);
- WINDOW_YSIZE - высота пользовательского окна (в пикселях);
- WINDOW_SAFE_AREA_X;
- WINDOW_SAFE_AREA_Y;
- WINDOW_SAFE_AREA_W;
- WINDOW_SAFE_AREA_H;
- FPS - последнее подсчитанное количество кадров в секунду;
- PPI - количество пикселей на дюйм;
- UI_SCALE - коэффициент масштабирования интерфейса (задан пользователем в глобальных настройках); пример использования: button_size = PPI * UI_SCALE * 0.5;
- UI_FONT_SCALE - аналогично UI_SCALE, но для масштабирования текста (размер шрифта).

Зарезервированные свойства контейнера

Эти свойства могут использоваться при проигрывании, сохранении и загрузке контейнеров:

- file_format;
- sample_rate;
- channels;
- loop_start - начальная позиция (номер звукового фрейма) петли;
- loop_len - длина петли (кол-во звуковых фреймов);
- loop_type - тип петли: 0-выкл.; 1-обычная; 2-двунаправленная (сначала в одну сторону, потом обратно);
- frames - количество кадров;
- frame - номер текущего кадра;
- fps - кол-во кадров в секунду;
- play - состояние автоматического проигрывания (0 - выкл, 1 - вкл); если включено, то кадры будут меняться автоматически во время вызова функции pixi();
- repeat - количество повторов (-1 - бесконечно);
- start_time - время начала (в системных тиках); автоматически заполняется после вызова play();
- start_frame - начальный кадр; автоматически заполняется после вызова play().

Встроенные функции

Работа с контейнерами

new

Создать новый контейнер с данными.

Сразу после создания контейнер может быть заполнен неопределенными значениями. Прежде чем читать из этого контейнера, его следует очистить функцией clean или заполнить полезными данными.

Параметры (xsize, ysize, type)

- xsize - ширина.
- ysize - высота.
- type - тип контейнера (а точнее - тип элементов, из которых состоит контейнер). Возможны следующие типы:
  - INT - знаковое целое (размер зависит от версии Pixilang);
  - INT8 - знаковое целое (8 бит);
  - INT16 - знаковое целое (16 бит);
  - INT32 - знаковое целое (32 бита);
  - INT64 - знаковое целое (64 бита);
  - FLOAT - плавающая запятая (размер зависит от версии Pixilang);
  - FLOAT32 - плавающая запятая (32 бита);
  - FLOAT64 - плавающая запятая (64 бита);
  - PIXEL - пиксель; после создания контейнера это значение превращается в INTx, где x - кол-во бит на пиксель.

Возвращаемое значение: ID контейнера или -1, если произошла ошибка при создании.

Примеры

```
p = new() //Создать контейнер 1x1. Тип = пиксели.
p = new( 4 ) //Создать контейнер 4x1. Тип = пиксели.
p = new( 4, 4 ) //Создать контейнер 4x4. Тип = пиксели.
p = new( 4, 4, FLOAT32 ) //Создать контейнер 4x1. Тип = 32-битные числа с плавающей запятой.
```

remove

Удалить контейнер.

Параметры (pixi)

- pixi - ID контейнера.

Примеры

```
p = new() //Создаем новый контейнер
remove( p ) //Удаляем его
```

remove_with_alpha

Удалить контейнер и связанный с ним контейнер альфа-канала (прозрачность).

Параметры (pixi)

- pixi - ID контейнера.

resize

Изменить параметры контейнера.

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

- pixi - ID контейнера;
- xsize - ширина;
- ysize - высота; опциональный;
- type - тип элементов, из которых состоит контейнер; опциональный; возможны следующие типы:
  - INT8 - знаковое целое (8 бит);
  - INT16 - знаковое целое (16 бит);
  - INT32 - знаковое целое (32 бита);
  - INT64 - знаковое целое (64 бита);
  - FLOAT32 - плавающая запятая (32 бита);
  - FLOAT64 - плавающая запятая (64 бита);
  - PIXEL - пиксель; после создания контейнера это значение превращается в INTx, где x - кол-во бит на пиксель;

- flags - опции масштабирования (сочетание флагов RESIZE_xxx); опциональный.

-1 в любом из параметров (кроме опций) будет означать, что данный параметр контейнера остается без изменений.

Возвращаемое значение: 0 - успешно; 1 - ошибка.

Примеры

```
p = new() //Создаем новый контейнер 1x1
resize( p, 32 ) //Изменяем его размер на 32x1
remove( p ) //Удаляем контейнер
```

rotate

Перевернуть контейнер на угол angle*90 градусов, по часовой стрелке.

Параметры (pixi, angle)

convert_type

Преобразовать значения контейнера к другому типу.

Параметры (pixi, new_type)

clean

Очистить контейнер (заполнить нулями или указанным значением).

Параметры (dest_cont, v, offset, count)

- dest_cont - ID контейнера;
- v - число, которым надо заполнить контейнер; опциональный; по умолчанию - 0;
- offset - смещение внутри dest_cont; опциональный; по умолчанию - 0;
- count - количество элементов, которые нужно заполнить; опциональный; по умолчанию - весь контейнер.

Примеры

```
p = new() //Создаем новый контейнер
clean( p ) //Очищаем его
remove( p ) //И удаляем
```

clone

Создать точную копию контейнера.

Параметры (pixi)

- pixi - ID контейнера.

Возвращаемое значение: ID нового контейнера или -1, если произошла ошибка при создании.

copy

Скопировать данные из контейнера src в контейнер dest.

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

- dest - приемник (куда);
- src - источник (откуда);
- dest_offset - номер первого копируемого элемента в приемнике;
- src_offset - номер первого копируемого элемента в источнике;
- count - количество элементов, которое нужно скопировать;
- dest_step - шаг, с которым записываются элементы в приемник (по умолчанию - 1);
- src_step - шаг, с которым считываются элементы из источника (по умолчанию - 1).
- flags - опции (сочетание флагов COPY_xxx); опциональный.

Примеры

```
//Скопировать все элементы из контейнера img1 в img2:
copy( img2, img1 )
//Скопировать элементы 8...200 из контейнера img1 в img2:
copy( img2, img1, 8, 8, 200 )
//Скопировать 200 элементов начиная с 8 из контейнера img1 в img2 с шагом 2:
copy( img2, img1, 8, 8, 200, 2, 2 )
```

Возвращаемое значение: количество элементов, которые удалось успешно скопировать.

get_size

Получить размер контейнера (кол-во элементов).

Параметры (pixi)

- pixi - ID контейнера.

Возвращаемое значение

Размер контейнера (кол-во элементов).

Примеры

```
p = new( 8, 8 ) //Создаем новый контейнер 8x8
size = get_size( p ) //Записываем его размер в переменную size
```

```
remove( p ) //Удаляем контейнер
```

get_xsize

Получить ширину контейнера.

Параметры (pixi)

- pixi - ID контейнера.

Возвращаемое значение: ширина контейнера.

Примеры

```
p = new( 8, 8 ) //Создаем новый контейнер 8x8
xsize = get_xsize( p ) //Записываем его ширину в переменную xsize
remove( p ) //Удаляем контейнер
```

get_ysize

Получить высоту контейнера.

Параметры (pixi)

- pixi - ID контейнера.

Возвращаемое значение: высота контейнера.

Примеры

```
p = new( 8, 8 ) //Создаем новый контейнер 8x8
ysize = get_xsize( p ) //Записываем его высоту в переменную ysize
remove( p ) //Удаляем контейнер
```

get_esize

Получить размер элемента контейнера (в байтах).

Параметры (pixi)

- pixi - ID контейнера.

Возвращаемое значение: размер элемента контейнера (в байтах).

Примеры

```
p = new( 8, 8, INT16 ) //Создаем новый контейнер 8x8; тип элемента = INT16
esize = get_esize( p ) //Записываем размер его элемента в переменную esize
//Теперь в переменной esize находится число 2 (т.к. 16 бит - это два байта).
```

```
remove( p ) //Удаляем контейнер
```

get_type

Получить тип элемента контейнера.

Параметры (pixi)

- pixi - ID контейнера.

Возвращаемое значение: тип элемента контейнера.

Примеры

```
p = new( 8, 8, INT32 ) //Создаем новый контейнер 8x8; тип элемента = INT32
type = get_type( p ) //Записываем тип его элемента в переменную type
//Теперь в переменной type находится константа INT32.
remove( p ) //Удаляем контейнер
```

get_flags

Получить флаги контейнера. Под флагами подразумевается 32-битное число, каждый бит в котором отвечает за включение/выключение какой-то опции контейнера. Полный список флагов можно посмотреть в этом разделе.

Параметры (pixi)

- pixi - ID контейнера.

Возвращаемое значение: флаги контейнера.

set_flags

Установить флаги контейнера. Под флагами подразумевается 32-битное число, каждый бит в котором отвечает за включение/выключение какой-то опции контейнера. Полный список флагов можно посмотреть в этом разделе.

Параметры (pixi, flags)

- pixi - ID контейнера;
- flags - флаги.

Примеры

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

reset_flags

Сбросить флаги контейнера.

Параметры (pixi, flags)

- pixi - ID контейнера;
- flags - флаги.

get_prop

Получить значение свойства контейнера. У каждого контейнера может быть неограниченное количество свойств.

Другой путь получения свойства контейнера - использовать оператор . (точка). Например: value = image.fps

Параметры (pixi, prop_name, def_prop)

- pixi - ID контейнера;
- prop_name - имя свойства;
- def_prop - это значение возвратится функцией, если свойство не существует; опциональный.

Возвращаемое значение: числовое значение указанного свойства контейнера.

set_prop

Установить значение свойства контейнера.

Другой путь установки свойства контейнера - использовать оператор . (точка). Например: image.fps = 20

Параметры (pixi, prop_name, value)

- pixi - ID контейнера;
- prop_name - имя свойства;
- value - числовое значение.

Примеры

```
p = new( 8, 8, INT32 ) //Создаем новый контейнер 8x8; тип элемента = INT32
set_prop( p, "speed", 777 ) //Добавляем свойство "speed" у созданного
контейнера
v = get_prop( p, "speed" ) //Читаем значение этого свойства
//Теперь в переменной v лежит число 777
//Рассмотрим так же более простой способ работы со свойствами контейнера
//(при помощи оператора . (точка)):
p.speed = 777
v = p.speed
```

remove_props

Удалить все свойства контейнера.

Параметры (pixi)

show_memory_debug_messages

Включить/выключить отладочные сообщения менеджера управления памятью.

Параметры (enable)

zlib_pack

Запаковать контейнер библиотекой Zlib.

Параметры (source, level)

- source - контейнер, который будет запакован;
- level - уровень компрессии Zlib (Z_NO_COMPRESSION, Z_BEST_SPEED, Z_BEST_COMPRESSION, Z_DEFAULT_COMPRESSION); опциональный.

Возвращаемое значение: ID нового контейнера, который является запакованной копией source; или -1 в случае ошибки.

zlib_unpack

Распаковать контейнер библиотекой Zlib.

Параметры (source)

- source - контейнер, который будет распакован.

Возвращаемое значение: ID нового контейнера, который является распакованной копией source; или -1 в случае ошибки.

Текстовые строки

num_to_str

Альтернативные имена: num2str.

Конвертировать число из переменной в текстовую строку (в контейнер).

Параметры (str, num)

- str - контейнер для строки;
- num - число.

Примеры

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

str_to_num

Альтернативные имена: str2num.

Конвертировать текста (из контейнера) в число.

Параметры (str)

- str - контейнер со строкой.

Возвращаемое значение: числовое значение.

Примеры

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

strcat

Добавляет строку source к строке destination. Обе строки должны заканчиваться символом с кодом 0, если кол-во остальных символов в строке меньше размера контейнера, в котором строка находится. После выполнения этой функции размер контейнера destination может увеличиться, если в нем не хватит места для строки source.

Параметры (destination, source)

Параметры (dest, dest_offset, source, source_offset)

strcmp

Сравнивает две строки str1 и str2. Обе строки должны заканчиваться символом с кодом 0, если кол-во остальных символов в строке меньше размера контейнера, в котором строка находится.

Параметры (str1, str2)

Параметры (str1, str1_offset, str2, str2_offset)

Возвращаемое значение:

- Меньше нуля - str1 меньше str2.
- Больше нуля - str1 больше str2.
- 0 - str1 равна str2.

strlen

Возвращает длину строки. Завершающий символ с кодом 0 не учитывается. Строки должна заканчиваться символом с кодом 0, если кол-во остальных символов в строке меньше размера контейнера, в котором строка находится.

Параметры (str)

Параметры (str, str_offset)

Возвращаемое значение: длина строки.

strstr

Ищет первое вхождение подстроки str2 в строке str1. Обе строки должны заканчиваться символом с кодом 0, если кол-во остальных символов в строке меньше размера контейнера, в котором строка находится.

Параметры (str1, str2)

Параметры (str1, str1_offset, str2, str2_offset)

Возвращаемое значение: смещение подстроки str2 в строке str1 или -1, если подстрока не найдена.

sprintf

Форматирует и запоминает наборы символов и значений в str. Каждый аргумент (если он есть), преобразуется и выводится согласно соответствующей спецификации формата в format. Подробное описание формата можно почитать здесь: http://ru.wikipedia.org/wiki/Printf или в любой документации на функцию sprintf() языка Си/Си++. Контейнер str расширяется при необходимости.

Параметры (str, format, ...)

Возвращаемое значение: количество символов, записанных в str или отрицательное значение в случае ошибки.

Примеры

```
sprintf( str, "Some text" ) //Записать текст в контейнер str
//В результате контейнер str будет содержать следующую строку: "Some text"
sprintf( str, "Number: %d", 12 ) //Записать знаковое десятичное число
//В результате контейнер str будет содержать следующую строку: "Number: 12"
```

printf

То же самое, что sprintf, только результат (набор символов и значений) записывается в поток STDOUT.

Параметры (format, …)

fprintf

Форматный вывод текста в указанный поток (открытый функцией fopen() или fopen_mem()).

Параметры (stream, format, …)

Работа с логом (журналом событий)

logf

Форматный вывод текста в буфер для логов (журнал событий).

Параметры (format, …)

get_log

Получить буфер с логом (журнал событий).

Возвращаемое значение: ID нового контейнера, в котором лежит свежий лог; (контейнер нужно удалять вручную при помощи remove()).

get_system_log

Получить буфер с системным логом (журнал событий). Системный лог содержит сообщения от всех виртуальных машин Pixilang и различных системных функций (глобальная инициализация Pixilang).

Возвращаемое значение: ID нового контейнера, в котором лежит свежий лог; (контейнер нужно удалять вручную при помощи remove()).

Работа с файлами

load

Загрузить контейнер из файла.

Параметры (filename, options)

Возвращаемое значение: ID загруженного контейнера или -1 в случае ошибки.

Примеры

```
//Грузим файл:
c = load( "smile.jpg" )
if c >= 0
{
  //Получаем формат загруженного файла:
  file_format = c.file_format
  //Возможные значения переменной file_format:
  //  FORMAT_RAW;
  //  FORMAT_JPEG;
  //  FORMAT_PNG;
  //  FORMAT_GIF;
  //  FORMAT_WAVE;
  //  FORMAT_AIFF;
  //  FORMAT_PIXICONTAINER;
  //  или еще какое-то из раздела "Типы файлов".
}
```

fload

Загрузить контейнер из потока данных, открытого функцией fopen() или fopen_mem().

Параметры (stream, options)

Возвращаемое значение: ID загруженного контейнера или -1 в случае ошибки.

save

Сохранить контейнер в указанном формате.

Параметры (pixi, filename, format, options)

- pixi;
- filename;
- format (FORMAT_RAW, FORMAT_JPEG, FORMAT_PNG, FORMAT_GIF, FORMAT_WAVE, FORMAT_PIXICONTAINER);
- options - опции (необязательный параметр):
  - для JPEG: качество сжатия от 0 (наихудшее) до 100 (наилучшее);
  - для GIF: сочетание флагов GIF_GRAYSCALE, GIF_DITHER.

Возвращаемое значение: 0 в случае успешного сохранения.

Примеры

```
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

Сохранить контейнер в поток данных, открытый функцией fopen() или fopen_mem().

Параметры (pixi, stream, format, options)

Возвращаемое значение: 0 в случае успешного сохранения.

get_real_path

Преобразовать путь в стиле Pixilang (например, 1:/img.png) в стиль реальной файловой системы (например, C:/Documents and Settings/username/Application Data/img.png).

Параметры (path)

Возвращаемое значение: ID контейнера с новым именем файла; (контейнер нужно удалять вручную при помощи remove()).

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

Функции для получения списка файлов.

Примеры

```
path = CURRENT_PATH //Директория, в которой мы будем сканировать файлы.
```

```
mask = -1 //Примеры маски: "txt/doc", "avi"; или -1 для получения всех
файлов.
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 - файл; 1 - директория;
    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 //Переходим к следующему файлу
    {
      //Файлов больше нет
      break
    }
  }
  remove_flist( fl )
}
```

get_file_size

Получить размер файла.

Параметры (filename)

remove_file

Параметры (filename)

rename_file

Параметры (old_filename, new_filename)

copy_file

Скопировать файл из source_filename в destination_filename.

Параметры (source_filename, destination_filename)

create_directory

Создать директорию.

Параметры (directory_name, mode)

- directory_name - имя создаваемой директории;
- mode - режим доступа (только для тех систем, где это поддерживается); опциональный.

set_disk0

Использовать поток данных _stream_ (открытый функцией fopen() или fopen_mem()) как виртуальный диск 0:/ . При этом _stream_ должен указывать на открытый незапакованный TAR архив. Файлы внутри TAR архива будут доступны для всех файловых функций через обращение к диску 0:/.

Параметры (stream)

- stream - ID потока данных или 0, если вы хотите выключить диск 0:/

get_disk0

fopen

Параметры (filename, mode)

Возвращаемое значение: ID потока данных, связанного с указанным файлом; или 0 в случае ошибки.

Примеры

```
f = fopen( "/tmp/data.txt", "rb" ) //Открываем файл data.txt для чтения
fclose( f ) //...и закрываем его.
```

fopen_mem

Открыть контейнер _data_ как файл.

Параметры (data)

Возвращаемое значение: ID потока данных, связанного с указанным контейнером; или 0 в случае ошибки.

fclose

Параметры (stream)

Примеры

```
f = fopen( "/tmp/data.txt", "rb" ) //Открываем файл data.txt для чтения.
c = fgetc( f ) //Получаем байт из этого файла.
fclose( f ) //Закрываем файл.
```

fputc

Параметры (c, stream)

Примеры

```
f = fopen( "/tmp/data.txt", "wb" ) //Открываем файл data.txt для записи.
fputc( 0x12, f ) //Записываем байт 0x12 в этот файл.
fclose( f ) //Закрываем файл.
```

fputs

Параметры (s, stream)

Примеры

```
f = fopen( "/tmp/data.txt", "wb" ) //Открываем файл data.txt для записи.
str = "Hello!"
fputc( str, f ) //Записываем строку "Hello!" в этот файл.
fclose( f ) //Закрываем файл.
```

fwrite

Параметры (data, size, stream, data_offset_optional)

Примеры

```
f = fopen( "/tmp/data.txt", "wb" ) //Открываем файл data.txt для записи.
str = "Hello!"
fwrite( str, 2, f ) //Записываем первые два байта из контейнера str в этот
файл.
fclose( f ) //Закрываем файл.
```

fgetc

Параметры (stream)

fgets

Загрузить строку текста из потока stream.

Параметры (s, n, stream, offset)

Возвращаемое значение: длина полученной строки.

Примеры

```
string = new( 256, 1, INT8 )
f = fopen( "/tmp/data.txt", "rb" ) //Открываем файл data.txt для чтения.
fgets( string, 256, f ) //Получаем строку текста из этого файла.
//Полученная строка помещается в указанный выше контейнер string.
//Если в контейнере недостаточно места, то строка обрезается.
fclose( f ) //Закрываем файл.
```

fread

Загрузить блок данных из потока stream.

Параметры (data, size, stream, data_offset_optional)

feof

Параметры (stream)

fflush

Параметры (stream)

fseek

Параметры (stream, offset, origin)

ftell

Параметры (stream)

Примеры

```
//Один из способов получения размера файла:
f = fopen( "/tmp/data.txt", "rb" )
fseek( f, 0, SEEK_END )
```

```
size_of_file = ftell( f )
fclose( f )
```

setxattr

Установить значение одного из дополнительных атрибутов файла.

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

Возвращаемое значение: 0 в случае успешного завершения или -1 в случае ошибки.

Графика

frame

Вывести содержимое рабочего экрана на дисплей и подождать указанное количество миллисекунд.

Параметры (delay, x, y, xsize, ysize)

- delay - длина паузы (в миллисекундах), которую нужно выждать после вывода на экран. По разным причинам пауза может оказаться длиннее, например, если система притормозила, или в системе неточный таймер;
- x, y, xsize, ysize - необязательные параметры, указывающие, какой регион текущего экрана надо вывести.

Возвращаемое значение:

- 0 - успешное завершение;
- -1 - контейнер рабочего экрана не найден;
- -2 - тайм-аут (возможно, графический движок временно не работает).

vsync

Включить/выключить вертикальную синхронизацию (синхронизация кадровой частоты с частотой вертикальной развёртки монитора). vsync(1) - включить. vsync(0) - выключить.

Параметры (enable)

set_pixel_size

Изменить размер пикселей на экране. Размер по умолчанию (минимальный) = 1. Увеличение размера ведет к уменьшению разрешения экрана.

Параметры (size)

get_pixel_size

Получить размер экранного пикселя.

set_screen

Сделать указанный контейнер текущим рабочим экраном. ID контейнера с экраном по умолчанию - 0.

Параметры (pixi)

- pixi - ID контейнера.

get_screen

Получить ID контейнера, который в данный момент является рабочим экраном.

Возвращаемое значение: ID контейнера, который является рабочим экраном.

set_zbuf

Параметры (zbuf_container)

Указать контейнер, который будет буфером глубины (Z-Buffer) при рисовании трехмерных объектов. Подробнее про Z-буферизацию можно почитать здесь.

Контейнер должен иметь тип INT32 и по размерам совпадать с размером текущего экрана.

get_zbuf

clear_zbuf

get_color

Получить значение цвета с заданными характеристиками r,g,b (красный,зеленый,синий).

Параметры (red, green, blue)

- red - интенсивность красного (от 0 до 255);
- green - интенсивность зеленого (от 0 до 255);
- blue - интенсивность синего (от 0 до 255);

Возвращаемое значение: значение цвета; формат этого значения может меняться в зависимости от версии Pixilang; например, если Pixilang скомпилирован для устройств с 8-битным дисплеем, то значение цвета будет в диапазоне от 0 до 255.

get_red

Получить интенсивность красной составляющей в указанном цвете.

Параметры (color)

- color - цвет.

Возвращаемое значение: интенсивность красной составляющей; от 0 до 255.

get_green

Получить интенсивность зеленой составляющей в указанном цвете.

Параметры (color)

- color - цвет.

Возвращаемое значение: интенсивность зеленой составляющей; от 0 до 255.

get_blue

Получить интенсивность синей составляющей в указанном цвете.

Параметры (color)

- color - цвет.

Возвращаемое значение: интенсивность синей составляющей; от 0 до 255.

get_blend

Получить промежуточное значение цвета между двумя известными.

Параметры (c1, c2, v)

- c1 - первый цвет;
- c2 - второй цвет;
- v - положение между c1 и c2; 0 - ближе всего к c1; 255 - ближе всего к c2.

Возвращаемое значение: промежуточное значение цвета.

transp

Установить прозрачность для всех последующих функций.

Параметры (t)

- t - значение прозрачности от 0 (невидимый) до 255 (непрозрачный).

get_transp

Получить текущее значение прозрачности.

clear

Очистить текущий рабочий экран заданным цветом (или черным, если цвет не задан).

Параметры (color)

- color - цвет.

dot

Нарисовать точку.

Параметры (x, y, color)

- x - координата X;
- y - координата Y;
- color - цвет.

dot3d

Нарисовать точку в 3D.

Параметры (x, y, z, color)

get_dot

Получить цвет в указанной точке.

Параметры (x, y)

- x - координата X;
- y - координата Y.

Возвращаемое значение: значение цвета в указанной точке.

get_dot3d

Получить цвет в указанной точке.

Параметры (x, y, z)

Возвращаемое значение: значение цвета в указанной точке.

line

Нарисовать линию.

Параметры (x1, y1, x2, y2, color)

line3d

Нарисовать линию в 3D.

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

box

Нарисовать прямоугольник.

Параметры (x, y, xsize, ysize, color)

fbox

Нарисовать закрашенный прямоугольник.

Параметры (x, y, xsize, ysize, color)

pixi

Вывести на экран контейнер с картинкой.

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

- pixi_cont - ID контейнера с картинкой;
- x;
- y;
- color - цвет фильтра; опциональный; значение по умолчанию - WHITE (пропускаются все цвета);
- xscale - коэффициент масштабирования по оси X; опциональный; значение по умолчанию - 1;
- yscale - коэффициент масштабирования по оси Y; опциональный; значение по умолчанию - 1;
- src_x - смещение по оси X внутри контейнера pixi_cont; по умолчанию - 0;
- src_y - смещение по оси Y внутри контейнера pixi_cont; по умолчанию - 0;
- src_xsize - ширина области (внутри контейнера), которую нужно вывести на экран; по умолчанию равна ширине контейнера;

- src_ysize - высота области (внутри контейнера), которую нужно вывести на экран; по умолчанию равна высоте контейнера.

Примеры

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

triangles3d

Нарисовать массив треугольников.

Параметры (vertices, triangles, tnum)

- vertices - контейнер вершин; ширина контейнера = 8; высота контейнера = количество вершин; формат одной вершины: X, Y, Z, TextureX (от 0 до ширины текстуры), TextureY (от 0 до высоты текстуры), Unused, Unused, Unused;
- triangles - контейнер треугольников; ширина контейнера = 8; высота контейнера = количество треугольников; формат треугольника: NumberOfVertex1, NumberOfVertex2, NumberOfVertex3, Color, Texture (или -1), Opacity (0..255), Unused, Order (треугольники с малыми значениями Order будут рисоваться раньше остальных);
- tnum - кол-во треугольников; опциональный.

sort_triangles3d

Отсортировать массив треугольников по глубине так, чтобы при вызове triangles3d() сначала рисовались дальние треугольники, а потом ближние.

Параметры (vertices, triangles, tnum)

- vertices - контейнер вершин; формат такой же, как в triangles3d();
- triangles - контейнер треугольников; формат такой же, как в triangles3d();
- tnum - кол-во треугольников; опциональный.

set_key_color

Установить/сбросить цвет прозрачности у контейнера.

Параметры (pixi, color)

- pixi - ID контейнера;
- color - цвет, который будет прозрачным; если цвет не указать, прозрачность для контейнера pixi выключится.

get_key_color

set_alpha

Привязать к контейнеру другой контейнер с альфа-каналом. Альфа-канал должен иметь тип INT8.

Параметры (pixi, alpha)

- pixi - ID контейнера;
- alpha - ID контейнера с альфа-каналом; если этот параметр не указать, то альфа-канал выключится для контейнера pixi.

get_alpha

Получить ID контейнера с альфа-каналом, привязанного к указанному контейнеру.

Параметры (pixi)

Возвращаемое значение: ID контейнера или -1 (если альфа-канал отсутствует).

print

Вывести текст на экран.

Параметры (text, x, y, color, align, max_xsize)

- text - ID контейнера с текстом;
- x, y - координаты точки, относительно которой происходит выравнивание;
- color - цвет;
- align - выравнивание; опциональный;
- max_xsize - максимальный размер (кол-во пикселей) текста по горизонтали; опциональный.

Примеры

```
print( "Hello Pixi!", 0, 0 ) //цвет - белый; выравнивание - по центру;
print( "line1\nline2", 50, 50, RED ) //выравнивание - по центру;
print( "line1\nline2", -50, 50, RED, TOP | LEFT ) //выравнивание - по
верхнему левому краю;
```

get_text_xsize

Параметр (text, align, max_xsize)

- text - ID контейнера с текстом;
- align - выравнивание; опциональный;
- max_xsize - максимальный размер (кол-во пикселей) текста по горизонтали; опциональный.

Возвращаемое значение: ширина текста в пикселях.

get_text_ysize

Параметр (text, align, max_xsize)

- text - ID контейнера с текстом;
- align - выравнивание; опциональный;
- max_xsize - максимальный размер (кол-во пикселей) текста по горизонтали; опциональный.

Возвращаемое значение: высота текста в пикселях.

set_font

Параметры (first_char_utf32, font_image, xchars, ychars)

get_font

Параметры (char_utf32)

Возвращаемое значение: ID контейнера, в котором находится картинка шрифта для указанного символа.

effector

Наложить эффект на выделенный участок экрана. На координаты этой функции не действует трансформация.

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

color_gradient

Нарисовать цветной градиент в указанном прямоугольнике. На координаты этой функции не действует трансформация.

Параметры (color1, transp1, color2, transp2, color3, transp3, color4, transp4, x, y, xsize, ysize, x_step, y_step)

split_rgb

Разбить картинку по каналам (красный, зеленый, синий) или наоборот собрать. Канал в данном случае - это контейнер любого типа, в который будут записываться (или считываться) значения яркости красной, зеленой или синей составляющей изображения.

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

- direction: 0 - ковертация из image в RGB; 1 - конвертация из RGB в image;
- image - контейнер с картинкой (тип PIXEL);
- red_channel - контейнер со значениями красного; опциональный; может быть -1, если нужно игнорировать этот канал;
- green_channel - контейнер со значениями зеленого; опциональный; может быть -1, если нужно игнорировать этот канал;
- blue_channel - контейнер со значениями синего; опциональный; может быть -1, если нужно игнорировать этот канал;
- image_offset - смещение в контейнере image; опциональный;
- channel_offset - смещение в контейнерах с каналами; опциональный;
- size - количество пикселей; опциональный.

Примеры

```
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 ) //Разбиваем картинку img на составляющие r, g, b
//Получаем яркость красного (от 0 до 255) для первого пикселя изображения:
value = r[ 0 ]
```

split_ycbcr

Аналогична split_rgb(), только для преобразования в/из формата YCbCr.

OpenGL основа

Версия Pixilang с поддержкой OpenGL основана на стандартах OpenGL ES 2.0 и OpenGL ES Shading Language 1.0 (GLSL ES).
Краткое изложение стандарта на нескольких страницах: OpenGL ES 2.0 Quick Reference Card.

set_gl_callback

Установить gl_callback - функцию, которая будет отвечать за OpenGL-отрисовку кадра.

Параметры (gl_callback, user_data)

- gl_callback - функция с параметрами ($user_data); почти все графические функции внутри gl_callback() будут выполняться через OpenGL, в обход стандартного пиксельного движка Pixilang;

- user_data - какие-то пользовательские данные, которые будут переданы функции gl_callback() во время перерисовки кадра.

Примеры

```
fn gl_callback( $user_data )
{
  set_screen( GL_SCREEN ) //Включить режим рисования в буфер OpenGL
  clear( YELLOW )
  set_screen( 0 ) //Возвращаемся в обычный режим рисования
}

set_gl_callback(
  gl_callback,
  0 )

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

set_gl_callback( -1 ) //Удалить gl_callback
```

remove_gl_data

Удалить из контейнера все OpenGL данные, которые были созданы автоматически во время перерисовки OpenGL кадра внутри gl_callback().

Параметры (container)

gl_draw_arrays

Гибрид OpenGL функций glColor4ub(), glBindTexture(), glVertexPointer(), glColorPointer(), glTexCoordPointer(), glDrawArrays().
Можно вызывать только внутри кода, заданного set_gl_callback().

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

- mode - режим рисования:
  - GL_POINTS;
  - GL_LINE_STRIP;
  - GL_LINE_LOOP;
  - GL_LINES;
  - GL_TRIANGLE_STRIP;
  - GL_TRIANGLE_FAN;
  - GL_TRIANGLES;
- first - номер первой вершины в указанных массивах;

- count - количество вершин;
- color_r, color_g, color_b, color_a - цвет RGBA (только, если не задан color_array);
- texture - контейнер с текстурой или -1;
- vertex_array - контейнер типа INT8, INT16 или FLOAT32 с вершинами;
- color_array - контейнер типа INT8 или FLOAT32 с RGBA цветами вершин или -1; опциональный;
- texcoord_array - контейнер типа INT8, INT16 или FLOAT32 с текстурными координатами; опциональный.

gl_blend_func

Полный аналог OpenGL функции glBlendFunc() или glBlendFuncSeparate() (если заданы sfactor_alpha и dfactor_alpha).
Можно вызывать только внутри кода, заданного set_gl_callback().

Вызывайте эту функцию без параметров, если нужно сбросить все значения в исходное состояние.

Параметры (sfactor, dfactor, sfactor_alpha, dfactor_alpha)

- sfactor;
- dfactor;
- sfactor_alpha; опциональный;
- dfactor_alpha; опциональный.

gl_bind_framebuffer

Превратить указанный контейнер cnum в OpenGL framebuffer (с прикрепленной текстурой) и сделать его текущим - то есть, все последующие операции рисования отправлять не на экран, а в этот framebuffer. Для отключения и перехода обратно на основной экран - вызовите эту функцию без параметров.
Можно вызывать только внутри кода, заданного set_gl_callback().

Параметры (cnum)

gl_bind_texture

Привязать указанный контейнер cnum к текстурному блоку texture_unit.
Можно вызывать только внутри кода, заданного set_gl_callback().

Параметры (cnum, texture_unit)

- cnum - номер контейнера;
- texture_unit - номер текстурного блока: 1, 2, 3, ...; блок с номером 0 является основным и всегда используется при отрисовке пиксельных контейнеров.

Пример

```
fn gl_callback( $userdata )
```

```
{
    set_screen( GL_SCREEN )
    gl_bind_texture( some_image2, 1 ) //привязать контейнер some_image2 к
текстурному блоку 1
    gl_bind_texture( some_image3, 2 ) //привязать контейнер some_image3 к
текстурному блоку 2
    gl_use_prog( gl_prog ) //Use user-defined GLSL program
    gl_uniform( gl_prog.g_texture2, 1 ) //установка переменной шейдера:
g_texture2 = текстурный блок 1
    gl_uniform( gl_prog.g_texture3, 2 ) //установка переменной шейдера:
g_texture3 = текстурный блок 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; //главный текстурный блок 0 (задается из
pixi())
uniform sampler2D g_texture2; //текстурный блок 1
uniform sampler2D g_texture3; //текстурный блок 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

Получить значение параметра состояния в целочисленном формате. Полный аналог OpenGL функции glGetIntegerv().
Можно вызывать только внутри кода, заданного set_gl_callback().

Параметры (pname)

gl_get_float

Получить значение параметра состояния в формате числа с плавающей точкой. Полный аналог OpenGL функции glGetFloatv().
Можно вызывать только внутри кода, заданного set_gl_callback().

Параметры (pname)

OpenGL шейдеры

Версия Pixilang с поддержкой OpenGL может использовать вершинные (vertex) и фрагментные (fragment) шейдеры, которые описываются специальным языком (основанном на ANSI C) OpenGL ES Shading Language 1.0 (GLSL ES).
Вершинный шейдер обрабатывает параметры вершин многогранников (из которых построены все объекты в OpenGL): координаты, текстурные координаты, цвет и т.д.
Фрагментный шейдер обрабатывает цвет каждого пикселя во время рисования многогранника.

В вершинных шейдерах используйте следующие правила (для получения максимальной кросс-платформенности в Pixilang):

- пишите **IN** вместо квалификаторов **attribute** и **in**;
- пишите **OUT** вместо квалификаторов **varying** и **out**;
- пишите **LOWP**, **MEDIUMP** и **HIGHP** вместо квалификаторов **lowp**, **mediump** и **highp**;
- пишите **PRECISION(P, T)** вместо **precision P, T**.

В фрагментных шейдерах используйте следующие правила

- пишите **IN** вместо квалификаторов **varying** и **in**;
- пишите **LOWP**, **MEDIUMP** и **HIGHP** вместо квалификаторов **lowp**, **mediump** и **highp**;
- пишите **PRECISION(P, T)** вместо **precision P, T**.

gl_new_prog

Создать новую GLSL программу - контейнер, совмещающий вершинный и фрагментный шейдеры. Эту функцию можно вызывать в любом месте Pixilang-кода. Непосредственная сборка программы (компиляция шейдеров и их объединение) будет происходить позже, когда вы передадите этот контейнер в функцию gl_use_prog().

Параметры (vertex_shader, fragment_shader)

- vertex_shader - контейнер с исходным кодом вершинного шейдера или одна из констант GL_SHADER_xxx (встроенные системные шейдеры);
- fragment_shader - контейнер с исходным кодом фрагментного шейдера или одна из констант GL_SHADER_xxx (встроенные системные шейдеры).

Возвращаемое значение: ID контейнера с GLSL программой или отрицательное значение в случае ошибки; (контейнер нужно удалять вручную при помощи remove()).

gl_use_prog

Использовать GLSL программу, созданную ранее при помощи gl_new_prog(). Если программа используется впервые, то будет произведена компиляция и объединение шейдеров, находящихся внутри этой программы.
Можно вызывать только внутри кода, заданного set_gl_callback().

Параметры (prog)

gl_uniform

Pixilang может передавать данные в GLSL переменные с квалификатором uniform. Uniform-переменные являются глобальными и могут использоваться как в вершинных, так и во фрагментных шейдерах. Функции типа gl_uniform() используются для передачи значений в эти переменные.
Можно вызывать только внутри кода, заданного set_gl_callback().

Через gl_uniform() можно также менять содержимое массивов в шейдерах, если использовать эту функцию следующим образом: gl_uniform(var_location, src_container, vector_size, first_vector, count), где count - это количество векторов, которые будут записаны в массив из контейнера src_container.

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

- var_location - ID переменной внутри текущей GLSL программы; получить ID нужной переменной можно, используя запись такого формата: ПРОГРАММА.ИМЯ_ПЕРЕМЕННОЙ;
- v0, v1, v2, v3 - значения, которые будут сохраняться в переменную; v1, v2 и v3 - опциональные; количество значений зависит от размерности переменной (например, для трехмерного вектора нужно использовать v0, v1 и v2).

Примеры

```
gl_use_prog( gl_prog ) //Используем GLSL программу gl_prog
gl_uniform( gl_prog.g_time, get_timer( 0 ) ) //Задаем значение uniform-переменной g_time
```

gl_uniform_matrix

Функция аналогичная gl_uniform(), но для работы с матрицами.
Можно вызывать только внутри кода, заданного set_gl_callback().

Параметры (size, matrix_location, transpose, matrix)

- size - размерность матрицы: 2 = 2×2; 3 = 3×3; 4 = 4×4;
- var_location - ID матрицы внутри текущей GLSL программы; получить ID нужной матрицы можно, используя запись такого формата: ПРОГРАММА.ИМЯ_МАТРИЦЫ;
- transpose - транспонировать матрицу (заменить строки на столбцы): 0 - нет; 1 - да;
- matrix - ID контейнера с матрицей.

Примеры

```
gl_use_prog( gl_prog ) //Использовать GLSL программу gl_prog
gl_uniform( 4, gl_prog.g_mat, 0, source_matrix ) //Сохранить данные из контейнера source_matrix в матрицу g_mat
```

Анимация контейнеров

pack_frame

Запаковать текущее содержимое контейнера в кадр. Номер кадра должен быть в свойстве “frame” этого контейнера.

Параметры (pixi)

unpack_frame

Распаковать кадр в текущее содержимое контейнера. Номер кадра должен быть в свойстве “frame” этого контейнера.

Параметры (pixi)

create_anim

Создать анимацию - скрытую область контейнера, в которой будут располагаться запакованные кадры.

Параметры (pixi)

remove_anim

Удалить анимацию в указанном контейнере.

Параметры (pixi)

clone_frame

Продублировать текущий кадр в указанном контейнере. Номер кадра должен быть в свойстве “frame” этого контейнера.

Параметры (pixi)

remove_frame

Удалить текущий кадр в указанном контейнере. Номер кадра должен быть в свойстве “frame” этого контейнера.

Параметры (pixi)

play

Включить режим авто-проигрывания для указанного контейнера. В этом режиме нужные кадры будут распаковываться автоматически во время вызова функции pixi().

Параметры (pixi)

stop

Выключить режим авто-проигрывания для указанного контейнера.

Параметры (pixi)

Трансформация

Трансформация системы координат.

t_reset

Сброс (очистка текущей матрицы трансформации).

t_rotate

Параметры (angle, x, y, z)

- angle - угол поворота в градусах;
- x, y, z - координаты вектора, задающего ось поворота.

t_translate

Параметры (x, y, z)

t_scale

Параметры (x, y, z)

t_push_matrix

Сохранить текущую матрицу трансформации в стек.

t_pop_matrix

Загрузить последнюю сохраненную матрицу трансформации из стека.

t_get_matrix

Получить текущую матрицу трансформации (4×4 FLOAT).

Параметры (matrix_container)

t_set_matrix

Установить текущую матрицу трансформации (4×4 FLOAT).

Параметры (matrix_container)

t_mul_matrix

Умножить текущую матрицу трансформации на matrix_container.

Параметры (matrix_container)

t_point

Пересчитать координаты точки с учетом текущей матрицы трансформации.

Параметры (point_coordinates)

- point_coordinates - контейнер с тремя элементами типа FLOAT.

Звук

set_audio_callback

Задать функцию, которая будет генерировать звуковой поток.

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

- callback - адрес функции;
- userdata - данные для функции;
- sample_rate - частота дискретизации; если 0, то будет использоваться частота, заданная в глобальных настройках;
- format - формат сэмпла;
- channels - кол-во каналов;

- flags - флаги AUDIO_FLAG_xxx.

Примеры

```
fn audio_callback( $stream, $userdata, $channels, $frames, $time )
{
  generator( OP_SIN, $channels[ 0 ], 0, 32767 / 2, 0.1, 0 ) //Левый канал
  generator( OP_SIN, $channels[ 1 ], 0, 32767 / 2, 0.1, 0 ) //Правый канал
  ret(1)
}
//Запустить звук:
set_audio_callback( audio_callback, 0, 22050, INT16, 2, AUDIO_FLAG_INTERP2 )
```

```
//Остановить звук:
set_audio_callback( -1 )
```

get_audio_sample_rate

Получить локальную или глобальную частоту дискретизации.

Параметры (source)

- source: 0 - локальная частота дискретизации, которая была установлена через set_audio_callback(); 1 - глобальная частота дискретизации из настроек Pixilang, может не совпадать с локальной, не меняется в процессе выполнения Pixilang-программ.

Возвращаемое значение: частота дискретизации в Гц.

enable_audio_input

Параметры (disable_enable)

get_note_freq

Параметры (note, finetune)

- note - номер ноты; 0 = C-0; 1 = C#0; 2 = D-0 ...
- finetune - подстройка от -64 (предыдущая нота) до 64 (следующая нота).

Возвращаемое значение: частота ноты в Герцах.

MIDI

midi_open_client

Параметры (client_name)

Возвращаемое значение: ID клиента.

midi_close_client

Параметры (client_id)

midi_get_device

Параметры (client_id, device_num, flags)

Возвращаемое значение: имя устройства под номером device_num или -1, если нет такого устройства.

midi_open_port

Параметры (client_id, port_name, device_name, flags)

Возвращаемое значение: номер открытого порта (port ID).

midi_reopen_port

Параметры (client_id, port_id)

midi_close_port

Параметры (client_id, port_id)

midi_get_event

Параметры (client_id, port_id, data_cont)

Возвращаемое значение: размер (в байтах) текущего MIDI события.

midi_get_event_time

Параметры (client_id, port_id)

Возвращаемое значение: время (в системных тиках) текущего MIDI события

midi_next_event

Перейти на следующее сообщения в очереди.

Параметры (client_id, port_id)

midi_send_event

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

Время

start_timer

Запустить таймер.

Параметры (timer_num)

- timer_num - номер таймера.

get_timer

Получить значение таймера в миллисекундах.

Параметры (timer_num)

- timer_num - номер таймера.

Возвращаемое значение: 32-битное значение таймера в миллисекундах.

get_year

get_month

get_day

get_hours

get_minutes

get_seconds

get_ticks

Получить значение системного 32-битного таймера высокого разрешения. Единица измерения - 1 тик.

get_tps

Получить количество системных тиков в секунду.

sleep

Заснуть на указанный промежуток времени.

Параметры (delay)

- delay - длина задержки в миллисекундах.

События

get_event

Получить очередное событие от системы.

Возвращаемое значение: 0 - нет новых событий; 1 - очередное событие получено, и оно находится в контейнере EVT.

set_quit_action

Установить поведение программы при получении события EVT_QUIT.

Параметры (action)

- action - номер действия, которое нужно выполнять при получении события EVT_QUIT.

Возможные значения параметра action:

- QA_NONE - ничего не делать;
- QA_CLOSE_VM (по умолчанию) - закрыть текущую виртуальную машину, но не выходить из Pixilang.

Многопоточность

thread_create

Создать новый поток выполнения, в котором сразу после создания будет запущена функция thread_function($thread_id, $user_data)

Параметры (thread_function, user_data, flags_optional)

Возвращаемое значение: ID потока или -1 в случае ошибки.

Примеры

```
fn thread_body( $thread_id, $user_data )
{
  printf( "Thread code\n" )
}
thread_id = thread_create( thread_body, 0 ) //Создаем новый поток
err = thread_destroy( thread_id, 1000 ) //Ждем до 1000 миллисекунд, пока
поток завершится
if err == 0 { printf( "Поток успешно завершен и закрыт\n" ) }
if err == 1 { printf( "Тайм-аут. Стоит попробовать еще раз\n" ) }
if err == 2 { printf( "Возникла ошибка в функции thread_destroy()\n" ) }
```

thread_destroy

Закрыть поток выполнения.

Параметры (thread_id, timeout_ms)

- thread_id;
- timeout_ms - тайм-аут в миллисекундах; отрицательные значения - не пытаться закрыть поток (небезопасно) в случае тайм-аута; INT_MAX - бесконечное ожидание.

Возвращаемое значение:

- 0 - поток успешно закрыт;
- 1 - тайм-аут;
- 2 - какая-то ошибка.

mutex_create

Примеры

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

```
mutex_destroy( new_mutex )
```

mutex_destroy

mutex_lock

mutex_trylock

mutex_unlock

Математика

acos(x); acosh(x); asin(x); asinh(x); atan(x); atanh(x); ceil(x); cos(x); cosh(x); exp(x); exp2(x); expm1(x); abs(x); floor(x); mod(x, y); log(x); log2(x); log10(x); pow(base, exp); sin(x); sinh(x); sqrt(x); tan(x); tanh(x);

rand() - получить случайное число в диапазоне от 0 до 32767; rand_seed(seed) - установить позицию (состояние) генератора случайных чисел.

Обработка данных

Примечание: функции обработки данных не работают с контейнерами динамического типа.

op_cn

Выполнить операцию обработки данных. Операнды: - контейнер C1; - числовое значение N.

Для каждого элемента контейнера C1 будет выполняться следующее:

```
C1[ i ] = C1[ i ] OP N,
где i - номер элемента; OP - выбранная операция.
```

Параметры (opcode, C1, N) - для всего контейнера C1

Параметры (opcode, C1, N, x, xsize) - область задана в 1D координатах

Параметры (opcode, C1, N, x, y, xsize, ysize) - область задана в 2D координатах

* opcode - код операции; * C1 - контейнер, над которым будет производиться выбранная операция; результат будет помещен в него же; * N - числовое значение; * x,y,xsize,ysize - регион, в котором будет выполняться операция.

Примеры

```
//Прибавить число 32 к каждому элементу контейнера img:
op_cn( OP_ADD, img, 32 )

//Прибавить число 32 к элементам 128...256:
op_cn( OP_ADD, img, 32, 128, 128 )

//Прибавить число 32 к элементам в регионе (8,8,32,32):
op_cn( OP_ADD, img, 32, 8, 8, 32, 32 )
```

op_cc

Выполнить операцию обработки данных. Операнды: - контейнер C1; - контейнер C2.

Для каждого элемента контейнера C1 будет выполняться следующее:

```
C1[ i ] = C1[ i ] OP C2[ i ],
где i - номер элемента; OP - выбранная операция.
```

Параметры (opcode, C1, C2) - для всей области контейнера C1

Параметры (opcode, C1, C2, dest_x, src_x, xsize) - область задана в 1D координатах

Параметры (opcode, C1, C2, dest_x, dest_y, src_x, src_y, xsize, ysize) - область задана в 2D координатах

op_ccn

Выполнить операцию обработки данных. Операнды: - контейнер C1; - контейнер C2; - числовое значение N.

Для каждого элемента контейнера C1 будет выполняться следующее:

```
C1[ i ] = C1[ i ] OP C2[ i ] OP2 N,
где i - номер элемента; OP - выбранная операция; OP2 - дополнительная
операция.
```

Параметры (opcode, C1, C2, N) - для всей области контейнера C1

Параметры (opcode, C1, C2, N, dest_x, src_x, xsize) - область задана в 1D координатах

Параметры (opcode, C1, C2, N, dest_x, dest_y, src_x, src_y, xsize, ysize) - область задана в 2D координатах

generator

Генератор сигнала.

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

- opcode - код операции;
- pixi - ID контейнера;
- phase - начальная фаза;
- amplitude - амплитуда;
- delta_x - скорость изменения фазы по горизонтали;
- delta_y - скорость изменения фазы по вертикали;
- x,y,xsize,ysize - регион, в котором будет выполняться операция.

Примеры

```
//Сгенерировать синусоиду в контейнер img:
generator( OP_SIN, img, 0, 1, 0.1, 0.1 )

//Сгенерировать синусоиду по более быстрому, но менее точному алгоритму:
generator( OP_SIN8, img, 0, 1, 0.1, 0.1 )

//Сгенерировать синусоиду в элементы 8...128 контейнера img:
generator( OP_SIN, img, 0, 1, 0.1, 0.1, 8, 120 )

//Сгенерировать синусоиду в регион (8,8,32,32) контейнера img:
generator( OP_SIN, img, 0, 1, 0.1, 0.1, 8, 8, 32, 32 )
```

wavetable_generator

Очень быстрый многоканальный сэмплер (набор из неограниченного количества примитивных генераторов), где сэмпл имеет фиксированный размер (32768 отсчетов) и всегда зациклен (loop).

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

- dest - INT16 или FLOAT32 контейнер, в который будет происходить запись;
- dest_offset - смещение;
- dest_length - длина генерируемого участка;
- table - контейнер с базовой волной (сэмплом) (поддерживаются такие форматы: 32768 x INT16, 32768 x FLOAT32);
- amp - INT32 массив амплитуд (fixed point 16.16);
- amp_delta - INT32 массив значений delta для амплитуды (fixed point 16.16);
- pos - INT32 массив с текущими смещениями внутри table (fixed point 16.16);
- pos_delta - INT32 массив со скоростями проигрывания (fixed point 16.16);
- gen_offset - номер первого генератора;
- gen_step - играть каждый gen_step генератор;
- gen_count - общее количество генераторов, которые нужно проиграть.

Возвращаемое значение: 0 в случае успешного завершения.

sampler

Параметры (sample_info)

Примеры

```
sample_data = new( 256, 1, INT16 ) //создаем 16-битный сэмпл
sample_info = new( SMP_INFO_SIZE, 1, INT32 ) //создаем контейнер для
сэмплера
clean( sample_info )
sample_info[ SMP_DEST ] = buffer //Контейнер, в который будем записывать
sample_info[ SMP_DEST_OFF ] = 0 //Смещение в контейнере
sample_info[ SMP_DEST_LEN ] = 256 //Размер области, которую будем заполнять
sample_info[ SMP_SRC ] = sample_data
sample_info[ SMP_SRC_OFF_H ] = 0 //Смещение сэмпла (левая часть числа с
фиксированной точкой)
sample_info[ SMP_SRC_OFF_L ] = 0 //Смещение сэмпла (правая часть числа с
фиксированной точкой, от 0 до 65535)
sample_info[ SMP_SRC_SIZE ] = 0 //Размер сэмпла (0 - весь сэмпл целиком)
sample_info[ SMP_LOOP ] = 0 //Начало лупа (повторяемого участка сэмпла)
sample_info[ SMP_LOOP_LEN ] = 128 //Длина лупа (или 0, если луп отключен)
sample_info[ SMP_VOL1 ] = 0 //Начальная громкость
sample_info[ SMP_VOL2 ] = 32768 //Конечная громкость (32768 = 1.0)
sample_info[ SMP_DELTA ] = ( 1 << 16 ) //Дельта (скорость проигрывания);
fixed point (real_value * 65536)
sample_info[ SMP_FLAGS ] = SMP_FLAG_INTERP4 | SMP_FLAG_PINGPONG //Кубическая
сплайновая интерполяция и луп в режиме ping-pong
sampler( sample_info ) //Записываем сэмпл с выбранными параметрами в
контейнер buffer
```

envelope2p

Линейная интерполяция (по двум точкам) усиления и DC-смещения в выделенной области контейнера. Без ограничения (защиты от переполнения).

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

- data_cont - исходный контейнер с данными;
- v1 - начальное значение амплитуды (0 - отсутствие сигнала; 32768 - исходная амплитуда; 32768*2 - двукратное усиление);
- v2 - конечное значение амплитуды (0 - отсутствие сигнала; 32768 - исходная амплитуда; 32768*2 - двукратное усиление);
- offset - смещение в контейнере data_cont; опциональный; по умолчанию - 0;
- size - размер активной области контейнера; опциональный; по умолчанию - весь контейнер;
- dc_offset1 - начальное DC смещение; опциональный; по умолчанию - 0;
- dc_offset2 - конечное DC смещение; опциональный; по умолчанию - 0.

gradient

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

fft

Выполнить быстрое преобразование Фурье.

Параметры (inverse, im, re, size)

new_filter

Создать новый фильтр, в основе которого следующая функция:

```
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;
```

Параметры (flags_for_future_use)

Возвращаемое значение: ID нового контейнера, в котором находятся данные фильтра.

remove_filter

Параметры (filter)

reset_filter

Параметры (filter)

init_filter

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

- filter;
- a - контейнер с коэффициентами входного сигнала;
- b - контейнер с коэффициентами обратной связи; опциональный; может быть -1 (если фильтр нерекурсивный);
- rshift - битовый сдвиг вправо для операций с фиксированной точкой; опциональный;
- flags - флаги на будущее; опциональный.

Возвращаемое значение: 0 в случае успешной инициализации фильтра.

apply_filter

Применить фильтр.

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

- filter;
- output - контейнер с выходными данными;
- input - контейнер с входными данными;
- flags - флаги на будущее; опциональный;
- offset - смещение внутри контейнеров output и input; опциональный;
- size - размер обрабатываемого блока; опциональный; по умолчанию - весь контейнер.

Возвращаемое значение: 0 в случае успешного применения фильтра.

replace_values

Подстановка значений контейнера. Типы контейнеров dest и values должны быть одинаковыми. Для каждого элемента контейнера dest будет выполняться следующее действие: dest[i] = values[(unsigned)src[i]]

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

- dest - куда;
- src - откуда;
- values - подставляемые значения;
- dest_offset - смещение в контейнере dest; опциональный;
- src_offset - смещение в контейнере src; опциональный;
- size - количество элементов, которые нужно изменить; опциональный.

Примеры

```
//Преобразовать 8-битную картинку в формат пикселей текущего экрана:
//src - это, например, контейнер основного экрана;
//img8 - контейнер с 8-битной картинкой (256 цветов);
//palette - контейнер с палитрой из 256 цветов;
replace_values( scr, img8, palette )
```

copy_and_resize

Скопировать (и растянуть/сжать, если нужно) указанную область контейнера src в указанную область контейнера dest.

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

- dest - куда;
- src - откуда;
- flags - флаги RESIZE_xxx; опциональный; по умолчанию - RESIZE_INTERP1;
- dest_x, dest_y, dest_rect_xsize, dest_rect_ysize, src_x, src_y, src_rect_xsize, src_rect_ysize - опциональные параметры.

Диалоги

file_dialog

Открыть диалоговое окно выбора файла.

Параметры (dialog_name, mask, id, default_name)

- dialog_name - название окна;
- mask - типы файлов (например, "gif/jpg" для отображения .gif и .jpg файлов; или "" для отображения всех файлов);
- id - имя файла, в который будет сохраняться состояние текущего диалога (например, "myprogram_jpeg_files");
- default_name - имя файла по умолчанию; опциональный.

Возвращаемое значение: ID контейнера с именем выбранного файла или -1, если файл не выбран; (контейнер нужно удалять вручную при помощи remove()).

prefs_dialog

Открыть диалоговое окно с глобальными настройками Pixilang.

Сеть

open_url

Открыть указанный URL в браузере.

Параметры (url_string)

Нативный код

dlopen

Открыть динамически подключаемую библиотеку (например, .DLL для Windows или .SO для Linux).

Параметры (lib_file_name)

Возвращаемое значение: ID открытой библиотеки или -1 в случае ошибки.

Примеры

```
//Например, мы имеем какую-то C функцию int show_info( int x, int y, void*
```

```
data )
//в библиотеке mylib.dll.
//Вызовем ее из Pixilang:
dl = dlopen( "mylib.dll" ) //Открываем библиотеку
if dl >= 0
{
  f = dlsym( dl, "show_info", "iip" ) //Получаем ID функции show_info()
  // "iip" - int int pointer
  if f >= 0
  {
    retval = dlcall( dl, f, 1, 2, "blahblah" ) //вызываем функцию
show_info() с параметрами 1, 2, "blahblah"
  }
  dlclose( dl ) //Закрываем библиотеку
}
```

dlclose

Закрыть динамически подключаемую библиотеку.

Параметры (lib_id)

dlsym

Получить ID символа (функции или переменной) из динамической библиотеки.

Параметры (lib_id, symbol_name, format, calling_convention)

- lib_id;
- symbol_name - имя функции или переменной;
- format - формат функции; опциональный; это текстовая строка следующего вида: “R(P)”, где R - тип возвращаемого значения (один ascii символ), P - типы параметров (несколько ascii символов или их полное отсутствие); ниже приведен список ascii символов, из которых эта строка строится:
  - 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 - одна из констант CCONV_xxx; опциональный.

Возвращаемое значение: ID символа или -1 в случае ошибки.

dlcall

Вызвать функцию из динамической библиотеки.

Параметры (lib_id, symbol_id, optional_function_parameters)

Системные функции

system

Выполнить системную команду.

Параметры (command)

Примеры

```
//Удаляем файл в Unix-совместимой ОС:
system( "rm /tmp/data.txt" )
```

argc

Функция возвращает количество аргументов, переданных программе.

argv

Функция возвращает ID контейнера со строкой, в которой записан аргумент под номером n.

Параметры (n)

Примеры

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

exit

Выйти из Pixilang.

Параметры (exit_code)

Примеры

```
exit( 4 ) //Выйти с кодом 4
```