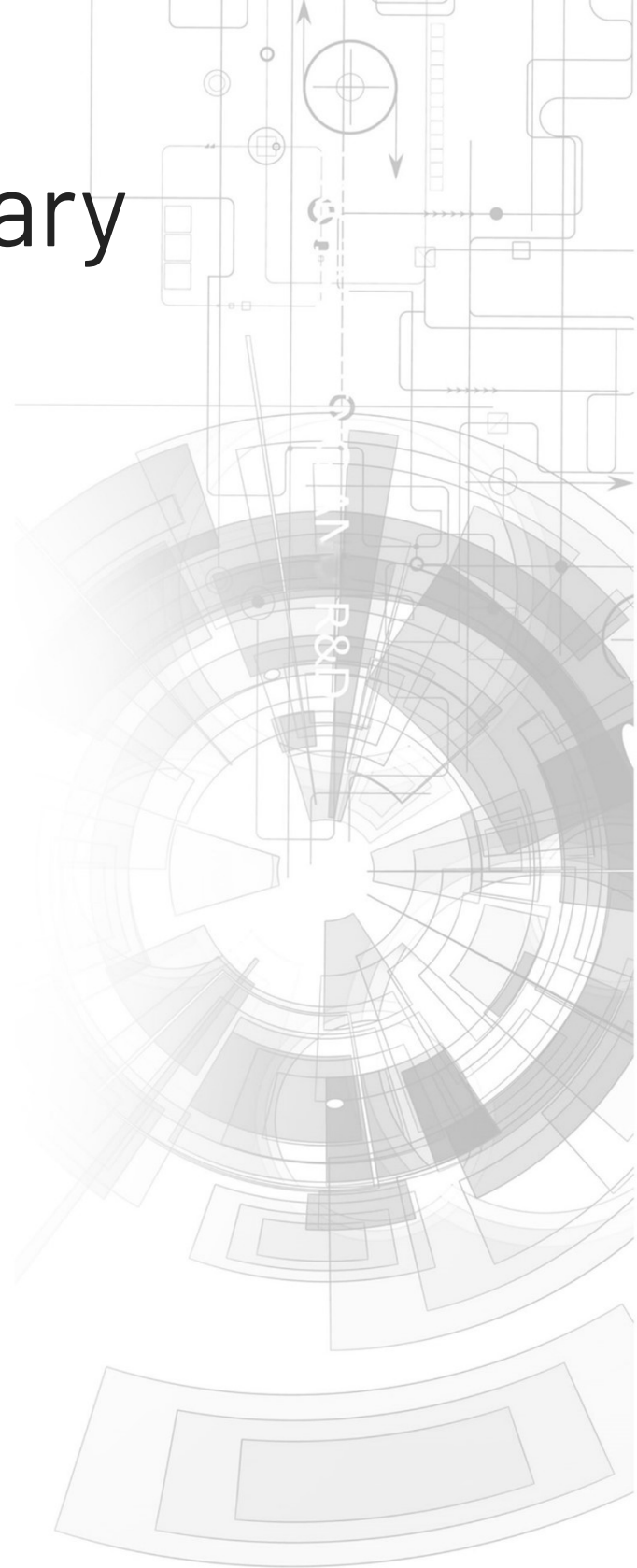


Graphics4D Library



Manual

Revision 1.2

Copyright © 2025 4D Systems

Content may change at any time. Please refer to the resource centre for latest documentation.

Contents

1. Introduction	5
2. Library Setup	5
3. Initializing the Library	6
4. Basic Graphics Functions	8
4.1. Initialize	8
4.2. GetFramebuffer	9
4.3. SetFramebuffer	10
4.4. SendFramebuffer	11
4.5. SetBacklightLevel	12
4.6. Contrast	13
4.7. GetWidth	14
4.8. GetHeight	15
4.9. ScreenMode	16
4.10. SetAddressWindow	17
4.11. BlendColor	18
4.12. Cls	19
4.13. RectangleFilled	20
4.14. RectangleFilled (with color array)	21
4.15. RectangleFilled (with buffer)	22
4.16. RectangleFilledWithAlpha	23
4.17. Hline	24
4.18. Vline	25
4.19. PutPixel	26
4.20. Line	27
4.21. Ellipse	28
4.22. EllipseFilled	29
4.23. Circle	30
4.24. Arc	31

4.25. ArcFilled	32
4.26. CircleFilled	33
4.27. Triangle	34
4.28. TriangleFilled	35
4.29. Polyline	36
4.30. Polygon	37
4.31. PolygonFilled	38
4.32. SetBackgroundColor	39
4.33. ClipWindow	40
4.34. SetFont	41
4.35. SetFontForeground (TextArea4D)	42
4.36. SetFontBackground (TextArea4D)	43
4.37. SetFontForeground	44
4.38. SetFontBackground	45
4.39. MoveTo	46
4.40. MoveRel	47
4.41. print	48
4.42. printf	49
4.43. CreateTextArea	50
4.44. print (TextArea4D)	51
4.45. printf (TextArea4D)	52
5. Image Control Functions	52
5.1. LoadImageControl (Pointer)	52
5.2. LoadImageControl (Filename)	53
5.3. GetCount	54
5.4. GetFile	55
5.5. GetInfo	56
5.6. SetProperties	57
5.7. SetValue	58
5.8. GetValue	59
5.9. SetPosition	60

5.10. GetFrames	61
5.11. Show	62
5.12. Clear	63
6. Touch Handling Functions	64
6.1. Initialize	64
6.2. Calibrate	65
6.3. GetPoints	66
6.4. GetStatus	67
6.5. GetID	68
6.6. GetX	69
6.7. GetY	70
7. Legal Notice	71
7.1. Proprietary Information	71
7.2. Disclaimer of Warranties & Limitations of Liabilities	71

1. Introduction

The Graphics4D Library is provided by 4D Systems for use with gen4-RP2350-XX product series. This library provides users access to the graphics and touch features of 4D Systems' RP2350 display modules.

Note however that some functionalities might not be supported by a certain product types, depending on its specifications. For instance, non-touch variants have no access to all touch-related functions. For more information on the specifications of a product, refer to its datasheet.

It is recommended to use Workshop5 IDE to get the most out of the library functions.

Note

Workshop5 is a **Windows-only** application.

2. Library Setup

When using Workshop5, the library is automatically included in the code.

```
#include "Graphics4D.h"  
#include "GeneratedConsts.h"
```

The library can be included to projects using the header file `Graphics4D.h`

Additionally, Workshop5 generates a project specific header file `GeneratedConsts.h` which includes constants and useful functions.

By including the Graphics4D library, the main user code gains access to `gfx`, `img` and `touch` functions.

3. Initializing the Library

Before using library functions to draw to the screen or detect touch for touch variants. The `gfx`, `img` and `touch` needs to be initialized.

For `gfx` and `touch`, this can be done by simply using the following functions as shown:

```
gfx.Initialize();  
touch.Initialize();
```

These two are responsible for basic graphical library features and touch handling.

For `img`, you'll need to prepare a handle and prepare the graphical resources available in flash memory or microSD card.

```
int main() {  
    gfx.Initialize();  
    touch.Initialize();  
    hndl = img.LoadImageControl(GRAPHICS_GCX);  
}
```

Where `GRAPHICS_GCX` can be a string indicating the graphics file to load from microSD or a pointer to a flash memory region containing the graphical resources. Note that the library only supports the graphical resources generated by Workshop5 IDE based on the user interface.

Workshop5 automatically generates the graphics file depending on project contents and defines `GRAPHICS_GCX` in `GeneratedConsts.h`. The handle `hndl` is also declared in this file. Additionally, Workshop5 also generates a utility function `SetupMedia` that handles loading the graphical resources as well as providing useful parameters of the Workshop5 widgets added to the project.

A typical project will contain the code as shown:

```
#include "Graphics4D.h"
#include "GeneratedConsts.h"

int main() {

    // this is required for programming without manually entering bootloader
    stdio_init_all(); // must also enable usb stdio in CMakeLists
    // otherwise, manually put the board to bootloader mode

    gfx.Initialize();
    gfx.ScreenMode(PROJECT_ORIENTATION);
    gfx.Contrast(8);

    // touch.Initialize(); // uncomment when using touch

    // Uncomment when using widgets
    // SetupMedia(); // autogenerated helper function used to load graphical widgets

    // put your setup code here, to run once:

    while (true) {
        // put your main code here, to run repeatedly:

    }
}
```

4. Basic Graphics Functions

The following functions are included in `gfx`. These are basic draw functions including simple shapes.

4.1. Initialize

Initializes the graphics library and prepares the display for drawing.

Syntax: `gfx.Initialize();`

Return: `bool` - Returns `true` if initialization is successful; otherwise, `false`.

Example

```
if (gfx.Initialize()) {  
    // Initialization successful  
}
```


4.2. GetFrameBuffer

Retrieves the pointer to the current RGB565 framebuffer.

Syntax: `gfx.GetFrameBuffer();`

Return: `uint16_t*` - Pointer to the current RGB565 framebuffer.



Example

```
uint16_t* buffer = gfx.GetFrameBuffer();  
// Gets a pointer to the framebuffer
```

4.3. SetFramebuffer

Changes the library's target framebuffer to the specified pointer. The framebuffer should be able to hold enough pixel data as the screen size.

Syntax: `gfx.SetFramebuffer(buffer);`

Argument	Type	Description
buffer	uint16_t*	Pointer to the RGB565 framebuffer memory

Return: None (void)

Example

2.4-inch to 3.5-inch

```
uint16_t * mainFb = gfx.GetFrameBuffer(); // Store the pointer to the current/main
framebuffer
uint16_t * newFb = rp_mem_malloc(LCD_WIDTH * LCD_HEIGHT * 2);
// do something in newFb directly...
gfx.SetFramebuffer(newFb); // Sets the new framebuffer for the library to use
// use gfx to draw using library functions...
gfx.SendFramebuffer(0, 0, LCD_WIDTH - 1, LCD_HEIGHT - 1); // Send contents of newFb
gfx.SetFramebuffer(mainFb); // revert back to mainFb
```

4.3-inch to 9.0-inch

```
uint16_t * mainFb = gfx.GetFrameBuffer(); // Store the pointer to the current/main
framebuffer
uint16_t * newFb = rp_mem_malloc(LCD_WIDTH * LCD_HEIGHT * 2);
// do something in newFb directly...
gfx.SetFramebuffer(newFb); // Sets the new framebuffer for the library to use
// be careful when doing this since the framebuffer is constantly being sent to the
RGB display
// therefore, this might result to visual glitches
```

4.4. SendFramebuffer

Sends a defined portion of the framebuffer to the display.

Note

This is only applicable for MCU-16 variants, sizes 2.4-inch to 3.5-inch. RGB variants constantly sends framebuffer data to the display.

Syntax: `gfx.SendFramebuffer(x1, y1, x2, y2);`

Argument	Type	Description
x1	uint	Horizontal starting position of the framebuffer area
y1	uint	Vertical starting position of the framebuffer area
x2	uint	Horizontal ending position of the framebuffer area
y2	uint	Vertical ending position of the framebuffer area

Return: None (*void*)

Example

```
gfx.SendFramebuffer(0, 0, 50, 50);  
// Sends the portion of the current framebuffer from (0, 0) to (50, 50) to the display
```

4.5. SetBacklightLevel

Sets the raw backlight level (0-1023) of the display. The actual brightness output of this function is not linear. If linear output is desired, use `gfx.Contrast` instead.

Syntax: `gfx.SetBacklightLevel(level);`

Argument	Type	Description
level	uint16_t	Backlight intensity level (0-1023)

Return: None (*void*)



Example

```
gfx.SetBacklightLevel(1023);  
// Sets the backlight level to maximum
```

4.6. Contrast

Adjusts the display brightness from 0 to 15. This produces a linear brightness output. For more control, `gfx.SetBacklightLevel` can be used.

Syntax: `gfx.Contrast(level);`

Argument	Type	Description
level	uint8_t	Contrast level (0-15)

Return: None (*void*)



Example

```
gfx.Contrast(15);  
// Sets the contrast level to maximum
```

4.7. GetWidth

Returns the width of the display in pixels.

Syntax: `gfx.GetWidth();`

Return: `uint` - Width of the display in pixels.



Example

```
uint width = gfx.GetWidth();  
// Retrieves the display width
```

4.8. GetHeight

Returns the height of the display in pixels.

Syntax: `gfx.GetHeight();`

Return: `uint` - Height of the display in pixels.

Example

```
uint height = gfx.GetHeight();  
// Retrieves the display height
```

4.9. ScreenMode

Sets the screen orientation mode.

Syntax: `gfx.ScreenMode(orientation);`

Argument	Type	Description
orientation	uint8_t	Screen orientation setting (<code>PORTRAIT</code> , <code>PORTRAIT_R</code> , <code>LANDSCAPE</code> , <code>LANDSCAPE_R</code>)

Return: None (*void*)



Example

```
gfx.ScreenMode(PORTRAIT);  
// Sets the screen orientation mode to PORTRAIT
```


4.10. SetAddressWindow

Defines a rectangular area for subsequent drawing operations.

Note

This is only applicable for MCU-16 variants, sizes 2.4-inch to 3.5-inch.

Syntax: `gfx.SetAddressWindow(x_start, y_start, x_end, y_end);`

Argument	Type	Description
x_start	uint16_t	Starting x-coordinate of the rectangle
y_start	uint16_t	Starting y-coordinate of the rectangle
x_end	uint16_t	Ending x-coordinate of the rectangle
y_end	uint16_t	Ending y-coordinate of the rectangle

Return: None (*void*)

Example

```
gfx.SetAddressWindow(10, 10, 100, 100);  
// Defines a rectangular area from (10, 10) to (100, 100)
```

4.11. BlendColor

Blends two colors based on a specified alpha value.

Syntax: `gfx.BlendColor(base_color, new_color, alpha);`

Argument	Type	Description
base_color	uint16_t	The base color (RGB565) to blend
new_color	uint16_t	The color (RGB565) to blend with the base
alpha	uint8_t	Alpha value for blending (0-255)

Return: `uint16_t` - The resulting color (RGB565) after blending.

Example

```
uint16_t blended = gfx.BlendColor(BLUE, RED, 128);  
// Blends BLUE and RED with 50% opacity for RED
```

4.12. Cls

Clears the screen, optionally updating the framebuffer.

Syntax: `gfx.Cls();`

Argument	Type	Description
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: None (void)

Example

```
gfx.Cls();  
// Clears the screen and updates the framebuffer
```

4.13. RectangleFilled

Draws a solid rectangle with a specified RGB565 color.

Syntax: `gfx.RectangleFilled(x1, y1, x2, y2, color);`

Argument	Type	Description
x1	int	Horizontal position of the first endpoint
y1	int	Vertical position of the first endpoint
x2	int	Horizontal position of the second endpoint
y2	int	Vertical position of the second endpoint
color	uint16_t	RGB565 color of the rectangle
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: None (void)

Example

```
gfx.RectangleFilled(10, 10, 50, 50, GREEN);  
// Draws a green filled rectangle from (10, 10) to (50, 50)
```

4.14. RectangleFilled (with color array)

Draws a solid rectangle using an array of colors.

Syntax: `gfx.RectangleFilled(x1, y1, x2, y2, colors);`

Argument	Type	Description
x1	int	Horizontal position of the first endpoint
y1	int	Vertical position of the first endpoint
x2	int	Horizontal position of the second endpoint
y2	int	Vertical position of the second endpoint
colors	const uint16_t*	Array of RGB565 colors for filling the rectangle
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: None (void)

Example

```
gfx.RectangleFilled(10, 10, 50, 50, myColorArray);  
// Draws a filled rectangle with colors from myColorArray
```

4.15. RectangleFilled (with buffer)

Draws a solid rectangle using a buffer.

Syntax: `gfx.RectangleFilled(x1, y1, x2, y2, buffer);`

Argument	Type	Description
x1	int	Horizontal position of the first endpoint
y1	int	Vertical position of the first endpoint
x2	int	Horizontal position of the second endpoint
y2	int	Vertical position of the second endpoint
buffer	const uint8_t*	Buffer of color data for the rectangle
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: None (void)

Example

```
gfx.RectangleFilled(10, 10, 50, 50, myBuffer);  
// Draws a filled rectangle using data from myBuffer
```

4.16. RectangleFilledWithAlpha

Draws a solid rectangle with alpha transparency.

Syntax: `gfx.RectangleFilledWithAlpha(x1, y1, x2, y2, buffer);`

Argument	Type	Description
x1	int	Horizontal position of the first endpoint
y1	int	Vertical position of the first endpoint
x2	int	Horizontal position of the second endpoint
y2	int	Vertical position of the second endpoint
buffer	const uint8_t*	Buffer with RGB565 color and alpha data
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: None (void)



Example

```
gfx.RectangleFilledWithAlpha(10, 10, 50, 50, myAlphaBuffer);  
// Draws a filled rectangle with alpha transparency using myAlphaBuffer
```



Note

This is typically used internally for GCX based widgets. Buffer data contains the format:

- alpha level 0-255
- if alpha level is 0, next byte (0 to 255) is number of fully transparent pixels, 0 designates 256 pixels.
- if alpha level is 255, next byte (0 to 255) is number of fully transparent pixels, 0 designates 256 pixels. This is followed by a color array (RGB565) of the same size.
- otherwise, if alpha neither 0 nor 255, it is followed by the RGB565 color

This format continues until the whole rectangle area is filled

4.17. Hline

Draws a horizontal line at a specified vertical position.

Syntax: `gfx.Hline(y, x1, x2, color);`

Argument	Type	Description
y	int	Vertical position for the line
x1	int	Starting horizontal position for the line
x2	int	Ending horizontal position for the line
color	uint16_t	RGB565 Color of the line
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: None (void)

Example

```
gfx.Hline(20, 0, 100, RED, true);  
// Draws a red horizontal line from (0, 20) to (100, 20)
```


4.18. Vline

Draws a vertical line at a specified horizontal position.

Syntax: `gfx.Vline(x, y1, y2, color);`

Argument	Type	Description
x	int	Horizontal position for the line
y1	int	Starting vertical position for the line
y2	int	Ending vertical position for the line
color	uint16_t	RGB565 Color of the line
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: None (void)

Example

```
gfx.Vline(10, 0, 100, BLUE);  
// Draws a blue vertical line from (10, 0) to (10, 100)
```

4.19. PutPixel

Sets a pixel at the specified coordinates with the given color.

Syntax: `gfx.PutPixel(x, y, color);`

Argument	Type	Description
x	int	X-coordinate of the pixel
y	int	Y-coordinate of the pixel
color	uint16_t	RGB565 Color of the pixel
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
gfx.PutPixel(10, 20, 0xFFFF);  
// Sets a white pixel at (10, 20)
```

4.20. Line

Draws a line between two points with the specified color.

Syntax: `gfx.Line(x1, y1, x2, y2, color);`

Argument	Type	Description
x1	int	X-coordinate of the starting point
y1	int	Y-coordinate of the starting point
x2	int	X-coordinate of the endpoint
y2	int	Y-coordinate of the endpoint
color	uint16_t	RGB565 Color of the line
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
gfx.Line(10, 10, 50, 50, 0x07E0);  
// Draws a green line from (10, 10) to (50, 50)
```

4.21. Ellipse

Draws an ellipse centered at the specified coordinates with the given radii and color.

Syntax: `gfx.Ellipse(x, y, x_rad, y_rad, color);`

Argument	Type	Description
x	int	X-coordinate of the ellipse center
y	int	Y-coordinate of the ellipse center
x_rad	uint	Horizontal radius of the ellipse
y_rad	uint	Vertical radius of the ellipse
color	uint16_t	RGB565 Color of the ellipse
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: void

Example

```
gfx.Ellipse(40, 30, 20, 10, 0xF800);  
// Draws a red ellipse centered at (40, 30)
```

4.22. EllipseFilled

Draws a filled ellipse centered at the specified coordinates with the given radii and color.

Syntax: `gfx.EllipseFilled(x, y, x_rad, y_rad, color);`

Argument	Type	Description
x	int	X-coordinate of the ellipse center
y	int	Y-coordinate of the ellipse center
x_rad	uint	Horizontal radius of the ellipse
y_rad	uint	Vertical radius of the ellipse
color	uint16_t	RGB565 Color of the filled ellipse
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`



Example

```
gfx.EllipseFilled(40, 30, 20, 10, 0x001F);  
// Draws a filled blue ellipse centered at (40, 30)
```

4.23. Circle

Draws a circle centered at the specified coordinates with the given radius and color.

Syntax: `gfx.Circle(x, y, radius, color);`

Argument	Type	Description
x	int	X-coordinate of the circle center
y	int	Y-coordinate of the circle center
radius	uint	Radius of the circle
color	uint16_t	RGB565 Color of the circle
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
gfx.Circle(50, 50, 15, 0xFFE0);  
// Draws a yellow circle centered at (50, 50)
```

4.24. Arc

Draws an arc centered at the specified coordinates with the given radius and start angle.

Syntax: `gfx.Arc(x, y, radius, sa, color);`

Argument	Type	Description
x	int	X-coordinate of the arc center
y	int	Y-coordinate of the arc center
radius	uint	Radius of the arc
sa	int	Start angle of the arc
color	uint16_t	RGB565 Color of the arc
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
gfx.Arc(30, 30, 10, 45, 0x07E0);  
// Draws an arc with 45-degree start angle centered at (30, 30)
```

4.25. ArcFilled

Draws a filled arc centered at the specified coordinates with the given radius, start angle, and end angle.

Syntax: `gfx.ArcFilled(x, y, radius, sa, ea, color);`

Argument	Type	Description
x	int	X-coordinate of the arc center
y	int	Y-coordinate of the arc center
radius	uint	Radius of the arc
sa	int	Start angle of the arc
ea	int	End angle of the arc
color	uint16_t	RGB565 Color of the filled arc
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
gfx.ArcFilled(30, 30, 10, 45, 90, 0x07E0);  
// Draws a filled arc from 45 to 90 degrees centered at (30, 30)
```


4.26. CircleFilled

Draws a filled circle centered at the specified coordinates with the given radius and color.

Syntax: `gfx.CircleFilled(x, y, radius, color);`

Argument	Type	Description
x	int	X-coordinate of the circle center
y	int	Y-coordinate of the circle center
radius	uint	Radius of the filled circle
color	uint16_t	RGB565 Color of the filled circle
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
gfx.CircleFilled(50, 50, 15, 0xFFE0);  
// Draws a filled yellow circle centered at (50, 50)
```

4.27. Triangle

Draws a triangle using the specified vertices and color.

Syntax: `gfx.Triangle(x1, y1, x2, y2, x3, y3, color);`

Argument	Type	Description
x1	int	X-coordinate of vertex 1
y1	int	Y-coordinate of vertex 1
x2	int	X-coordinate of vertex 2
y2	int	Y-coordinate of vertex 2
x3	int	X-coordinate of vertex 3
y3	int	Y-coordinate of vertex 3
color	uint16_t	RGB565 Color of the triangle
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`



Example

```
gfx.Triangle(10, 10, 20, 30, 40, 10, 0xF800);  
// Draws a red triangle with vertices (10, 10), (20, 30), and (40, 10)
```

4.28. TriangleFilled

Draws a filled triangle using the specified vertices and color.

Syntax: gfx.TriangleFilled(x1, y1, x2, y2, x3, y3, color);

Argument	Type	Description
x1	int	X-coordinate of vertex 1
y1	int	Y-coordinate of vertex 1
x2	int	X-coordinate of vertex 2
y2	int	Y-coordinate of vertex 2
x3	int	X-coordinate of vertex 3
y3	int	Y-coordinate of vertex 3
color	uint16_t	RGB565 Color of the filled triangle
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: void

 **Example**

```
gfx.TriangleFilled(10, 10, 20, 30, 40, 10, 0xF800);  
// Draws a filled red triangle with vertices (10, 10), (20, 30), and (40, 10)
```

4.29. Polyline

Draws a polyline defined by a series of vertices with the specified color.

Syntax: `gfx.Polyline(len, vx, vy, color);`

Argument	Type	Description
len	uint	Number of vertices in the polyline
vx	const int*	Array of X-coordinates of the vertices
vy	const int*	Array of Y-coordinates of the vertices
color	uint16_t	RGB565 Color of the polyline
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
int x[] = {10, 20, 30, 40};
int y[] = {10, 20, 10, 20};
gfx.Polyline(4, x, y, 0xFFFF);
// Draws a polyline connecting the specified vertices in white
```

4.30. Polygon

Draws a polygon defined by a series of vertices with the specified color.

Syntax: `gfx.Polygon(len, vx, vy, color);`

Argument	Type	Description
len	uint	Number of vertices in the polygon
vx	const int*	Array of X-coordinates of the vertices
vy	const int*	Array of Y-coordinates of the vertices
color	uint16_t	RGB565 Color of the polygon
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
int x[] = {10, 20, 30};
int y[] = {10, 30, 10};
gfx.Polygon(3, x, y, 0xF800);
// Draws a red polygon connecting the specified vertices
```

4.31. PolygonFilled

Draws a filled polygon defined by a series of vertices with the specified color.

Syntax: gfx.PolygonFilled(len, vx, vy, color);

Argument	Type	Description
len	uint	Number of vertices in the polygon
vx	const int*	Array of X-coordinates of the vertices
vy	const int*	Array of Y-coordinates of the vertices
color	uint16_t	RGB565 Color of the filled polygon
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: void

 Example

```
int x[] = {10, 20, 30};
int y[] = {10, 30, 10};
gfx.PolygonFilled(3, x, y, 0x07E0);
// Draws a filled green polygon connecting the specified vertices
```

4.32. SetBackgroundColor

Sets the background color of the display.

Syntax: `gfx.SetBackgroundColor(color);`

Argument	Type	Description
color	uint16_t	Color to set as the background

Return: `uint16_t` - Previous background color.

Example

```
uint16_t oldColor = gfx.SetBackgroundColor(0x0000);  
// Sets the background color to black and saves the previous color
```

4.33. ClipWindow

Defines a clipping window for rendering.

Syntax: `gfx.ClipWindow(x1, y1, x2, y2);`

Argument	Type	Description
x1	int	X-coordinate of the top-left corner
y1	int	Y-coordinate of the top-left corner
x2	int	X-coordinate of the bottom-right corner
y2	int	Y-coordinate of the bottom-right corner

Return: `bool` - `true` if the clipping window was set successfully, `false` otherwise.

Example

```
if (gfx.ClipWindow(10, 10, 100, 100)) {  
    // Clipping window set successfully  
}
```


4.34. SetFont

Sets the font for text rendering.

Syntax: `gfx.SetFont(font);`

Argument	Type	Description
font	const uint8_t*	Pointer to the font data

Return: `const uint8_t*` - Pointer to the previous font.

Example

```
const uint8_t* previousFont = gfx.SetFont(myFont);  
// Sets a new font and saves the previous one
```

Note

Font format is similar to 4DGL fonts and can be generated using the [4D Font Tool](#)

4.35. SetFontForeground (TextArea4D)

Sets the foreground color for text in a specified area.

Syntax: `gfx.SetFontForeground(area, color);`

Argument	Type	Description
area	TextArea4D	Area for which to set the color
color	uint16_t	Foreground color (RGB565)

Return: `uint16_t` - Previous foreground color.

Example

```
uint16_t oldColor = gfx.SetFontForeground(myTextArea, 0xFFFF);  
// Sets the foreground color to white for the specified area
```

4.36. SetFontBackground (TextArea4D)

Sets the background color for text in a specified area.

Syntax: `gfx.SetFontBackground(area, color);`

Argument	Type	Description
area	TextArea4D	Area for which to set the color
color	uint16_t	Background color (RGB565)
transparent	bool	(Optional) Whether the background is transparent, default: false

Return: `uint16_t` - Previous background color.

Example

```
uint16_t oldColor = gfx.SetFontBackground(myTextArea, 0x0000, true);  
// Sets the background color to black for the specified area with transparency
```

4.37. SetFontForeground

Sets the foreground color for text rendering.

Syntax: `gfx.SetFontForeground(color);`

Argument	Type	Description
color	uint16_t	Foreground color (RGB565)

Return: `uint16_t` - Previous foreground color.



Example

```
uint16_t oldColor = gfx.SetFontForeground(0xFFFF);  
// Sets the foreground color to white
```

4.38. SetFontBackground

Sets the background color for text rendering.

Syntax: `gfx.SetFontBackground(uint16_t color);`

Argument	Type	Description
color	uint16_t	Background color
transparent	bool	(Optional) Whether the background is transparent, default: false

Return: `uint16_t` - Previous background color.

Example

```
uint16_t oldColor = gfx.SetFontBackground(0x0000, false);  
// Sets the background color to black without transparency
```

4.39. MoveTo

Moves the current drawing position to the specified coordinates.

Syntax: `gfx.MoveTo(int x, int y);`

Argument	Type	Description
x	int	X-coordinate of the new position
y	int	Y-coordinate of the new position

Return: `bool` - `true` if the move was successful, `false` otherwise.



Example

```
if (gfx.MoveTo(50, 100)) {  
    // Move to coordinates (50, 100) successfully  
}
```

4.40. MoveRel

Moves the current drawing position relative to its current position.

Syntax: `gfx.MoveRel(int x_offset, int y_offset);`

Argument	Type	Description
x_offset	int	X-offset to move from current position
y_offset	int	Y-offset to move from current position

Return: `bool` - `true` if the move was successful, `false` otherwise.



Example

```
if (gfx.MoveRel(10, -5)) {  
    // Move 10 units right and 5 units up from the current position  
}
```

4.41. print

Prints a string at the current position.

Syntax: `gfx.print(str);`

Argument	Type	Description
str	const char*	String to print
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `size_t` - Number of characters printed.

Example

```
size_t charsPrinted = gfx.print("Hello, World!");  
// Prints "Hello, World!" at the current position
```


4.42. printf

Prints a formatted string at the current position.

Syntax: `gfx.printf(const char *format, ...);`

Argument	Type	Description
format	const char*	Format string
...	...	Additional arguments as needed

Return: `size_t` - Number of characters printed.

Example

```
size_t charsPrinted = gfx.printf("Value: %d", 42);  
// Prints "Value: 42" at the current position
```

4.43. CreateTextArea

Creates a text area for rendering text.

Syntax: `gfx.CreateTextArea(int x1, int y1, int x2, int y2, uint16_t fg_color, uint16_t bg_color);`

Argument	Type	Description
x1	int	X-coordinate of the top-left corner
y1	int	Y-coordinate of the top-left corner
x2	int	X-coordinate of the bottom-right corner
y2	int	Y-coordinate of the bottom-right corner
fg_color	uint16_t	Foreground color (RGB565)
bg_color	uint16_t	Background color (RGB565)

Return: `TextArea4D` - The created text area.

Example

```
TextArea4D area = gfx.CreateTextArea(10, 10, 100, 50, 0xFFFF, 0x0000);  
// Creates a text area with specified dimensions and colors
```

4.44. print (TextArea4D)

Prints a string in the specified text area.

Syntax: `gfx.print(area, str);`

Argument	Type	Description
area	TextArea4D	The text area to print into
str	const char*	String to print
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `size_t` - Number of characters printed.

Example

```
size_t charsPrinted = gfx.print(area, "Hello, Text Area!");  
// Prints "Hello, Text Area!" in the specified text area
```

4.45. printf (TextArea4D)

Prints a formatted string in the specified text area.

Syntax: `gfx.printf(area, format, ...);`

Argument	Type	Description
area	TextArea4D	The text area to print into
format	const char*	Format string
...	...	Additional arguments as needed

Return: `size_t` - Number of characters printed.

Example

```
size_t charsPrinted = gfx.printf(area, "Value: %d", 42);  
// Prints "Value: 42" in the specified text area
```

5. Image Control Functions

The following functions are included in `img`. These allows displaying of widgets generated by Workshop5 into to flash memory or as a file in the microSD.

5.1. LoadImageControl (Pointer)

Loads an image control from a pointer to image data.

Syntax: `img.LoadImageControl(ptr);`

Argument	Type	Description
ptr	const uint8_t*	Pointer to the image data

Return: `ImageControl4D` - Handle to the loaded image control.

Example

```
ImageControl4D hndl = img.LoadImageControl(imageDataPtr);  
// Loads an image control from a pointer to image data
```

5.2. LoadImageControl (Filename)

Loads an image control from a specified file.

Syntax: `img.LoadImageControl(filename);`

Argument	Type	Description
filename	const char*	Path to the image file

Return: `ImageControl4D` - Handle to the loaded image control.

Example

```
ImageControl4D hndl = img.LoadImageControl("filename.gcx");  
// Loads an image control from a specified file
```

5.3. GetCount

Retrieves the number of images in the image control.

Syntax: `img.GetCount(hndl);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control

Return: `uint` - Number of images in the control.

Example

```
uint count = img.GetCount(hndl);  
// Retrieves the number of images in the image control
```

5.4. GetFile

Retrieves a pointer to the file associated with the image control.

Syntax: `img.GetFile(hndl);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control

Return: `FIL*` - Pointer to the file associated with the image control.

Example

```
FIL* filePtr = img.GetFile(hndl);  
// Retrieves a pointer to the file associated with the image control
```

5.5. GetInfo

Retrieves media information for a specified image.

Syntax: `img.GetInfo(hndl, index);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image

Return: `MediaInfo4D` - Information about the specified image.



Example

```
MediaInfo4D info = img.GetInfo(hndl, 0);  
// Retrieves media information for the first image
```


5.6. SetProperties

Sets properties for a specified image in the image control.

Syntax: `img.SetProperties(hndl, index, properties);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image
properties	const uint16_t*	Pointer to an array of properties

Return: `void`

Example

```
uint16_t props[] = {100};  
img.SetProperties(hndl, iSliderA0, props);  
// Sets properties for the first image in the control
```

Note

This is typically used to set the behavior of Sliders and Knobs. It is recommended to simply use `SetupMedia` function that is automatically generated instead of doing this manually. For better examples, create a project and add Slider and Knobs and check the 'Generated' tab.

5.7. SetValue

Sets a value for a specified property of an image.

Syntax: `img.SetValue(hndl, index, value);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image
value	uint16_t	Value to set for the property

Return: `uint16_t` - The new value of the property.

Example

```
uint16_t newValue = img.SetValue(hndl, 0, 5);  
// Sets the value of the first image's property
```

5.8. GetValue

Gets a value for a specified property of an image.

Syntax: `img.GetValue(hndl, index);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image

Return: `uint16_t` - The value of the specified property.

Example

```
uint16_t value = img.GetValue(hndl, 0);  
// Gets the value of the first image's property
```

5.9. SetPosition

Sets the position of a specified image in the image control.

Syntax: `img.SetPosition(hndl, index, x, y);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image
x	int16_t	X-coordinate of the position
y	int16_t	Y-coordinate of the position

Return: `void`



Example

```
img.SetPosition(hndl, 0, 50, 100);  
// Sets the position of the first image to (50, 100)
```

5.10. GetFrames

Gets the number of frames for a specified image.

Syntax: `img.GetFrames(ImageControl4D hndl, uint index);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image

Return: `uint16_t` - The number of frames for the specified image.

Example

```
uint16_t frames = img.GetFrames(hndl, 0);  
// Gets the number of frames for the first image
```

5.11. Show

Displays a specified image from the image control.

Syntax: `img.Show(hndl, index);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
img.Show(hndl, iSliderA0);  
// Displays the first image from the image control
```

5.12. Clear

Clears a specified image in the image control with a given color.

Syntax: `img.Clear(hndl, index, color);`

Argument	Type	Description
hndl	ImageControl4D	Handle to the image control
index	uint	Index of the image
color	uint16_t	RGB565 Color to clear the image with
draw_fb	bool	(Optional) Specifies whether to draw the framebuffer at the end of operation (default: true). Note: This is only applicable for MCU displays, sizes 2.4-inch to 3.5-inch

Return: `void`

Example

```
img.Clear(hndl, 0, 0xFFFF);  
// Clears the first image with white color
```

6. Touch Handling Functions

The following functions are included for touch support.

6.1. Initialize

Initializes the touch object.

Syntax: `touch.Initialize();`

Return: `bool` - `true` if initialization was successful, `false` otherwise.

Example

```
if (touch.Initialize()) {  
    // Touch object initialized successfully  
}
```


6.2. Calibrate

Calibrates the touch screen.



Note

This is only applicable for resistive touch displays

Syntax: `touch.Calibrate();`

Return: `bool` - `true` if calibration was successful, `false` otherwise.



Example

```
if (touch.Calibrate()) {  
    // Touch screen calibrated successfully  
}
```

6.3. GetPoints

Performs touch reading operations and returns the number of points.

Syntax: `touch.GetPoints();`

Return: `uint8_t` - number of active touch points

Example

```
uint8_t points = touch.GetPoints();  
// Retrieves the number of touch points
```

6.4. GetStatus

Retrieves the status of the touch point.

Syntax: `touch.GetStatus();`

Argument	Type	Description
point	uint8_t	(Optional) Index specifying touch (usually ignored for single touch)

Return: `int8_t` - Status of the touch object.

Example

```
int8_t status = touch.GetStatus();  
// Retrieves the status of the touch object
```

6.5. GetID

Retrieves the ID of the currently active touch point.

Syntax: `touch.GetID();`

Argument	Type	Description
point	uint8_t	(Optional) Index specifying touch (usually ignored for single touch)

Return: `int16_t` - ID of the active touch point.

Example

```
int16_t id = touch.GetID();  
// Retrieves the ID of the currently active touch point
```

6.6. GetX

Retrieves the X-coordinate of the currently active touch point.

Syntax: `touch.GetX();`

Argument	Type	Description
point	uint8_t	(Optional) Index specifying touch (usually ignored for single touch)

Return: `int16_t` - X-coordinate of the active touch point.

Example

```
int16_t x = touch.GetX();  
// Retrieves the X-coordinate of the active touch point
```

6.7. GetY

Retrieves the Y-coordinate of the currently active touch point.

Syntax: `touch.GetY();`

Argument	Type	Description
point	uint8_t	(Optional) Index specifying touch (usually ignored for single touch)

Return: `int16_t` - Y-coordinate of the active touch point.

Example

```
int16_t y = touch.GetY();  
// Retrieves the Y-coordinate of the active touch point
```

7. Legal Notice

7.1. Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. 4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems. 4D Systems reserves the right to modify, update or make changes to Specifications or written material without prior notice at any time.

All trademarks belong to their respective owners and are recognised and acknowledged.

7.2. Disclaimer of Warranties & Limitations of Liabilities

4D Systems makes no warranty, either expressed or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

Images and graphics used throughout this document are for illustrative purposes only. All images and graphics used are possible to be displayed on the 4D Systems range of products, however the quality may vary.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail - safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems' products and devices in 'High Risk Activities' and in any other application is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.