




MSP Graphics Library version 3.30.00.19

USER'S GUIDE

Copyright

Copyright © Texas Instruments Incorporated. All rights reserved.

 Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments
Post Office Box 655303
Dallas, TX 75265
<http://www.ti.com/msp430>



Revision Information

This is version 3.30.00.19 of this document, last updated on June 12, 2017.

Document License

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 United States License (CC BY-SA 3.0). To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/us/> or send a letter to

Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Contributors to this document

Copyright © 2015 Texas Instruments Incorporated - <http://www.ti.com/>

Table of Contents

Copyright	2
Revision Information	2
Document License	2
Contributors to this document	2
1 Using Template Driver files	5
2 Circle API	7
2.0.1 API Functions	7
3 Context API	9
3.0.2 API Functions	9
4 Image API	11
4.0.3 API Functions	11
5 Line API	13
5.0.4 API Functions	13
6 Rectangle API	15
6.0.5 API Functions	15
7 String API	17
7.0.6 API Functions	17
8 Button API	19
8.0.7 API Functions	19
9 ImageButton API	21
9.0.8 API Functions	21
10 RadioButton API	23
10.0.9 API Functions	23
11 CheckBox API	25
11.0.10 API Functions	25
12 Using the MSP Image Reformer Tool	27
IMPORTANT NOTICE	28

1 Using Template Driver files

Modifying the Template Driver File ??

This template driver is intended to be modified for creating new LCD drivers. It is setup so that only `Template_DriverPixelDraw()` and `DPYCOLORTRANSLATE()` and some LCD size configuration settings in the header file `Template_Driver.h` are **REQUIRED** to be written. These functions are marked with the string "TemplateDisplayFix" in the comments so that a search through `Template_Driver.c` and `Template_Driver.h` can quickly identify the necessary areas of change.

`Template_DriverPixelDraw()` is the base function to write to the LCD display. Functions like `WriteData()`, `WriteCommand()`, and `SetAddress()` are suggested to be used to help implement the `Template_DriverPixelDraw()` function, but are not required. `SetAddress()` should be used by other pixel level functions to help optimize them.

This is not an optimized driver and will significantly impact performance. It is highly recommended to first get the prototypes working with the single pixel writes, and then go back and optimize the driver. Please see application note [SLAA548](#) for more information on how to fully optimize LCD driver files. In short, driver optimizations should take advantage of the auto-incrementing of the LCD controller. This should be utilized so that a loop of `WriteData()` can be used instead of a loop of `Template_DriverPixelDraw()`. The pixel draw loop contains both a `SetAddress()` + `WriteData()` compared to `WriteData()` alone. This is a big time saver especially for the line draws and `Template_DriverPixelDrawMultiple()`. More optimization can be done by reducing function calls by writing macros, eliminating unnecessary instructions, and of course taking advantage of other features offered by the LCD controller. With so many pixels on an LCD screen each instruction can have a large impact on total drawing time.

2 Circle API

[Introduction](#) ??
[API Functions](#) 7

[Programming Example](#) .. ?? The Circle API provides simple functions to draw a circle on the display. There are two different functions used to draw a circle; one which draws the outline, and the other which draws a filled-in circle. The clipping of the circle is performed within the routine; the display driver's circle fill routine is used to perform the actual circle fill.

The code for this API is contained in `<SDK_INSTALL_DIR>/source/ti/grlib/circle.c`, with `<SDK_INSTALL_DIR>/source/ti/circle.h` containing the API definitions for use by applications.

2.0.1 API Functions

The Circle API is broken into two separate functions both of which write to the display.

The function which draws a circle is handled by

[Graphics_drawCircle\(\)](#)

The function which draws a filled-in circle is handled by

[Graphics_fillCircle\(\)](#)

[Graphics_Context](#) g_sContext;

```
// // Initialize the graphics context // Graphics_initContext(g_sContext,  
g_sKitronix320x240x16_SSD2119, g_sKitronix320x240x16_SSD2119_funcs);  
Graphics_setForegroundColor(g_sContext, GRAPHICS_COLOR_BLACK); Graph-  
ics_setBackgroundColor(g_sContext, GRAPHICS_COLOR_WHITE);
```

```
Graphics_clearDisplay(g_sContext);
```

```
Graphics\_drawCircle(g_sContext, 275, 100, 30); Graphics\_fillCircle(g_sContext, 50, 100, 30);
```


3 Context API

Introduction	??
API Functions	9
Programming Example	??

The Context API provides simple functions to initialize a drawing context, preparing it for use on the display. The display driver will be used for all subsequent graphics operations.

The code for this API is contained in `<SDK_INSTALL_DIR>/source/ti/grlib/context.c`, with `<SDK_INSTALL_DIR>/source/ti/grlib/context.h` containing the API definitions for use by applications.

3.0.2 API Functions

The Context API is broken into two separate functions both of which initialize the context for the display, but differ in the way they set the clipping regions of the screen. The clipping region is not allowed to exceed the extents of the screen, but may be a portion of the screen. The supplied coordinates are inclusive for the clipping region. As a consequence, the clipping region must contain at least one row and one column.

The function which initializes the context and whose clipping region is set to the extent of the entire screen is handled by

```
Graphics_initContext()
```

The function which initializes the context and also sets a clipping region is handled by

```
Graphics_setClipRegion()
```

```
Graphics_Context g_sContext;
```

```
// // Initialize the graphics context // Graphics_initContext(g_sContext,
g_sKitronix320x240x16_SSD2119, g_sKitronix320x240x16_SSD2119_funcs);
Graphics_setBackgroundColor(g_sContext, GRAPHICS_COLOR_BLACK); Graph-
ics_setForegroundColor(g_sContext, GRAPHICS_COLOR_WHITE);
__no_operation();
```


4 Image API

Introduction??
 API Functions11
 Programming Example ..?? The Image API provides simple functions to draw images on the screen. There are two different functions used to draw a image; one which converts the palette of a bitmap image and the other which renders the bitmap image onto the screen.

The code for this API is contained in `<SDK_INSTALL_DIR>/source/ti/grlib/image.c`, with `<SDK_INSTALL_DIR>/source/ti/grlib/image.h` containing the API definitions for use by applications.

4.0.3 API Functions

The Image API is broken into two separate functions, one to convert the palette and the other to render to the display. Calling the `GrImageDraw()` function also invokes `GrPaletteConversion()` as well so the user only needs to be concerned with the `GrImageDraw()` function.

The image may be either 1-, 4-, or 8-bits per pixel by using a palette supplied in the image data. The image palette is in 24-bit RGB form and by calling `GrPaletteConversion()`, the palette can then be sent to the LCD using `DpyColorTranslate` function. The converted palette is contained in a global buffer while the original image remains the same. The palette can be uncompressed data or it can be compressed using several different compression types. Compression options are either 4-, 7- or 8-bit run length encoding, or a custom run length encoding variation written for complex 8-bit per pixel images.

The function which draws a bitmap image is handled by

`Graphics_drawImage()`

`Graphics_Context g_sContext;`

```
// // Initialize the graphics context // Graphics_initContext(g_sContext,
g_sKitronix320x240x16_SSD2119, g_sKitronix320x240x16_SSD2119_funcs);
Graphics_setForegroundColor(g_sContext, GRAPHICS_COLOR_BLACK); Graphics_
setBackgroundColor(g_sContext, GRAPHICS_COLOR_WHITE);
```

`Graphics_clearDisplay(g_sContext);`

`Graphics_drawImage(sContext, imageName, 200, 70);`

5 Line API

Introduction	??
API Functions	13
Programming Example	??

?? The Line API provides simple functions to draw lines on the display. There are five different functions used to draw a line; two optimized functions for horizontal and vertical drawing, one generic line drawing function, two functions for clipping. The user needs only to be concerned with the generic line drawing function, GrLineDraw(), as it incorporates the use of all the other functions automatically.

The code for this API is contained in `<SDK_INSTALL_DIR>/source/ti/grlib/line.c`, with `<SDK_INSTALL_DIR>/source/ti/grlib/line.h` containing the API definitions for use by applications.

5.0.4 API Functions

The Line API is broken into two separate functions; one for drawing and the other for clipping (internal functions).

The functions that draw a line are handled by

Graphics_drawLineV()

Graphics_drawLineH()

Graphics_drawLine()

The user needs only to be concerned with the generic line drawing function, GrLineDraw(), as it incorporates the use of all the other functions automatically.

Graphics_Context g_sContext;

```
// // Initialize the graphics context // Graphics_initContext(g_sContext,
g_sKitronix320x240x16_SSD2119, g_sKitronix320x240x16_SSD2119_funcs);
Graphics_setForegroundColor(g_sContext, GRAPHICS_COLOR_BLACK); Graphics_
setBackgroundColor(g_sContext, GRAPHICS_COLOR_WHITE);
```

```
Graphics_clearDisplay(g_sContext);
```

```
Graphics_drawLine(sContext, 130, 30, 275, 200 ); Graphics_drawLineH(sContext, 20, 180, 220);
Graphics_drawLineV(sContext, 30, 50, 160);
```


6 Rectangle API

[Introduction](#) ??
[API Functions](#) 15
[Programming Example](#) ?? The Rectangle API provides simple functions to draw a rectangle on the display. There are two different functions used to draw a rectangle; one which draws the outline, and the other which draws a filled-in rectangle. The clipping of the rectangle is performed within the routine; the display driver's rectangle fill routine is used to perform the actual rectangle fill.

The code for this API is contained in `<SDK_INSTALL_DIR>/source/ti/grlib/rectangle.c`, with `<SDK_INSTALL_DIR>/source/ti/grlib/rectangle.h` containing the API definitions for use by applications.

6.0.5 API Functions

The Rectangle API is broken into two groups; one that draws to the screen and the other which perform checks(internal functions).

The functions which draw rectangles are handled by

`Graphics_drawRectangle()`

`Graphics_fillRectangle()`

```
Graphics_Context g_sContext; Graphics_Rectangle myRectangle1 = { 60, 60, 120, 120}; Graphics_Rectangle myRectangle2 = { 160, 60, 220, 120};
```

```
// // Initialize the graphics context // Graphics_initContext(g_sContext,
g_sKitronix320x240x16_SSD2119, g_sKitronix320x240x16_SSD2119_funcs);
Graphics_setBackgroundColor(g_sContext, GRAPHICS_COLOR_BLACK); Graphics_
setForegroundColor(g_sContext, GRAPHICS_COLOR_WHITE);
```

```
Graphics_clearDisplay(g_sContext);
```

```
Graphics_drawRectangle(g_sContext, myRectangle1); Graphics_fillRectangle(g_sContext,
myRectangle2);
```


7 String API

[Introduction](#) ??
[API Functions](#) 17
[Programming Example](#) ?? The String API provides simple functions to draw strings on the screen. There are several different functions used to draw a string; one which counts the number of leading zeroes, one for obtaining the display width of the string, one for drawing the string to the display, one for setting the location of the current string table, one to set the current language, and the last one for grabbing the string from the current string table. The user should not directly call NumLeadingZeroes() as it is used internally.

The code for this API is contained in `<SDK_INSTALL_DIR>/source/ti/grlib/string.c`, with `<SDK_INSTALL_DIR>/source/ti/grlib/string.h` containing the API definitions for use by applications.

7.0.6 API Functions

The String API available are classified as below.

The functions which calculate and set up parameters are handled by

`Graphics_getStringWidth()`

The function which draws a string to the display is handled by

`Graphics_drawString()`

`Graphics_Context g_sContext;`

```
// // Initialize the graphics context // Graphics_initContext(g_sContext,
g_sKitronix320x240x16_SSD2119, g_sKitronix320x240x16_SSD2119_funcs);
Graphics_setBackgroundColor(g_sContext, GRAPHICS_COLOR_BLACK); Graph-
ics_setForegroundColor(g_sContext, GRAPHICS_COLOR_WHITE);
```

`Graphics_clearDisplay(g_sContext);`

`Graphics_setFont(g_sContext, g_sFontCm26); Graphics_drawString(g_sContext, "Welcome to ", -1, 20, 8, 0);`

`Graphics_setFont(g_sContext, g_sFontCm30); Graphics_drawString(g_sContext, "Dallas TX", -1, 20, 180, 0);`

8 Button API

Introduction	??
API Functions	19
Programming Example ??	The Button API provides simple functions to draw a button on the display.

8.0.7 API Functions

The Button API is broken into four separate functions both of which write to the display.

The function which draws a button is handled by

`Graphics_drawButton()`

The function which draws a selected button

`Graphics_drawSelectedButton()`

The function which draws a released button

`Graphics_drawReleasedButton()`

The function which determines if button has been pressed

`Graphics_isButtonSelected()`

`Graphics_Button yesButton;`

```
yesButton.xMin = 80; yesButton.xMax = 150; yesButton.yMin = 80; yesButton.yMax =
120; yesButton.borderWidth = 1; yesButton.selected = false; yesButton.fillColor = GRAPH-
ICS_COLOR_RED; yesButton.borderColor = GRAPHICS_COLOR_RED; yesButton.selectedColor
= GRAPHICS_COLOR_BLACK; yesButton.textColor = GRAPHICS_COLOR_BLACK; yes-
Button.selectedTextColor = GRAPHICS_COLOR_RED; yesButton.textXPos = 100; yesBut-
ton.textYPos = 90; yesButton.text = "YES"; yesButton.font = g_sFontCm18;
```

`Graphics_drawButton(g_sContext, yesButton);`

9 ImageButton API

Introduction	??
API Functions	21
Programming Example ??	The ImageButton API provides simple functions to draw a imageButton on the display.

9.0.8 API Functions

The ImageButton API is broken into four separate functions both of which write to the display.

The function which draws a imageButton is handled by

Graphics_drawImageButton()

The function which draws a selected imageButton

Graphics_drawSelectedImageButton()

The function which draws a released imageButton

Graphics_drawReleasedImageButton()

The function which determines if imageButton has been pressed

Graphics_isImageButtonSelected()

Graphics_ImageButton primitiveButton;

```
primitiveButton.xPosition=20; primitiveButton.yPosition=50; primitiveButton.borderWidth=5; primitiveButton.selected=false; primitiveButton.imageWidth=Primitives_Button4BPP_UNCOMP.xSize; primitiveButton.imageHeight=Primitives_Button4BPP_UNCOMP.ySize; primitiveButton.borderWidth=GRAPHICS_COLOR_WHITE; primitiveButton.selectedColor=GRAPHICS_COLOR_RED; primitiveButton.image=Primitives_Button4BPP_UNCOMP;
```

Graphics_drawImageButton(g_sContext, primitiveButton);

10 RadioButton API

Introduction	??
API Functions	23
Programming Example . ??	The RadioButton API provides simple functions to draw a radioButton on the display.

10.0.9 API Functions

The RadioButton API is broken into four separate functions both of which write to the display.

The function which draws a radioButton is handled by

`Graphics_drawRadioButton()`

The function which draws a selected radioButton

`Graphics_drawSelectedRadioButton()`

The function which draws a released radioButton

`Graphics_drawReleasedRadioButton()`

The function which determines if radioButton has been pressed

`Graphics_isRadioButtonSelected()`

```
Graphics_RadioButton radioButton1 = { 5, 15, true, 4, GRAPHICS_COLOR_BLACK, 9, GRAPHICS_COLOR_BLACK, GRAPHICS_COLOR_WHITE, g_sFontFixed6x8, "Option 1" };
```

```
Graphics_drawRadioButton(g_sContext, radioButton1);
```

11 CheckBox API

Introduction	??
API Functions	25
Programming Example . ??	The CheckBox API provides simple functions to draw a checkBox on the display.

11.0.10 API Functions

The CheckBox API is broken into four separate functions both of which write to the display.

The function which draws a checkBox is handled by

`Graphics_drawCheckBox()`

The function which draws a selected checkBox

`Graphics_drawSelectedCheckBox()`

The function which draws a released checkBox

`Graphics_drawReleasedCheckBox()`

The function which determines if checkBox has been pressed

`Graphics_isCheckBoxSelected()`

```
Graphics_CheckBox checkBox1 = { 5, 15, false, 4, GRAPHICS_COLOR_BLACK, GRAPHICS_COLOR_WHITE, GRAPHICS_COLOR_BLACK, 9, g_sFontFixed6x8, "Option 1" };
```

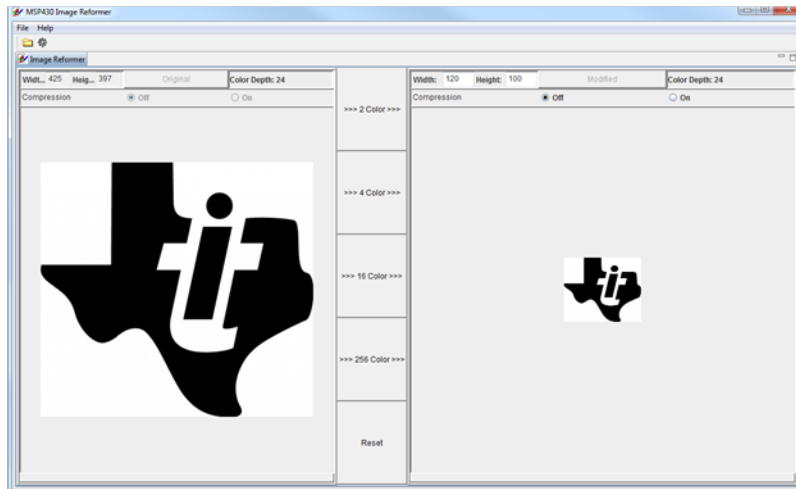
```
Graphics_drawCheckBox(g_sContext, checkBox1);
```


12 Using the MSP Image Reformer Tool

Introduction ??

Running MSP Image Reformer Tool ??

Image Reformer converts images into C code that can be used with the MSP Graphics Library. Import your source image, make your bpp and size settings, generate C code, and then add the resulting file into your project.



To run the tool go to `<SDK_INSTALL_DIR>\tools\image-reformer` and run `image_ReFormer.jar`

In order to keep MSP Graphics Library and Open Source Project the JRE is not shipped with the Library and it requires that the users have Java 1.5 or later installed in their machines. Currently the tool only has support for Windows OS support.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © , Texas Instruments Incorporated