NGN_CANVAS.H

(Funciones de la capa de dibujado)

NGN_Canvas(
float position_x = 0.0f,
// Posicion X (0 por defecto)
float position_y = 0.0f,
// Posicion Y (0 por defecto)
int32_t _width = DEFAULT_VALUE,
// Ancho de la capa (Toda la pantalla por defecto)
int32_t _height = DEFAULT_VALUE
// Alto de la capa (Toda la pantalla por defecto)
);

Crea un nuevo canvas, usando los parámetros especificados.

```
NGN_Canvas* canvas = new NGN_Canvas(100, 50, 200, 64);
```

void Position(float position_x, float position_y);
void Position(Vector2 pos);

Posiciona el canvas en la coordenada dada.

```
canvas->Position(10, 20);
```

void Translate(float speed_x, float speed_y);
void Translate(Vector2 spd);

Mueve el canvas en la dirección y velocidades dadas.

```
canvas->Translate(0.0f, -2.0f);
```

void Size(float w, float h);

Cambia el tamaño del canvas.

```
canvas->Size(640, 480);
```

void Scale(float w, float h);
void Scale(float scale);

Escala el canvas según el factor dado. Según la sobrecarga usada, escalara los ejes en conjunto o por separado. La escala por defecto es 1.0f.

```
canvas->Scale(1.5f);
canvas->Scale(2.0f, 0.75f);
```

void Rotate(double degrees);

Rota el canvas cada frame el número de unidades dado, en grados.

```
canvas->Rotate(1.2f);
```

void SetCenter(float x, float y);

Específica, en coordenadas relativas y desde el centro real canvas, donde se ubicará el centro de rotación del mismo.

```
canvas->SetCenter(-10, -5);
```

Vector2 position

Posición del canvas en pantalla.

float width
float height

Tamaño del canvas.

bool visible

Indica si el canvas es o no visible.

int32_t alpha

Nivel de transparencia del canvas, entre 0 y 255.

double rotation

Rotación del canvas, en grados.

bool flip_h
bool flip_v

Volteado vertical y horizontal del canvas.

Nota:
Los cambios de tamaño o escala no afectan al tamaño original del contenedor, solo se cambia el tamaño del contenido al representarse en la pantalla.

(Funciones de dibujado)

void Cls(uint32_t color = 0x00000000);

Borra el contenido del canvas y si se especifica, lo rellena con el color dado.

```
textbox->Cls(0x0080FFFF);       // RRGGBBAA
```

void Point(uint32_t x, uint32_t y, uint32_t color);

Dibuja un punto de 1x1 pixels del color especificado en las coordenadas del canvas dadas.

```
canvas->Point(100, 50, 0x00FF00FF);
```

void Points(const std::vector<CanvasPoint> &points);

Dibuja todos los puntos contenidos en el vector dado en una sola llamada a la función.

```
std::vector<CanvasPoint> mypoints;
mypoints.resize(3);
mypoints[0].x = 100; mypoints[0].y = 100; mypoints[0].color = 0xFF0000FF;
mypoints[1].x = 101; mypoints[1].y = 101; mypoints[1].color = 0x00FF00FF;
mypoints[2].x = 102; mypoints[2].y = 102; mypoints[2].color = 0x0000FFFF;
canvas->Points(mypoints);
mypoints.clear();
```

void Line(
    uint32_t x1, uint32_t y1,   // Punto A
    uint32_t x2, uint32_t y2,   // Punto B
    uint32_t color   // Color (RGBA)
);

Dibuja una línea entre dos puntos con el color especificado.

```
canvas->Line(10, 10, 200, 200, 0xFF0000FF);
```

void Lines(const std::vector<CanvasLine> &lines);

Dibuja todas las líneas contenidas en el vector dado en una sola llamada a la función.

```
std::vector<CanvasLine> mylines;
mylines.resize(2);
mypoints[0].x1 = 100; mypoints[0].y1 = 100;
mypoints[0].x2 = 200; mypoints[0].y2 = 200;
mypoints[0].color = 0xFFFFFFFF;
mypoints[1].x1 = 200; mypoints[1].y1 = 200;
mypoints[1].x2 = 300; mypoints[1].y2 = 300;
mypoints[1].color = 0xFFFFFFFF;
canvas->Lines(mylines);
mylines.clear();
```

```
void Box(
    uint32_t x1, uint32_t y1,    // Vértice superior izquierdo
    uint32_t x2, uint32_t y2,    // Vértice inferior derecho
    uint32_t color,              // Color (RGBA)
    bool paint = false           // Relleno?
);
```

Dibuja una caja entre los vértices especificados con el color dado. Puede dibujarse con o sin relleno.

```
canvas->Box(10, 10, 200, 200, 0xFF00FFFF, true);
canvas->Box(10, 10, 200, 200, 0xFFFFFFFF);
```

```
void Circle(
    uint32_t x, uint32_t y,           // Coordenadas del centro
    uint32_t r,                       // Radio horizontal
    uint32_t color,                   // Color (RGBA)
    uint32_t r2 = DEFAULT_VALUE,      // Radio vertical
    double in_angle = 0.0f,           // Ángulo inicial (RAD)
    double out_angle = 0.0f           // Ángulo final   (RAD)
);
```

Dibuja un circulo o arco con los parámetros especificados. Si no se especifica el radio vertical, se usara el horizontal en su lugar. De no especificar los ángulos iniciales y finales se dibujará un círculo completo.

```
canvas->Circle(320, 240, 32, 0xFFFFFFFF);
canvas->Circle(320, 240, 32, 0xFFFFFFFF, 32, 3.14f, 6.28f);
```

```
void FilledCircle(
    uint32_t x, uint32_t y,       // Coordenadas del centro
    uint32_t r,                   // Radio horizontal
    uint32_t color,               // Color (RGBA)
    uint32_t r2 = DEFAULT_VALUE   // Radio vertical
);
```

Dibuja un circulo relleno con los parámetros especificados. Si no se especifica el radio vertical, se usara el horizontal en su lugar.

```
canvas->FilledCircle(320, 240, 100, 0xFFFFFFFF);
```