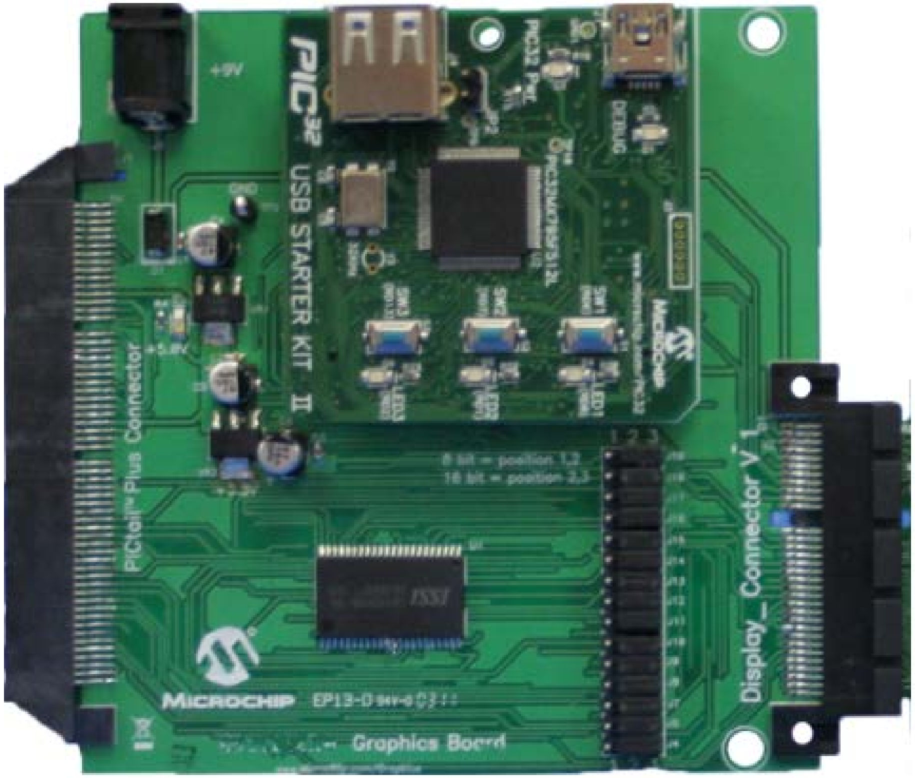
MICROCHIP

*Regional Training Centers*

# HIF2132A

## Designing Stylish HMIs with Microchip's Graphics Library and Visual Graphics Display Designer using PIC32 LCC Graphics

# The Board



- PSRAM for external frame buffer.
- Internal/External Memory Jumpers
- Part Number:
    - AC164144

# Class Objectives

After this class you will be able to:

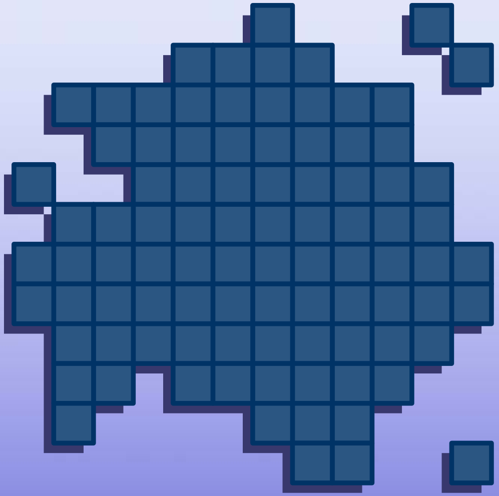
- Use a PC based graphical design tool to create a customized user interface code template.
- Implement application callback functions to allow users to fully interact with the system
- Use advanced features in Microchip's Graphics Library to enhance your GUI appearance
- Learn about the LCC Graphics Solution

# Agenda

- Microchip Graphics Library Intro
- Graphics Basics and Terminology
    - Colors and Images
    - Fonts
    - Widgets and Style Schemes
- Lab 1 – Creating a Splash Screen
- Lab 2 – Creating a Simple Icon Menu
- Interfacing the User (Message Interface)
- Lab 3 – User Interface using Message Callback
- Advanced Features (Drawing Callback)

# Agenda

- LCC Interfacing to TFT LCD Panels
- PIC32 Peripherals Used for Graphics
  - DMA
  - PMP
- Controller/Driver Tasks
- Lab 4 – LCC and Primitive Layer Demo
- Customizing for YOUR application
- Summary

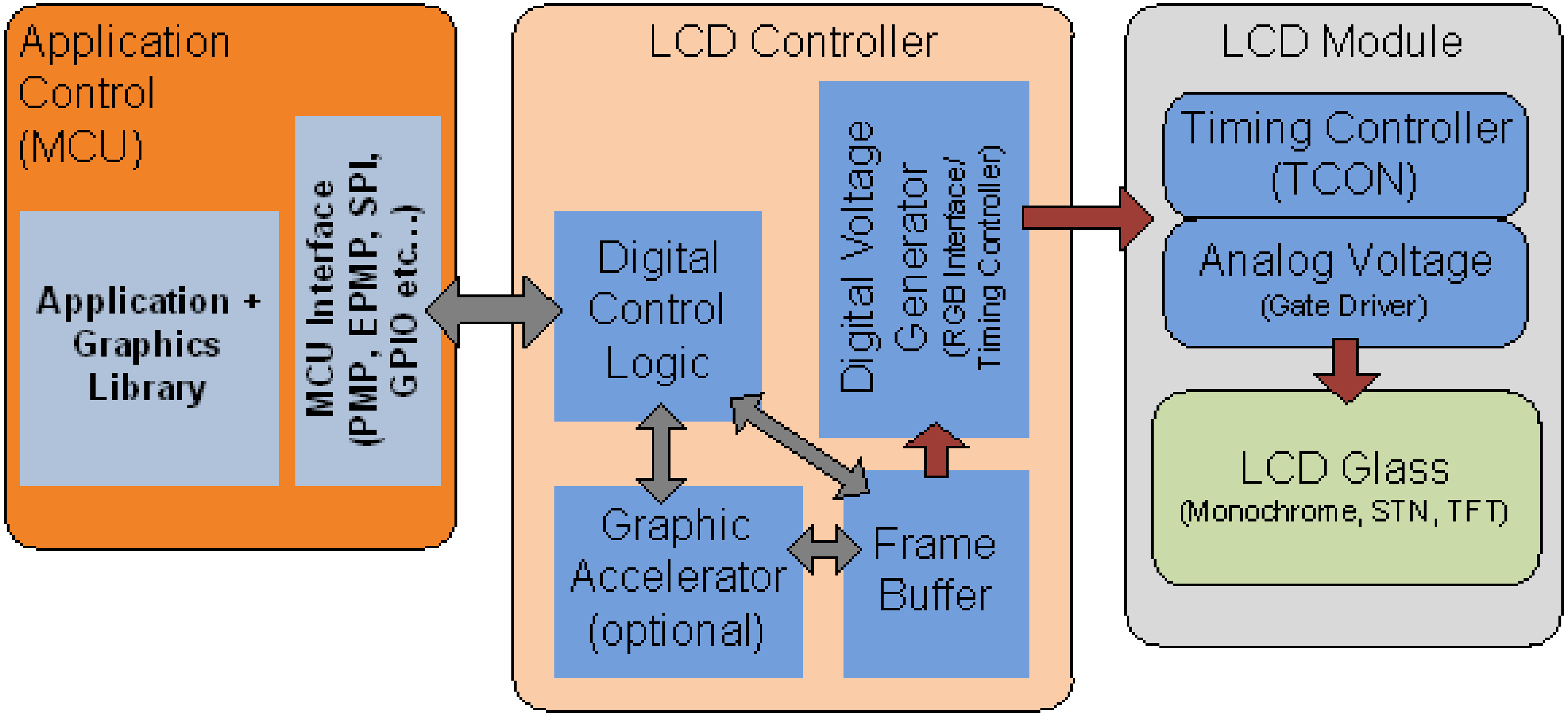


# Embedded Graphics Systems

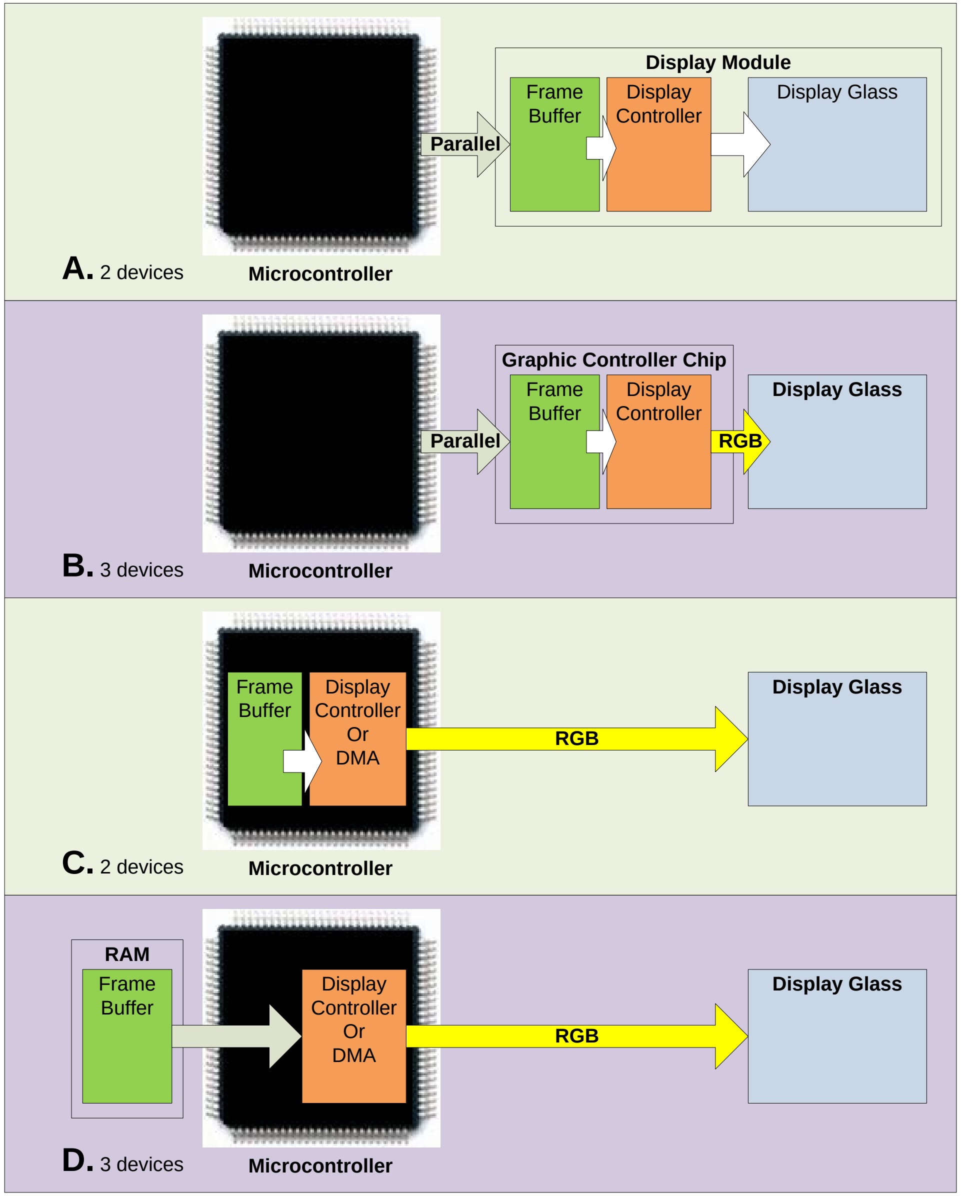
## Overview

Regional Training Centers

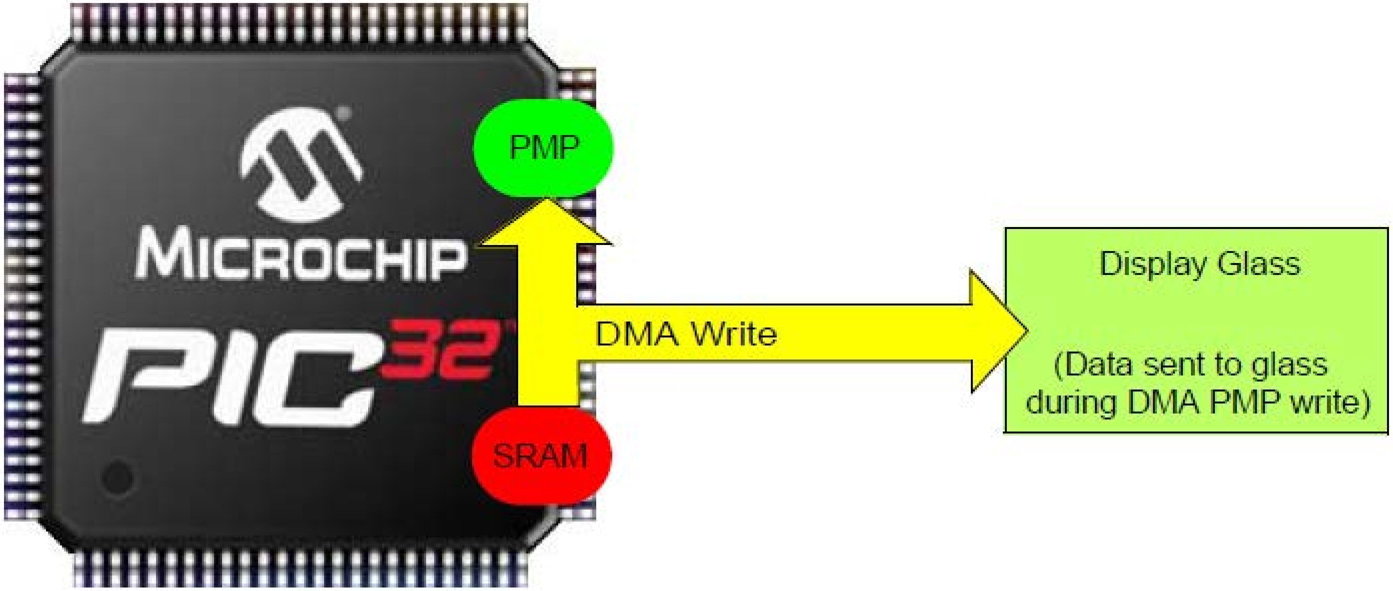
# How it works …


Application Control (MCU)
Application + Graphics Library
MCU Interface (PMP, EPMP, SPI, GPIO etc..)
LCD Controller
Digital Control Logic
Digital Voltage Generator (RGB Interface/ Timing Controller)
Graphic Accelerator (optional)
Frame Buffer
LCD Module
Timing Controller (TCON)
Analog Voltage (Gate Driver)
LCD Glass (Monochrome, STN, TFT)

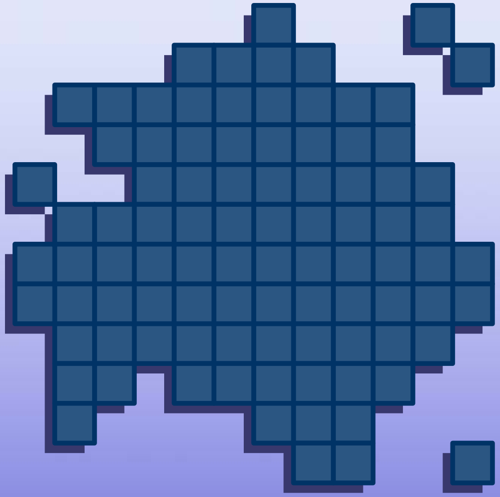
# Summary of Graphical Setups



# LCC Controllerless method



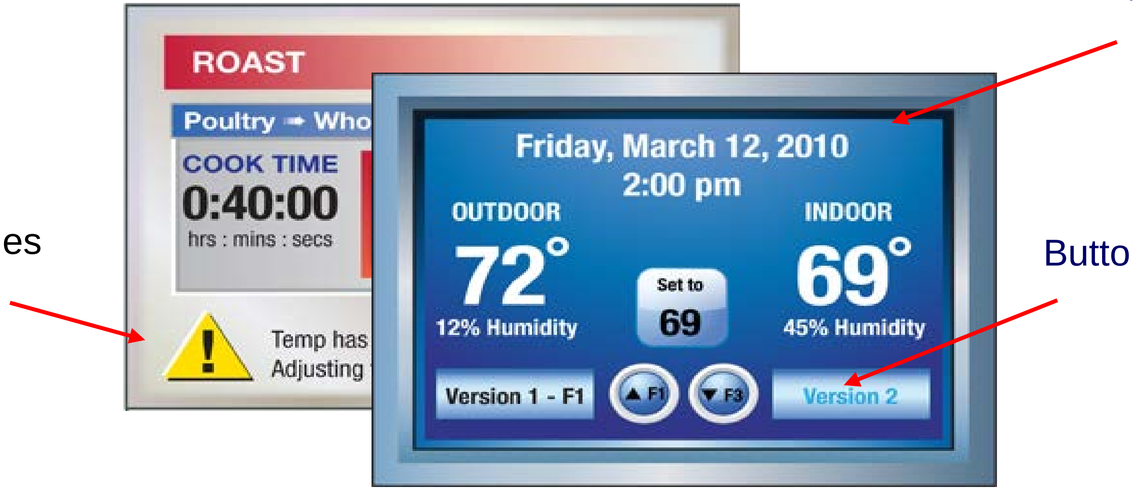
- Cost Effective
- Works on many platforms



# Microchip Graphics Library

Overview

# Graphics Library



Static Text, Multiple Fonts

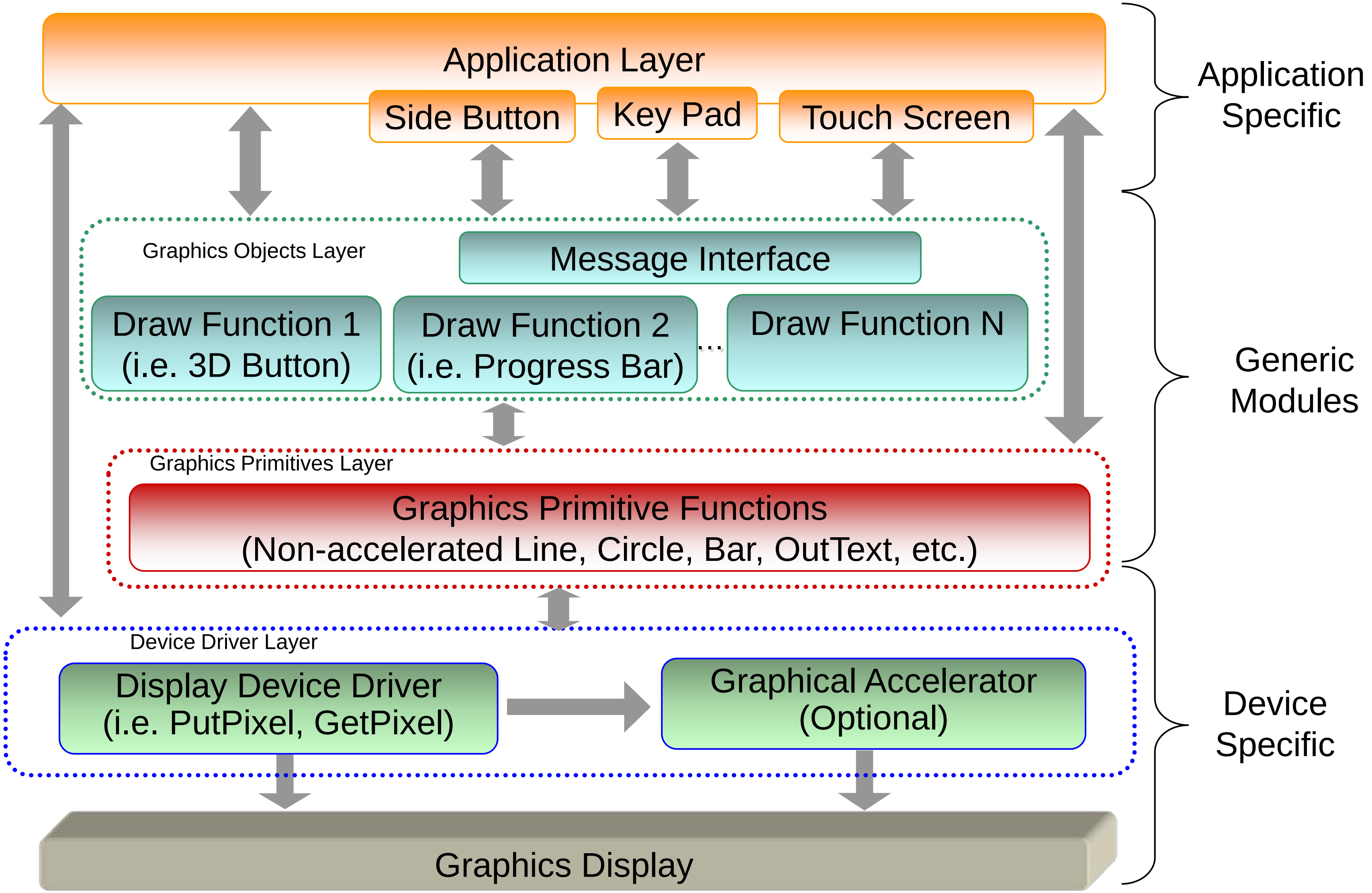
Images

Button

- Works with 16- and 32-bit PIC® MCUs
- Modular design – compile only what you need!
- Supports up to 24 bpp color depth
- Supports gradients, transparency, alpha blending, and transitions
- Demo source provided for our low-cost, full featured development tools
- Free to Microchip customers
  - Source code included
  - Multiple display controller drivers included

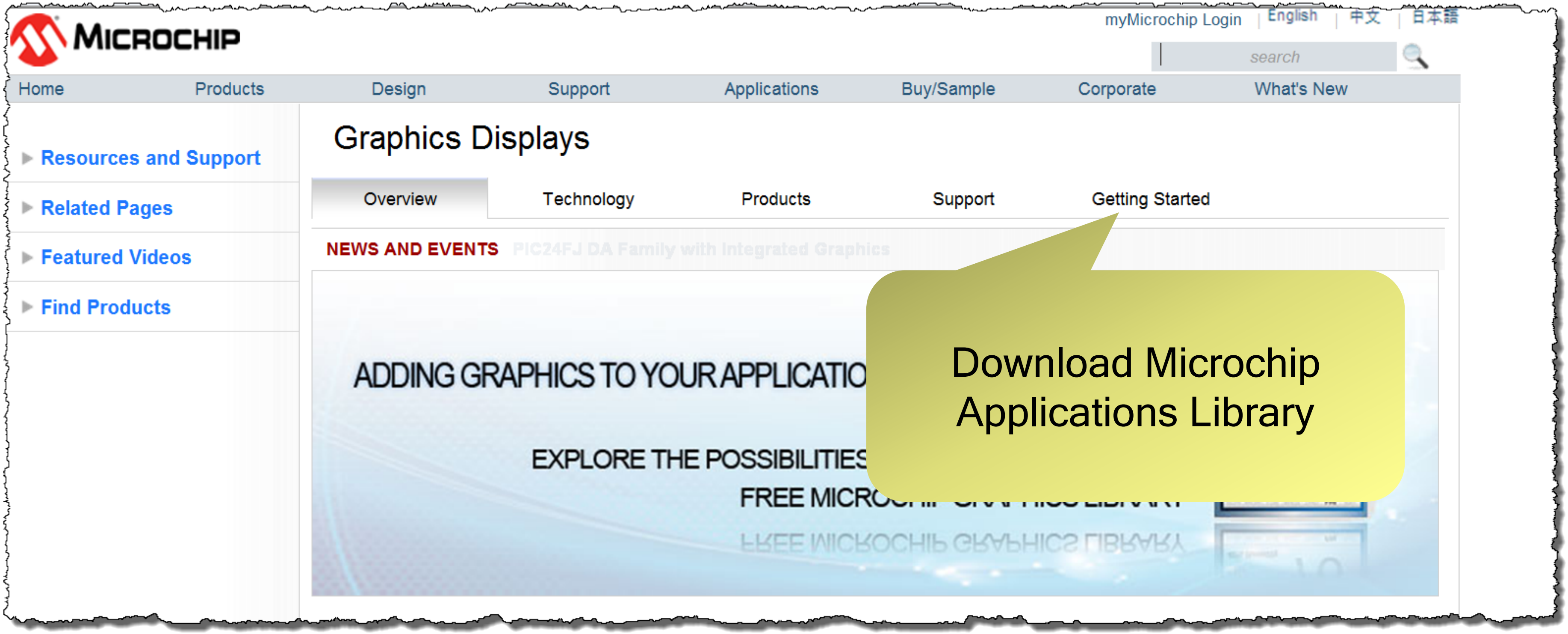
Regional Training Centers

# Library Overview


Application Layer
Side Button
Key Pad
Touch Screen
Graphics Objects Layer
Message Interface
Draw Function 1 (i.e. 3D Button)
Draw Function 2 (i.e. Progress Bar)
…
Draw Function N
Graphics Primitives Layer
Graphics Primitive Functions (Non-accelerated Line, Circle, Bar, OutText, etc.)
Device Driver Layer
Display Device Driver (i.e. PutPixel, GetPixel)
Graphical Accelerator (Optional)
Graphics Display
Application Specific
Generic Modules
Device Specific

# Graphics Design Center

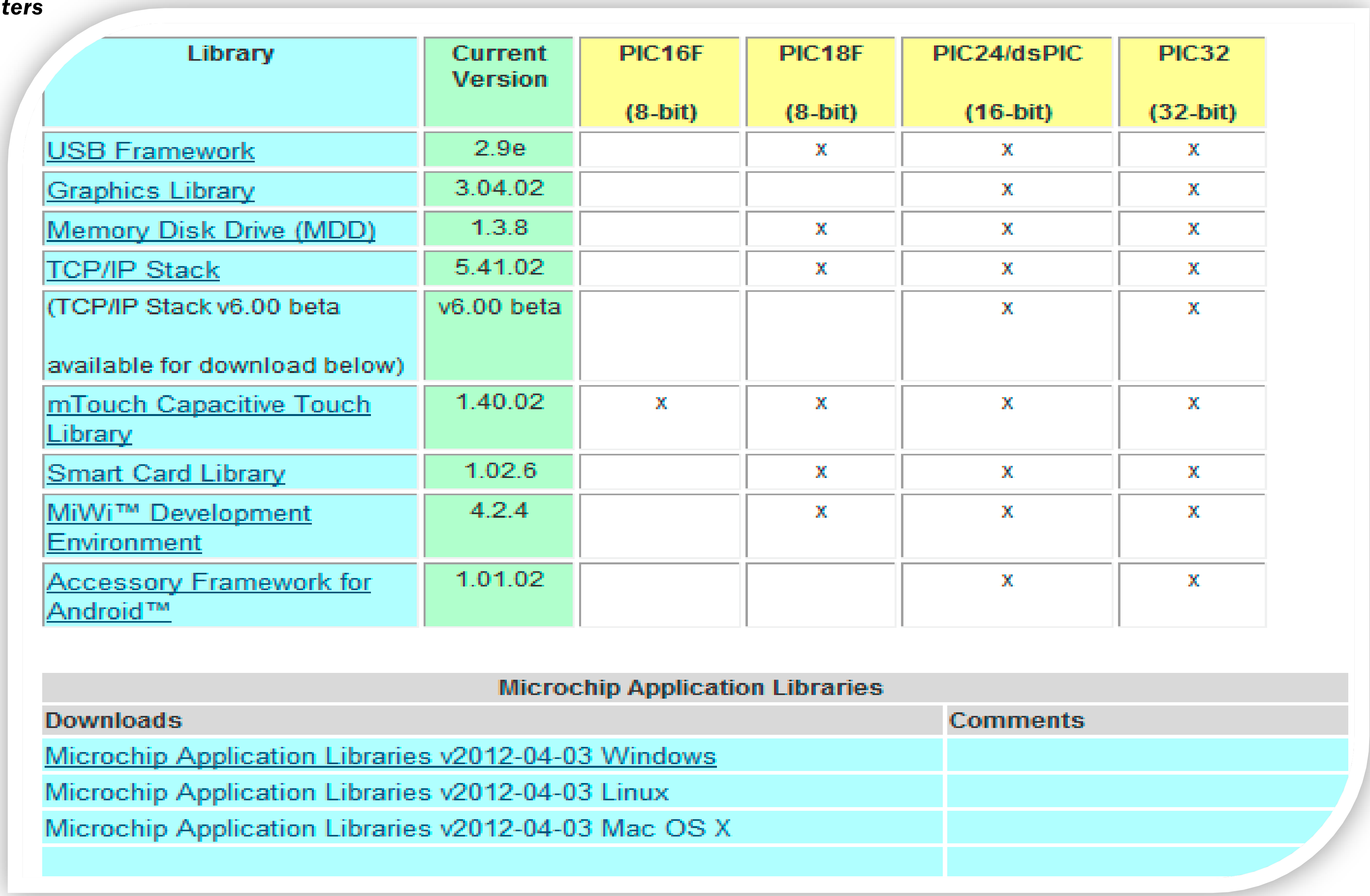
http://www.microchip.com/graphics



Download Microchip Applications Library

# Microchip Libraries for Applications

http://www.microchip.com/MLA

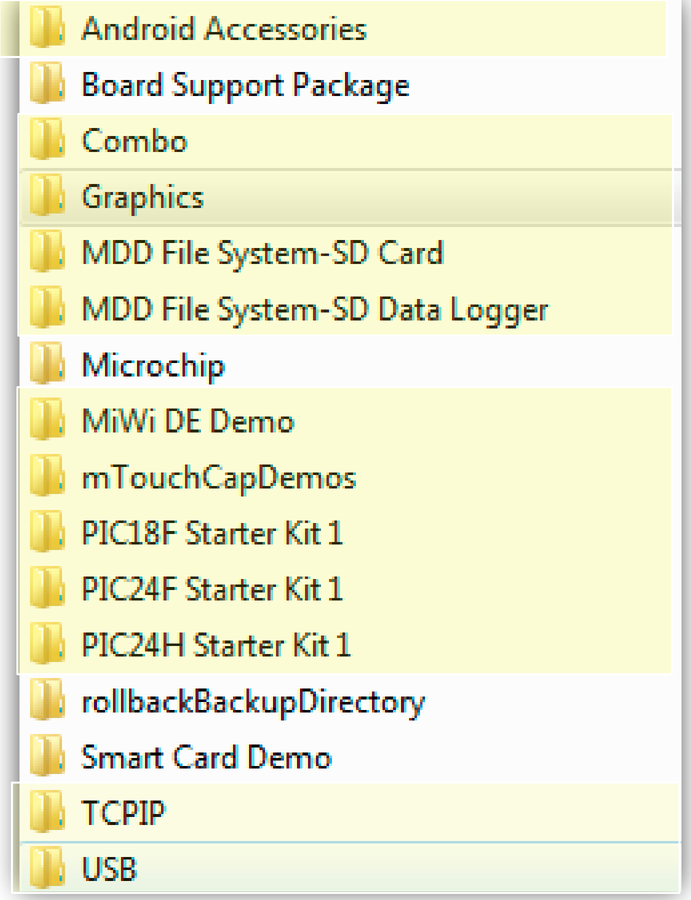


| Library | Current Version | PIC16F (8-bit) | PIC18F (8-bit) | PIC24/dsPIC (16-bit) | PIC32 (32-bit) |
|---|---|---|---|---|---|
| USB Framework | 2.9e | | x | x | x |
| Graphics Library | 3.04.02 | | | x | x |
| Memory Disk Drive (MDD) | 1.3.8 | | x | x | x |
| TCP/IP Stack | 5.41.02 | | x | x | x |
| (TCP/IP Stack v6.00 beta available for download below) | v6.00 beta | | | x | x |
| mTouch Capacitive Touch Library | 1.40.02 | x | x | x | x |
| Smart Card Library | 1.02.6 | | x | x | x |
| MiWi™ Development Environment | 4.2.4 | | x | x | x |
| Accessory Framework for Android™ | 1.01.02 | | | x | x |

| Microchip Application Libraries | |
|---|---|
| **Downloads** | **Comments** |
| Microchip Application Libraries v2012-04-03 Windows | |
| Microchip Application Libraries v2012-04-03 Linux | |
| Microchip Application Libraries v2012-04-03 Mac OS X | |
| | |

# Microchip Solutions Directory

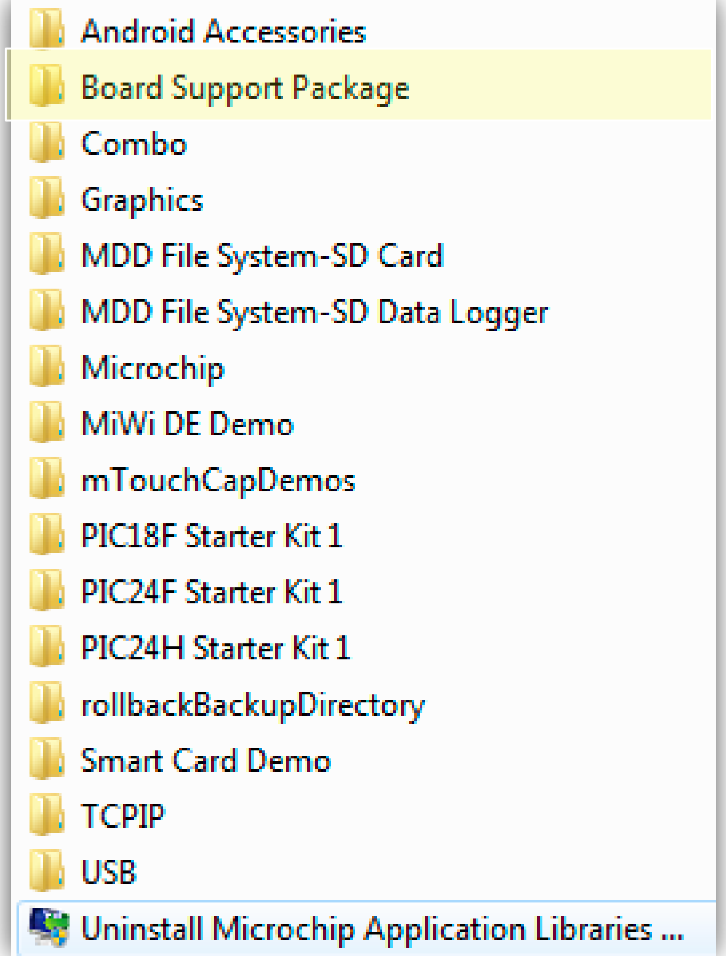
- Reference Design Projects
  - Full source code
- Find individual projects in these directories
- All paths relative to the Microchip directory



Android Accessories
Board Support Package
Combo
Graphics
MDD File System-SD Card
MDD File System-SD Data Logger
Microchip
MiWi DE Demo
mTouchCapDemos
PIC18F Starter Kit 1
PIC24F Starter Kit 1
PIC24H Starter Kit 1
rollbackBackupDirectory
Smart Card Demo
TCPIP
USB

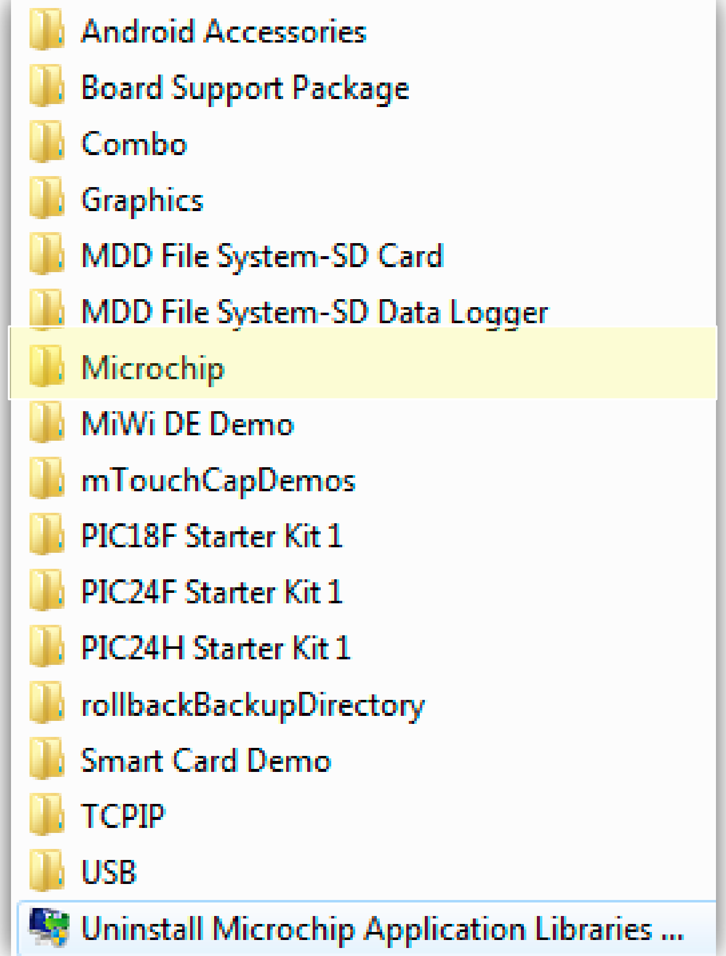
# Microchip Solutions
# Board Support

- Microchip development / evaluation board support
  - Touch screen
  - Memory
  - Potentiometer
  - Accelerometer
  - etc…


Android Accessories
Board Support Package
Combo
Graphics
MDD File System-SD Card
MDD File System-SD Data Logger
Microchip
MiWi DE Demo
mTouchCapDemos
PIC18F Starter Kit 1
PIC24F Starter Kit 1
PIC24H Starter Kit 1
rollbackBackupDirectory
Smart Card Demo
TCPIP
USB
Uninstall Microchip Application Libraries ...

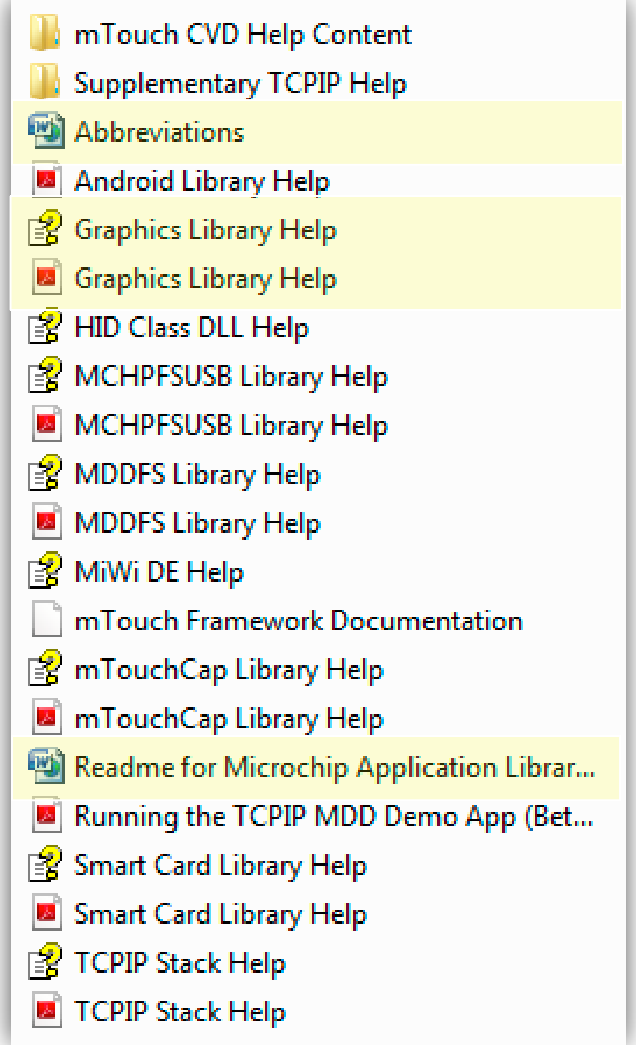
# Microchip Solutions Library Source Code

- Source files for all libraries
- Help files for all libraries



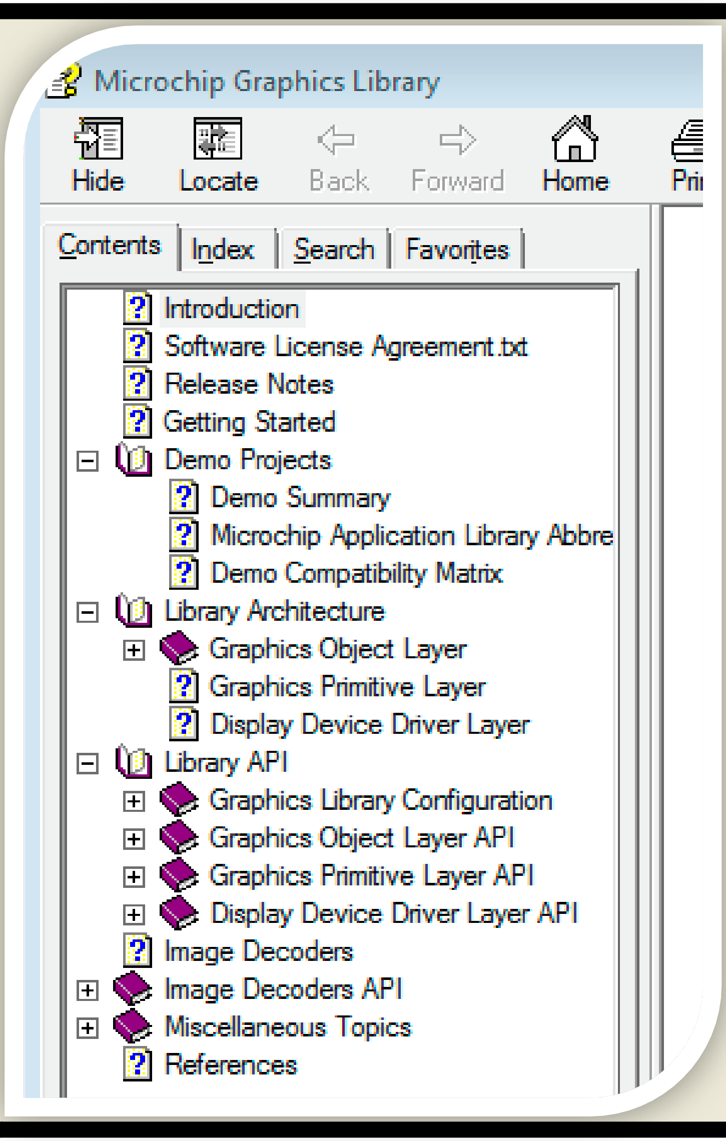
# Microchip Solutions
# Library Help Files

- ../Microchip/Help
- Abbreviations for the Microchip development boards used in demos
- Graphics Library APIs and demo user guides



mTouch CVD Help Content
Supplementary TCPIP Help
Abbreviations
Android Library Help
Graphics Library Help
Graphics Library Help
HID Class DLL Help
MCHPFSUSB Library Help
MCHPFSUSB Library Help
MDDFS Library Help
MDDFS Library Help
MiWi DE Help
mTouch Framework Documentation
mTouchCap Library Help
mTouchCap Library Help
Readme for Microchip Application Librar...
Running the TCPIP MDD Demo App (Bet...
Smart Card Library Help
Smart Card Library Help
TCPIP Stack Help
TCPIP Stack Help

# Library Help



- Introduction
- Software License Agreement.txt
- Release Notes
- Getting Started
- Demo Projects
  - Demo Summary
  - Microchip Application Library Abbre
  - Demo Compatibility Matrix
- Library Architecture
  - Graphics Object Layer
  - Graphics Primitive Layer
  - Display Device Driver Layer
- Library API
  - Graphics Library Configuration
  - Graphics Object Layer API
  - Graphics Primitive Layer API
  - Display Device Driver Layer API
- Image Decoders
- Image Decoders API
- Miscellaneous Topics
- References

Help files are included as part of the Microchip Graphics Library installation and are located in the following directory:

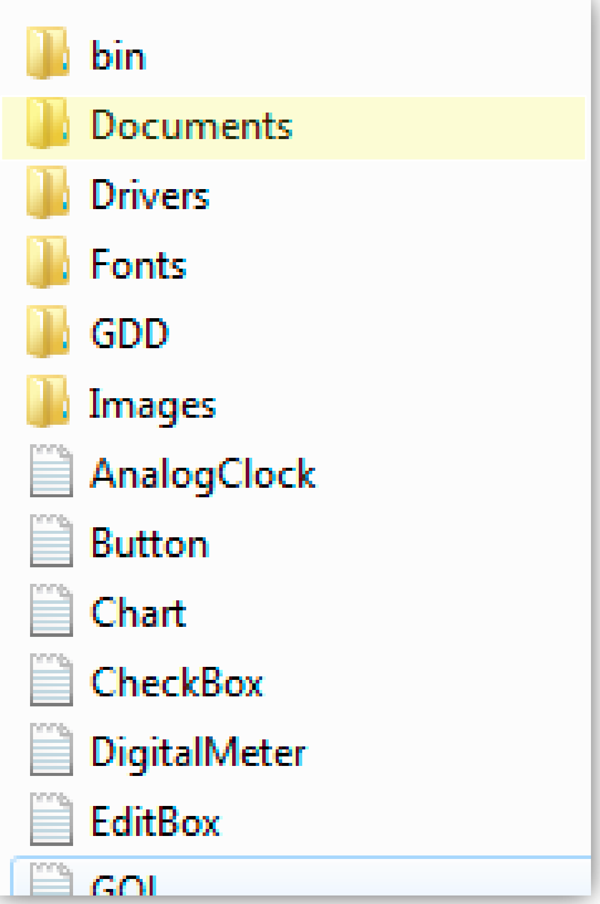
..\Microchip\Help

Graphics Library Help.chm

Graphics Library Help.pdf

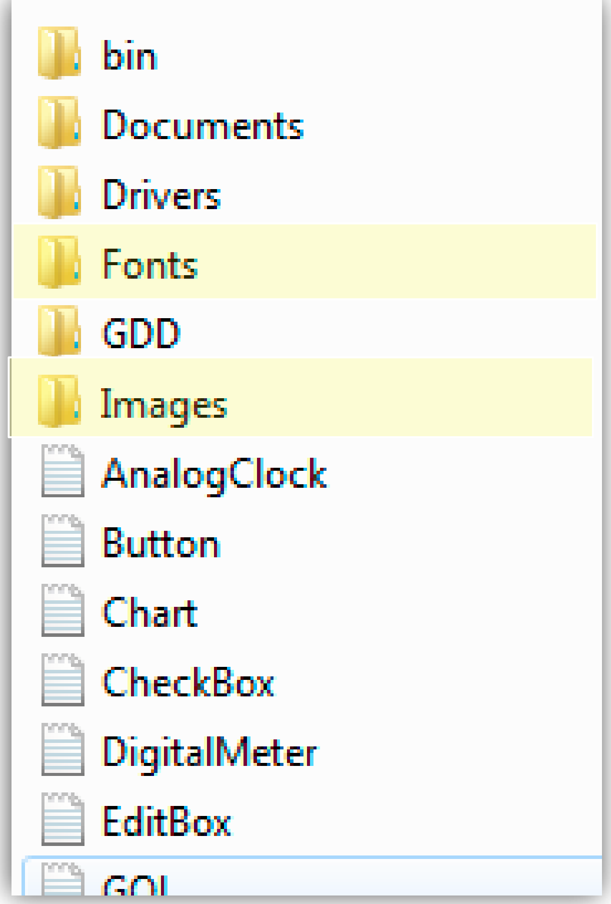
# Microchip Solutions Graphics Library

- ../Microchip/Graphics
- Schematics for the Microchip Graphics development boards
- User’s guides for Microchip Graphics demo projects


bin
Documents
Drivers
Fonts
GDD
Images
AnalogClock
Button
Chart
CheckBox
DigitalMeter
EditBox
GOL

# Microchip Solutions Graphics Library

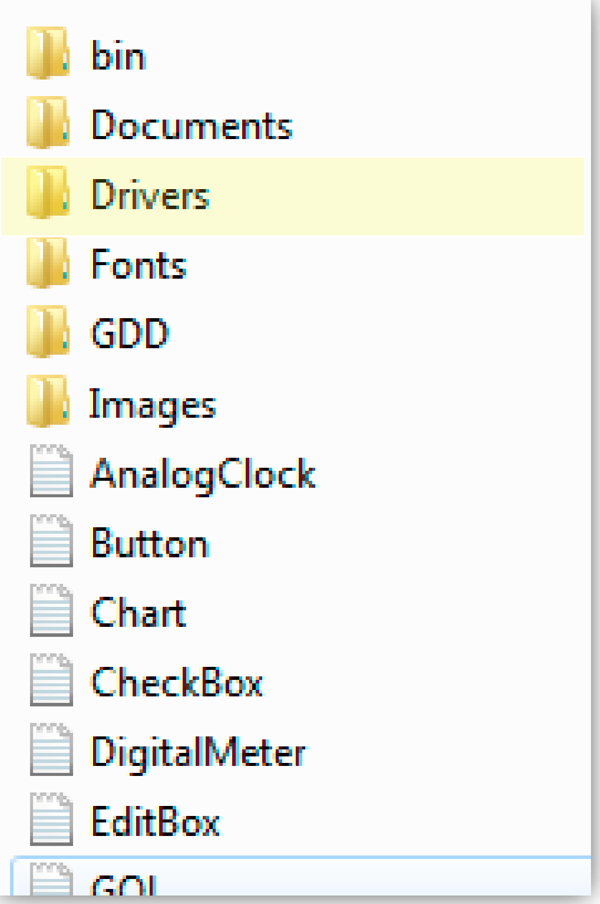
- ../Microchip/Graphics
- Free fonts
- Free images and icons



bin
Documents
Drivers
Fonts
GDD
Images
AnalogClock
Button
Chart
CheckBox
DigitalMeter
EditBox
GOL

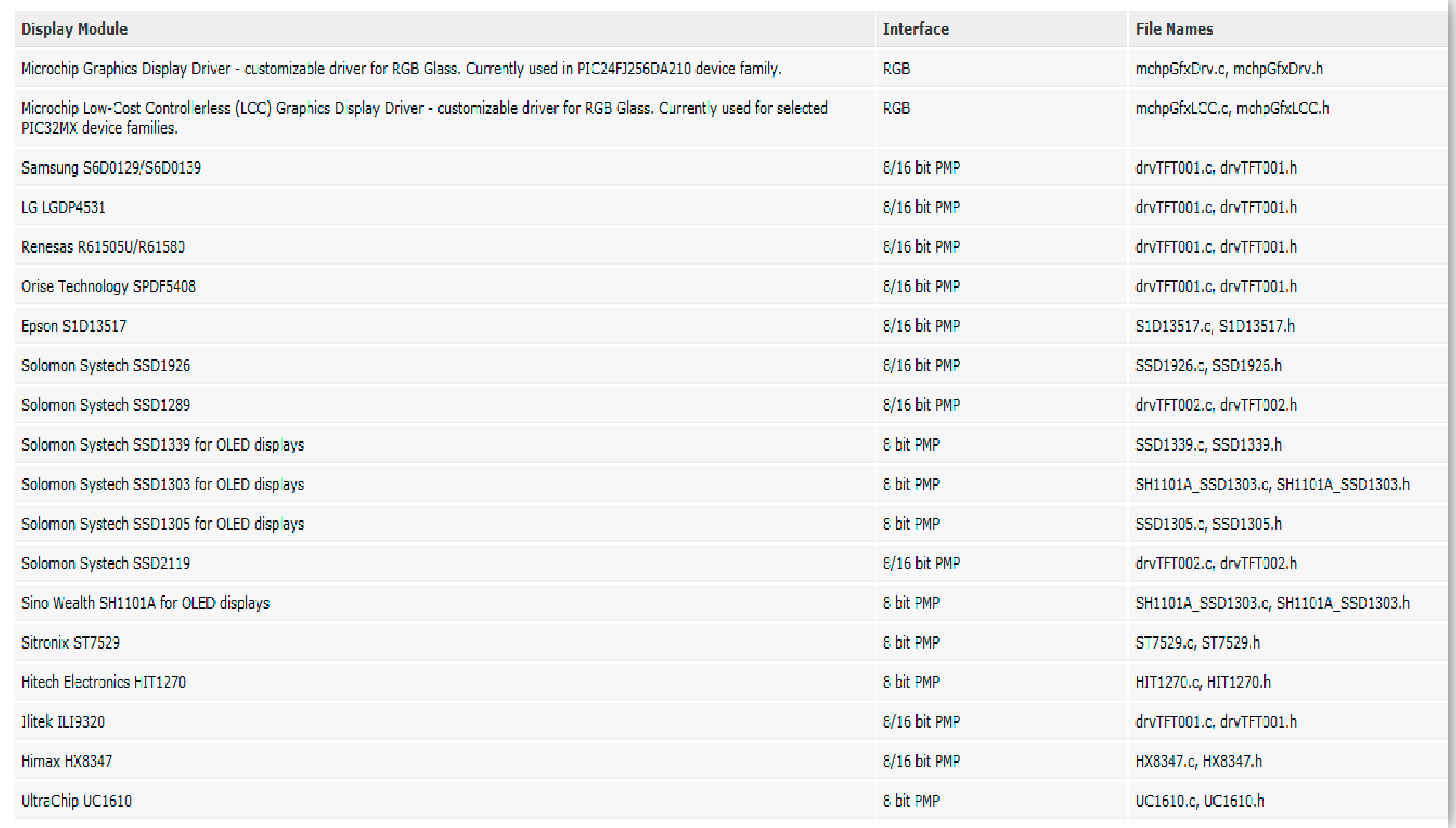
# Microchip Solutions Graphics Library

- ../Microchip/Graphics
- ‘C’ source files for graphics (LCD) controllers

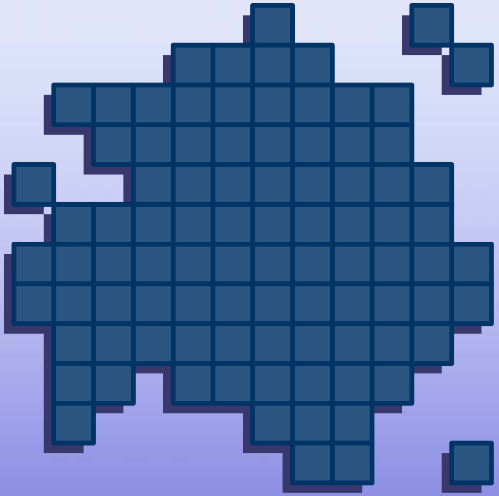


# Microchip Graphics Library

## Help -> Release Notes



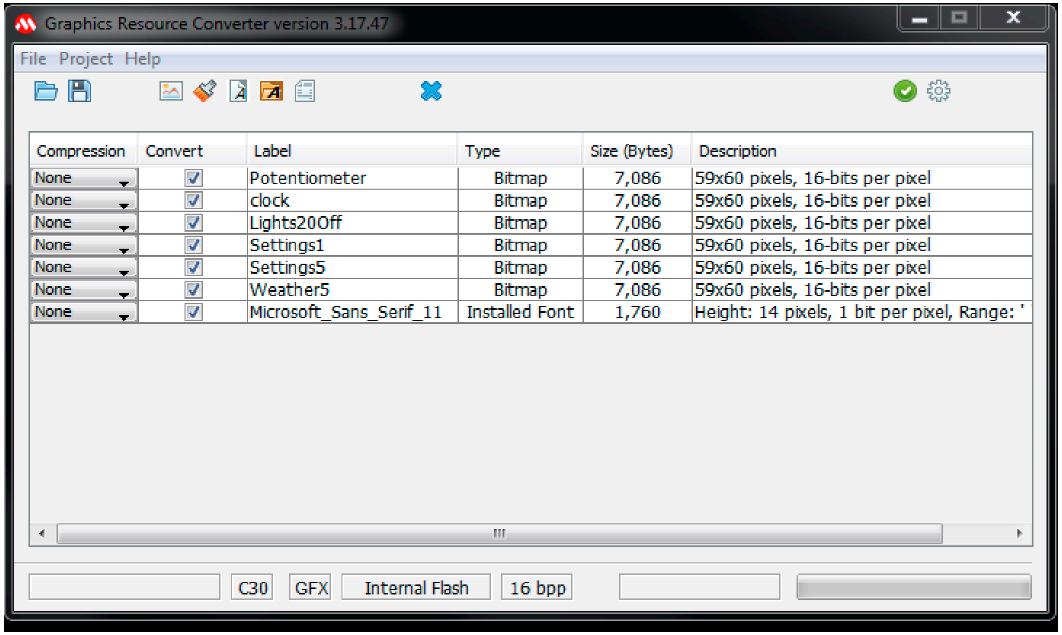
| Display Module | Interface | File Names |
|---|---|---|
| Microchip Graphics Display Driver - customizable driver for RGB Glass. Currently used in PIC24FJ256DA210 device family. | RGB | mchpGfxDrv.c, mchpGfxDrv.h |
| Microchip Low-Cost Controllerless (LCC) Graphics Display Driver - customizable driver for RGB Glass. Currently used for selected PIC32MX device families. | RGB | mchpGfxLCC.c, mchpGfxLCC.h |
| Samsung S6D0129/S6D0139 | 8/16 bit PMP | drvTFT001.c, drvTFT001.h |
| LG LGDP4531 | 8/16 bit PMP | drvTFT001.c, drvTFT001.h |
| Renesas R61505U/R61580 | 8/16 bit PMP | drvTFT001.c, drvTFT001.h |
| Orise Technology SPDF5408 | 8/16 bit PMP | drvTFT001.c, drvTFT001.h |
| Epson S1D13517 | 8/16 bit PMP | S1D13517.c, S1D13517.h |
| Solomon Systech SSD1926 | 8/16 bit PMP | SSD1926.c, SSD1926.h |
| Solomon Systech SSD1289 | 8/16 bit PMP | drvTFT002.c, drvTFT002.h |
| Solomon Systech SSD1339 for OLED displays | 8 bit PMP | SSD1339.c, SSD1339.h |
| Solomon Systech SSD1303 for OLED displays | 8 bit PMP | SH1101A_SSD1303.c, SH1101A_SSD1303.h |
| Solomon Systech SSD1305 for OLED displays | 8 bit PMP | SSD1305.c, SSD1305.h |
| Solomon Systech SSD2119 | 8/16 bit PMP | drvTFT002.c, drvTFT002.h |
| Sino Wealth SH1101A for OLED displays | 8 bit PMP | SH1101A_SSD1303.c, SH1101A_SSD1303.h |
| Sitronix ST7529 | 8 bit PMP | ST7529.c, ST7529.h |
| Hitech Electronics HIT1270 | 8 bit PMP | HIT1270.c, HIT1270.h |
| Ilitek ILI9320 | 8/16 bit PMP | drvTFT001.c, drvTFT001.h |
| Himax HX8347 | 8/16 bit PMP | HX8347.c, HX8347.h |
| UltraChip UC1610 | 8 bit PMP | UC1610.c, UC1610.h |



# Microchip Graphics Library

## Design Utilities

# Graphics Resource Converter

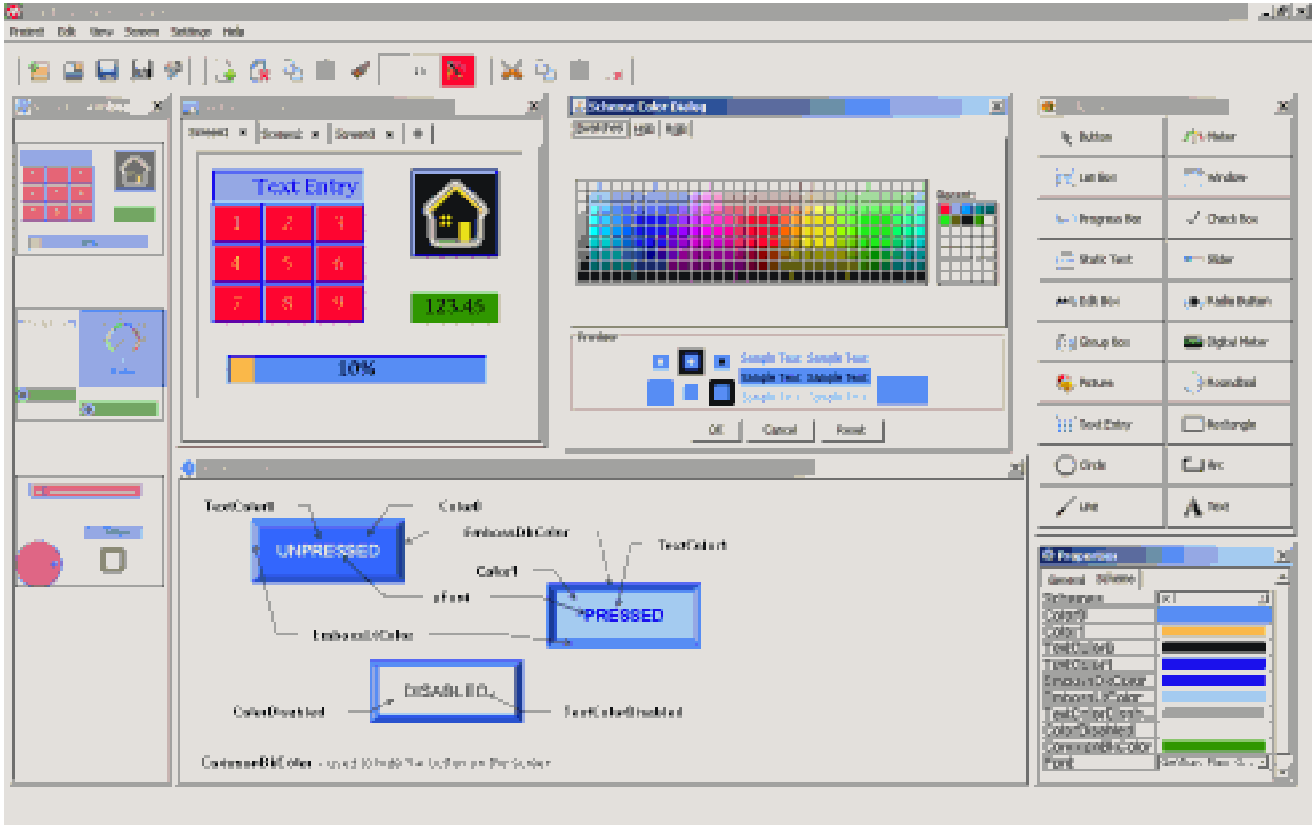


| Compression | Convert | Label | Type | Size (Bytes) | Description |
|---|---|---|---|---|---|
| None | ☑ | Potentiometer | Bitmap | 7,086 | 59x60 pixels, 16-bits per pixel |
| None | ☑ | clock | Bitmap | 7,086 | 59x60 pixels, 16-bits per pixel |
| None | ☑ | Lights2OOff | Bitmap | 7,086 | 59x60 pixels, 16-bits per pixel |
| None | ☑ | Settings1 | Bitmap | 7,086 | 59x60 pixels, 16-bits per pixel |
| None | ☑ | Settings5 | Bitmap | 7,086 | 59x60 pixels, 16-bits per pixel |
| None | ☑ | Weather5 | Bitmap | 7,086 | 59x60 pixels, 16-bits per pixel |
| None | ☑ | Microsoft_Sans_Serif_11 | Installed Font | 1,760 | Height: 14 pixels, 1 bit per pixel, Range: ' |

C30 GFX Internal Flash 16 bpp

- Converts graphical resources (images, fonts, binary files) into formats the Microchip Graphics Library can use
  - Bitmaps and fonts optimized for PIC® microcontrollers
  - JPEG encoding maintained
- Generate color palettes for use with Microchip Graphics Module

# Graphical Display Designer

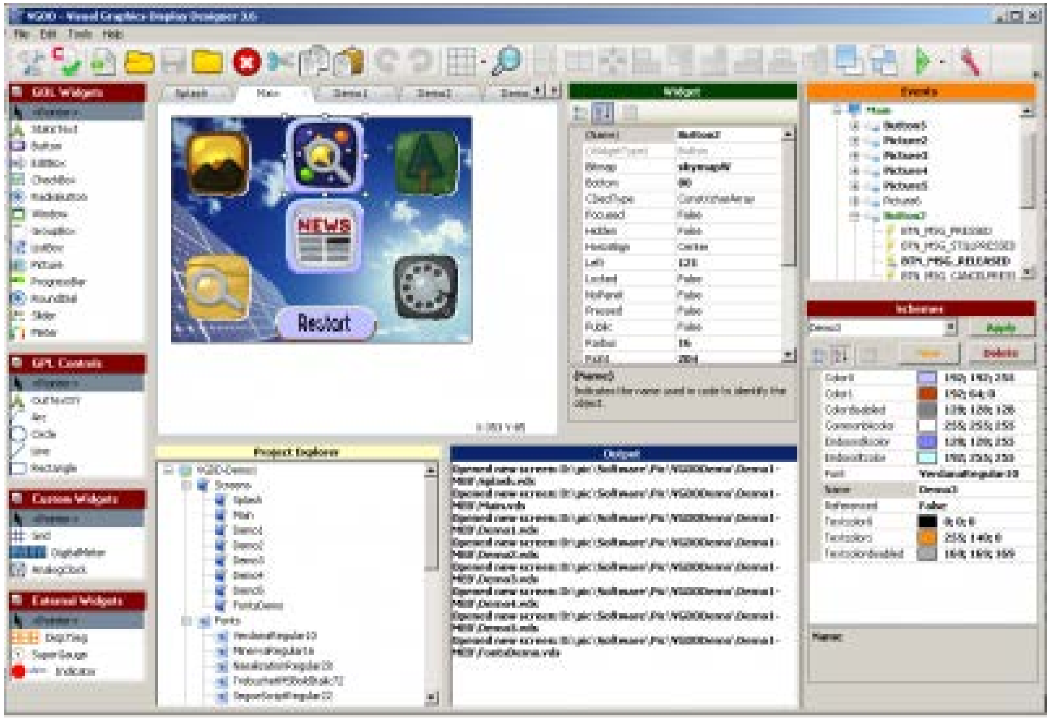


- WYSIWYG GUI Design utility with code generation
- Project Handling
- Stand alone or MPLAB® X IDE plug-in (Windows, Linux, Mac)

# Third Party- VGDD
# Visual Graphics Display Designer

http://virtualfab.it/mediawiki/index.php VGDD:Visual_Graphis_Display_Designer

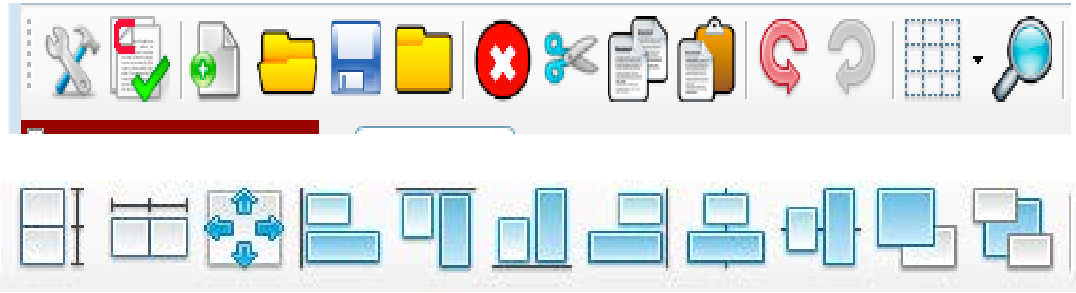
HIGH FUNCTION
LOW COST



- WYSIWYG GUI Design utility with code generation
  - Uses Microchip Graphics Library
- Project Generation using integrated MPLAB® X IDE Wizard
- Stand alone program (Windows only)

# Visual Graphics Display Designer Features

- Provides visual screen creation
  - WYSIWYG results using Microchip Graphics Library
  - Integrated resource conversion for fonts and images
- Graphics system hardware configuration
  - Must use library supported LCD controller and display drivers
- C Code generation
  - Application templates with all callback functions
  - Required hardware set up and drivers included
- Editing features
  - Cut, copy and paste
  - Alignment guides
  - Snap to grid
  - Undo / Redo engine



# Visual Graphics Display Designer Features

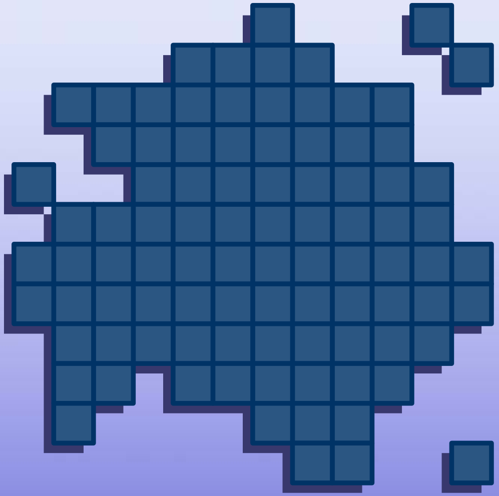
- Generate MPLAB® X IDE project structure
  - Application code template provided
  - All required library files populated into the project
  - Project properties set (heap allocation, memory optimizations, etc)
- Import / Export screen files
  - Allows reuse in multiple application
- Customization support
  - Design with your own colors
  - Select your own fonts and images
  - Support for user defined widgets
- Widget event Handling
  - Uses Microchip defined states and callback functions
  - Screen navigation using state machines
- Screen simulation
  - Player integrated with full version of VGDD
  - Visually see the GUI screen movement

# Microchip Graphics Library

## Basics

# Terminology

- **Application Programming Interface (API):** A set of functions that can be called from an application to access features of another program or library.
- **Graphics Primitive:** An elementary graphics building block such as a point, line, arc, etc.
- **Graphics Object:** Any of the various shapes (e.g. buttons, charts, dials, etc) your program can render to the screen and control. These objects can be used to provide user input to your system.
- **Widget:** Another name for a graphics object.
- **Glyph:** A graphical representation, in a particular typeface, of all the individual symbols of any of the world's writing systems (e.g. alphabetic letters, Chinese characters, numerals, punctuation marks, etc.).



# Microchip Graphics Library

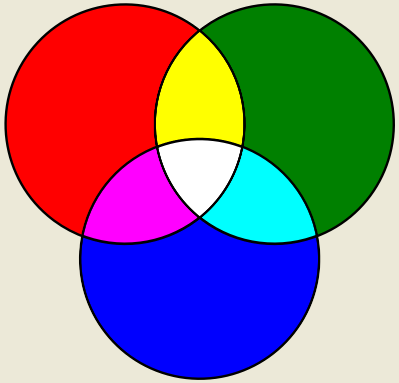
## Basics:
## Images and Color

# RGB Color Model

## Definition

The RGB Color Model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. Colors are expressed as a triplet (RGB) Source: Wikipedia

- 16-bit (65,536 colors)
- Red: 5 bits
- Green: 6 bits
- Blue: 5 bits
- RGB 565



- 18-bit (262,144 colors)
- Red: 6 bits
- Green: 6 bits
- Blue: 6 bits
- RGB 666

- 24-bit (1.678M colors)
- Red: 8 bits
- Green: 8 bits
- Blue: 8 bits
- RGB 888

# Color Depth

## Definition

Color Depth is the number of bits used to represent the color of a single pixel in an image or in a frame buffer. Color Depth is usually specified using a bpp (bits per pixel) notation. A higher color depth gives a broader range of distinct colors, but also requires more memory to store the image or frame.

- True Color => 24 bpp (16,777,216 colors)
- High Color => 16 bpp (65,536 colors)
- Indexed Color => Use of a color lookup table
  - Palette stored as part of image file
  - Required for images 8 bpp or less
- LCD System specifications – color depth refers to the capacity of the hardware
- Image Files – color depth impacts memory size

# Frame Buffer Sizes

- PIC24 options up to 96KB RAM
- PIC32 options up to 128KB RAM
- Use external SRAM for frame buffers > 128KB

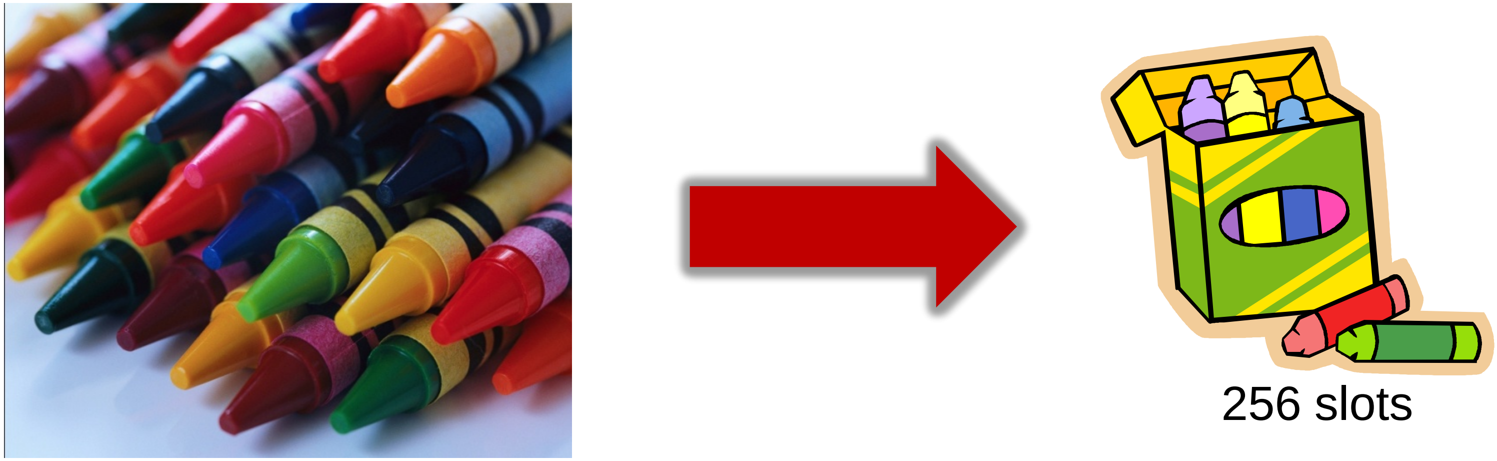
| Display Resolution Typical Sizes | | | Color Depth/ Memory Requirement in (Bytes) | | | | |
|---|---|---|---|---|---|---|---|
| | | | **1 bpp (Mono)** | **2 bpp (4 shades)** | **4 bpp (16 shades)** | **8 bpp (256 colors)** | **16 bpp (65K colors)** |
| WVGA | 800x480 | 7” | 🟩 48,000 | 🟩 96,000 | 🟦 192,000 | 🟦 384,000 | 🟦 768,000 |
| VGA | 640x480 | 5.7” | 🟩 38,400 | 🟩 76,800 | 🟦 153,600 | 🟦 307,200 | 🟦 614,400 |
| WQVGA | 480x272 | 4.3” | 🟩 16,320 | 🟩 32,640 | 🟩 65,280 | 🟦 130,560 | 🟦 261,120 |
| QVGA | 320x240 | 3.2” | 🟩 9,600 | 🟩 19,200 | 🟩 38,400 | 🟩 76,800 | 🟦 153,600 |
| Common for OLED | 128x64 | 1”-2.7” | 🟩 1,024 | 🟩 2,048 | 🟩 4,096 | 🟩 8,192 | 🟩 16,384 |

🟩 **Internal SRAM on PIC24 DA or PIC32**

🟦 **External SRAM**

# Color Lookup Table (Indexed Color)

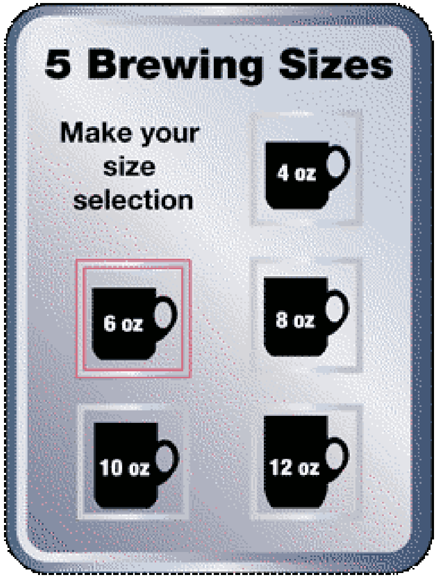
- Converts the indexed color numbers stored in each pixel of frame buffer into physical colors
- Microchip’s PIC24 DA family allows 256 entries of 16 bpp colors



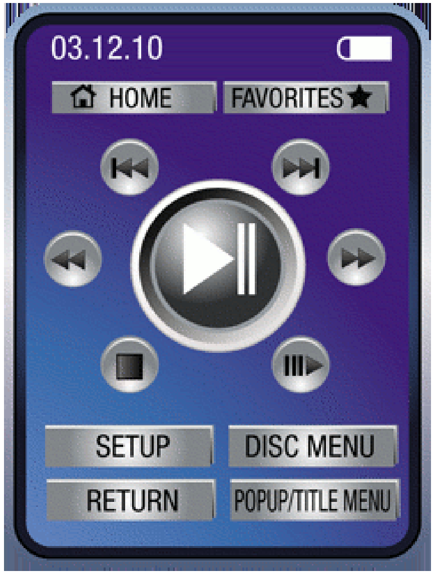
256 slots

65,536 colors

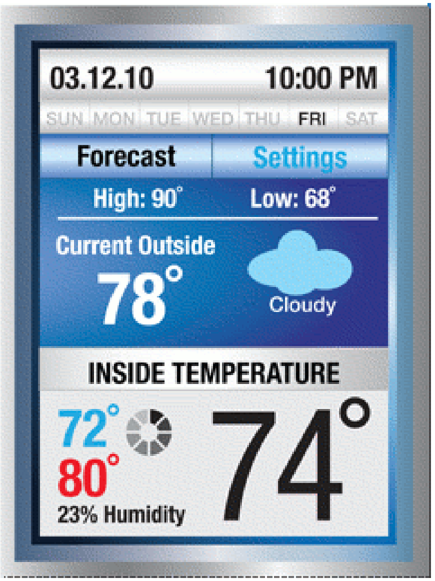
# Guess How Many Colors?



*16 Shades – 4bpp*
*320x240 Resolution*
*37.5K byte per frame*



*256 Colors – 8bpp*
*320x240 Resolution*
*75K byte per frame*



*256 Colors – 8bpp*
*320x240 Resolution*
*75K byte per frame*

# PIC24F DA Color Lookup Table

- Dynamic switching of different color palettes
- Each palette can utilize a different set of 256 colors
- Every screen update may have a different color palette, supporting different set of 256 colors

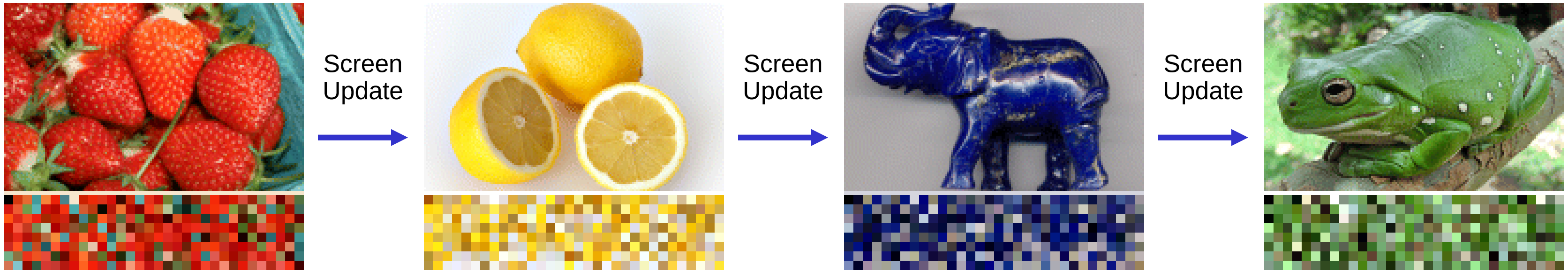


Image #1
(256 colors)

Image #2
(256 colors)

Image #3
(256 colors)

…

Image #N
(256 colors)

Color Lookup Table
(Palette #1)

Color Lookup Table
(Palette #2)

Color Lookup Table
(Palette #3)

…

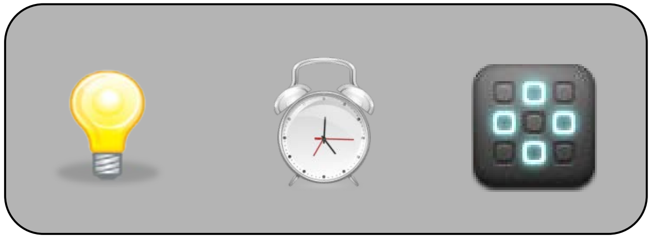
Color Lookup Table
(Palette #N)

# Image Support

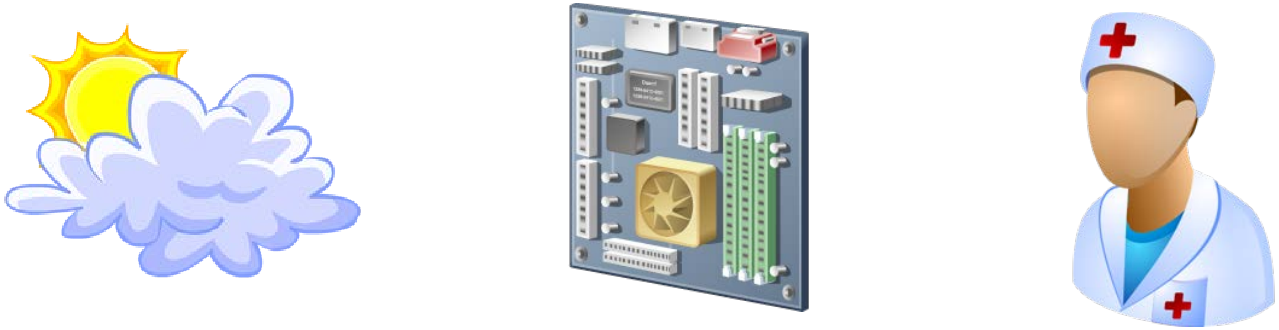
- Microchip Graphics Library supports both bitmap and jpeg formats
  - Color depth up to 24 bpp
- Images are converted using Graphics Resource Converter utility
  - May be compressed using run length encoding (RLE)
  - OR … Fixed Huffman if PIC24xxxDAxxx's Inflate Processing Unit (IPU) is used

# How are images used?

- Display company logos
- Create icons



- Provide user information (indicators)



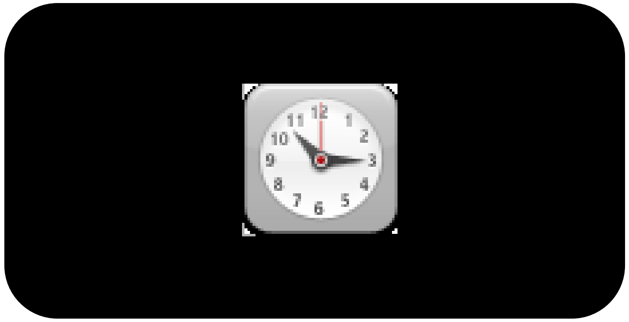
# How are images used?

- Create screen backgrounds

# Design considerations

- Image size color depth impact memory use
- Example:

Image Size: 59x60 pixels
Color Depth: 32 bpp



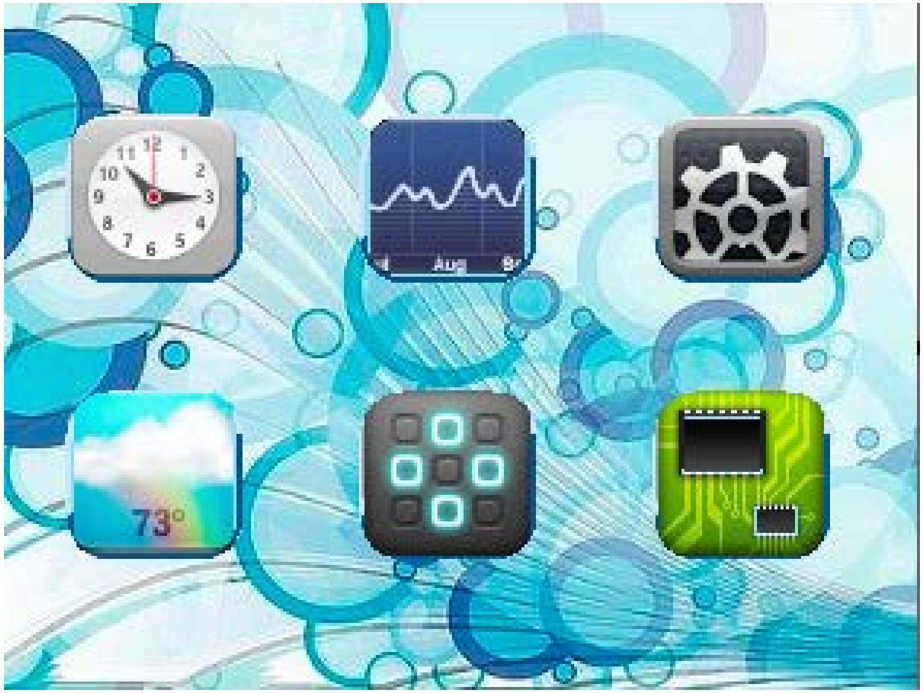
Requires:
(59x60x32)/8 = 14K bytes

Image Size: 59x60 pixels
Color Depth: 16 bpp

Requires:
(59x60x16)/8 = 7K bytes

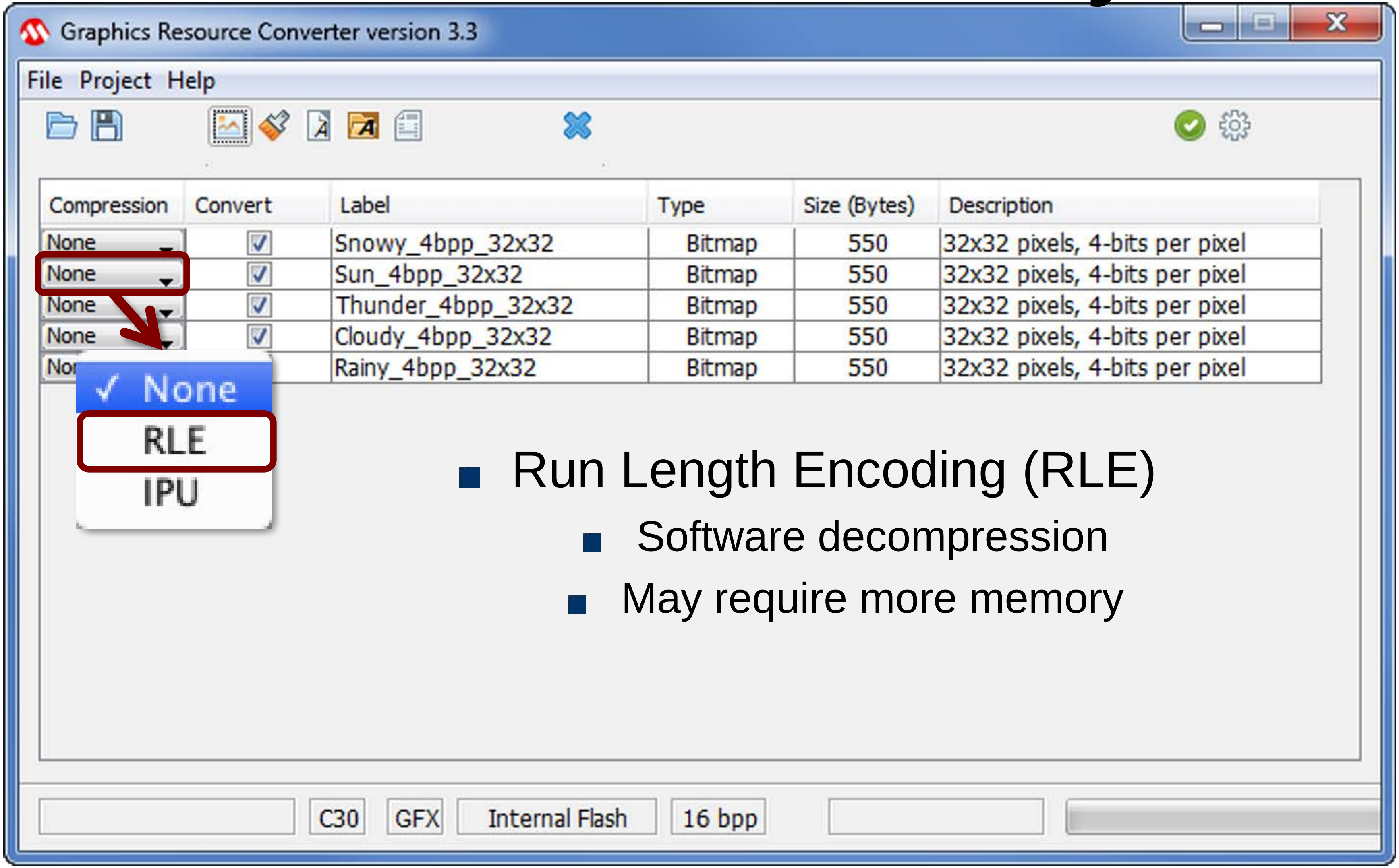
# Design Considerations

- Image size and color depth impact memory requirements
- Guess how much memory needed for this screen?



- ANSWER:  As drawn, this screen requires ~194,000 bytes

# Graphics Resource Converter Reduce Memory

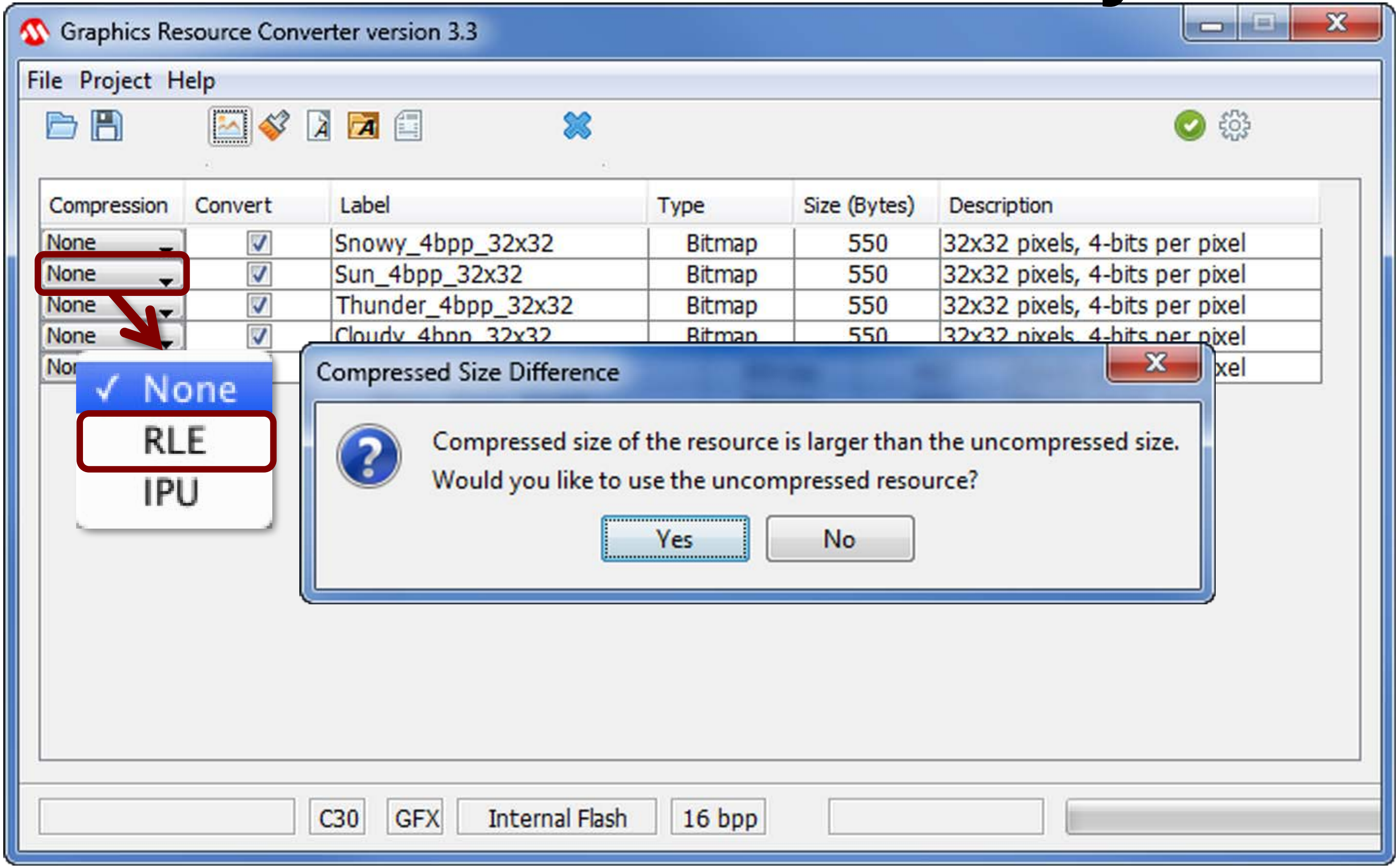


| Compression | Convert | Label | Type | Size (Bytes) | Description |
|---|---|---|---|---|---|
| None | ☑ | Snowy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Sun_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Thunder_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Cloudy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| Nor |  | Rainy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |

C30 | GFX | Internal Flash | 16 bpp

- Run Length Encoding (RLE)
  - Software decompression
  - May require more memory

# Graphics Resource Converter Reduce Memory

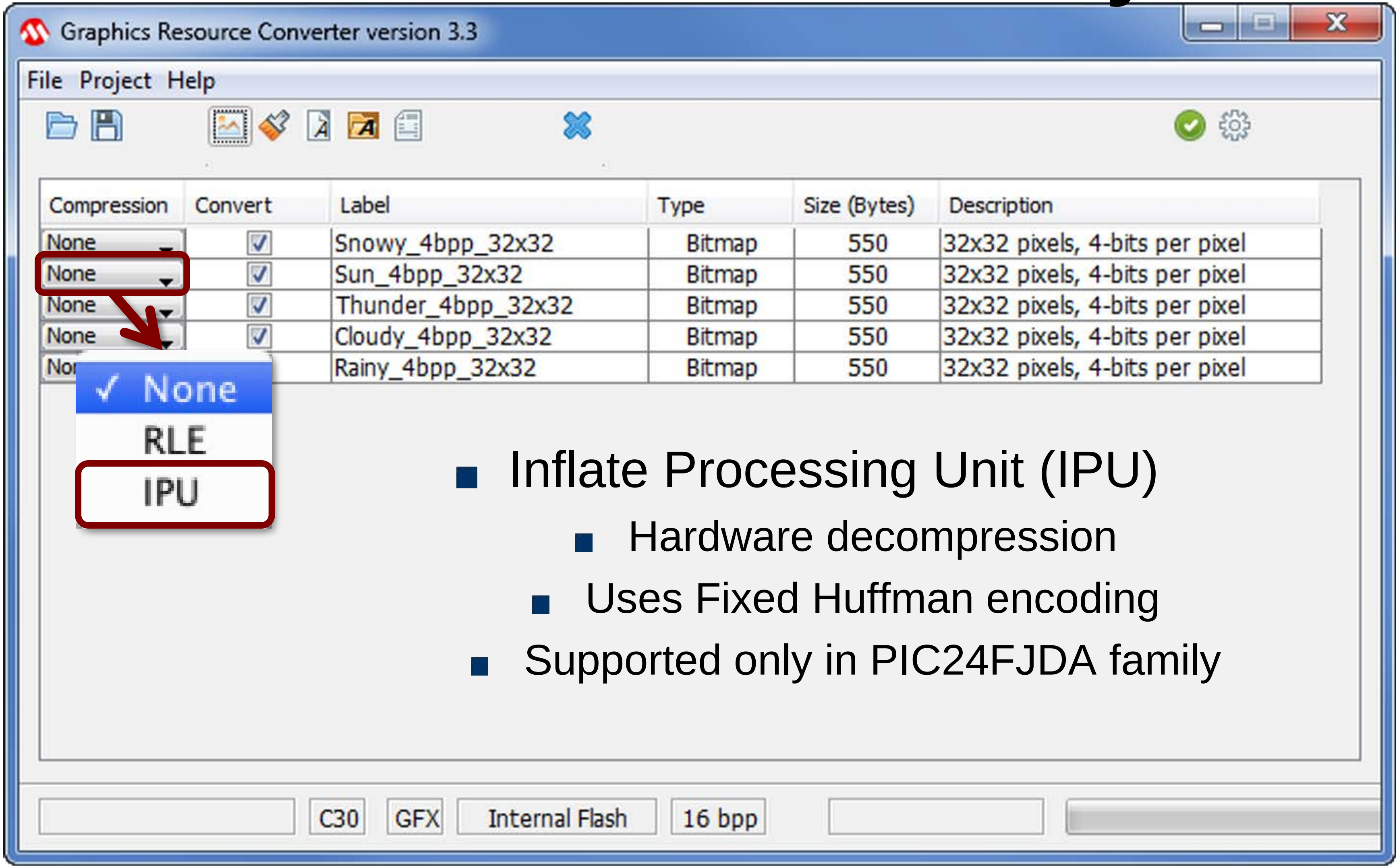

Graphics Resource Converter version 3.3
File Project Help
✓ None
RLE
IPU
Compressed Size Difference
Yes
No

| Compression | Convert | Label | Type | Size (Bytes) | Description |
|---|---|---|---|---|---|
| None | ☑ | Snowy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Sun_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Thunder_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Cloudy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| Nor | | | | | xel |

Compressed size of the resource is larger than the uncompressed size.
Would you like to use the uncompressed resource?

C30 | GFX | Internal Flash | 16 bpp

# Graphics Resource Converter Reduce Memory

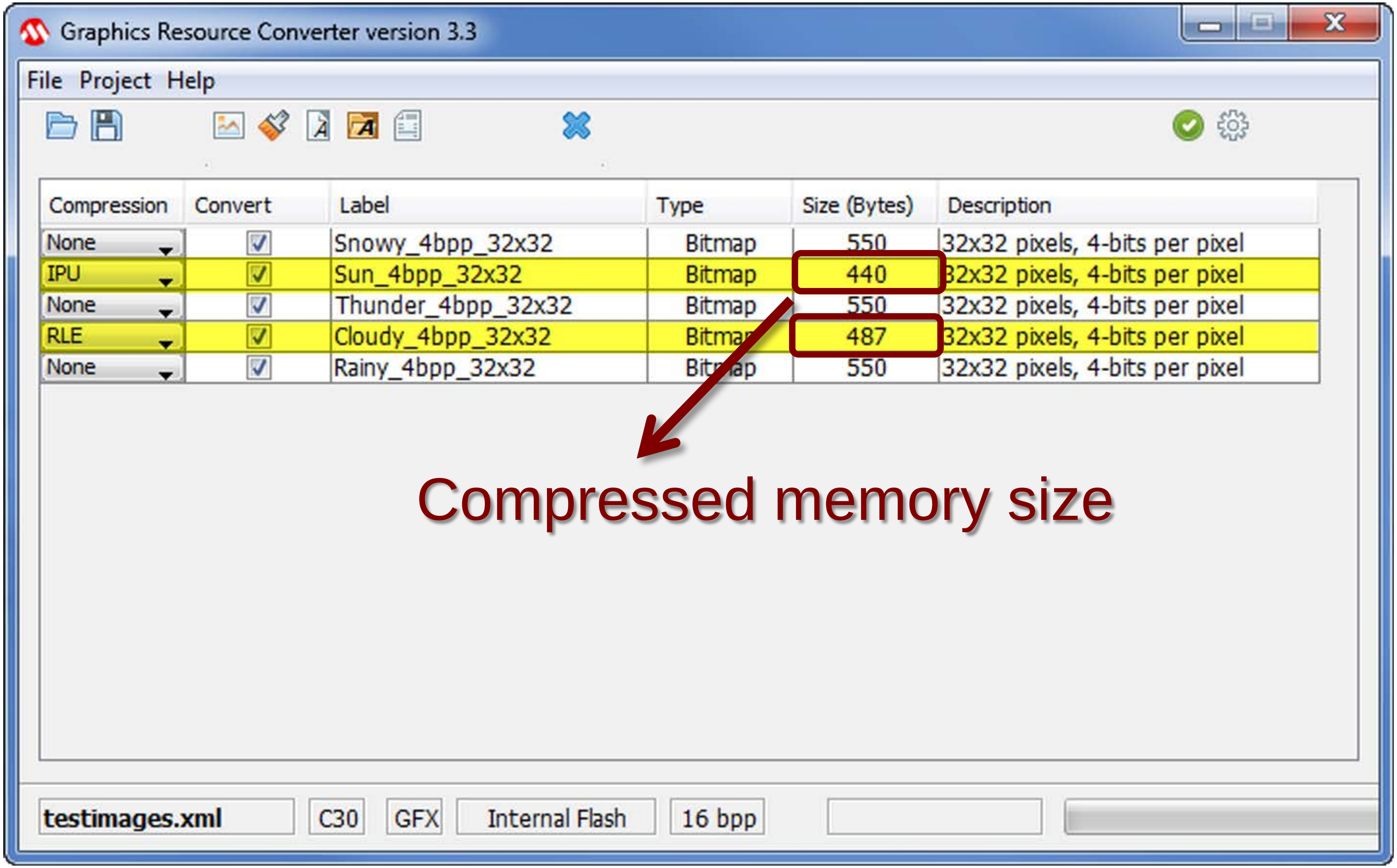


| Compression | Convert | Label | Type | Size (Bytes) | Description |
|---|---|---|---|---|---|
| None | ☑ | Snowy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Sun_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Thunder_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Cloudy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| No |  | Rainy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |

C30 | GFX | Internal Flash | 16 bpp

- Inflate Processing Unit (IPU)
  - Hardware decompression
  - Uses Fixed Huffman encoding
  - Supported only in PIC24FJDA family

# Graphics Resource Converter Reduce Memory

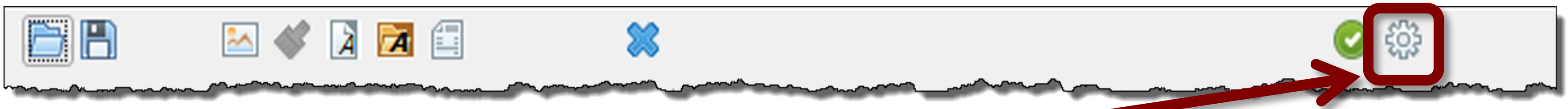


| Compression | Convert | Label | Type | Size (Bytes) | Description |
|---|---|---|---|---|---|
| None | ☑ | Snowy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| IPU | ☑ | Sun_4bpp_32x32 | Bitmap | 440 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Thunder_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |
| RLE | ☑ | Cloudy_4bpp_32x32 | Bitmap | 487 | 32x32 pixels, 4-bits per pixel |
| None | ☑ | Rainy_4bpp_32x32 | Bitmap | 550 | 32x32 pixels, 4-bits per pixel |

testimages.xml | C30 | GFX | Internal Flash | 16 bpp

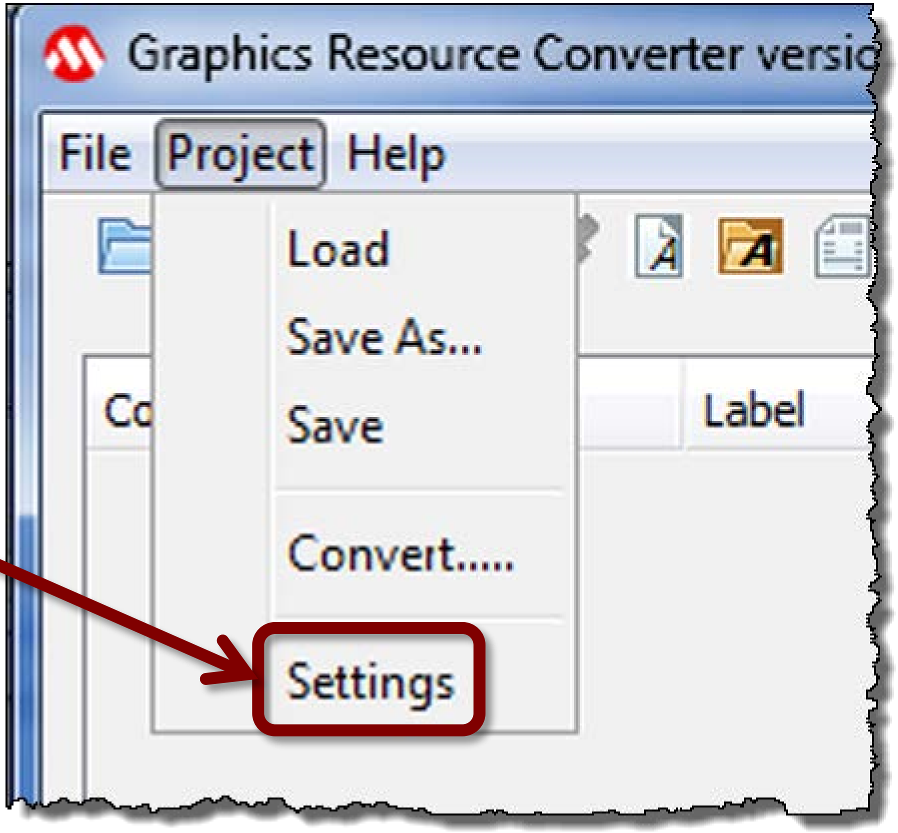
Compressed memory size

# Graphics Resource Converter Project Settings



Either click toolbar icon OR…

Select from Project menu

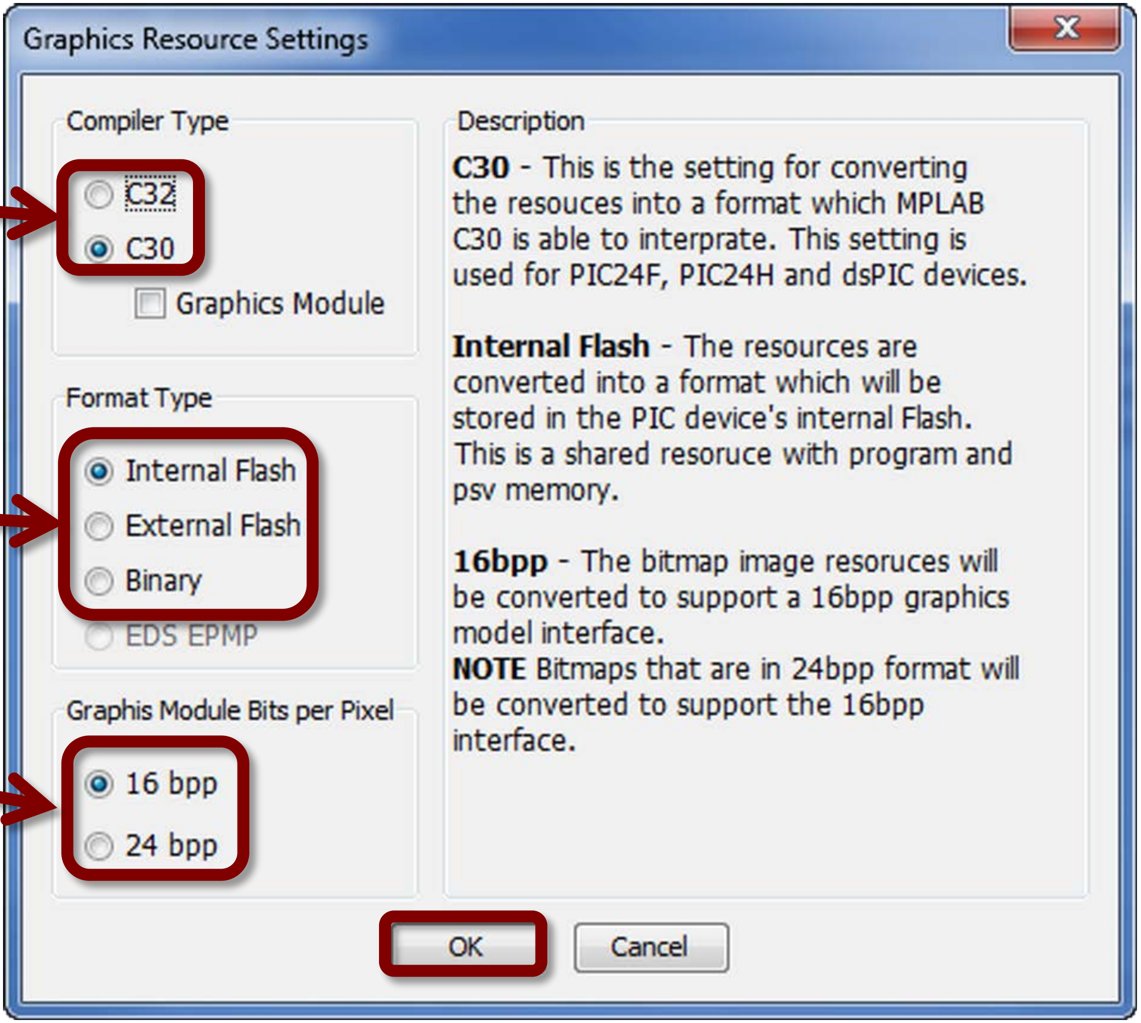


# Graphics Resource Converter Project Settings

Choose compiler

Choose output format

Choose output color depth



Graphics Resource Settings

Compiler Type

○ C32
● C30
☐ Graphics Module

Format Type

● Internal Flash
○ External Flash
○ Binary
○ EDS EPMP

Graphis Module Bits per Pixel

● 16 bpp
○ 24 bpp

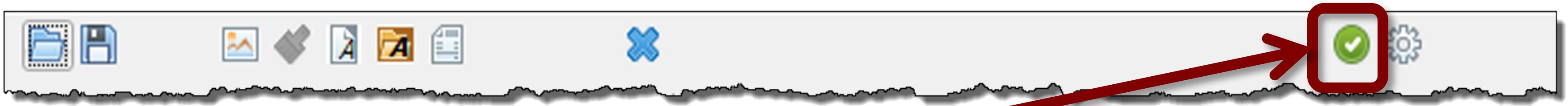
Description

**C30** - This is the setting for converting the resouces into a format which MPLAB C30 is able to interprate. This setting is used for PIC24F, PIC24H and dsPIC devices.

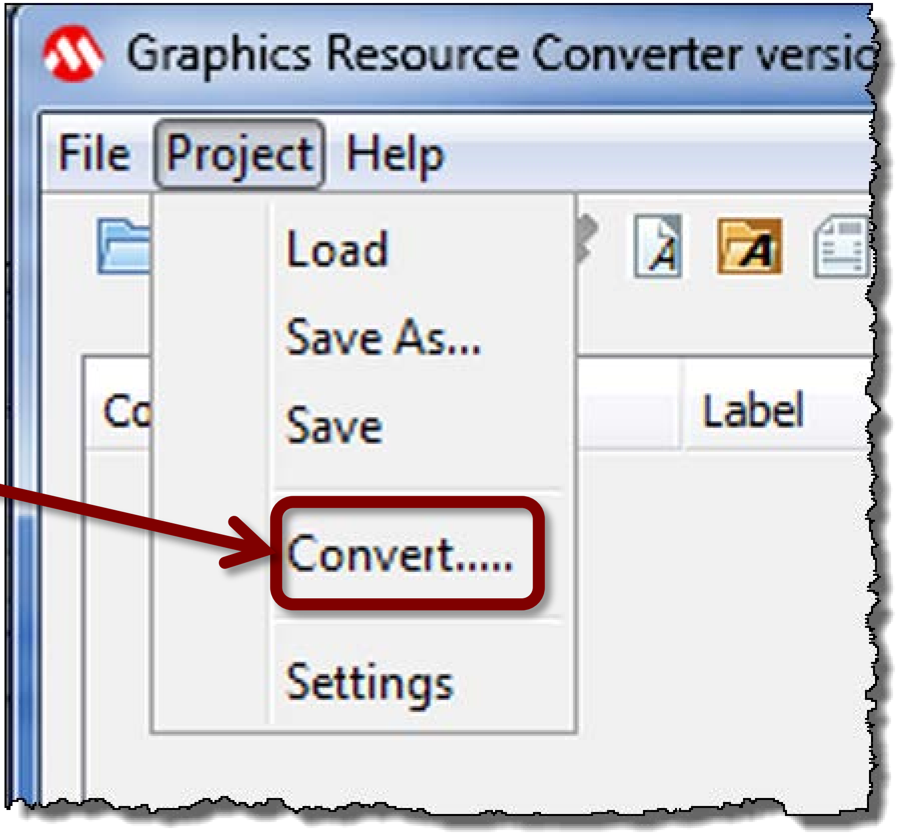
**Internal Flash** - The resources are converted into a format which will be stored in the PIC device's internal Flash. This is a shared resoruce with program and psv memory.

**16bpp** - The bitmap image resoruces will be converted to support a 16bpp graphics model interface.
**NOTE** Bitmaps that are in 24bpp format will be converted to support the 16bpp interface.

# Graphics Resource Converter Convert

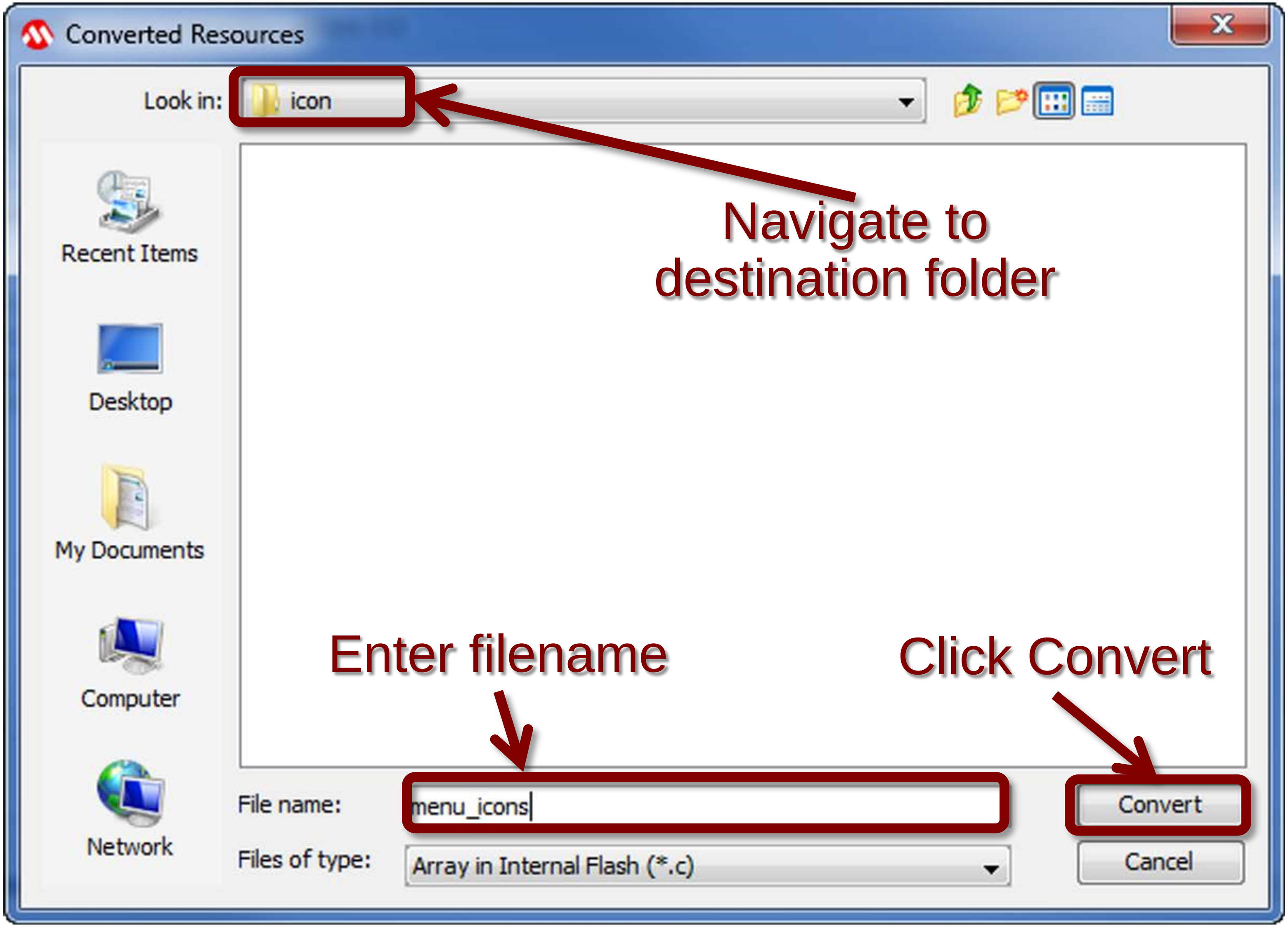


Either click toolbar icon OR…



Select from Project menu

# Graphics Resource Converter Convert



Converted Resources

Look in: icon

File name: menu_icons

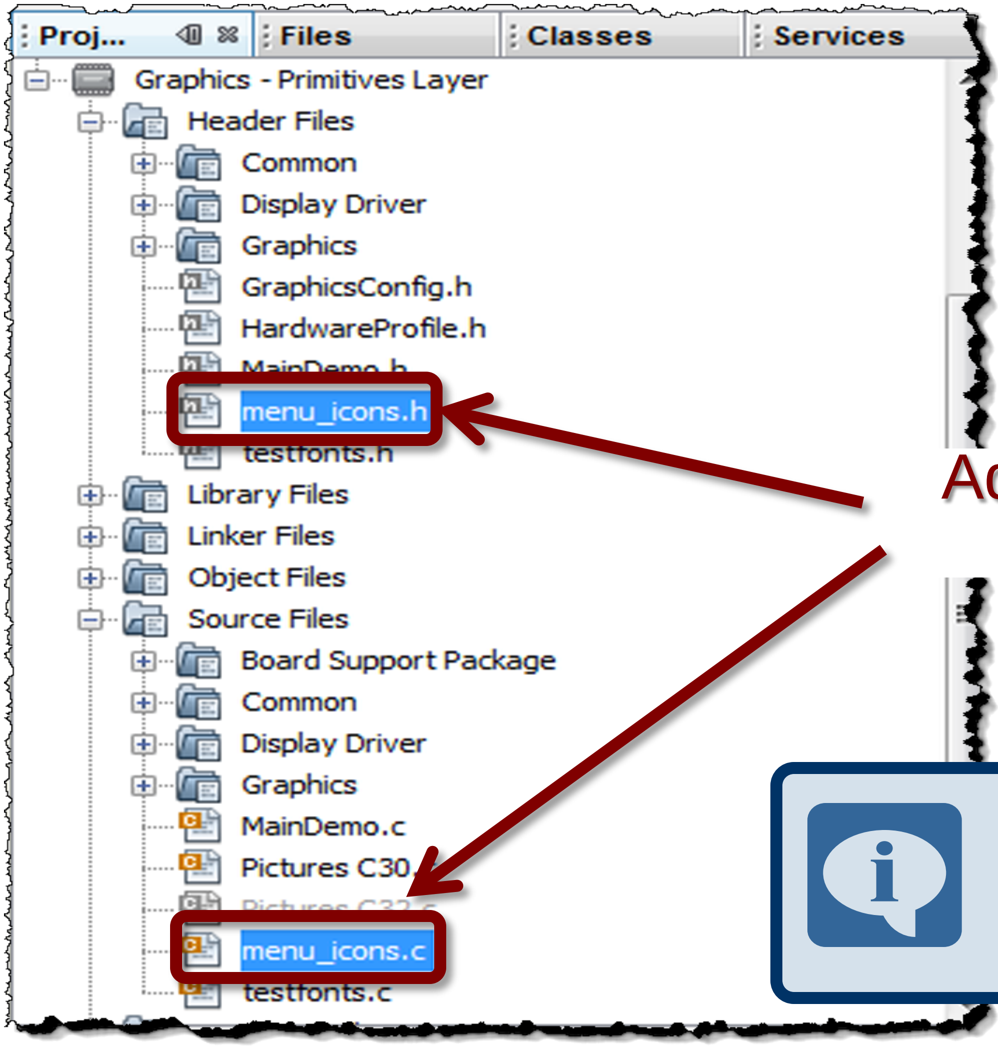
Files of type: Array in Internal Flash (*.c)

Navigate to destination folder

Enter filename

Click Convert

# Using Images

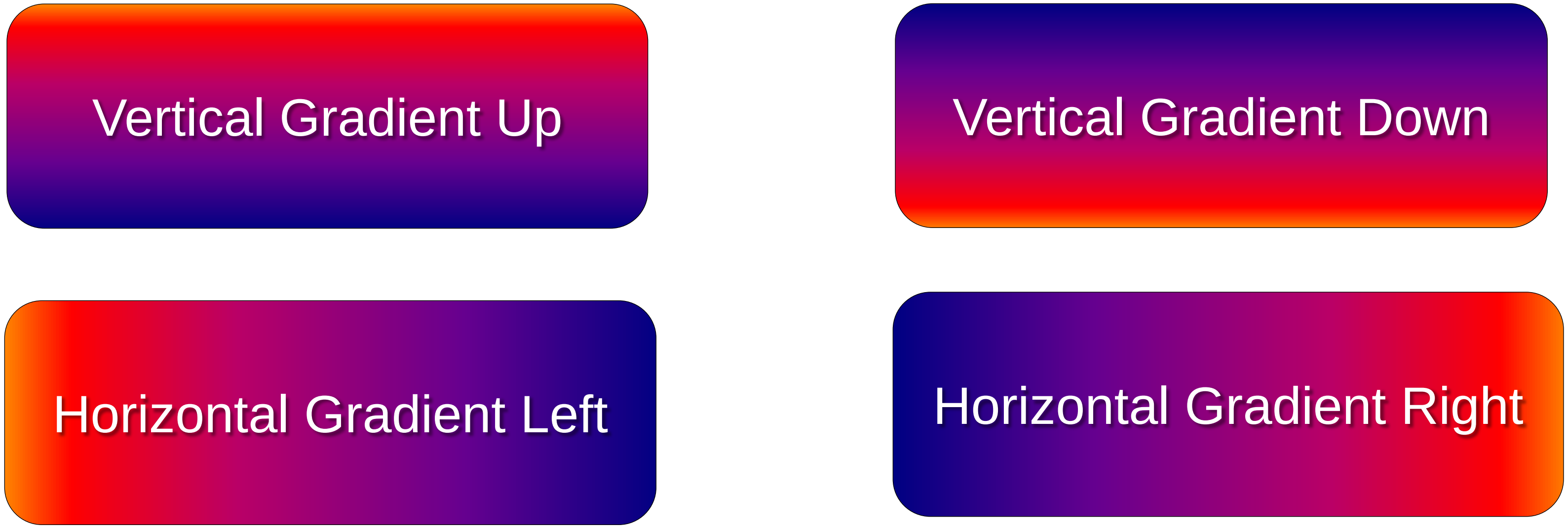


Add generated files to the project

If using external memory, use the External Memory Programmer utility to program generated .hex file into the external memory

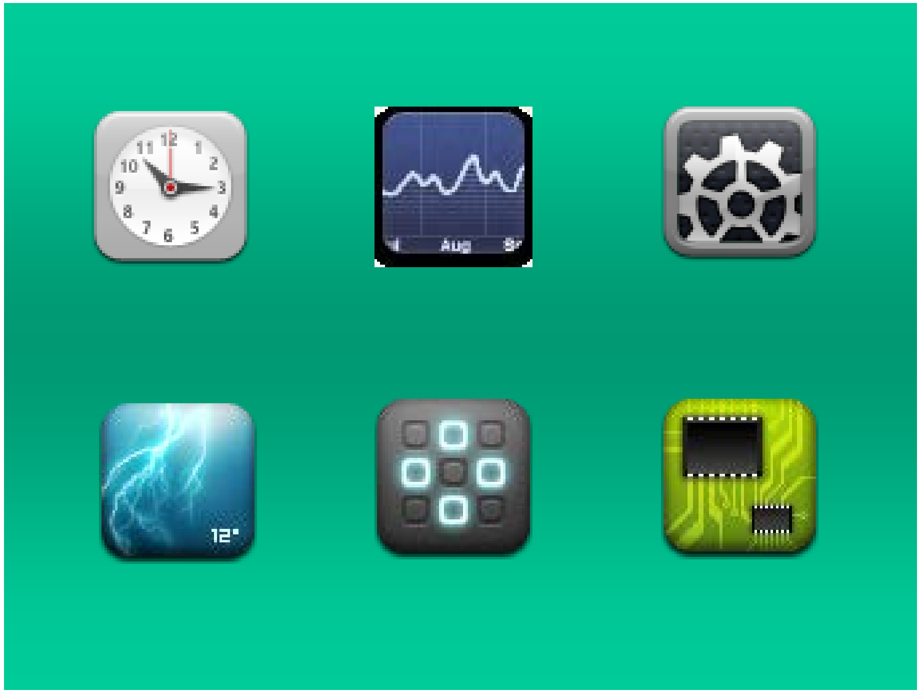
# Gradients

- Smooth transition from one color to another
- Implemented in the primitive layer of the library
- Vertical or horizontal are supported



# Gradient Background

- Enhances HMI appearance
- Doesn’t consume extra memory space



# Microchip Graphics Library
# Use Gradient

- In **GraphicsConfig.h**, enable gradients

```
#define USE_GRADIENT
```

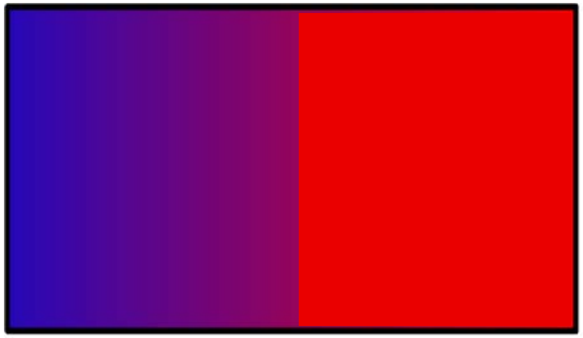
- In application file(s) …
  - Call `BarGradient(…)` or `BevelGradient(…)`

```
…
BarGradient( left, top, right, bottom,
BRIGHTBLUE, BRIGHTRED, 100, GRAD_RIGHT);
…
```

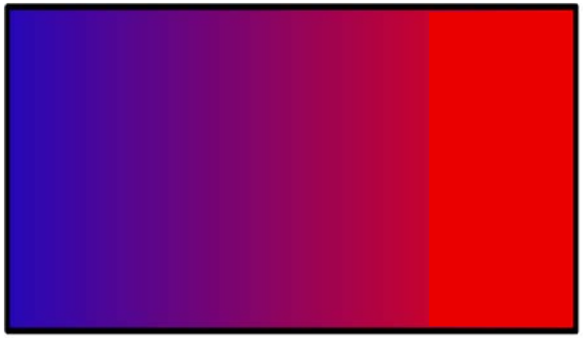
# Gradients
# ‘Length’ Parameter

- **Length describes where gradient will stop**

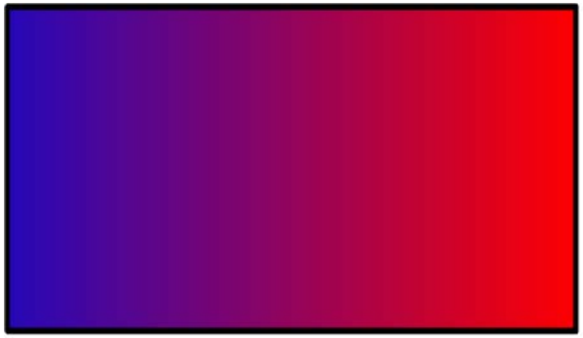
```
…
BarGradient( left, top, right, bottom,
BRIGHTBLUE, BRIGHTRED, length,
GRAD_RIGHT);
…
```



length = 50



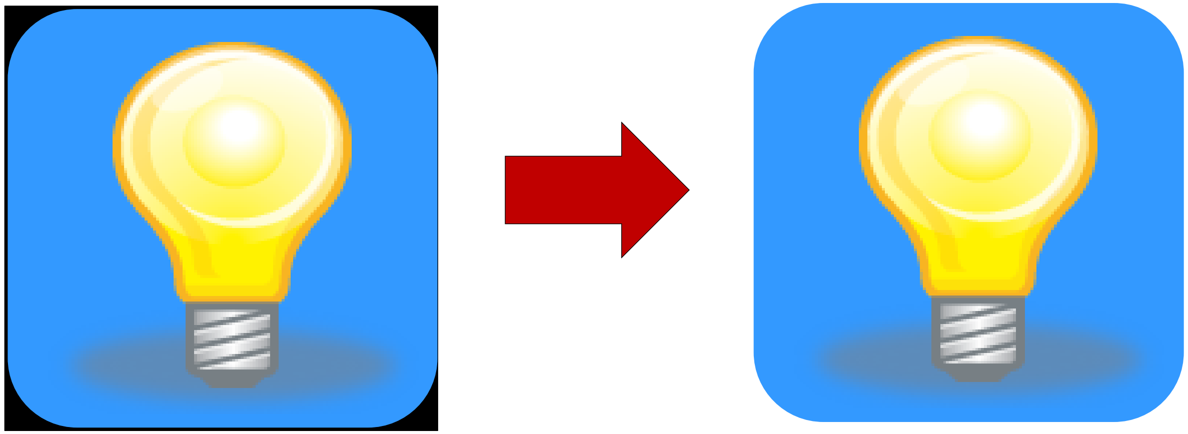
length = 75



length = 100

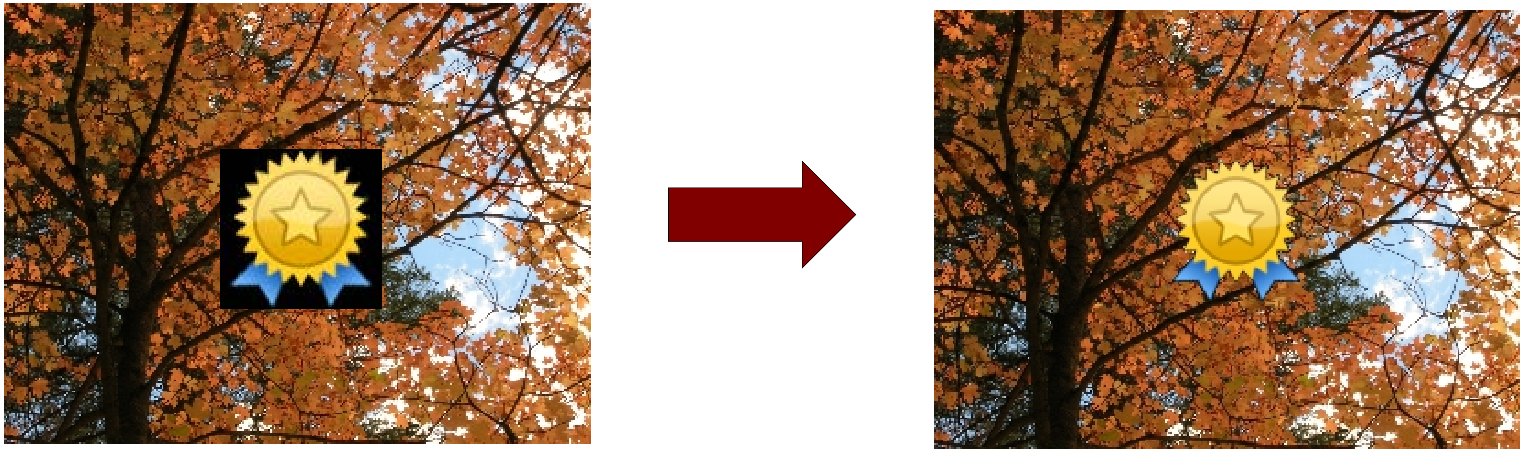
# Transparency

- Used to “hide” colors
  - Hardware accelerated (if available)
  - OR use primitive layer APIs
- May be used to hide icon edges



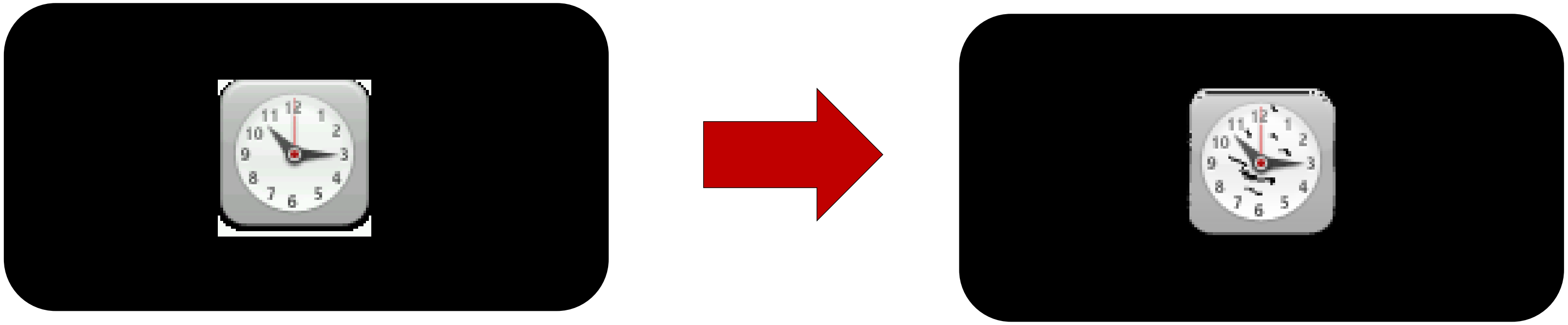
# Microchip Graphics Library Transparency Example

- Application requires ribbon icon over the screen background picture of a tree
- Problem:  The ribbon icon is a bitmap with a black background that does not blend with the picture
- Solution:  Use transparency to hide the icon’s black background



# Design Considerations

- Transparent colors are EXACT match
- Pixels of transparent color are ignored
- Example:
  - Transparent color set to white



- Use image processing tool (e.g. GIMP, Paint) to change corner color

# Microchip Graphics Library Transparency Example

- In **GraphicsConfig.h**, enable transparency

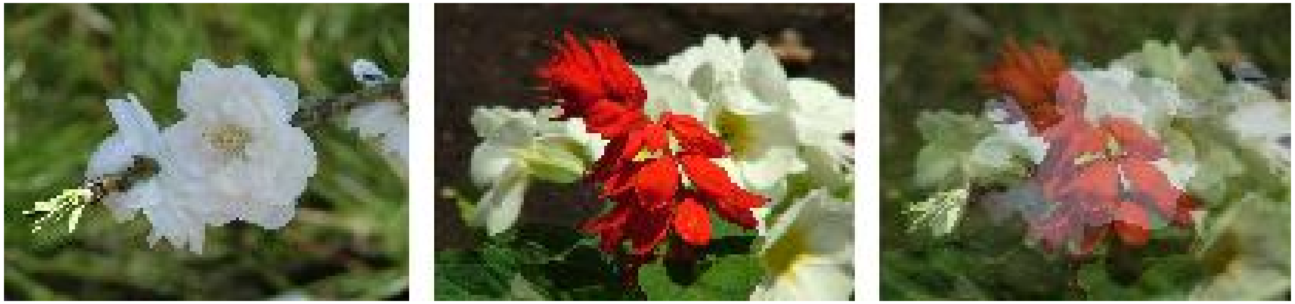
```
#define USE_BITMAP_FLASH
#define USE_TRANSPARENCY
```

- In **Main.c** …

```
                     …
   extern const BITMAP_FLASH RibbonIcon;
extern const BITMAP_FLASH ScreenBackground;
                     …
 PutImage(0,0, (void *)&ScreenBackground);
      TransparentColorEnable(BLACK);
    PutImage(0,0, (void *)&RibbonIcon);
        TransparentColorDisable();
                     …
```

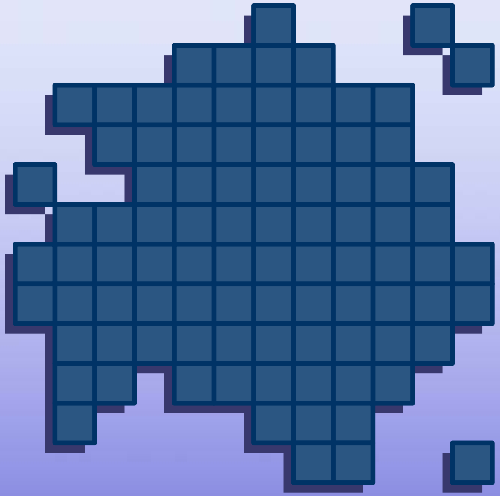
# Alpha Blending

- Layering effect where 2 images are blended
- Fading effect where a new image slowly appears as the old disappears
- Can be used to add an overlay to the HMI



# Double Buffering

- Two frames can be used for the GUI to ensure no renderring effect takes place to the user.



# Microchip Graphics Library

## Basics:  Fonts

# Fonts

Definition:

Fonts are electronic data files containing a set of glyphs, characters and symbols. Fonts are created with font editors and are often considered works of art. Pre-created fonts are available from many sources, but may be licensed. Often times they are copyrighted. Please read all licensing agreements before use.

- **Use Graphics Resource Converter to convert fonts**
  - **True Type (*.ttf)**
  - **Raster Fonts (*.fnt)**
- **Resulting font …**
  - **Stored as an array in internal flash (const section)**
  - **or external memory**
- **Unicode support via multi-byte characters**
  - **AN1182 -- Fonts in the Microchip Graphics Library**

# How to get fonts

- Purchase fonts
  - Thousands of sources available online
  - Read license terms!
- Many websites offer “free” fonts
  - Look for open source fonts (http://www.openfontlibrary.org)
  - Download Google Web Fonts (http://code.google.com/p/googlefontdirectory)
- Create your own fonts using a font editor
  - Fony – freeware
  - FontForge – freeware
  - Other editors range in price
- Convert from Open Type
  - Results vary depending on the converter

# Font Terminology

- Body
  - Imaginary area that encompasses each glyph in a font

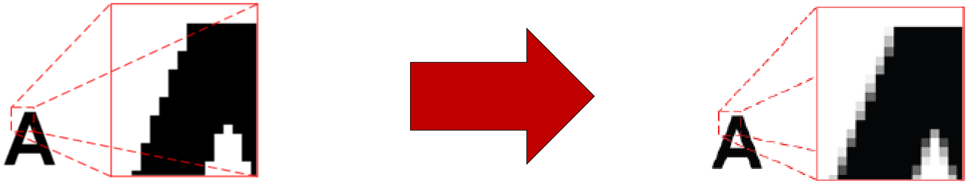
Font body impacts spacing
Font body impacts spacing
Font body impacts spacing
Font body impacts spacing

- Point Size
  - The height of the glyph body

24 pt Arial
24 pt Calibri
24 pt Cochin
24 pt Gabriola

# Font Terminology

- Anti-Aliasing
  - Blurring the edges to soften the look
  - Desirable for larger point sizes



- Style
  - *Italics*, Bold, Anti-Aliased, etc
- Extended Glyphs
  - Used render languages that use more than one byte to represent a single character

त + े = ते

# Microchip Graphics Library Font Requirements

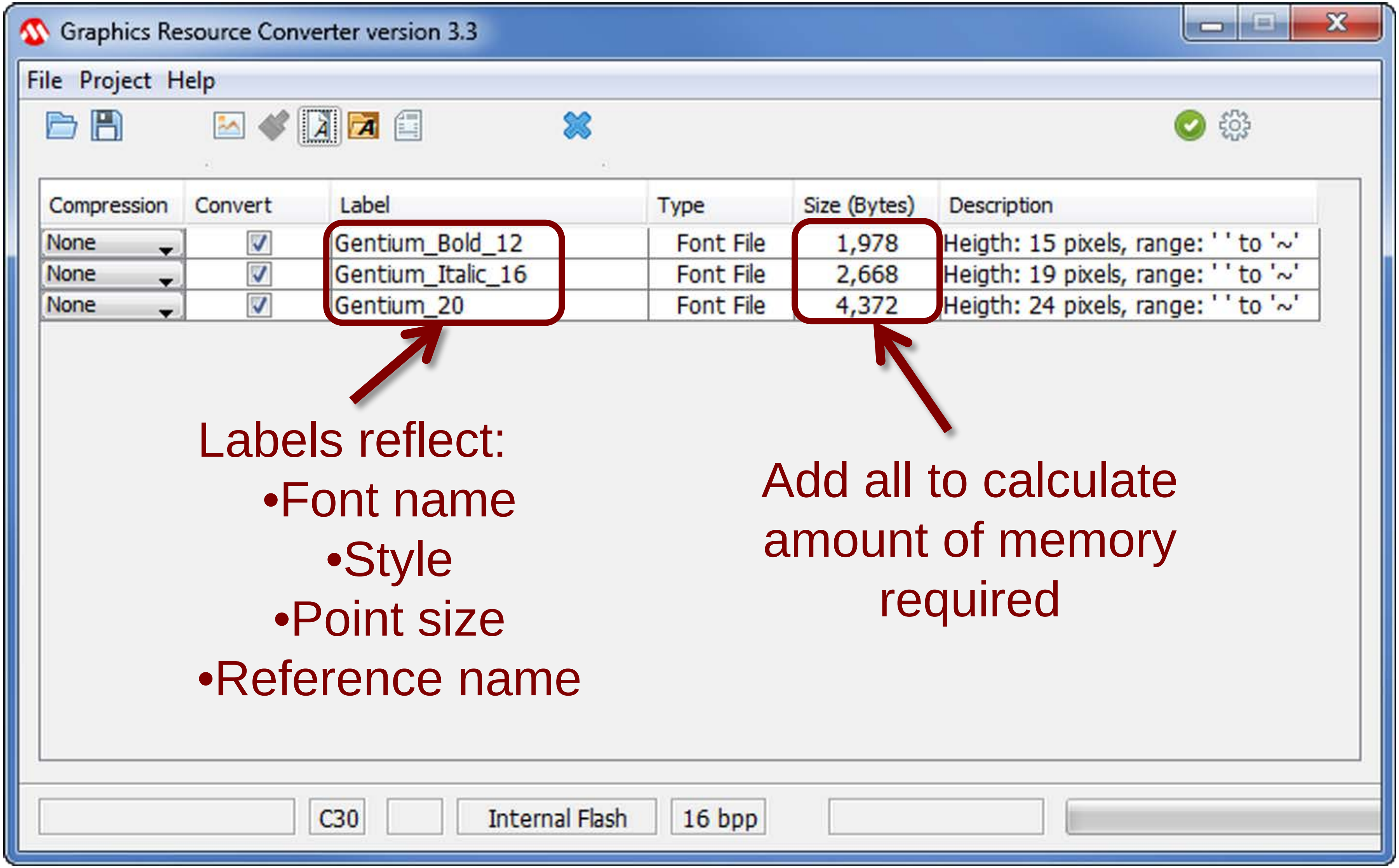
- Fonts must be converted
- Must have one font table per font type
  - EXAMPLE:
    - TimesNewRoman Bold 12 pts
    - TimesNewRoman Italic 12 pts
    - TimesNewRoman Bold 24 pts
    - => 3 font tables required
- May be stored in internal or external FLASH

# Design Considerations

- Memory consumption factors:
  - Using multiple font types
  - Point size and weight of the characters
  - Number of characters in the font
  - Anti-aliased fonts consume ~2x the memory
  - Extended glyphs need at least 2 bytes per character

- Reduce font memory footprint by:
  - Minimize number of font types
  - Use Microchip Graphics Resource Converter to:
    - Filter the font (only store characters used)
      - Especially for extended glyphs
    - Reduce the font range (e.g. only store numbers)
  - Limit use of anti-aliased fonts

# Graphics Resource Converter Fonts

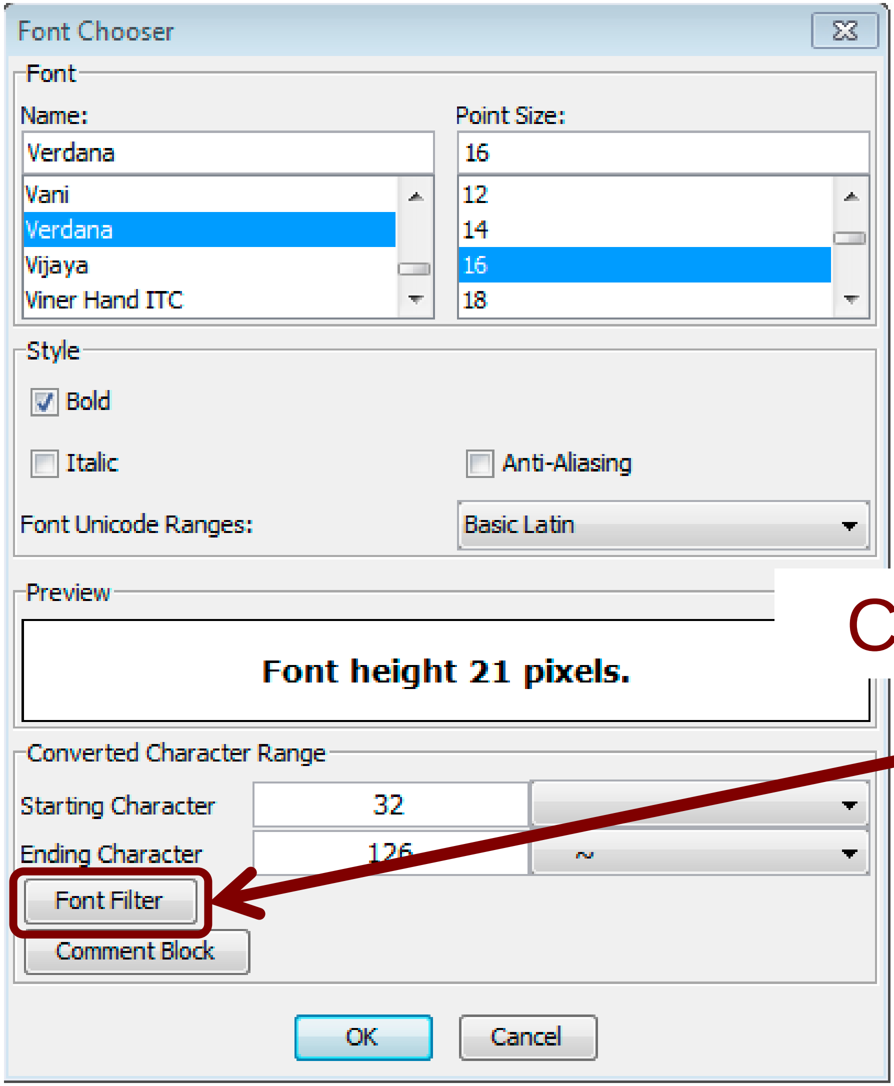


| Compression | Convert | Label | Type | Size (Bytes) | Description |
|---|---|---|---|---|---|
| None | ☑ | Gentium_Bold_12 | Font File | 1,978 | Heigth: 15 pixels, range: ' ' to '~' |
| None | ☑ | Gentium_Italic_16 | Font File | 2,668 | Heigth: 19 pixels, range: ' ' to '~' |
| None | ☑ | Gentium_20 | Font File | 4,372 | Heigth: 24 pixels, range: ' ' to '~' |

Labels reflect:
- Font name
- Style
- Point size
- Reference name

Add all to calculate amount of memory required

# Graphics Resource Converter
# Reducing Font Memory



Name: Verdana

Point Size: 16

☑ Bold

☐ Italic

☐ Anti-Aliasing

Font Unicode Ranges: Basic Latin

Font height 21 pixels.

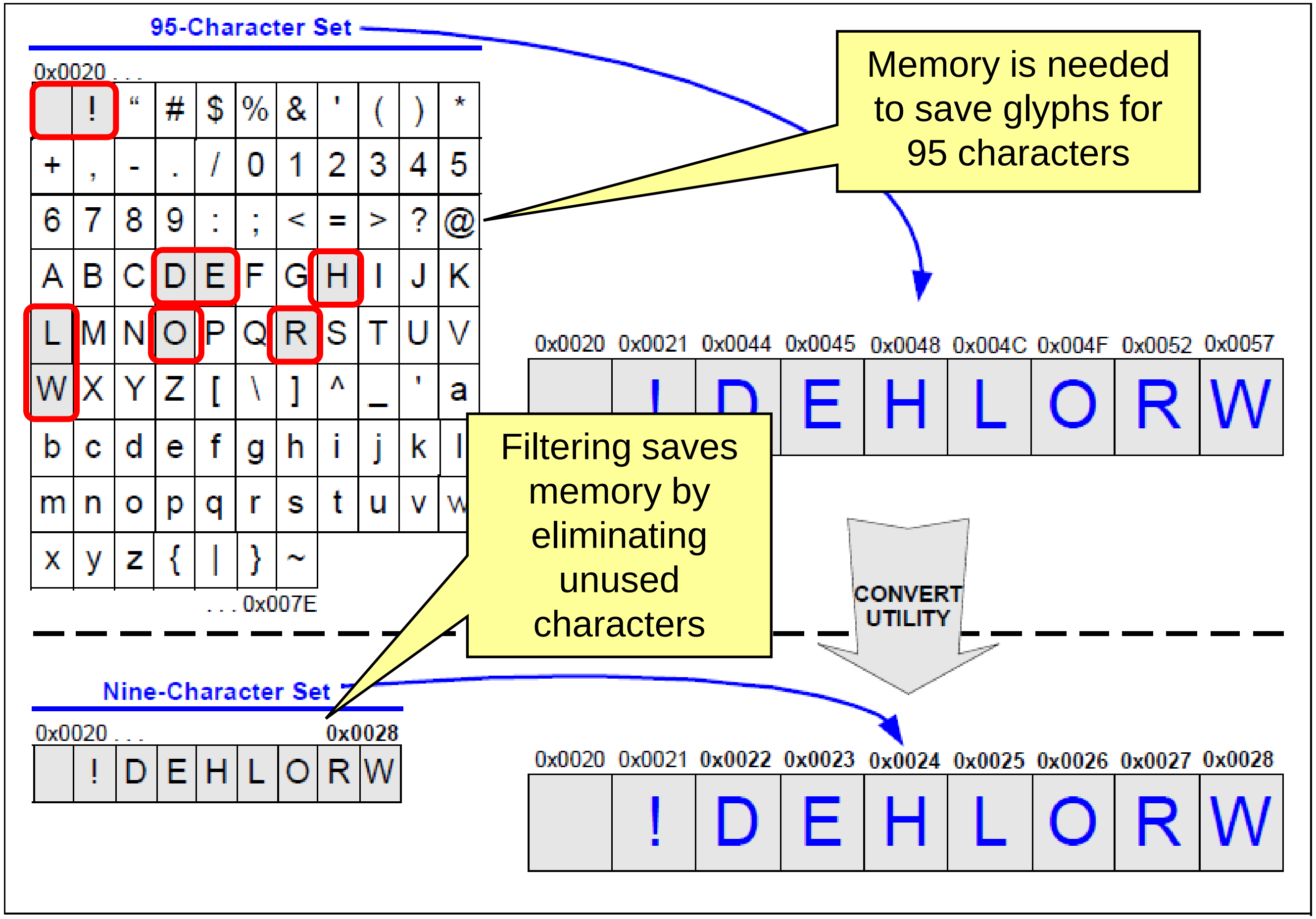
Starting Character 32

Ending Character 126 ~

Convert only specific strings

# Font Filtering

**Application will use the string “Hello World!”**
**Since we only need 9 characters, how can we save flash memory space?**



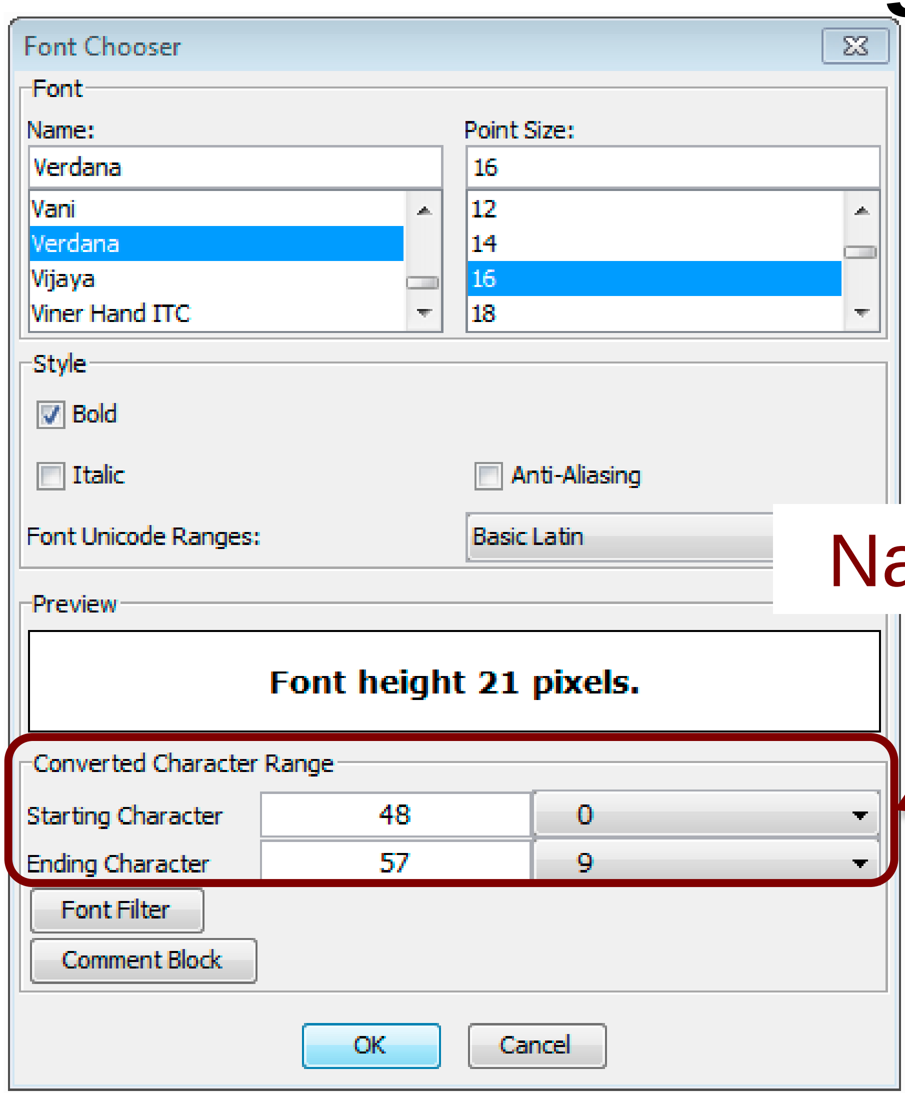
Memory is needed to save glyphs for 95 characters

Filtering saves memory by eliminating unused characters

# Font Filter
# Text File

- Editor must support unicode
  - Save in 16-bit unicode format
- Each line must have 3 sections:
  - `<String Label>:<String>//<comments>`
    - “//” Indicator required
    - Comments are optional
- Refer to the Graphics Resource Converter help file or to App Note 1182 for details

# Graphics Resource Converter Reducing Font Memory



Font

Name: Verdana

Point Size: 16

Style

☑ Bold

☐ Italic

☐ Anti-Aliasing

Font Unicode Ranges: Basic Latin

Preview

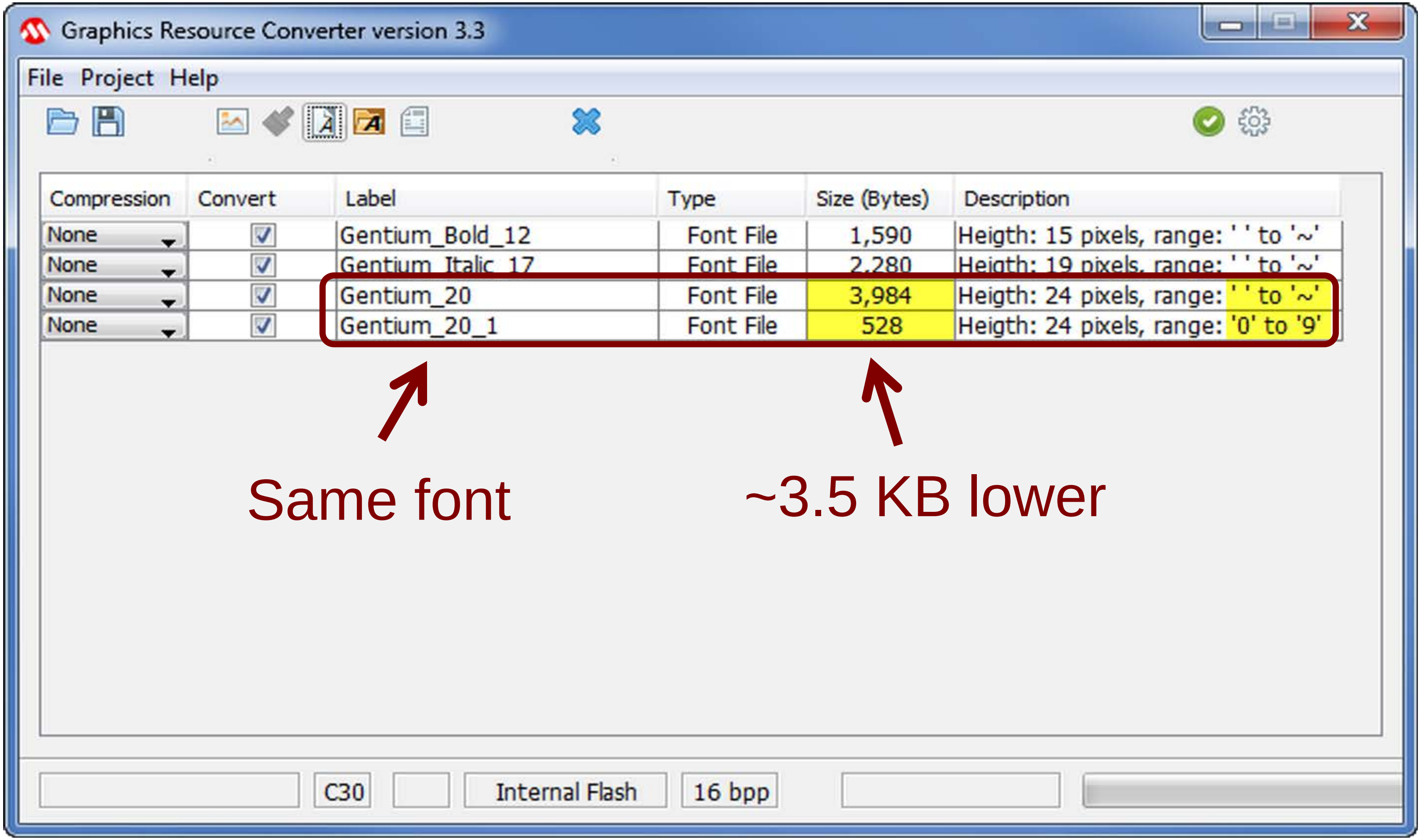
**Font height 21 pixels.**

Converted Character Range

| | | |
|---|---|---|
| Starting Character | 48 | 0 |
| Ending Character | 57 | 9 |

Narrow the character range

# Graphics Resource Converter
# Reducing Font Memory



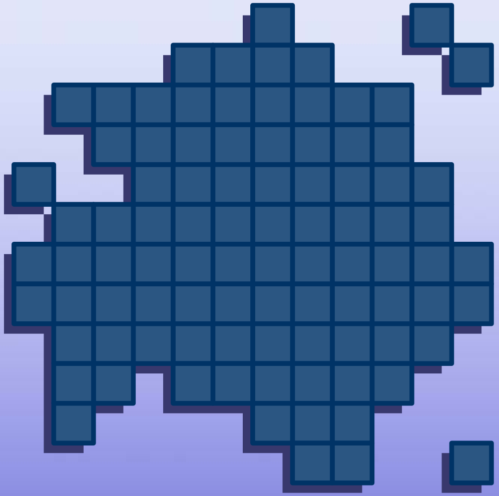
Graphics Resource Converter version 3.3

| Compression | Convert | Label | Type | Size (Bytes) | Description |
|---|---|---|---|---|---|
| None | ☑ | Gentium_Bold_12 | Font File | 1,590 | Heigth: 15 pixels, range: ' ' to '~' |
| None | ☑ | Gentium_Italic_17 | Font File | 2,280 | Heigth: 19 pixels, range: ' ' to '~' |
| None | ☑ | Gentium_20 | Font File | 3,984 | Heigth: 24 pixels, range: ' ' to '~' |
| None | ☑ | Gentium_20_1 | Font File | 528 | Heigth: 24 pixels, range: '0' to '9' |

C30 | Internal Flash | 16 bpp

Same font

~3.5 KB lower



# Microchip Graphics Library

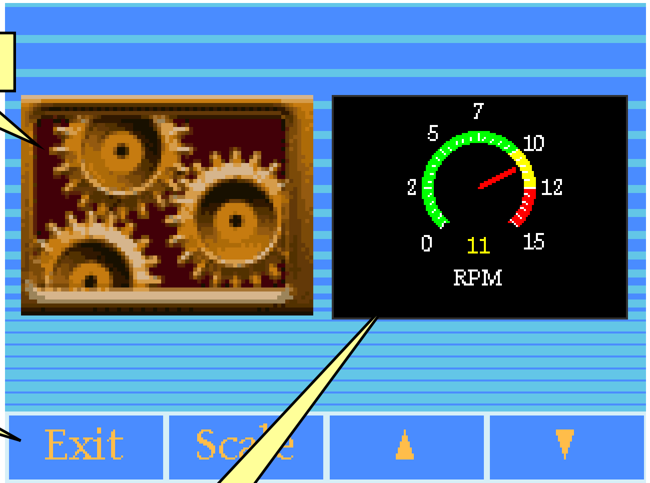
## Basics: Widgets

# Library Widgets

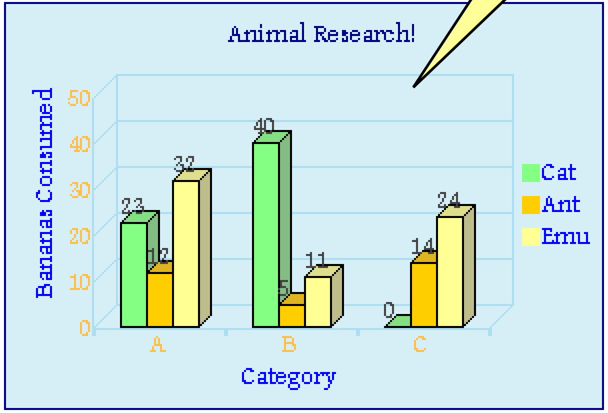
Picture

Buttons

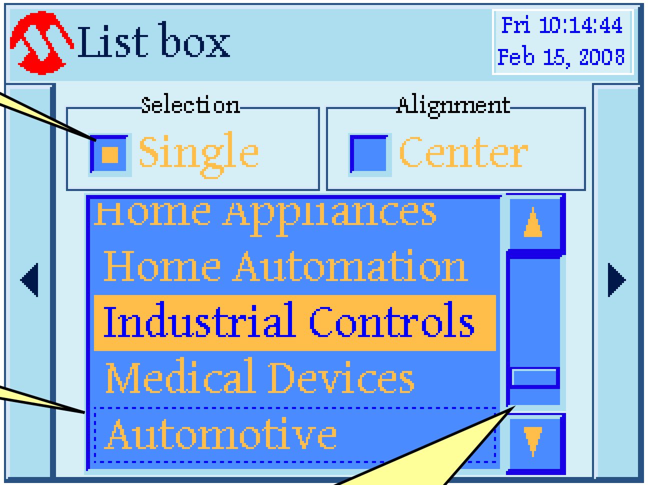
Meter



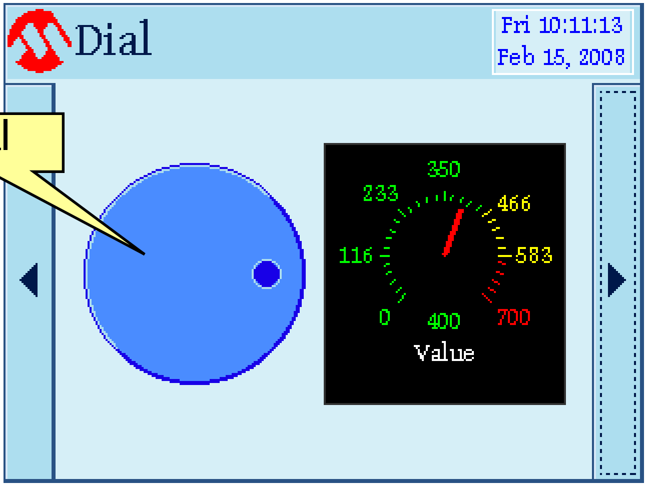
Chart



Checkbox



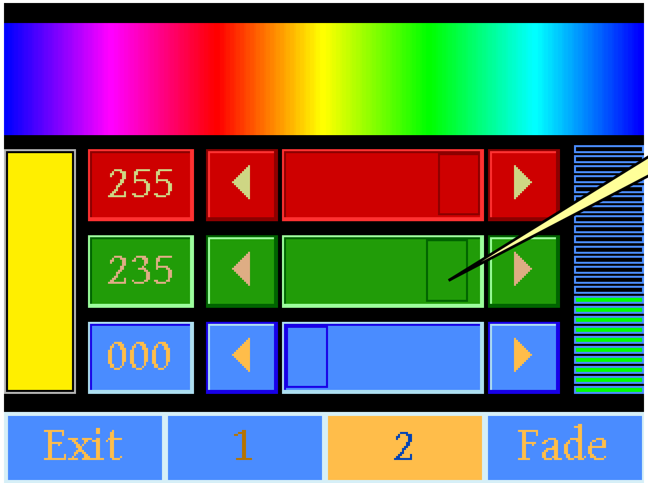
Dial



Listbox

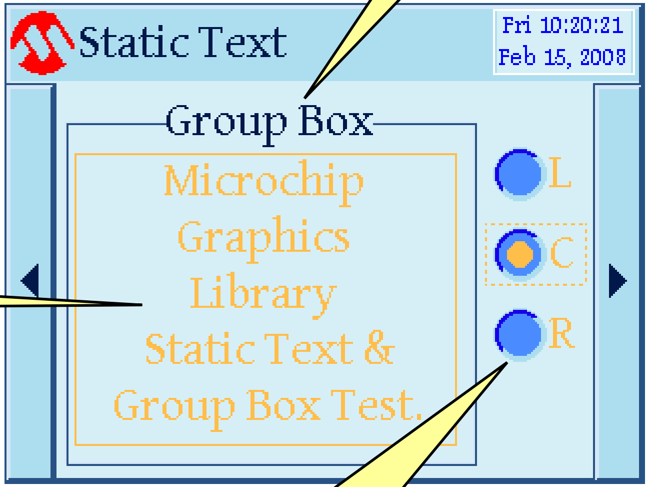
Scrollbar (using Buttons and Slider)

# Library Widgets



Slider

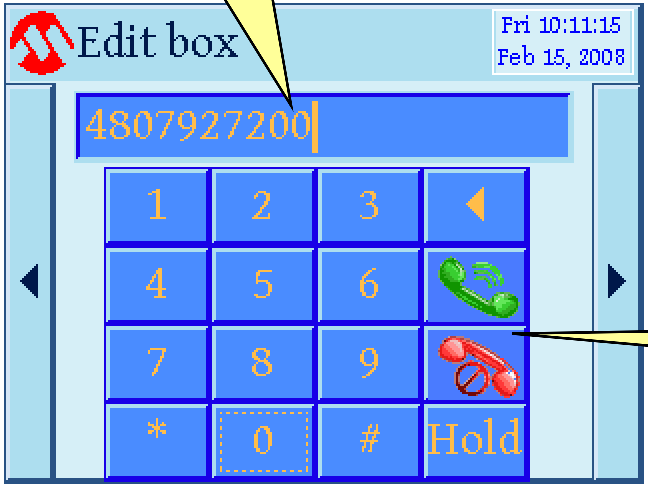
Group Box



Microchip Graphics Library Static Text & Group Box Test.

Edit Box

Static Text



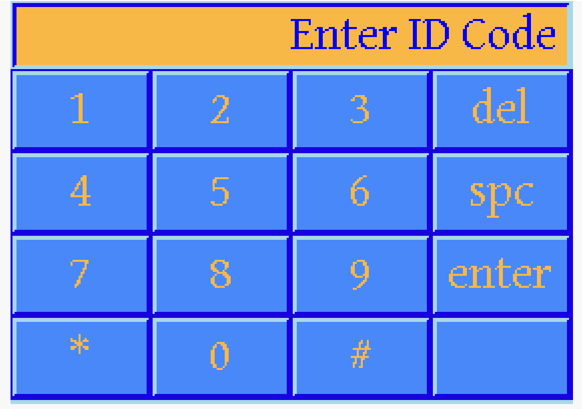
4807927200

Radio Buttons

Button with Image

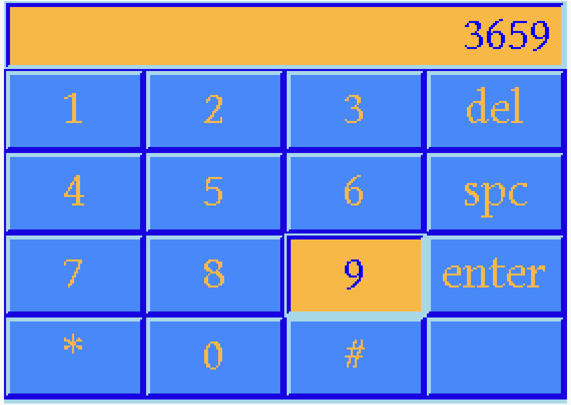
# Library Widgets

- Text Entry Widget
- Custom Widget creation described in AN1246



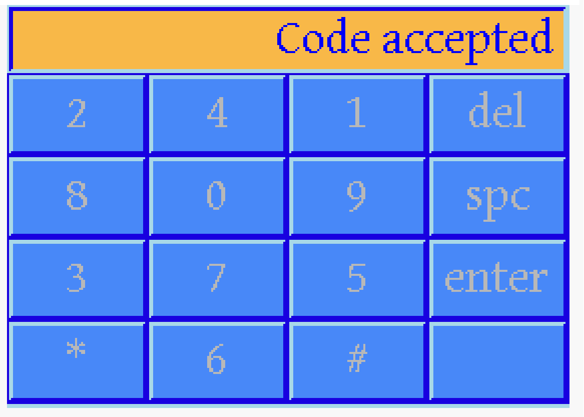
Enter ID Code

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | del |
| 4 | 5 | 6 | spc |
| 7 | 8 | 9 | enter |
| * | 0 | # | |



3659

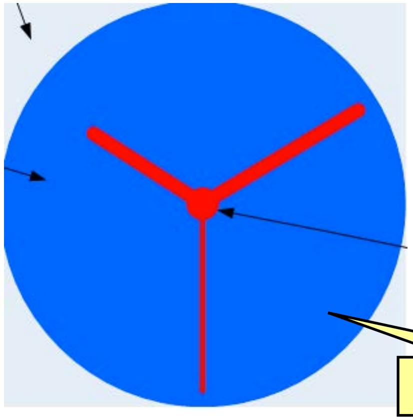
| | | | |
|---|---|---|---|
| 1 | 2 | 3 | del |
| 4 | 5 | 6 | spc |
| 7 | 8 | 9 | enter |
| * | 0 | # | |



Code accepted

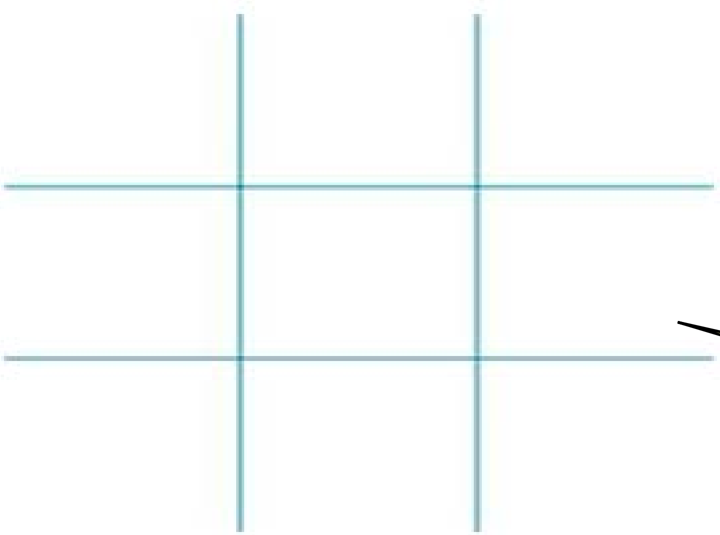
| | | | |
|---|---|---|---|
| 2 | 4 | 1 | del |
| 8 | 0 | 9 | spc |
| 3 | 7 | 5 | enter |
| * | 6 | # | |

# Library Widgets



Analog Clock

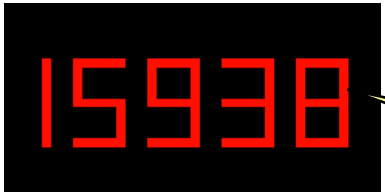
Digital Meter



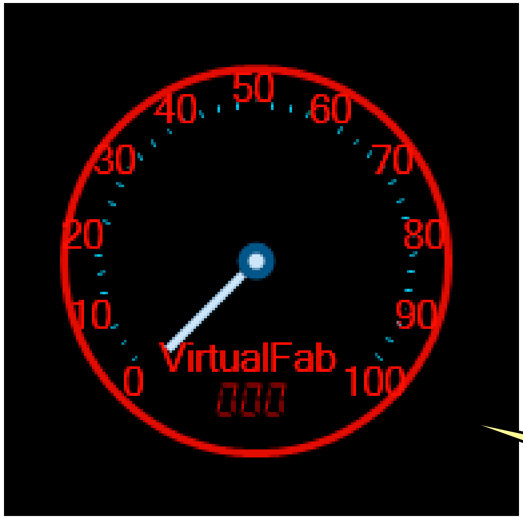
Grid

# VGDD External Widgets

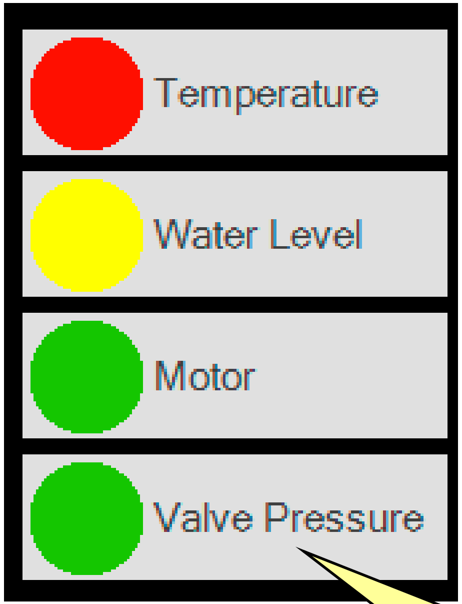
Require license for full version before they can be used



7 seg display



Super Gauge



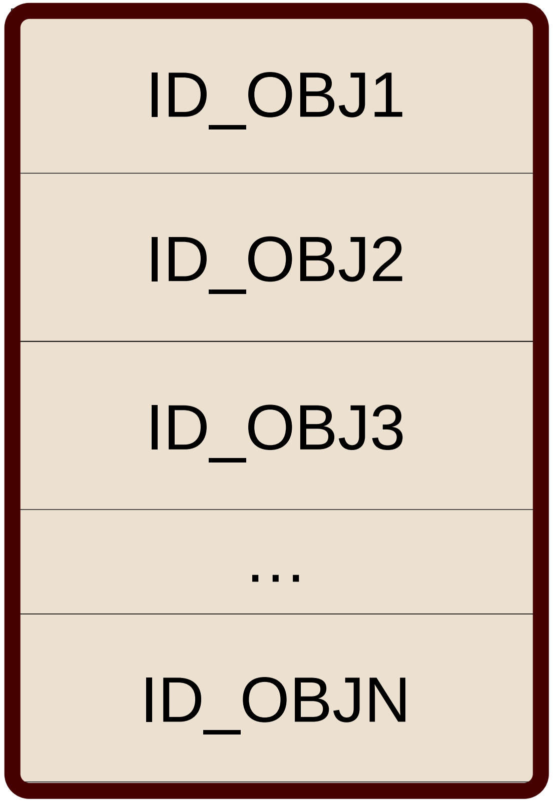
Alarm Indicators

# Widget Abbreviations

Obj =

| Widget Name | Obj Abbreviation |
|---|---|
| Analog Clock | Ac |
| Button | Btn |
| Chart | Ch |
| Checkbox | Cb |
| Round Dial | Rdia |
| Digital Meter | Dm |
| Edit Box | Eb |
| Grid | Grid |
| Group Box | Gb |
| List Box | Lb |
| Meter | Mtr |
| Picture Control | Pict |
| Progress Bar | Pb |
| Radio Button | Rb |
| Slider/Scroll Bar | Sld |
| Static Text | St |
| Text Entry | Te |
| Window | Wnd |

# Creating Widgets



ID_OBJ1

ID_OBJ2

ID_OBJ3

…

ID_OBJN

- `ObjCreate(,,,)`
  - Populates the widget structure
  - Adds widget to bottom of the active linked list
  - Returns a pointer to the widget structure
- Heap required
  - Widgets dynamically created
- More details on using the linked list later…

# Creating Widgets
# Object Header Members

- `ID`
  - Unique integer used to form a “Handle”
- `Location`
  - Top, left, bottom, right define placement
- `State`
  - 16-bit value that defines the widget state
- `Style Scheme pointer`
  - Structure that defines widget appearance

# What is a “heap”?

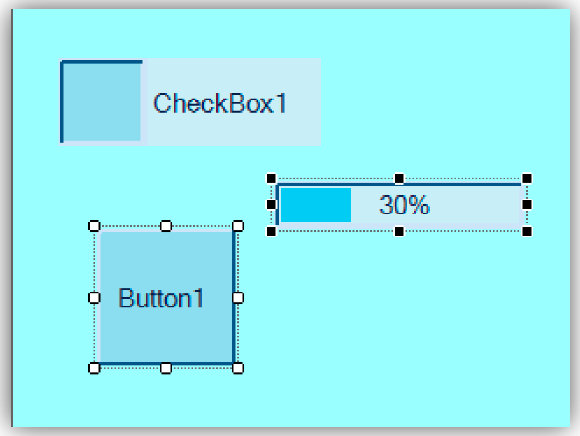
**Definition:**

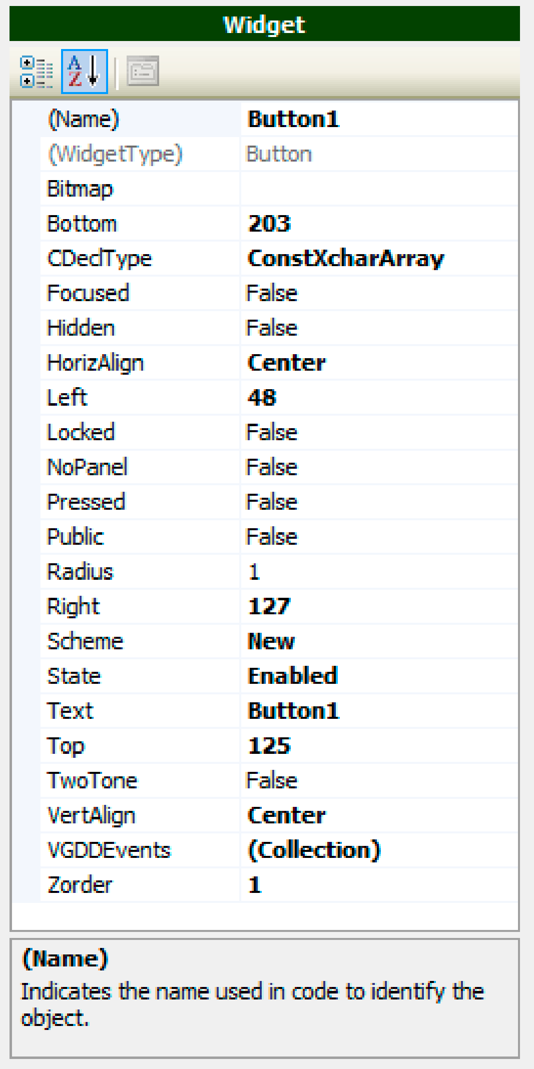
The heap is an unused pool of memory (similar to a stack) where a programmer may dynamically allocate memory using malloc() calls. In the embedded world, it is typically used to store recursive data structures (e.g. linked lists). Allocated memory must be freed by the application to avoid difficult runtime errors.

- **Proper memory management required**
  - **Memory Leak:**
    - **Program does not free memory no longer needed**
    - **Leads to heap overrun**
  - **Fragmentation:**
    - **May occur when memory is deallocated in chunks**
    - **New malloc() requests may no longer fit**
    - **Clean up with a defrag engine (not provided in library)**
  - **Both result in runtime errors**
- **`GOLFree()` – Graphics Library Function**
  - **Removes entire linked list from heap**

# Creating Widgets Using VGDD

- VGDD Widget Property Window
- Used to create widget structure
- See changes as they are made

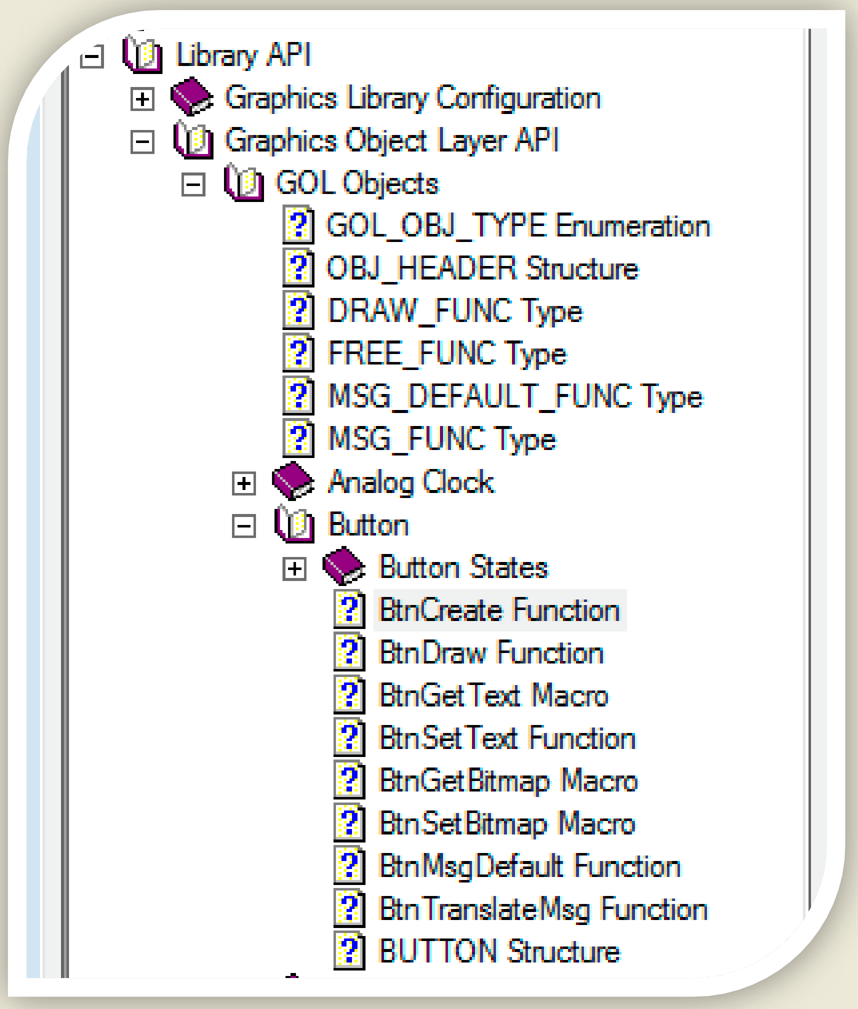



Widget

| | |
|---|---|
| (Name) | **Button1** |
| (WidgetType) | Button |
| Bitmap | |
| Bottom | **203** |
| CDeclType | **ConstXcharArray** |
| Focused | False |
| Hidden | False |
| HorizAlign | **Center** |
| Left | **48** |
| Locked | False |
| NoPanel | False |
| Pressed | False |
| Public | False |
| Radius | 1 |
| Right | **127** |
| Scheme | **New** |
| State | **Enabled** |
| Text | **Button1** |
| Top | **125** |
| TwoTone | False |
| VertAlign | **Center** |
| VGDDEvents | **(Collection)** |
| Zorder | **1** |

**(Name)**
Indicates the name used in code to identify the object.

# Widget Help



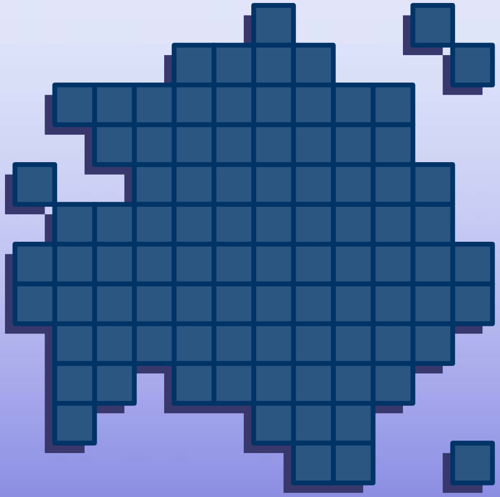
- Library API
  - Graphics Library Configuration
  - Graphics Object Layer API
    - GOL Objects
      - GOL_OBJ_TYPE Enumeration
      - OBJ_HEADER Structure
      - DRAW_FUNC Type
      - FREE_FUNC Type
      - MSG_DEFAULT_FUNC Type
      - MSG_FUNC Type
      - Analog Clock
      - Button
        - Button States
        - BtnCreate Function
        - BtnDraw Function
        - BtnGetText Macro
        - BtnSetText Function
        - BtnGetBitmap Macro
        - BtnSetBitmap Macro
        - BtnMsgDefault Function
        - BtnTranslateMsg Function
        - BUTTON Structure

To find the ObjCreate APIs, expand the desired widget and select the appropriate create function.

..\Microchip\Help

Graphics Library Help.chm

Graphics Library Help.pdf



# Microchip Graphics Library

## Basics:  Style Schemes

# Widget Style Schemes

Definition:

The Style Scheme is a structure used by the library to define the appearance of widgets by assigning style properties.

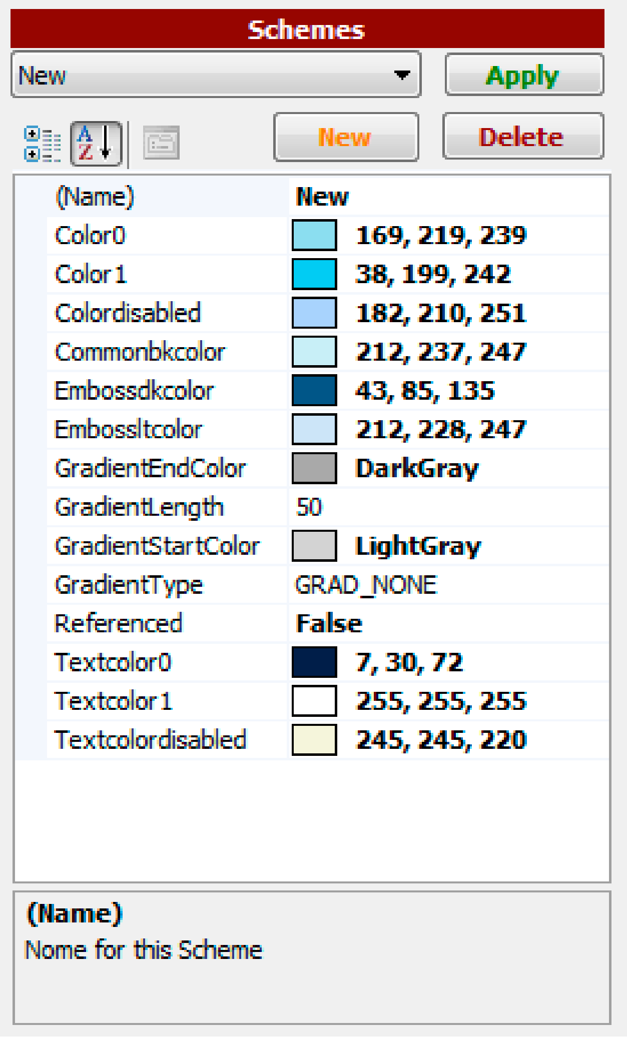
- Default scheme provided in GOLCreateScheme.c
  - May be overridden by the application
  - Widgets without assigned schemes use the default scheme
- `GOLCreateScheme()` dynamically creates style schemes
  - Populates a new scheme structure
  - Returns a pointer to that structure
- Style Schemes may be created at compile time
  - Use ‘&’ operator to reference scheme
- All widgets use style scheme member values differently
  - Help file provides pictures

# Style Scheme Structure Defined in GOL.h

| Type | Member | Description |
|---|---|---|
| `GFX_COLOR` | `EmbossDkColor` | 3D dark color |
| `GFX_COLOR` | `EmbossLtColor` | 3D light color |
| `GFX_COLOR` | `TextColor0` | Objects with text |
| `GFX_COLOR` | `TextColor1` | Objects with text |
| `GFX_COLOR` | `Color0` | Assigned to a state |
| `GFX_COLOR` | `Color1` | Assigned to a state |
| `GFX_COLOR` | `ColorDisabled` | Object in disabled state |
| `GFX_COLOR` | `TextColorDisabled` | Object in disabled state |
| `GFX_COLOR` | `CommonBkColor` | Used to hide objects |
| `void` | `*pFont` | Pointer to font selected |
| `BYTE` | `AlphaValue` | Define `USE_ALPHABLEND` |
| `GFX_GRADIENT_SCHEME` | `gradientScheme` | Define `USE_GRADIENT` |

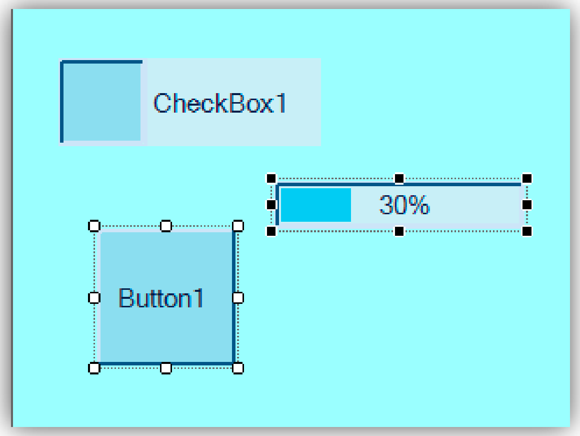
# Creating Style Schemes Using VGDD

- VGDD Style Scheme Window
- Used to style schemes
- See how changes impact other widgets in the design

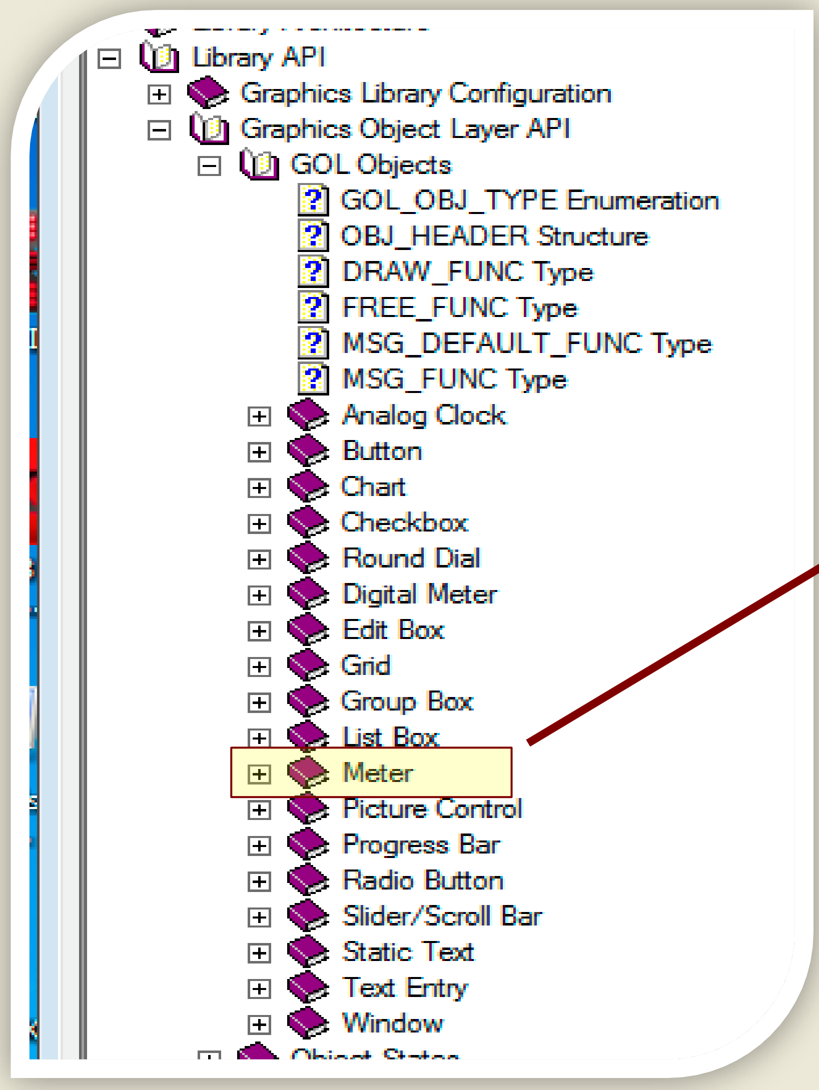


| | |
|---|---|
| (Name) | New |
| Color0 | 169, 219, 239 |
| Color1 | 38, 199, 242 |
| Colordisabled | 182, 210, 251 |
| Commonbkcolor | 212, 237, 247 |
| Embossdkcolor | 43, 85, 135 |
| Embossltcolor | 212, 228, 247 |
| GradientEndColor | DarkGray |
| GradientLength | 50 |
| GradientStartColor | LightGray |
| GradientType | GRAD_NONE |
| Referenced | False |
| Textcolor0 | 7, 30, 72 |
| Textcolor1 | 255, 255, 255 |
| Textcolordisabled | 245, 245, 220 |

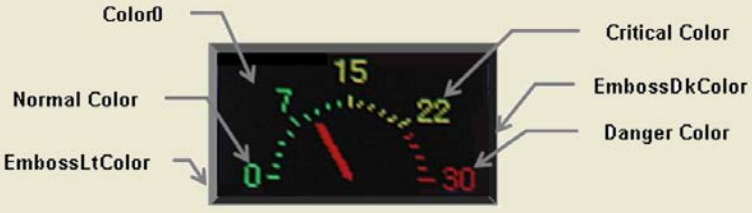
**(Name)**
Name for this Scheme



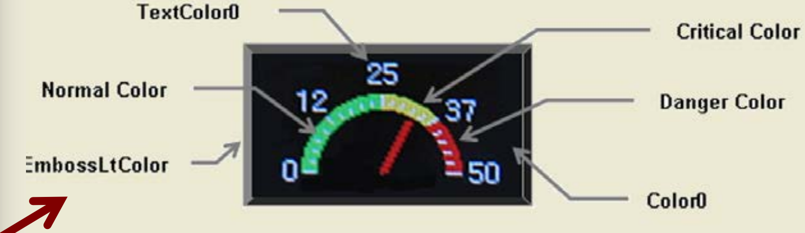
# Style Scheme Help



- Library API
  - Graphics Library Configuration
  - Graphics Object Layer API
    - GOL Objects
      - GOL_OBJ_TYPE Enumeration
      - OBJ_HEADER Structure
      - DRAW_FUNC Type
      - FREE_FUNC Type
      - MSG_DEFAULT_FUNC Type
      - MSG_FUNC Type
      - Analog Clock
      - Button
      - Chart
      - Checkbox
      - Round Dial
      - Digital Meter
      - Edit Box
      - Grid
      - Group Box
      - List Box
      - Meter
      - Picture Control
      - Progress Bar
      - Radio Button
      - Slider/Scroll Bar
      - Static Text
      - Text Entry
      - Window



Default Half Meter



Half Meter with Arc enabled

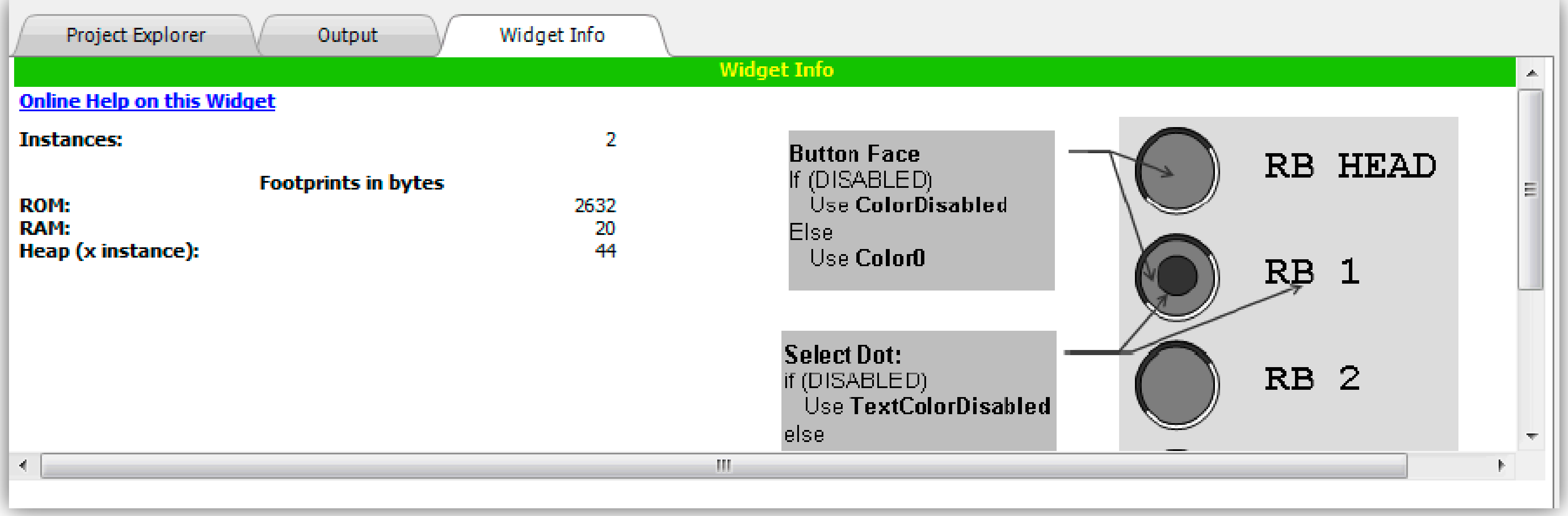
Graphics Library Help.chm

Graphics Library Help.pdf

../Microchip/Help

# Creating Style Schemes Using VGDD

- VGDD Widget Info Window
- See widget style scheme information while designing
- Get resource information



| | |
|---|---|
| Instances: | 2 |

Footprints in bytes

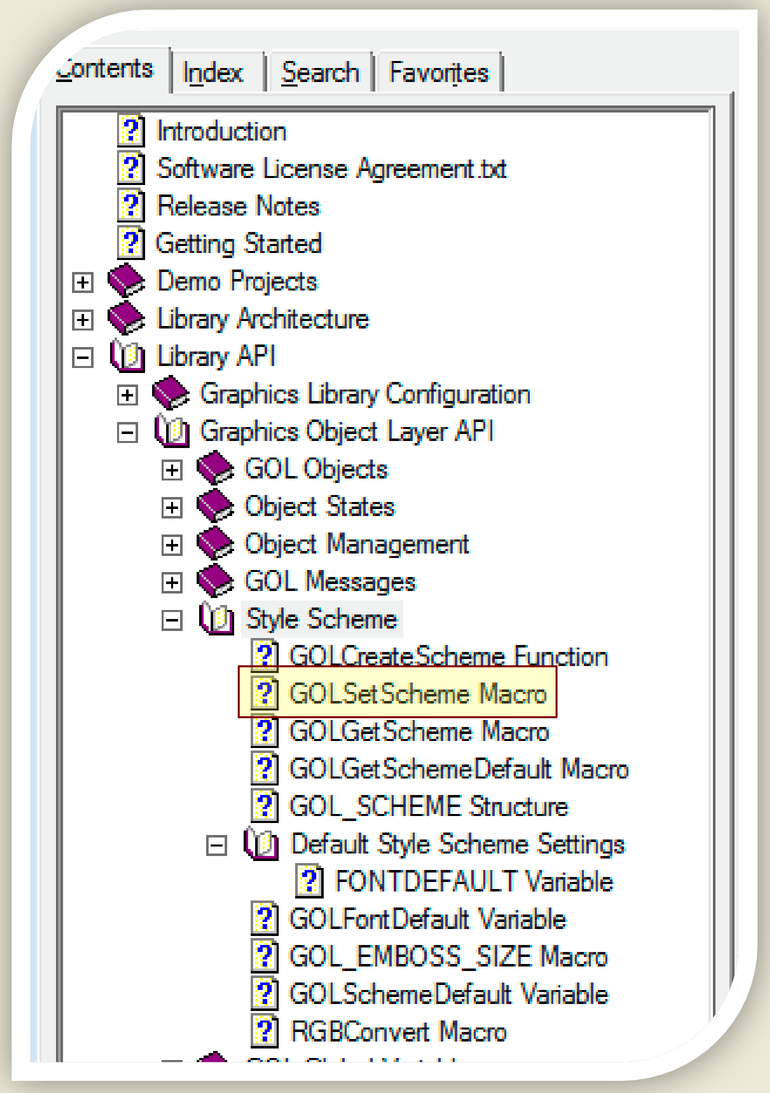
| | |
|---|---|
| ROM: | 2632 |
| RAM: | 20 |
| Heap (x instance): | 44 |

Button Face
If (DISABLED)
  Use ColorDisabled
Else
  Use Color0

Select Dot:
if (DISABLED)
  Use TextColorDisabled
else

Regional Training Centers

# Style Scheme APIs



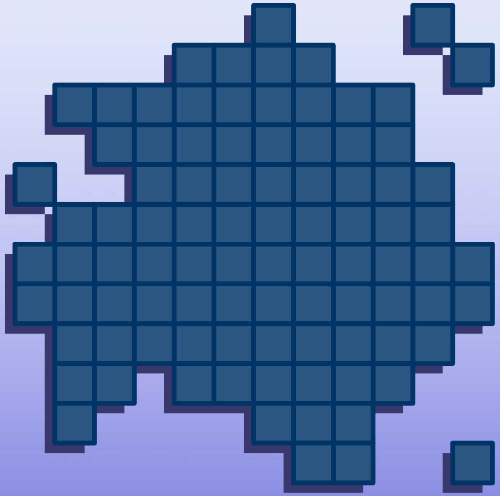
- Introduction
- Software License Agreement.txt
- Release Notes
- Getting Started
- Demo Projects
- Library Architecture
- Library API
  - Graphics Library Configuration
  - Graphics Object Layer API
    - GOL Objects
    - Object States
    - Object Management
    - GOL Messages
    - Style Scheme
      - GOLCreateScheme Function
      - GOLSetScheme Macro
      - GOLGetScheme Macro
      - GOLGetSchemeDefault Macro
      - GOL_SCHEME Structure
      - Default Style Scheme Settings
        - FONTDEFAULT Variable
      - GOLFontDefault Variable
      - GOL_EMBOSS_SIZE Macro
      - GOLSchemeDefault Variable
      - RGBConvert Macro

Descriptions for other APIs that affect the style scheme are found in the Graphics Library help file.

../Microchip/Help

Graphics Library Help.chm

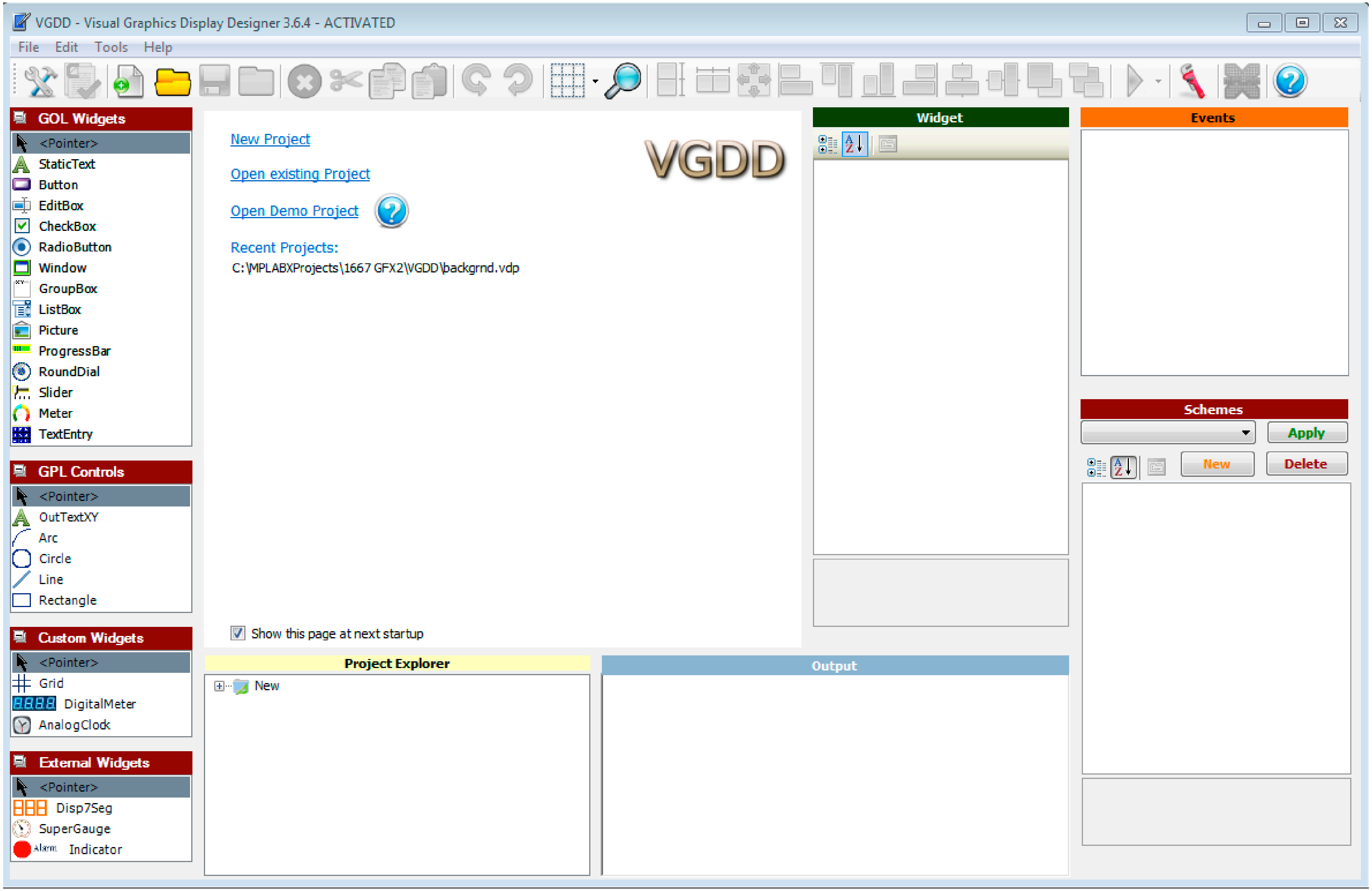
Graphics Library Help.pdf

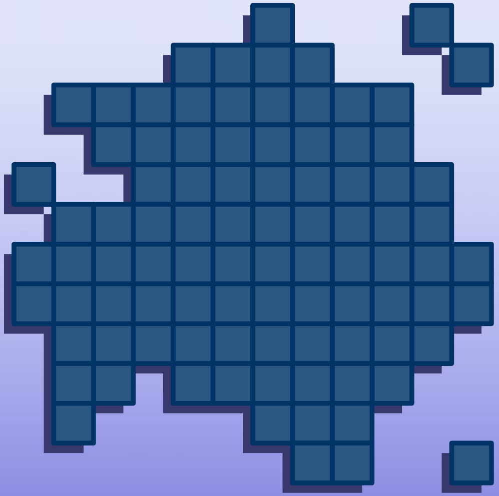


# Microchip Graphics Library

## Getting Started w/ VGDD
## (Visual Graphics Display Designer)

# Visual Graphics Display Designer
## FOLLOW ALONG DEMO





# Lab Exercise 1

## Create a Splash Screen

# Lab 1

## Create a Splash Screen

? Purpose

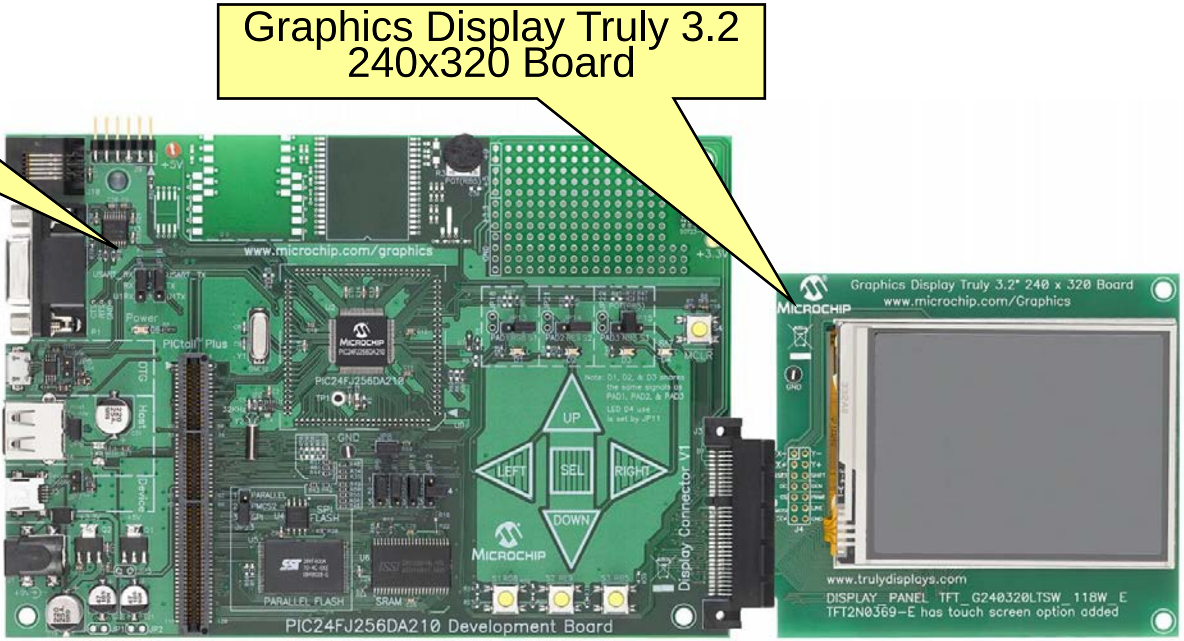
- In this exercise, you will demonstrate your ability to use the Microchip Graphics Library primitive layer functions to create a splash screen for our end application
- If time allows, you will explore the use of gradients and transparency to enhance the splash screen appearance

# Lab Exercise 1
# Create a Splash Screen

## Objective

- Use the Graphics Resource Converter to generate font and image files for use in the application code
- Use primitive layer functions to render image and text strings to the display
- Explore how gradients and transparency can enhance the splash screen appearance

# Low Cost Development Tools


PIC24FJ256DA210 Development Board

Graphics Display Truly 3.2 240x320 Board

OR

- PIC24FJ256DA210 Development Board (DM240312)
- Graphics Display Board (AC164127-x)
- MPLAB® ICD 3 In-Circuit Debugger (DV164035)
  - OR
- MPLAB REAL ICE™ In-Circuit Emulator (DV244005)

# LCC and DA210

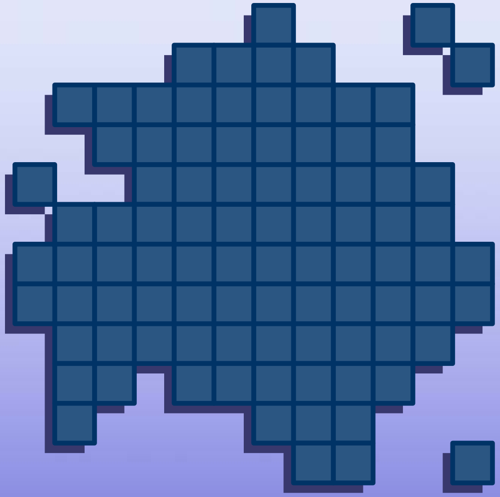
- DA210:
- Built in graphics accelerator including: font rendering, CLUT, bitBLT, and accelerated bar draw functions.
- 16 MIPS microcontroller.
- 76k internal frame buffer
- Max pixel clock support VGA at 30Hz
- LCC:
- 80 MIPS controller (this will get faster as the PIC32 portfolio grows)
- 128k internal frame buffer. (this will get larger as the PIC32 portfolio grows)
- Pixel Clock Jitter in External Frame Buffer mode (will not be present on updated PICs)
- Max pixel clock support VGA at 20Hz
- Can be done in 8bit PMP mode (frees up IO pins).

# Lab 1

## Create a Splash Screen

Procedure

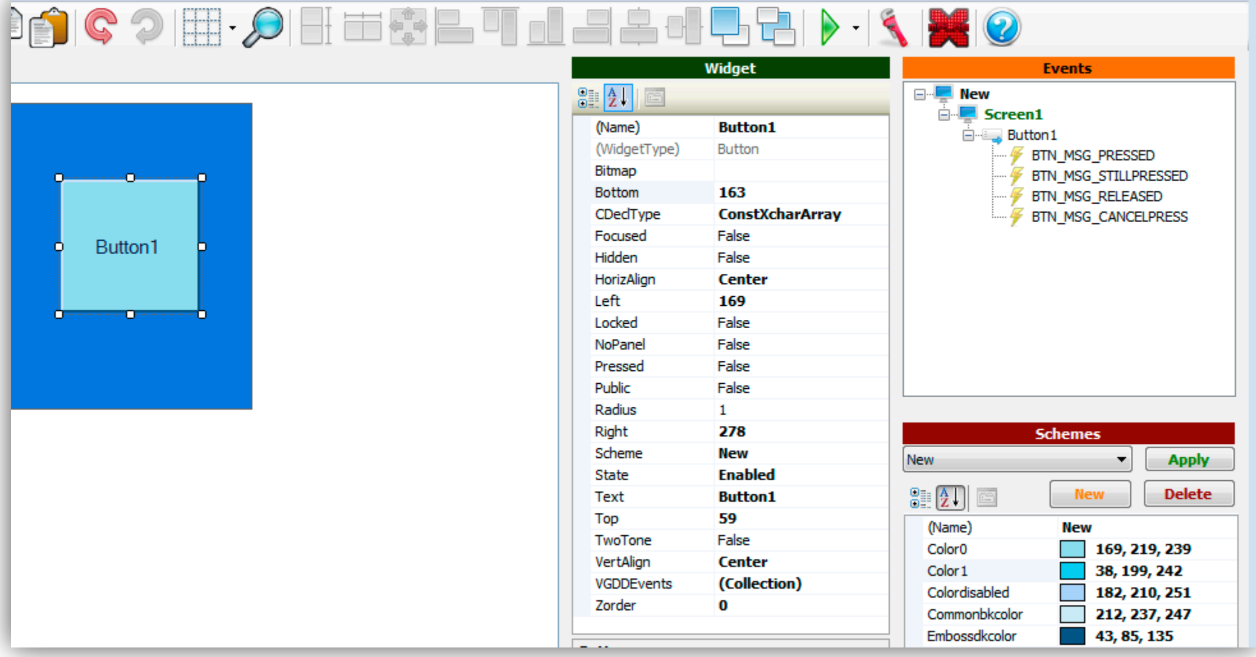
- **Follow the instructions in the lab manual starting on page 1-1**



# Lab Exercise 2

## Create a Menu Screen

# VGDD Style Schemes



| (Name) | Button1 |
|---|---|
| (WidgetType) | Button |
| Bitmap | |
| Bottom | 163 |
| CDeclType | ConstXcharArray |
| Focused | False |
| Hidden | False |
| HorizAlign | Center |
| Left | 169 |
| Locked | False |
| NoPanel | False |
| Pressed | False |
| Public | False |
| Radius | 1 |
| Right | 278 |
| Scheme | New |
| State | Enabled |
| Text | Button1 |
| Top | 59 |
| TwoTone | False |
| VertAlign | Center |
| VGDDEvents | (Collection) |
| Zorder | 0 |

- New
  - Screen1
    - Button1
      - BTN_MSG_PRESSED
      - BTN_MSG_STILLPRESSED
      - BTN_MSG_RELEASED
      - BTN_MSG_CANCELPRESS

New

| (Name) | New |
|---|---|
| Color0 | ■ 169, 219, 239 |
| Color1 | ■ 38, 199, 242 |
| Colordisabled | ■ 182, 210, 251 |
| Commonbkcolor | ■ 212, 237, 247 |
| Embossdkcolor | ■ 43, 85, 135 |

- VGDD contains a style scheme manager
- Demo – Use the style scheme manager to alter style schemes

# Lab 2

## Create a Menu Screen

## ? Purpose

- In this exercise, you will demonstrate your ability to use the VGDD to create a menu screen for our end application
- You will enhance the icons through the use of transparency

# Lab 2

## Create a Menu Screen

### Objective

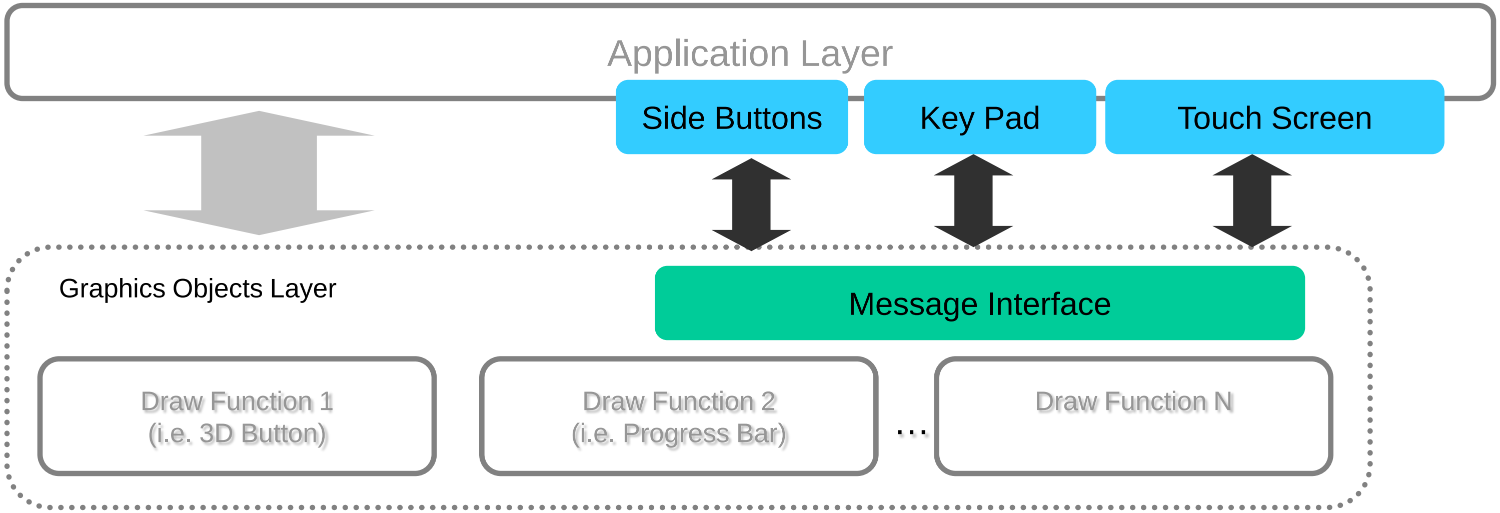
- Use VGDD to create a screen with:
    - 3 Rounded Buttons (one with text, two with image(no text))
    - 3 Static Texts to label the Buttons
- In VGDD, create and apply new style schemes
- Generate the code and program the microcontroller to examine the screen

# Microchip Graphics Library

## Interfacing the User:

## GOL Messaging Interface

# Message Interface

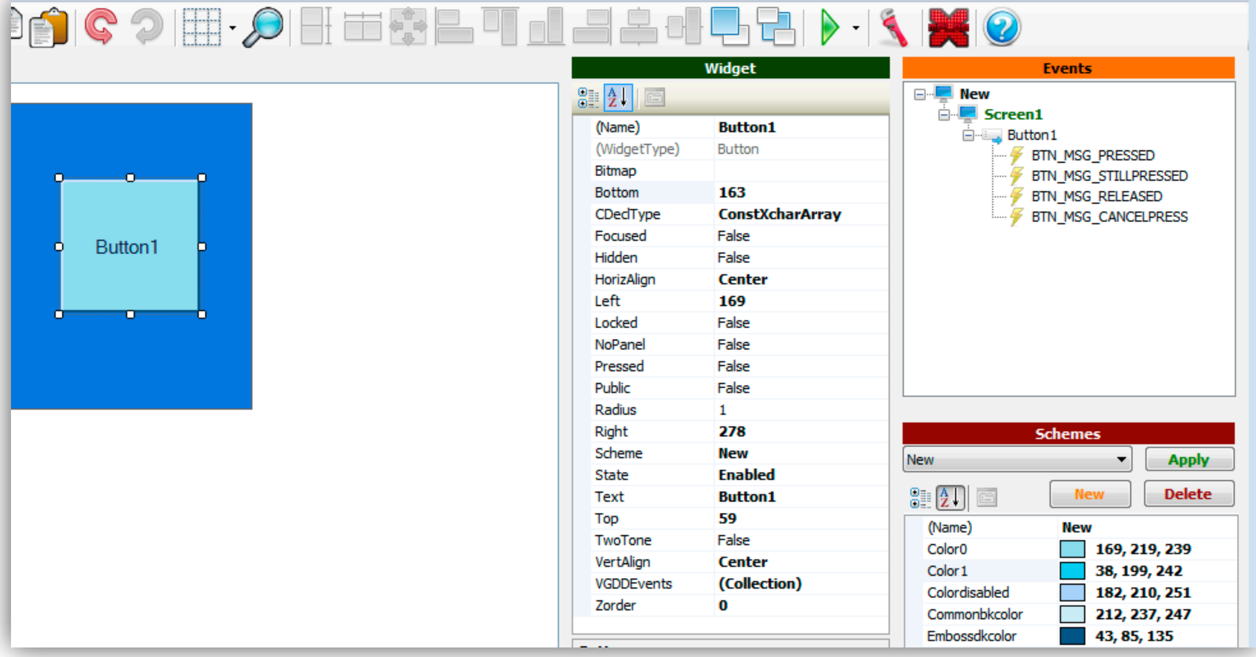


- Simplifies integration of user input devices
- Allow application to efficiently manage widgets
- Provides seamless interface for the user

# Interfacing the User Application Requirements Part 1

- Detect user input
    - Touchscreen driver files included in library
    - Sidebutton and keyboard detection up to user
- Populate message structure (`GOL_MSG` type)
    - Based on the input detected
- Call `GOLMsg(&msg)`
    - Where `&msg` is address of message structure
- `GOLMsg(&msg)` translates the message
    - Translation determines action (e.g. button looks pressed or not)
    - Application may customize widget response

# VGDD Event Handling

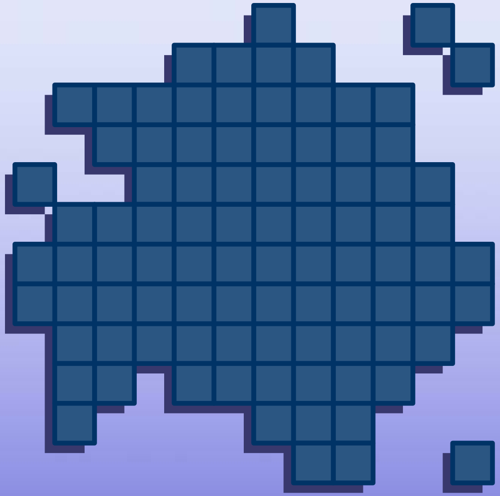


| | |
|---|---|
| (Name) | **Button1** |
| (WidgetType) | Button |
| Bitmap | |
| Bottom | **163** |
| CDeclType | **ConstXcharArray** |
| Focused | False |
| Hidden | False |
| HorizAlign | **Center** |
| Left | **169** |
| Locked | False |
| NoPanel | False |
| Pressed | False |
| Public | False |
| Radius | 1 |
| Right | **278** |
| Scheme | **New** |
| State | **Enabled** |
| Text | **Button1** |
| Top | **59** |
| TwoTone | False |
| VertAlign | **Center** |
| VGDDEvents | **(Collection)** |
| Zorder | **0** |

- **New**
  - **Screen1**
    - Button1
      - BTN_MSG_PRESSED
      - BTN_MSG_STILLPRESSED
      - BTN_MSG_RELEASED
      - BTN_MSG_CANCELPRESS

| | |
|---|---|
| (Name) | **New** |
| Color0 | **169, 219, 239** |
| Color1 | **38, 199, 242** |
| Colordisabled | **182, 210, 251** |
| Commonbkcolor | **212, 237, 247** |
| Embossdkcolor | **43, 85, 135** |

- VGDD contains an event handler
- Allows customization of widget behavior
- But first, we need to learn a little more about widgets and events …



# Microchip Graphics Library

## Interfacing the User:

## Widget States

# Widget States

- 16-bit value used to represent the widgets state
  - Member of widget structure
- Drawing states indicate widget needs to be:
  - Fully drawn
  - Partially drawn
  - Hidden
- Property states define action and appearance
  - Button is pressed or released
  - Text alignment in a static text widget
- Macros are used to identify state values
  - Example: `BTN_DRAW` indicates button should be fully drawn
  - Logical OR used to combine states
  - Example: `BTN_DRAW | BTN_PRESSED` indicates button should be drawn in the pressed state

# Button Drawing States

| Statebit Macro | Description |
|---|---|
| `BTN_DISABLED` | Button will not accept messages |
| `BTN_DRAW` | Button must be fully redrawn |
| `BTN_DRAW_FOCUS` | Button focus must be redrawn |
| `BTN_FOCUS` | Button is focused |
| `BTN_HIDE` | Button must be covered with CommonBkColor |
| `BTN_PRESSED` | Button is in the pressed state |
| `BTN_TEXTBOTTOM` | Button text is bottom aligned |
| `BTN_TEXTTOP` | Button text is top aligned |
| `BTN_TEXTLEFT` | Button text is left aligned |
| `BTN_TEXTRIGHT` | Button text is right aligned |
| `BTN_TOGGLE` | Button will have a toggle behavior |
| `BTN_TWOTONE` | Button will be flat, with two colors |
| `BTN_NOPANEL` | Button will be created, but not drawn (icons) |

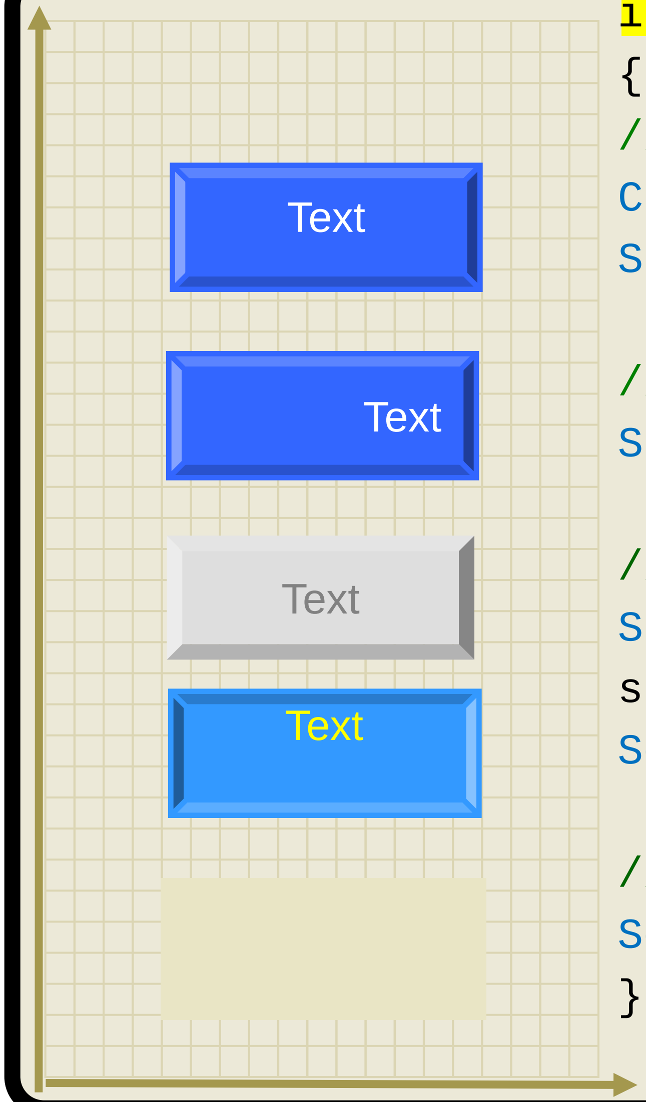
# Button Property States

| Statebit Macro | Description |
|---|---|
| `BTN_DISBABLED` | Button will not accept messages |
| `BTN_DRAW` | Button must be fully redrawn |
| `BTN_DRAW_FOCUS` | Button focus must be redrawn |
| `BTN_FOCUS` | Button is focused |
| `BTN_HIDE` | Button must be covered with CommonBkColor |
| `BTN_PRESSED` | Button is in the pressed state |
| `BTN_TEXTBOTTOM` | Button text is bottom aligned |
| `BTN_TEXTTOP` | Button text is top aligned |
| `BTN_TEXTLEFT` | Button text is left aligned |
| `BTN_TEXTRIGHT` | Button text is right aligned |
| `BTN_TOGGLE` | Button will have a toggle behavior |
| `BTN_TWOTONE` | Button will be flat, with two colors |
| `BTN_NOPANEL` | Button will be created, but not drawn (icons) |

# State Macros

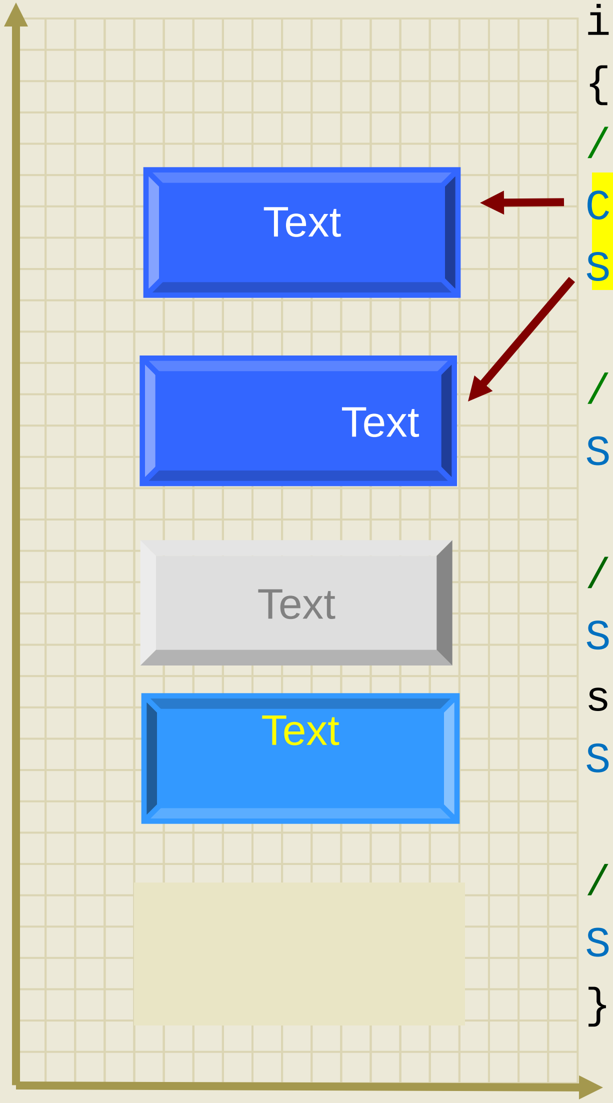
- `SetState(pObj, state)`
    - Set specified state
    - States remain set until cleared
- `ClrState(pObj, state)`
    - Clear specified state
- `GetState(pObj, state)`
    - Get a widget’s state
- Where…
    - `pObj` = pointer to the widget structure
    - `state` = desired state

# State Bits Example
# Button Widget



```
if (GOL_DRAW())
{
// Example Button text alignments
ClrState(pBtn, BTN_PRESSED);
SetState(pBtn, BTN_TEXTRIGHT);

// Example Button Focus
SetState(pBtn, BTN_FOCUSED);

// Example Button Action
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTLEFT;
SetState(pBtn, state);

// Example Hiding Button
SetState(pBtn, BTN_HIDE)
}
```

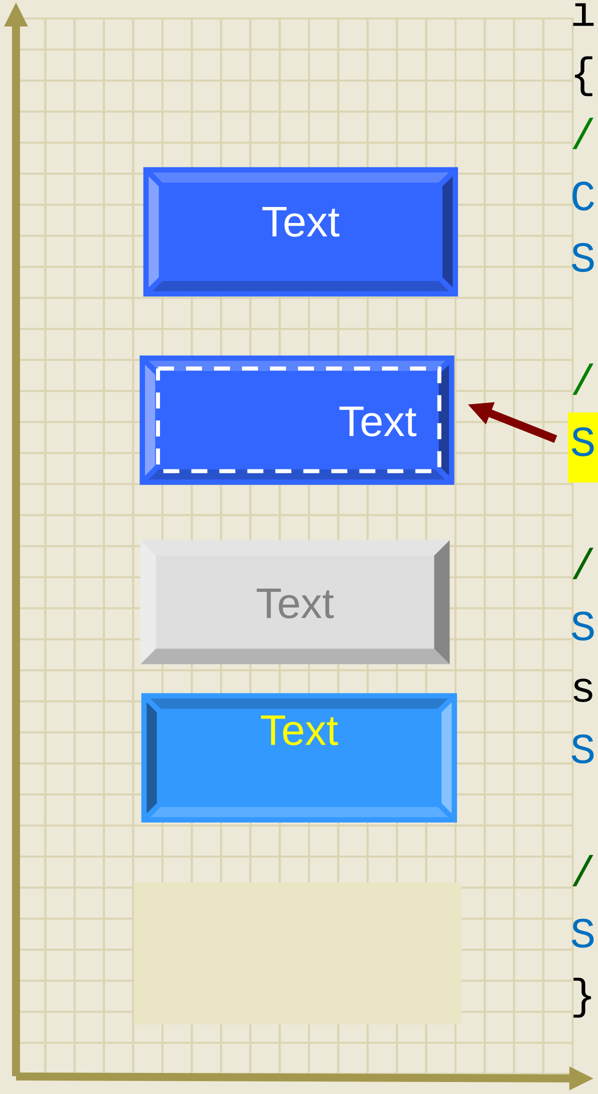
# Button States



```
if (GOL_DRAW())
{
// Example Button text alignments
ClrState(pBtn, BTN_PRESSED);
SetState(pBtn, BTN_TEXTRIGHT);

// Example Button Focus
SetState(pBtn, BTN_FOCUSED);

// Example Button Action
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTLEFT;
SetState(pBtn, state);

// Example Hiding Button
SetState(pBtn, BTN_HIDE)
}
```

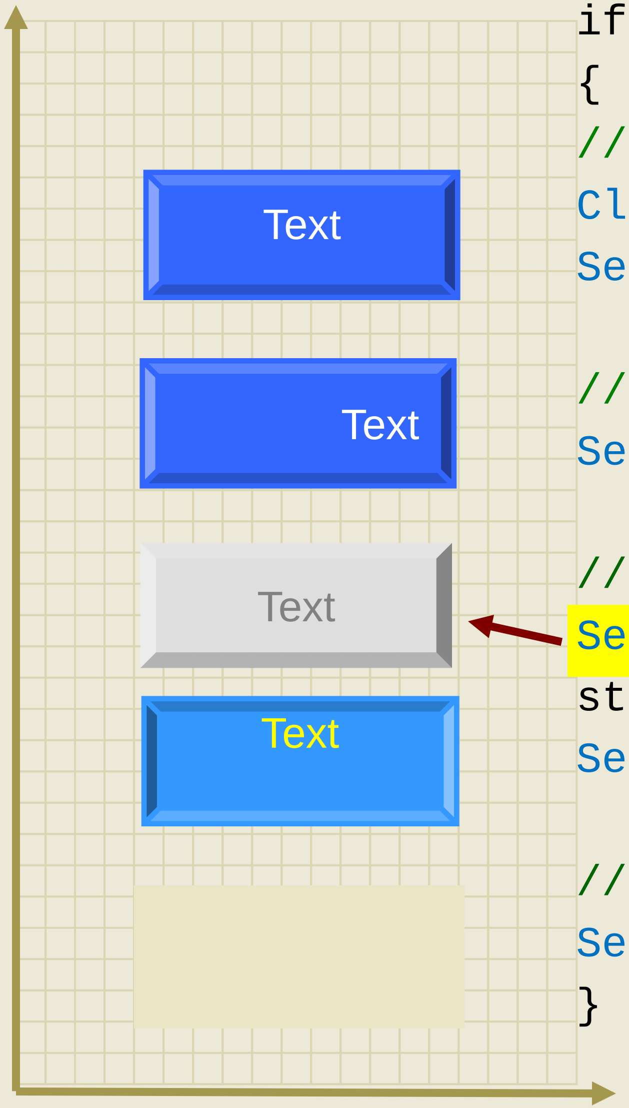
Microchip Regional Training Centers

# Button States



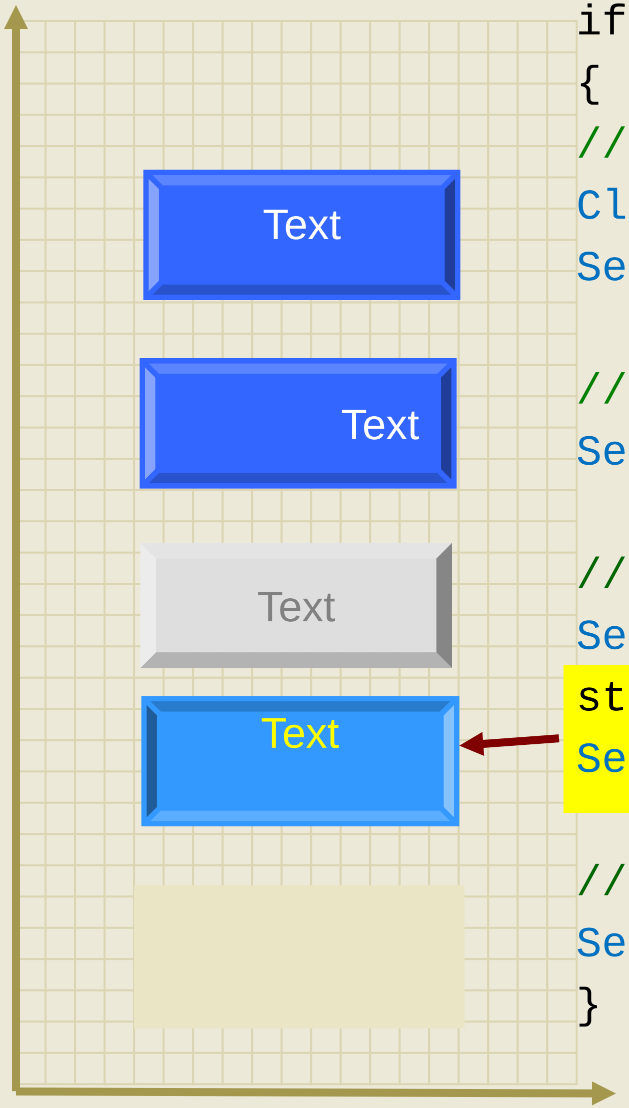
```
if (GOL_DRAW())
{
// Example Button text alignments
ClrState(pBtn, BTN_PRESSED);
SetState(pBtn, BTN_TEXTRIGHT);

// Example Button Focus
SetState(pBtn, BTN_FOCUSED);

// Example Button Action
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTLEFT;
SetState(pBtn, state);

// Example Hiding Button
SetState(pBtn, BTN_HIDE)
}
```

# Button States



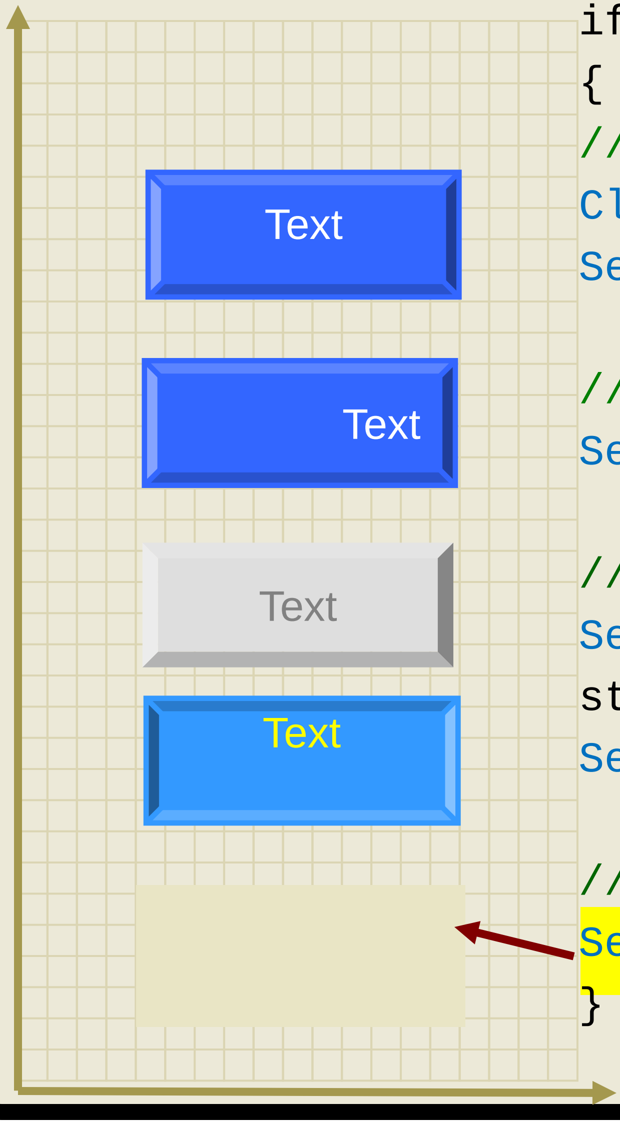
```
if (GOL_DRAW())
{
// Example Button text alignments
ClrState(pBtn, BTN_PRESSED);
SetState(pBtn, BTN_TEXTRIGHT);

// Example Button Focus
SetState(pBtn, BTN_FOCUSED);

// Example Button Action
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTLEFT;
SetState(pBtn, state);

// Example Hiding Button
SetState(pBtn, BTN_HIDE)
}
```

# Button States



```
if (GOL_DRAW())
{
// Example Button text alignments
ClrState(pBtn, BTN_PRESSED);
SetState(pBtn, BTN_TEXTRIGHT);

// Example Button Focus
SetState(pBtn, BTN_FOCUSED);

// Example Button Action
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTLEFT;
SetState(pBtn, state);

// Example Hiding Button
SetState(pBtn, BTN_HIDE)
}
```

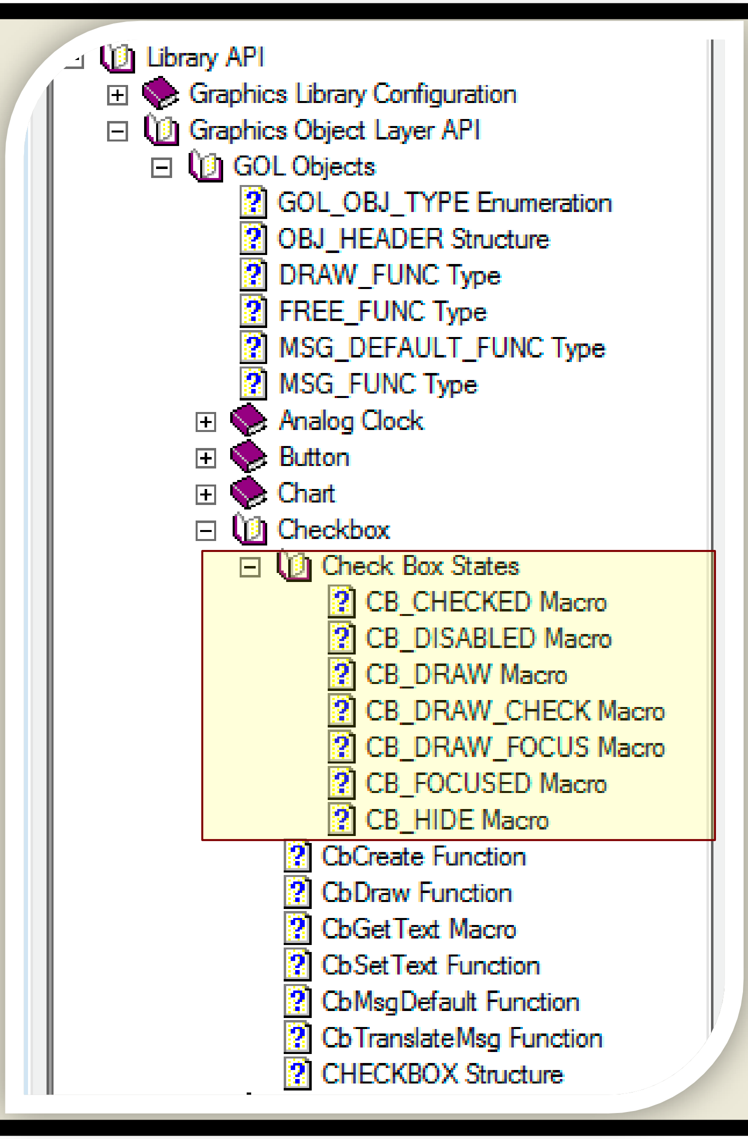
# Button States



```
if (GOL_DRAW())
{
// Example Button text alignments
ClrState(pBtn, BTN_PRESSED);
SetState(pBtn, BTN_TEXTRIGHT);

// Example Button Focus
SetState(pBtn, BTN_FOCUSED);

// Example Button Action
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTLEFT;
SetState(pBtn, state);

// Example Hiding Button
SetState(pBtn, BTN_HIDE)
}
```

Microchip Regional Training Centers

# Widget State Help



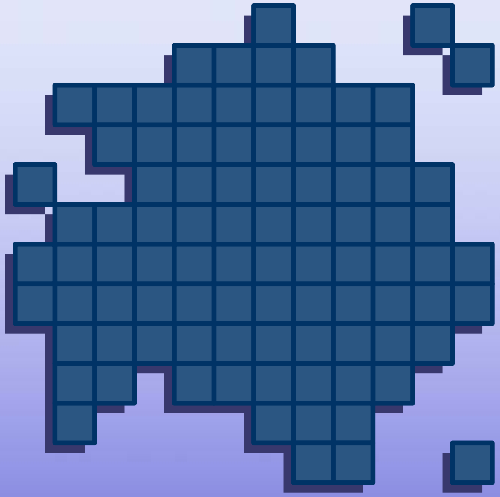
- Library API
  - Graphics Library Configuration
  - Graphics Object Layer API
    - GOL Objects
      - GOL_OBJ_TYPE Enumeration
      - OBJ_HEADER Structure
      - DRAW_FUNC Type
      - FREE_FUNC Type
      - MSG_DEFAULT_FUNC Type
      - MSG_FUNC Type
    - Analog Clock
    - Button
    - Chart
    - Checkbox
      - Check Box States
        - CB_CHECKED Macro
        - CB_DISABLED Macro
        - CB_DRAW Macro
        - CB_DRAW_CHECK Macro
        - CB_DRAW_FOCUS Macro
        - CB_FOCUSED Macro
        - CB_HIDE Macro
      - CbCreate Function
      - CbDraw Function
      - CbGetText Macro
      - CbSetText Function
      - CbMsgDefault Function
      - CbTranslateMsg Function
      - CHECKBOX Structure

Every widget also has unique state bits. These can be found in the library help file as shown…

../Microchip/Help

Graphics Library Help.chm

Graphics Library Help.pdf



# Microchip Graphics Library

## Referencing Widgets

# Pointer to Widget

## Assign pointer when widget is created

```
void CreateButtons(void)
{
…
BUTTON       *pBtn
#define       ID_BTN2     16
…
…
pBtn =       BtnCreate(
             ID_BTN2,     // 2nd Button ID
             x3, y3,      // left, top
             x4, y4,      // right, bottom
             Radius,      // Rounded edges
             BTN_DRAW,    // Display button
             &arrow,      // use this bitmap
             NULL,        // no text
             altScheme);  // style scheme
…
}
```

Declare the Pointer

Assign Widget

# Pointer to Widget Another Way…

- `GOLFindObject(ID)`
  - Returns ADDRESS of the widget
  - Avoid mismatch warning by typecasting

Declare the Pointer

Assign Widget

```
BUTTON *pBtn
…


pBtn =   (BUTTON*)GOLFindObject(ID_BTN1);

SetState (pBtn, BTN_DRAW);
pBtn =   (BUTTON*)GOLFindObject(ID_BTN2);
SetState (pBtn, BTN_DRAW);
```

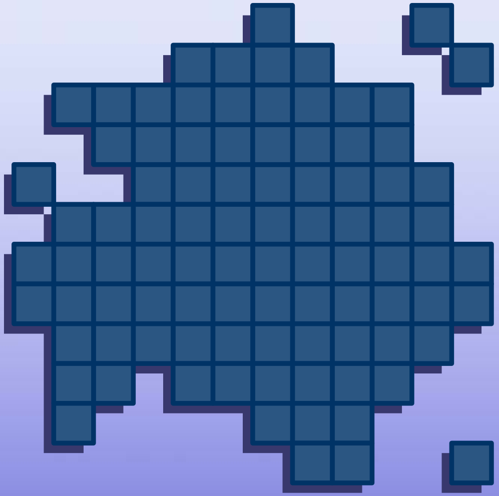
# Widget Identifier

- `GetObjID(*pObj)`
  - Returns ID member of the widget structure

```
#define ID_BTN1
BUTTON *pBtn;
…

currObjID = GetObjID(pBtn);
switch(currObjID){
   case ID_BTN1:
          //do something useful
   break;
…}
```

Declare the Pointer

Assign Widget



# Microchip Graphics Library

## Interfacing the User:
## Managing Widgets

# Widget Abbreviations

`Obj =`

| Widget Name | Obj Abbreviation |
|---|---|
| Analog Clock | Ac |
| Button | Btn |
| Chart | Ch |
| Checkbox | Cb |
| Round Dial | Rdia |
| Digital Meter | Dm |
| Edit Box | Eb |
| Grid | Grid |
| Group Box | Gb |
| List Box | Lb |
| Meter | Mtr |
| Picture Control | Pict |
| Progress Bar | Pb |
| Radio Button | Rb |
| Slider/Scroll Bar | Sld |
| Static Text | St |
| Text Entry | Te |
| Window | Wnd |

# Widget Management APIs

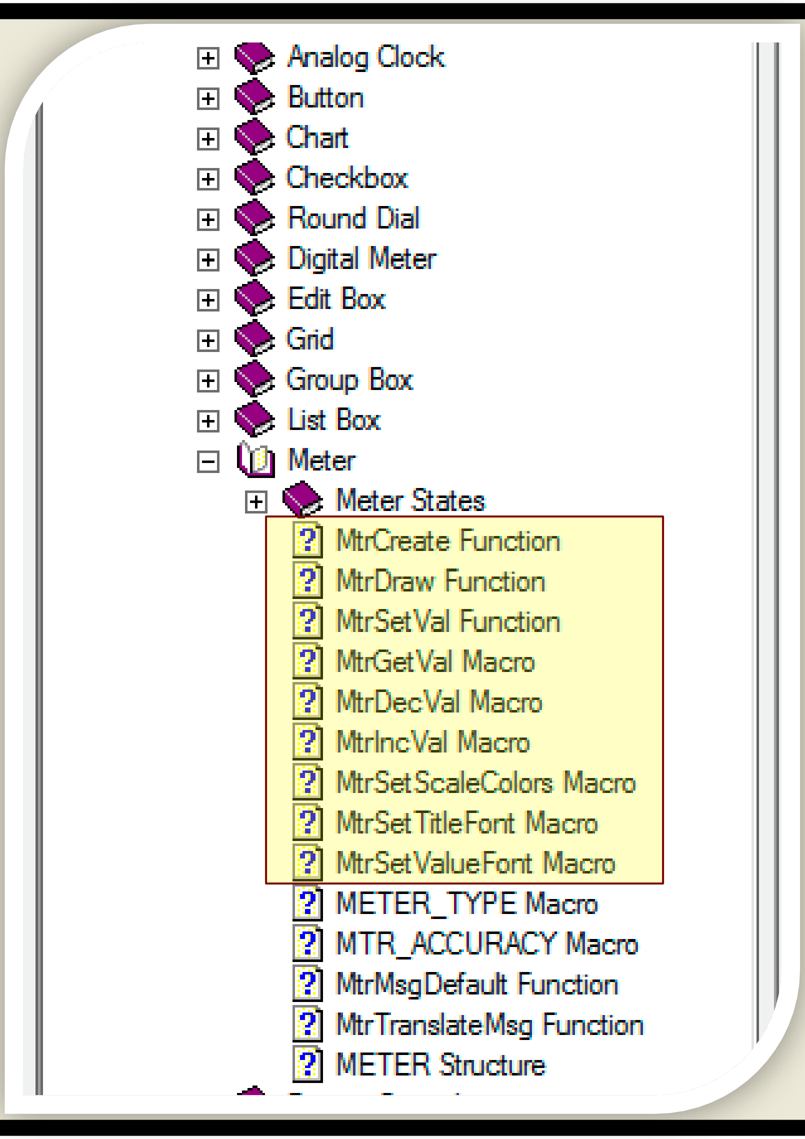
- Widget's with text strings
  - `ObjSetText(*pObj, *pText)`
  - `ObjGetText(*pObj)`
    - Returns a pointer to the text string in use
  - Works with button, checkbox, edit box, group box, radio button, static text, and window widgets

- Widget's with images:
  - `ObjSetBitmap(*pObj, *pImage)`
  - `ObjGetBitmap(*pObj)`
    - Returns a pointer to the bitmap in use
  - Works with picture and button widgets

# Widget Management APIs

- Positional Widgets
  - `ObjSetPos(*pObj, posValue)`
  - `ObjGetPos(*pObj)`
    - Returns the current position
  - `ObjSetRange(*pObj, rangeValue)`
  - `ObjGetRange(*pObj)`
    - Returns the current range
  - Works with progress bar and slider widgets
- Widgets have various management APIs available
- Refer to library help file for more details

# Widget Management API Help



- Analog Clock
- Button
- Chart
- Checkbox
- Round Dial
- Digital Meter
- Edit Box
- Grid
- Group Box
- List Box
- Meter
  - Meter States
  - MtrCreate Function
  - MtrDraw Function
  - MtrSetVal Function
  - MtrGetVal Macro
  - MtrDecVal Macro
  - MtrIncVal Macro
  - MtrSetScaleColors Macro
  - MtrSetTitleFont Macro
  - MtrSetValueFont Macro
  - METER_TYPE Macro
  - MTR_ACCURACY Macro
  - MtrMsgDefault Function
  - MtrTranslateMsg Function
  - METER Structure

The widget management APIs are found under the individual widgets in the Graphics Object Layer section of the help file.

../Microchip/Help

Graphics Library Help.chm

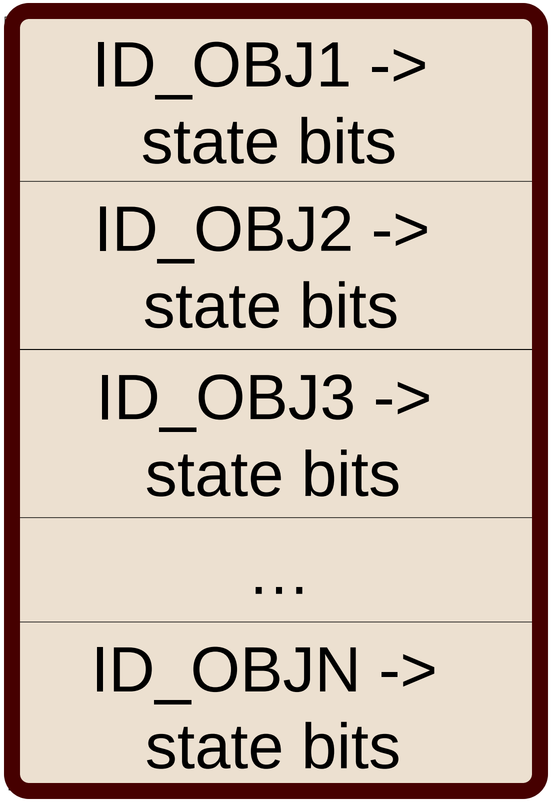
Graphics Library Help.pdf

# Microchip Graphics Library

## Receiving Messages

# Receiving Messages

Linked List



ID_OBJ1 -> state bits

ID_OBJ2 -> state bits

ID_OBJ3 -> state bits

…

ID_OBJN -> state bits

- `GOLMsg(&msg)`
  - User must supply pointer to a message structure
  - Immediately returns if `EVENT_INVALID`
  - Translates message
  - Finds affected widget in linked list
  - Modifies widget state
  - Returns TRUE when done

# Message Structure

- Message Structure

```
typedef struct {
  BYTE           type;
 BYTE          uiEvent;
  SHORT         param1;
  SHORT         param2;
   } GOL_MSG;
```

- `type`:    identify input device type
- `uiEvent`: identify the event
- `param1`:
- `param2`: provide information based on `type` and `uiEvent`

# Message Structure
# `type = TYPE_TOUCHSCREEN`

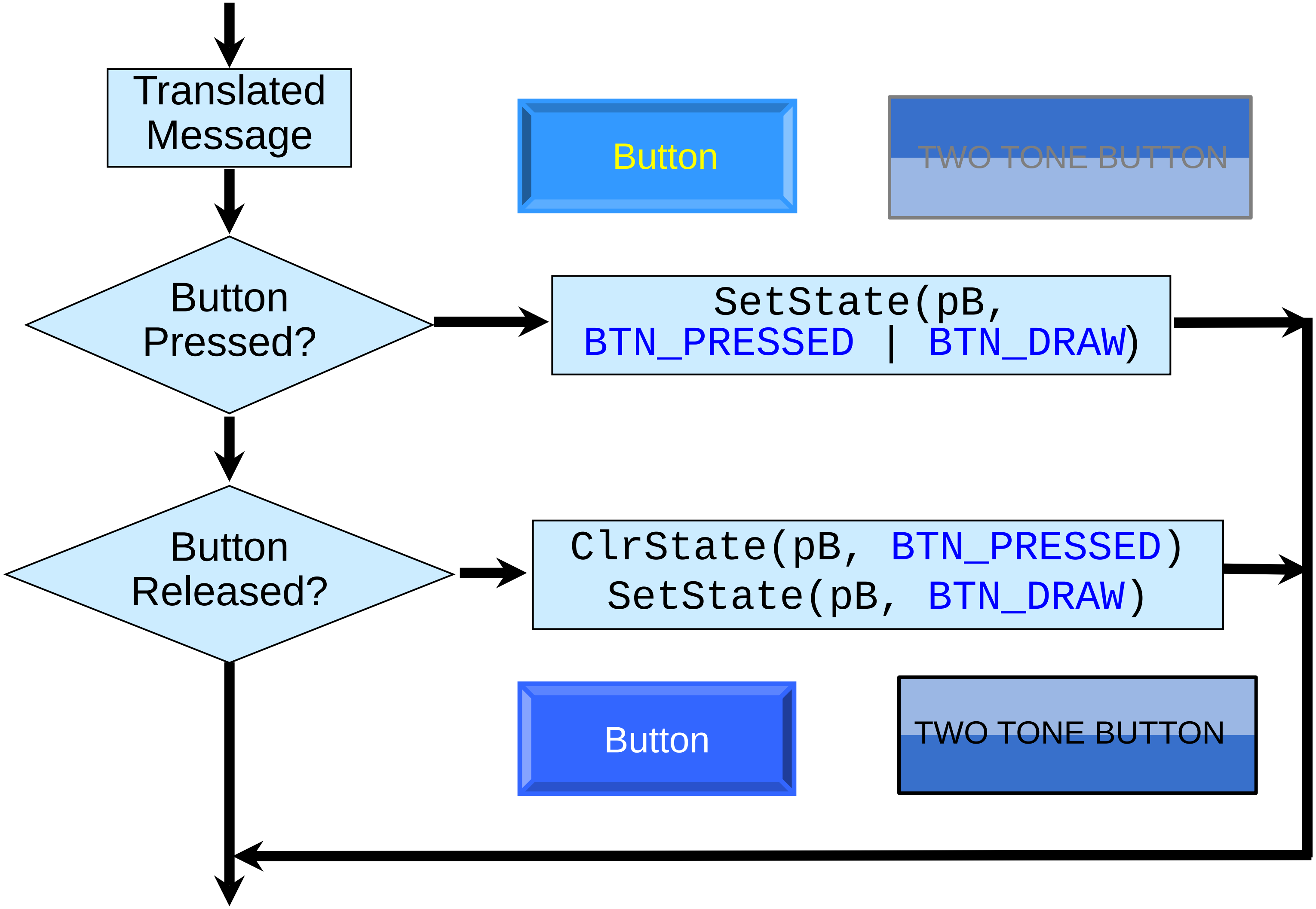
- Valid `uiEvent` values
  - `EVENT_PRESS`
  - `EVENT_STILLPRESS`
  - `EVENT_RELEASE`
  - `EVENT_MOVE`
  - `EVENT_INVALID`
    - No touch
- `param1` = x coordinate of touch
- `param2` = y coordinate of touch

# Message Structure
# `type = TYPE_KEYBOARD`

- `param1` = Object identifier of receiving widget
- Valid `uiEvent` values
  - `EVENT_KEYSCAN`
    - `param2` = Scan Code
  - `EVENT_CHARCODE`
    - `param2` = character to add
  - `EVENT_INVALID`
    - No key pressed

AT keyboard scan codes are provided in the library help file

# Widget Default Actions
# Button



Translated Message

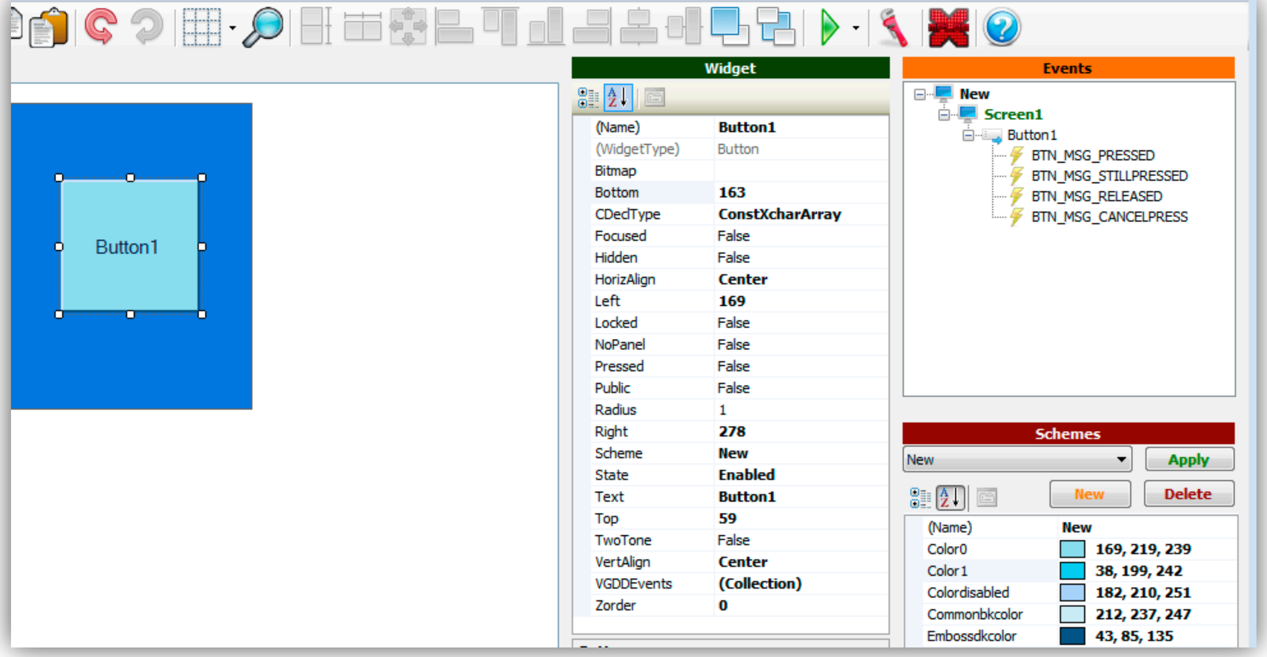
Button Pressed?

```
SetState(pB,
BTN_PRESSED | BTN_DRAW)
```

Button Released?

```
ClrState(pB, BTN_PRESSED)
SetState(pB, BTN_DRAW)
```

# VGDD Event Handling



**Widget**

| (Name) | Button1 |
|---|---|
| (WidgetType) | Button |
| Bitmap | |
| Bottom | 163 |
| CDeclType | ConstXcharArray |
| Focused | False |
| Hidden | False |
| HorizAlign | Center |
| Left | 169 |
| Locked | False |
| NoPanel | False |
| Pressed | False |
| Public | False |
| Radius | 1 |
| Right | 278 |
| Scheme | New |
| State | Enabled |
| Text | Button1 |
| Top | 59 |
| TwoTone | False |
| VertAlign | Center |
| VGDDEvents | (Collection) |
| Zorder | 0 |

**Events**

- New
  - Screen1
    - Button1
      - BTN_MSG_PRESSED
      - BTN_MSG_STILLPRESSED
      - BTN_MSG_RELEASED
      - BTN_MSG_CANCELPRESS

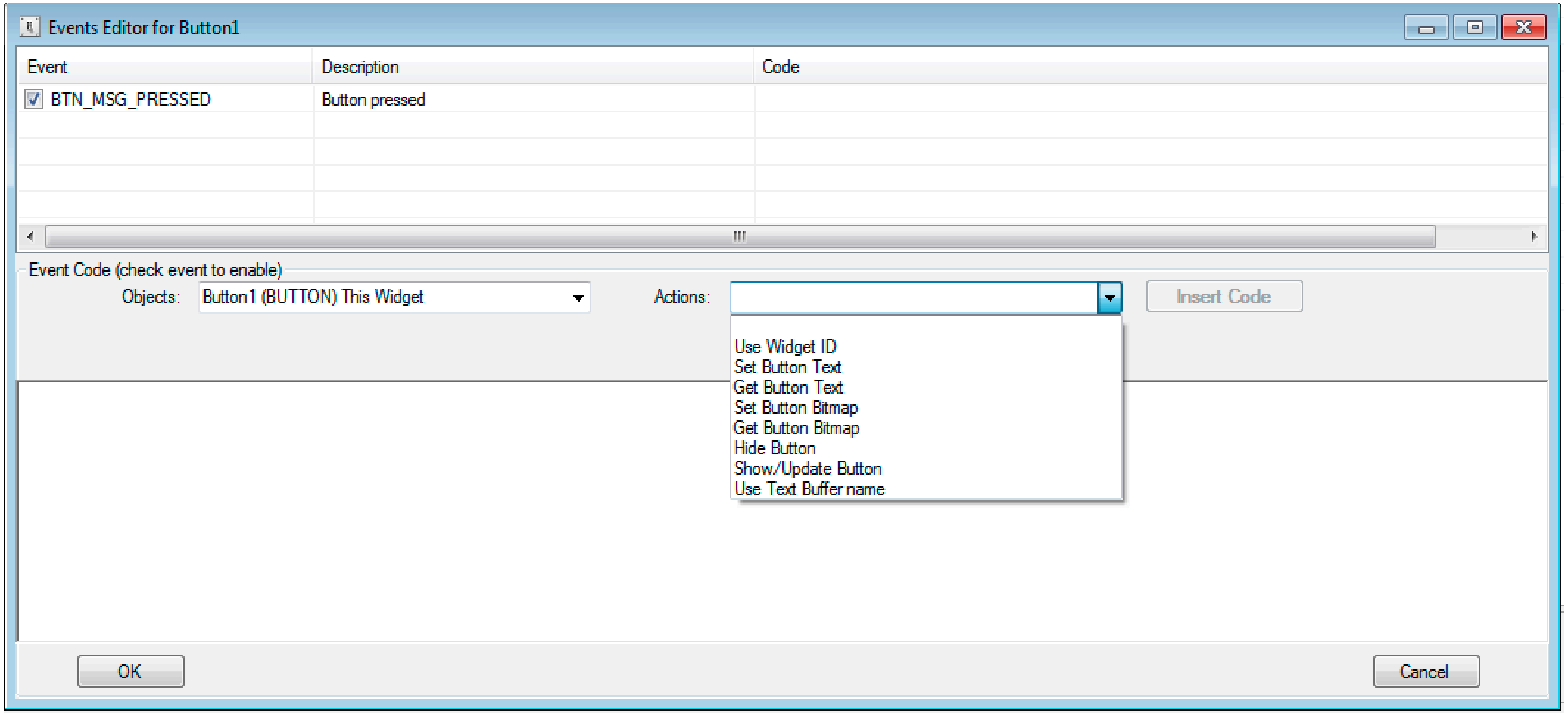
**Schemes**

New ▾ Apply
New Delete

| (Name) | New |
|---|---|
| Color0 | 169, 219, 239 |
| Color1 | 38, 199, 242 |
| Colordisabled | 182, 210, 251 |
| Commonbkcolor | 212, 237, 247 |
| Embossdkcolor | 43, 85, 135 |

- VGDD contains a simulator
- Touchscreen driver included in project
  - Will need to change if using a side buttons
- Demo to show widget default events

# Widget Management APIs VGDD Event Handler



Events Editor for Button1

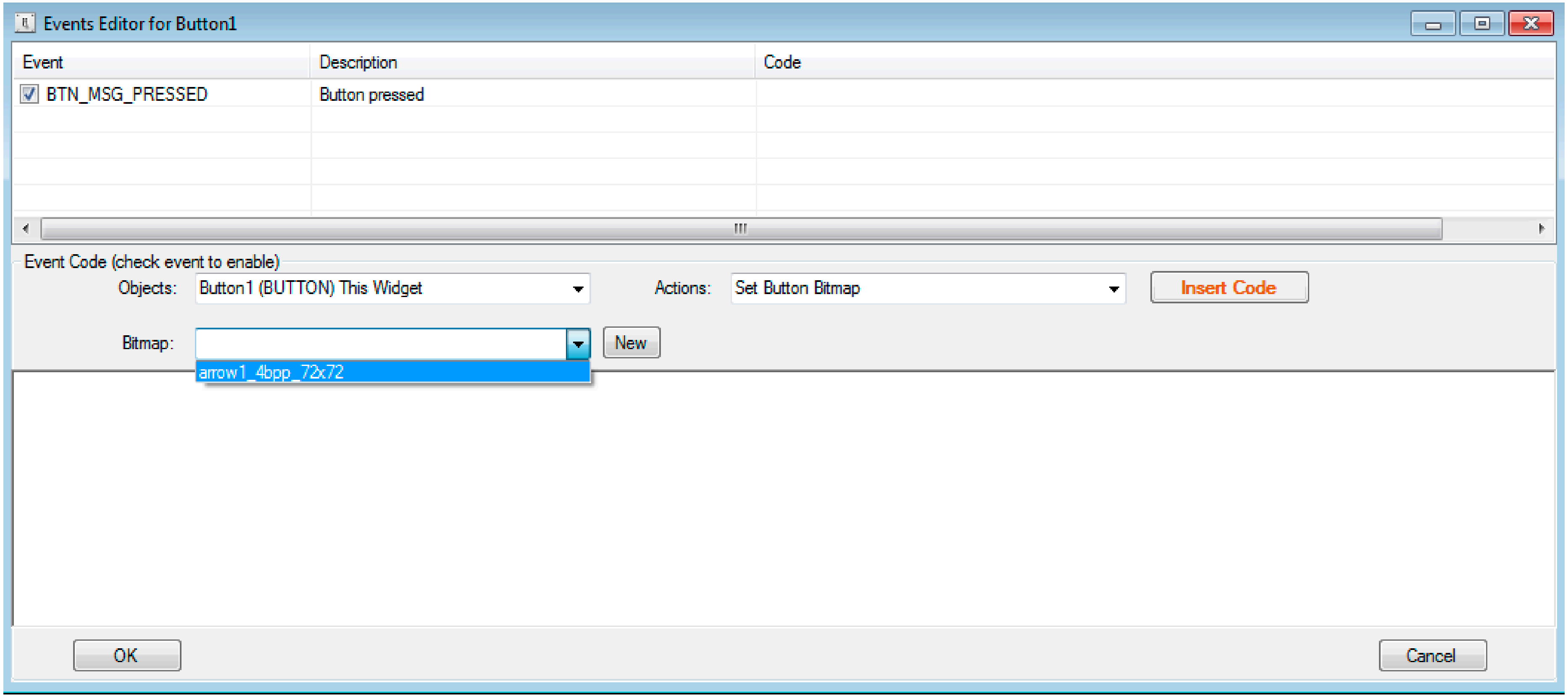
| Event | Description | Code |
|---|---|---|
| ☑ BTN_MSG_PRESSED | Button pressed | |

Event Code (check event to enable)

Objects: Button1 (BUTTON) This Widget

Actions:

# Widget Management APIs VGDD Event Handler



Events Editor for Button1

| Event | Description | Code |
|---|---|---|
| ☑ BTN_MSG_PRESSED | Button pressed | |

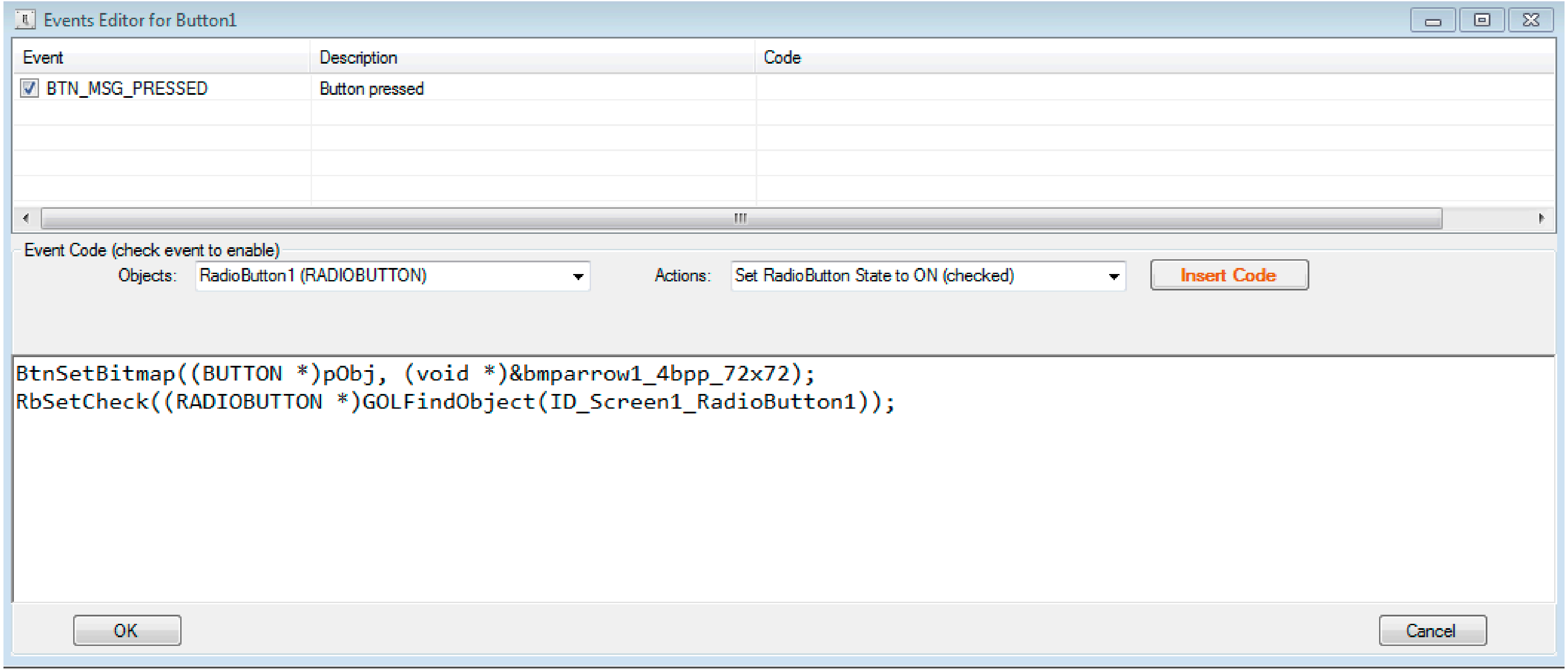
Event Code (check event to enable)

Objects: Button1 (BUTTON) This Widget

Actions: Set Button Bitmap

Bitmap: arrow1_4bpp_72x72

# Widget Management APIs VGDD Event Handler



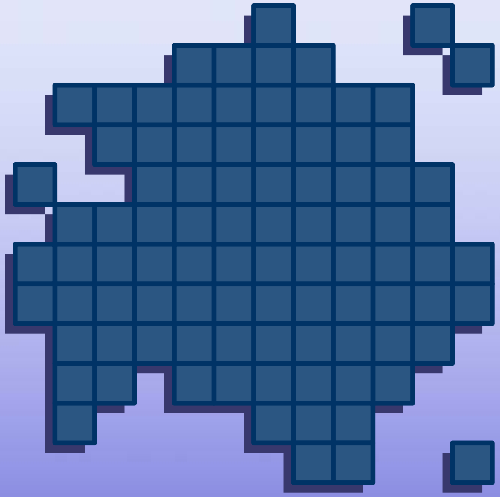
| Event | Description | Code |
|---|---|---|
| ☑ BTN_MSG_PRESSED | Button pressed | |

Event Code (check event to enable)

Objects: RadioButton1 (RADIOBUTTON)

Actions: Set RadioButton State to ON (checked)

```
BtnSetBitmap((BUTTON *)pObj, (void *)&bmparrow1_4bpp_72x72);
RbSetCheck((RADIOBUTTON *)GOLFindObject(ID_Screen1_RadioButton1));
```



# Microchip Graphics Library

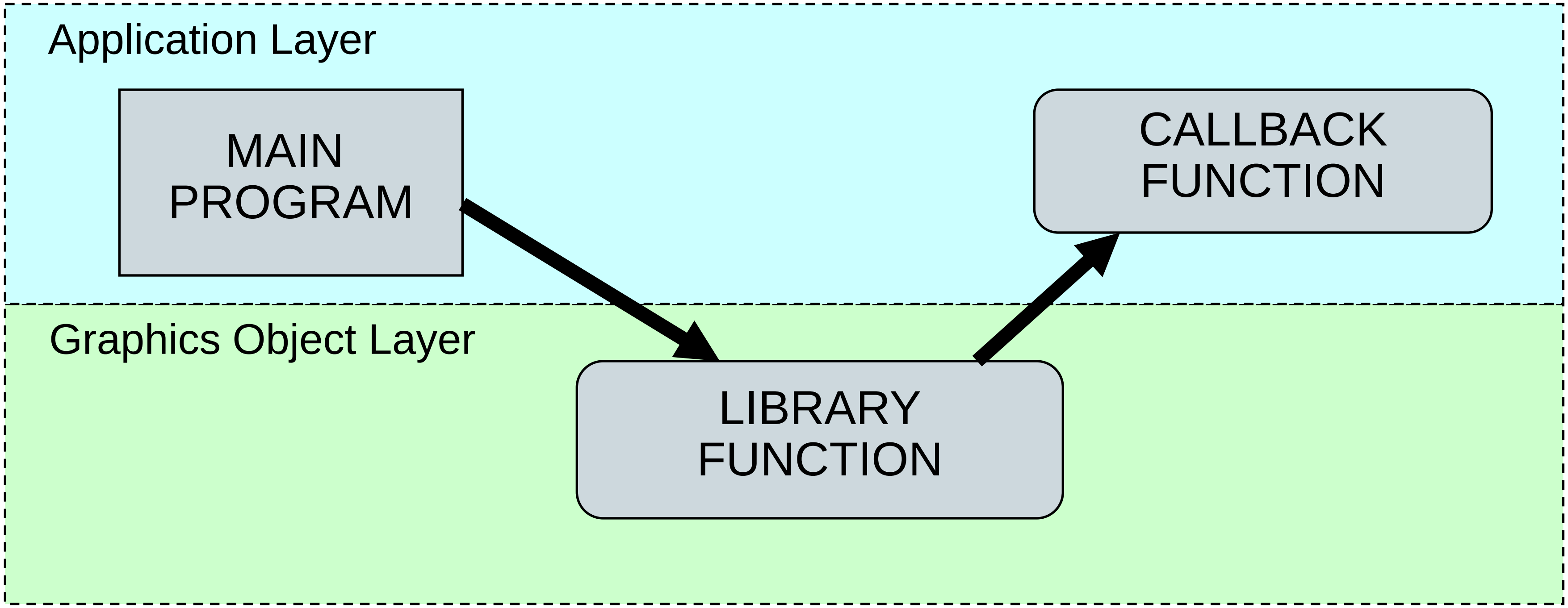
## Customizing widget events

`GOLMsgCallback()`

# What is a Callback?

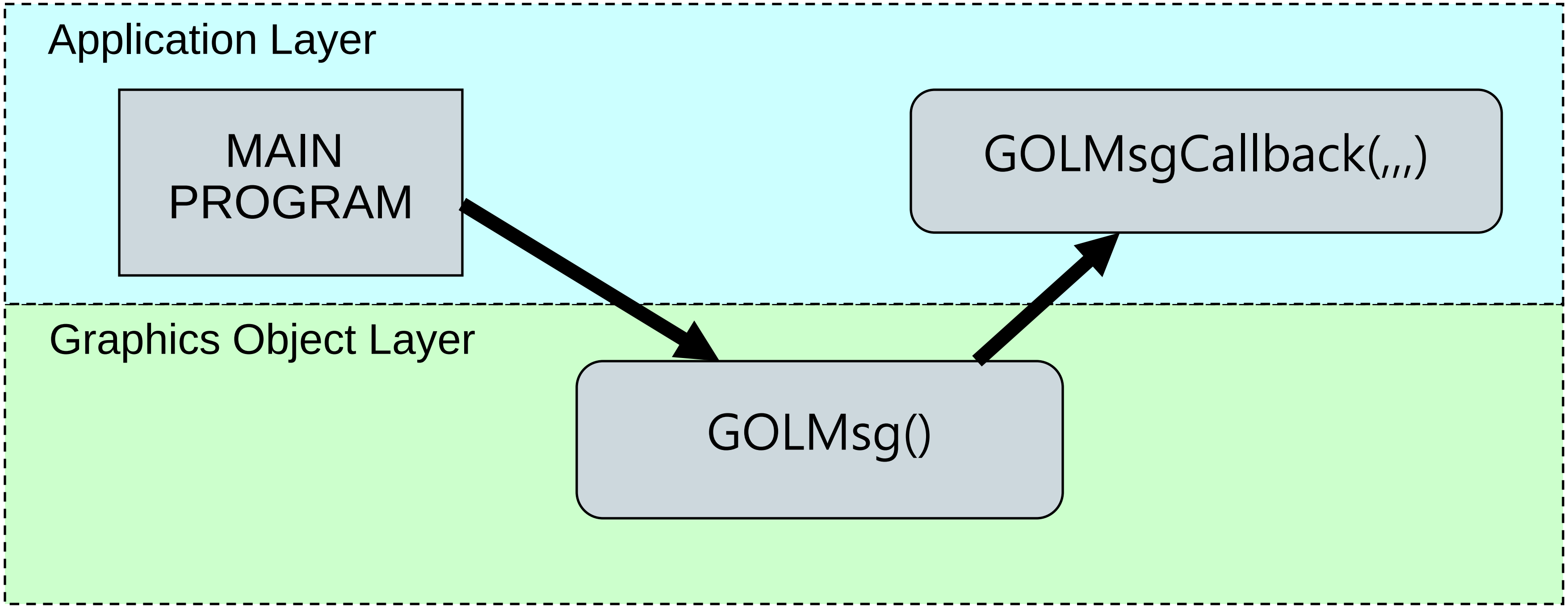
**Definition:**

A callback function allows a lower level software layer to call a subroutine defined in a higher layer.



# `GOLMsgCallback(,,,)`

- Called by `GOLMsg(&msg)`
- Only called if valid message is received



# Interfacing the User Application Requirements Part 1

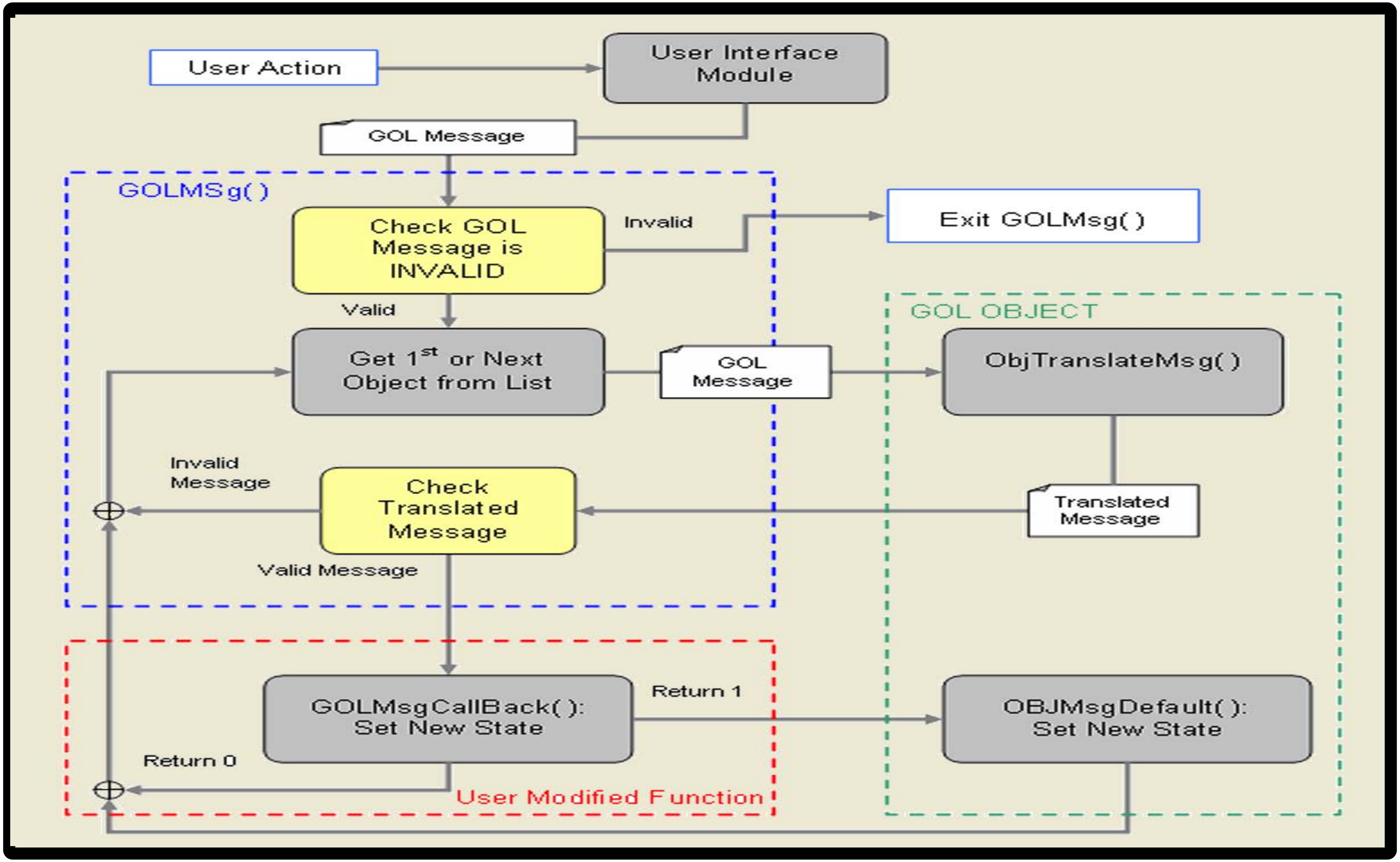
- **Provide callback functions (REQUIRED)**
  - **`GOLMsgCallback(,,,)`**
    - Called by **`GOLMsg()`** only if valid message received
    - Customize response to message event
      - **Example: Change button image on EVENT_PRESS**
      - **Example: Increase volume on EVENT_STILLPRESS**
      - **Example: Change screen state on EVENT_PRESS**
  - **`GOLDrawCallback()`**
    - Called by **`GOLDraw()`**when drawing is complete
    - Not dependent on message events
    - Customized drawing
      - **ONLY safe place to modify drawing properties**
      - **Example: Modify bars to form a signal strength meter**
    - Customized system response
      - **Example: Read a sensor and change a string**
      - **Example: Update a progress bar**

# `GOLMsgCallback(,,,)`

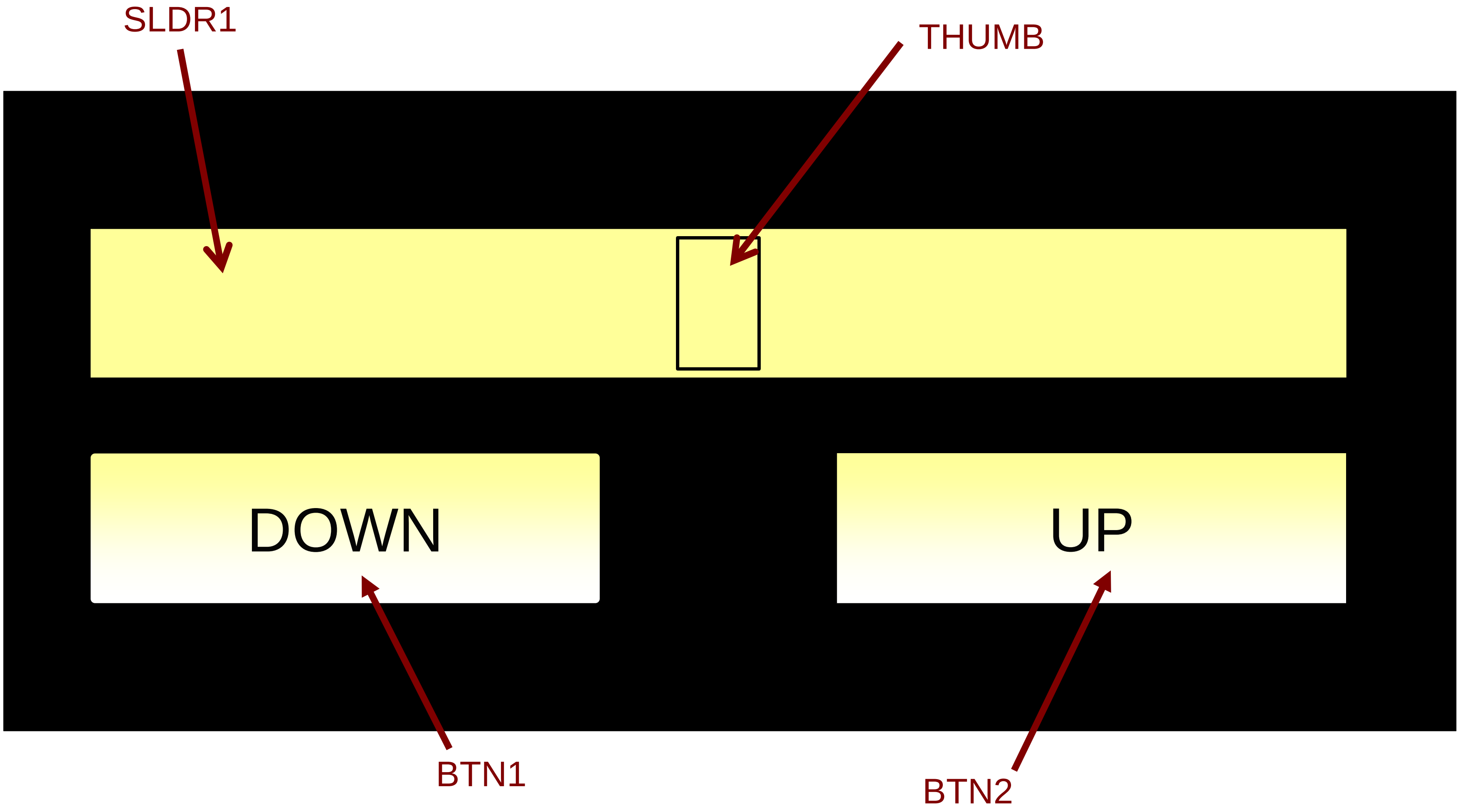
- Called by `GOLMsg(&msg)`
- ***MUST*** be provided in application code
- Perform custom actions:
  - Example: Change bitmap when button pressed
  - Example: Turn on LED when button pressed
- Input parameters:
  - `objMsg`: Translated message for the widget
  - `pObj`: Pointer to the widget
  - `pMsg`: Pointer to message structure
- Output:
  - `TRUE`: To perform default actions too
  - `'0'` : To skip default actions

Regional Training Centers

# GOLMsg() Flowchart

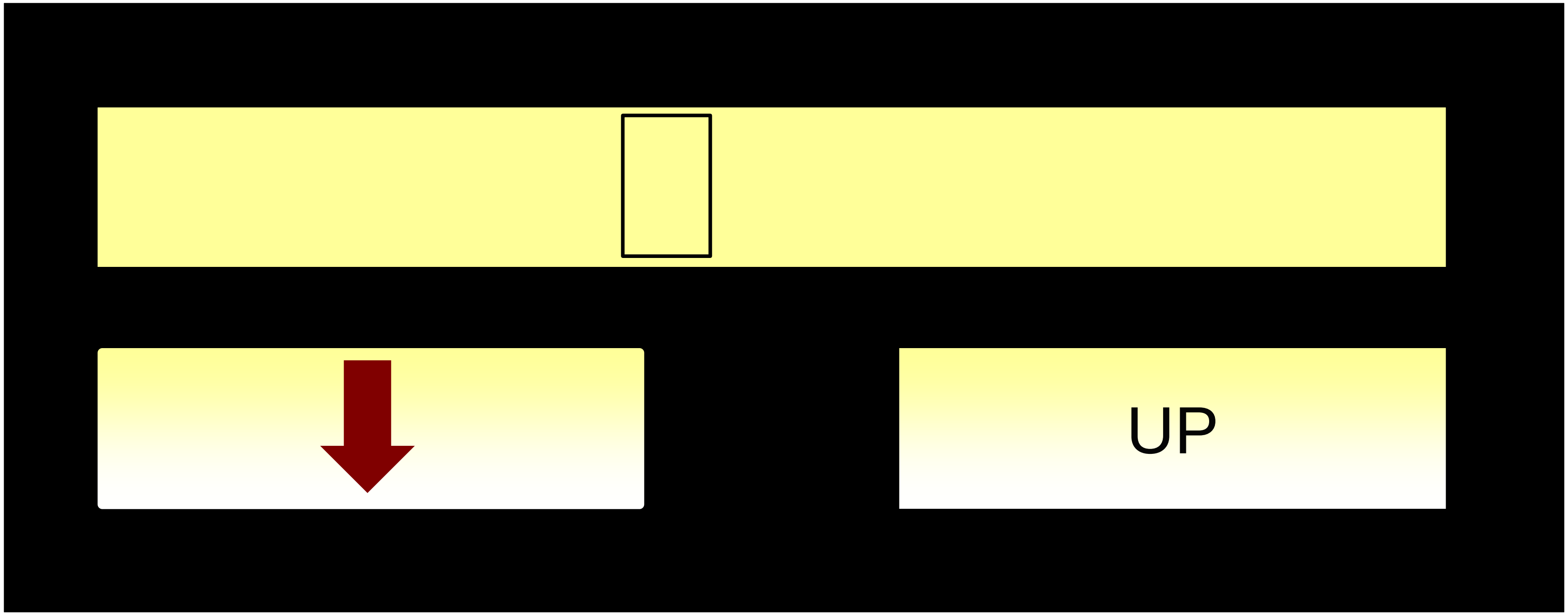


# `GOLMsgCallback(,,,)` Example



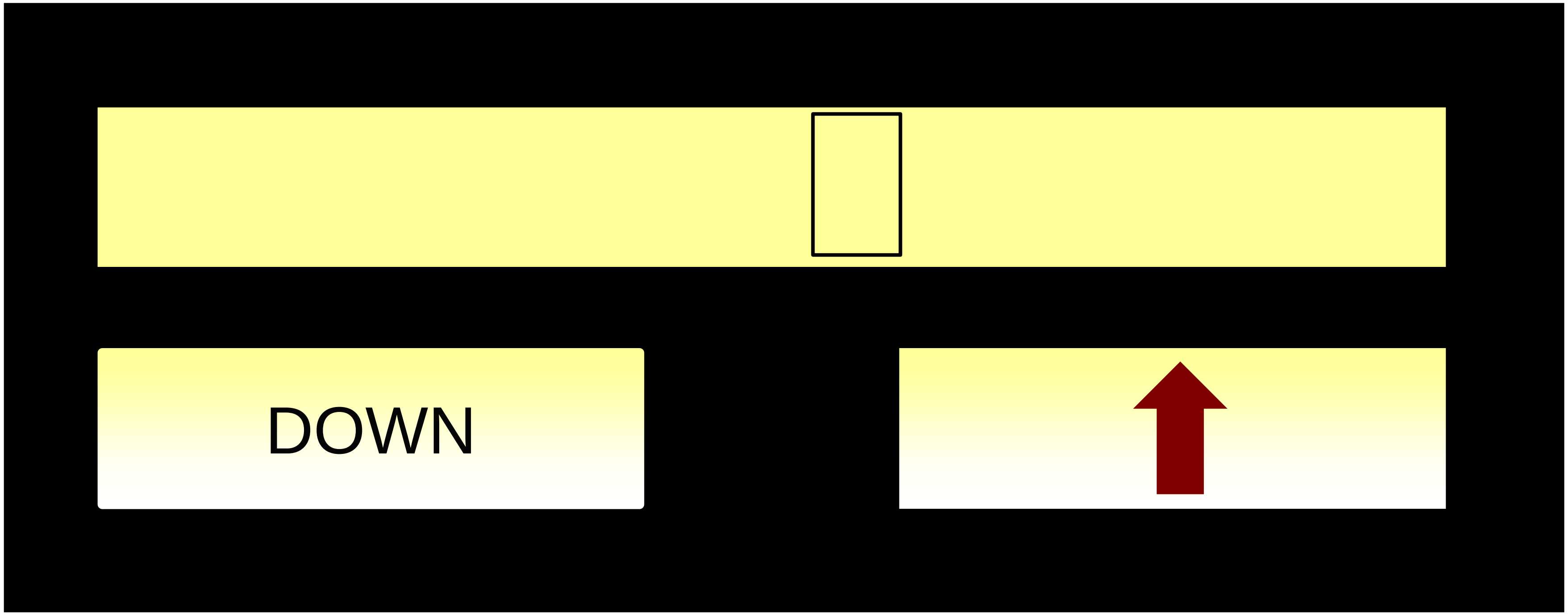
Regional Training Centers

# `GOLMsgCallback(,,,)` Example



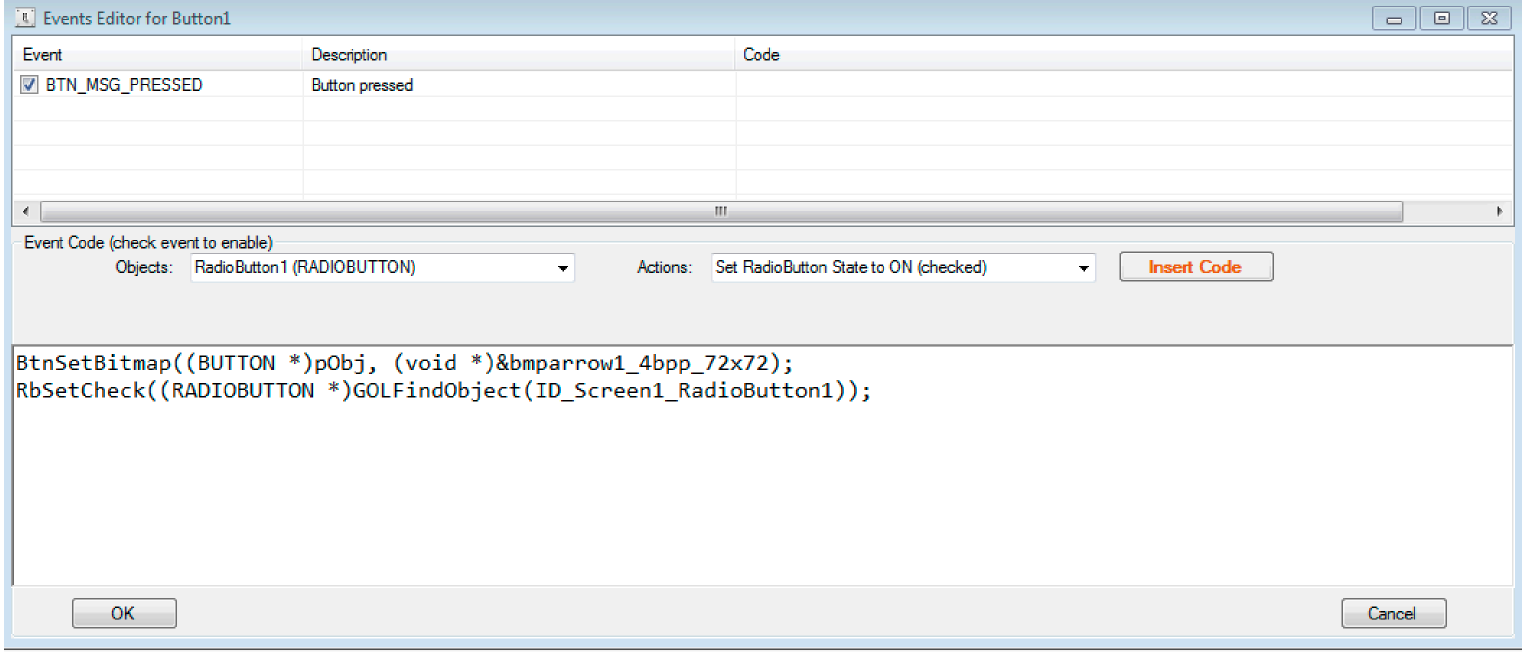
- Widget Actions:
  - Move slider thumb to the left
  - Add down arrow bitmap
- System Action:
  - Decrease motor speed

# `GOLMsgCallback(,,,)` Example



- Widget Actions:
  - Move slider thumb to the right
  - Add up arrow bitmap
- System Action:
  - Increase motor speed

# VGDD Event Handling



| Event | Description | Code |
|---|---|---|
| ☑ BTN_MSG_PRESSED | Button pressed | |

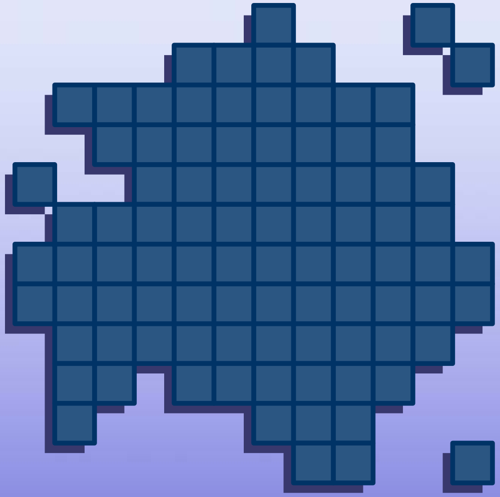
Event Code (check event to enable)

Objects: RadioButton1 (RADIOBUTTON)

Actions: Set RadioButton State to ON (checked)

```
BtnSetBitmap((BUTTON *)pObj, (void *)&bmparrow1_4bpp_72x72);
RbSetCheck((RADIOBUTTON *)GOLFindObject(ID_Screen1_RadioButton1));
```

- **Using VGDD event handler, code for the callback functions will be generated**
- **Event handler detects only one event at a time**
- **DEMO use of VGDD events**



# Lab Exercise 3

## LED Control Screen

# Lab 3

## Interfacing the User

?

### Purpose

- In this exercise you will demonstrate your ability to use VGDD to generate code to handle widget events
- Additionally, you will demonstrate your ability to control the system by adding application code to the generated C files

# Lab 3

## Interfacing the User

Objective

- Manage widget events using VGDD to generate the code
- Explore how to use a checkbox to light an LED on the board
- Use status texts to inform the user of the current system status

Regional Training Centers

# Lab 3

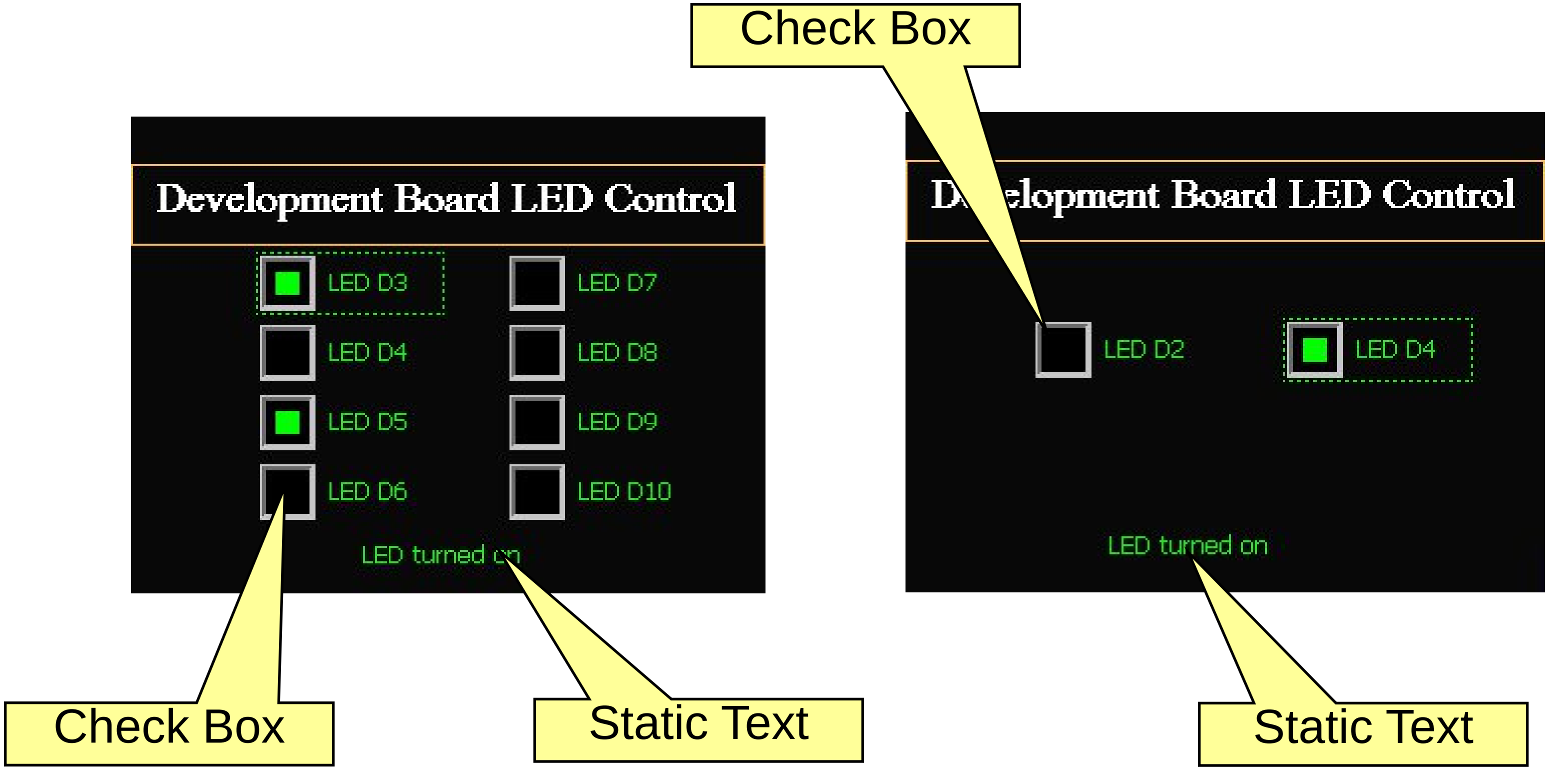
## Interfacing the User

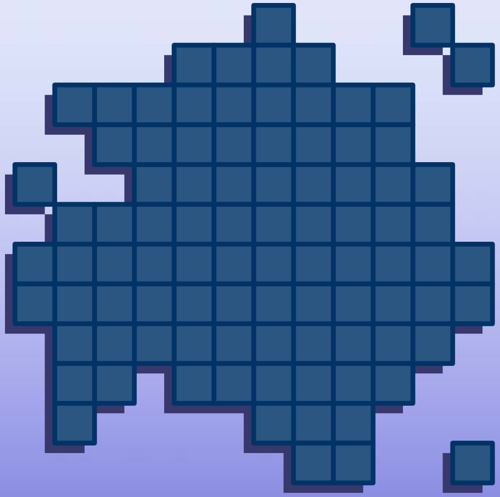
### Procedure

- **Follow the instructions in the lab manual starting on page 3-1**

# Lab 3
# Interfacing the User

## Results





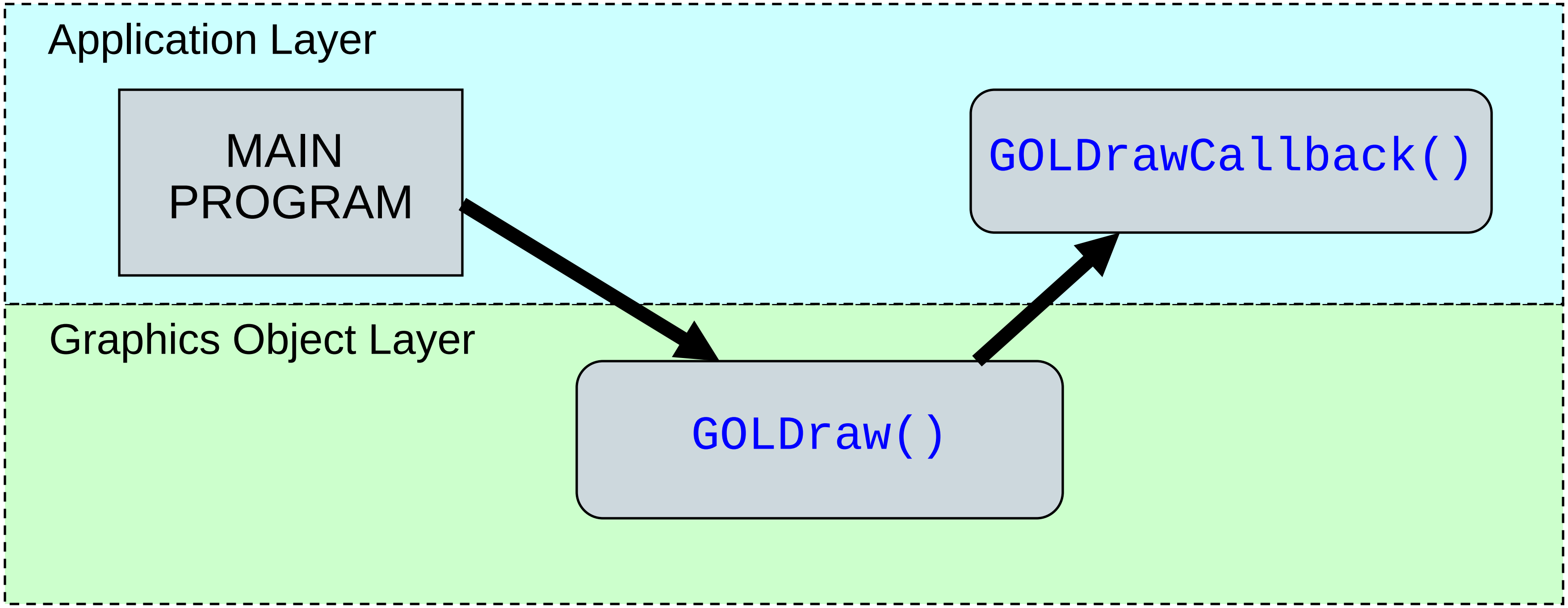
# Microchip Graphics Library

## Customizing the Response

`GOLMsgCallback()`

# Callback Reminder

- Called by `GOLDraw()`
- Used to add system or widget response to user inputs



Application Layer

MAIN PROGRAM

`GOLDrawCallback()`

Graphics Object Layer

`GOLDraw()`

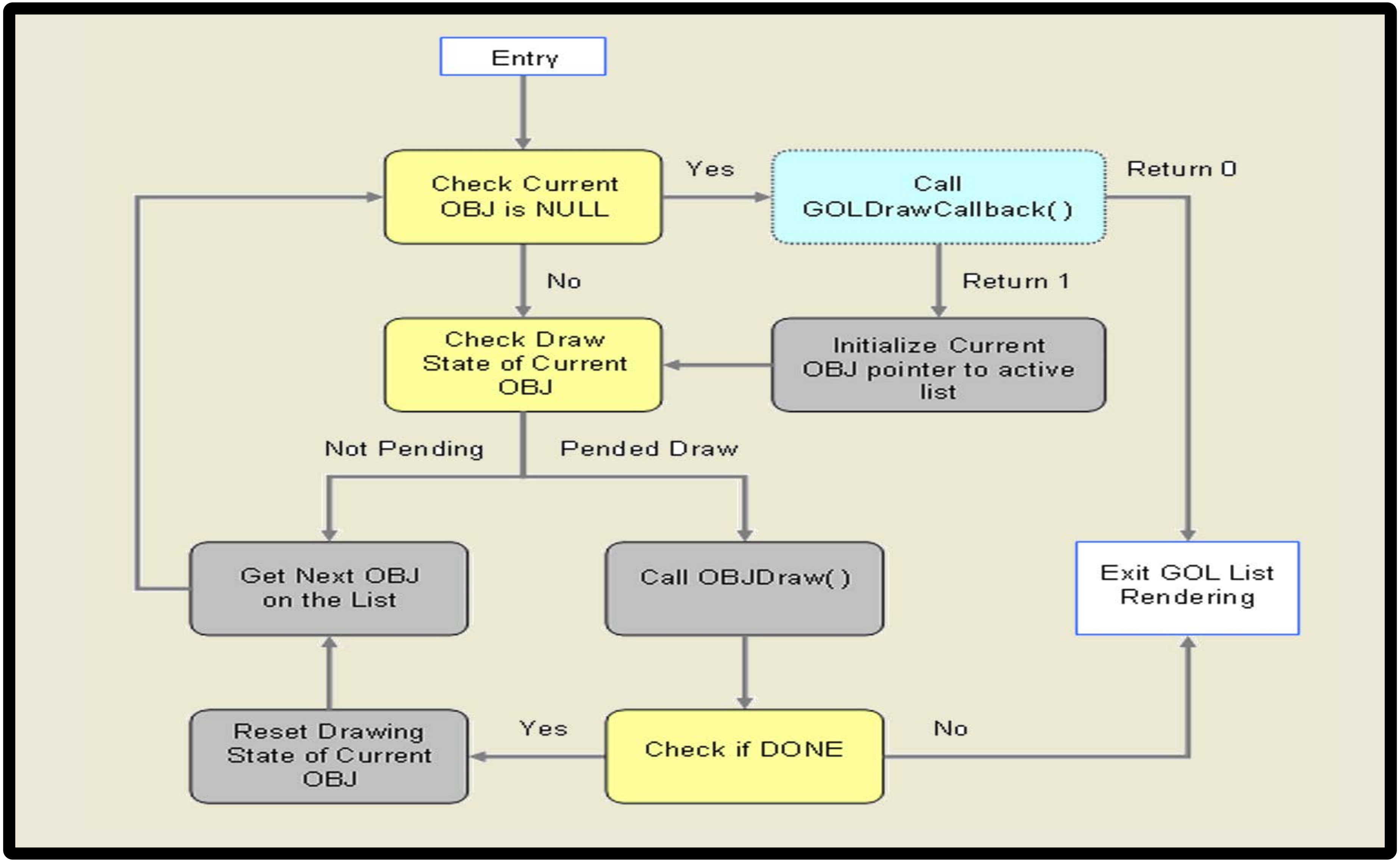
# Interfacing the User Application Requirements Part 2

- **Provide callback functions (REQUIRED)**
  - **`GOLMsgCallback(,,,)`**
    - Called by GOLMsg() only if a valid event occurs
    - Customize response to message event
      - **Example: Change button image on EVENT_PRESS**
      - **Example: Increase volume on EVENT_STILLPRESS**
      - **Example: Change screen state on** EVENT_PRESS
  - **`GOLDrawCallback()`**
    - Called by **`GOLDraw()`** at end of drawing sequence
    - Not dependent on message events
    - Customized drawing
      - **ONLY safe place to modify drawing properties**
      - **Example: Modify bars to form a signal strength meter**
    - Customized system response
      - **Example: Read a sensor and change a string**
      - **Example: Update a progress bar**
    - Create new screen

# Customized Drawing `GOLDrawCallback()`

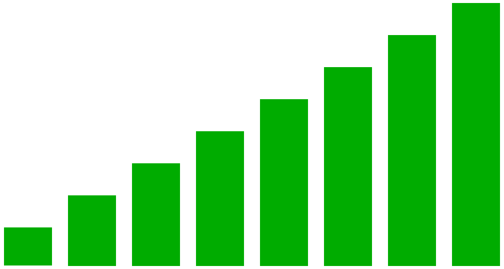
- Called by `GOLDraw()`
  - Drawing sequence is completed
- ***MUST*** be included in application code
  - Return `TRUE` to return control to `GOLDraw()`
  - Return ‘`0`’ to keep drawing control
- Perform customized drawing
  - Example:  Signal strength indicator
- Monitor and control for continuous events
  - Example:  Read from a sensor and update screen
  - Example:  Display a countdown timer
- Safest place to modify the linked list

# Flowchart for `GOLDraw()`

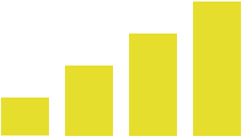


# `GOLDrawCallback()`

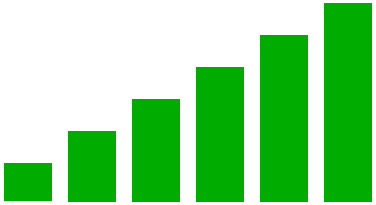
# Example: Battery Life Meter



Battery 100%



Battery 50%

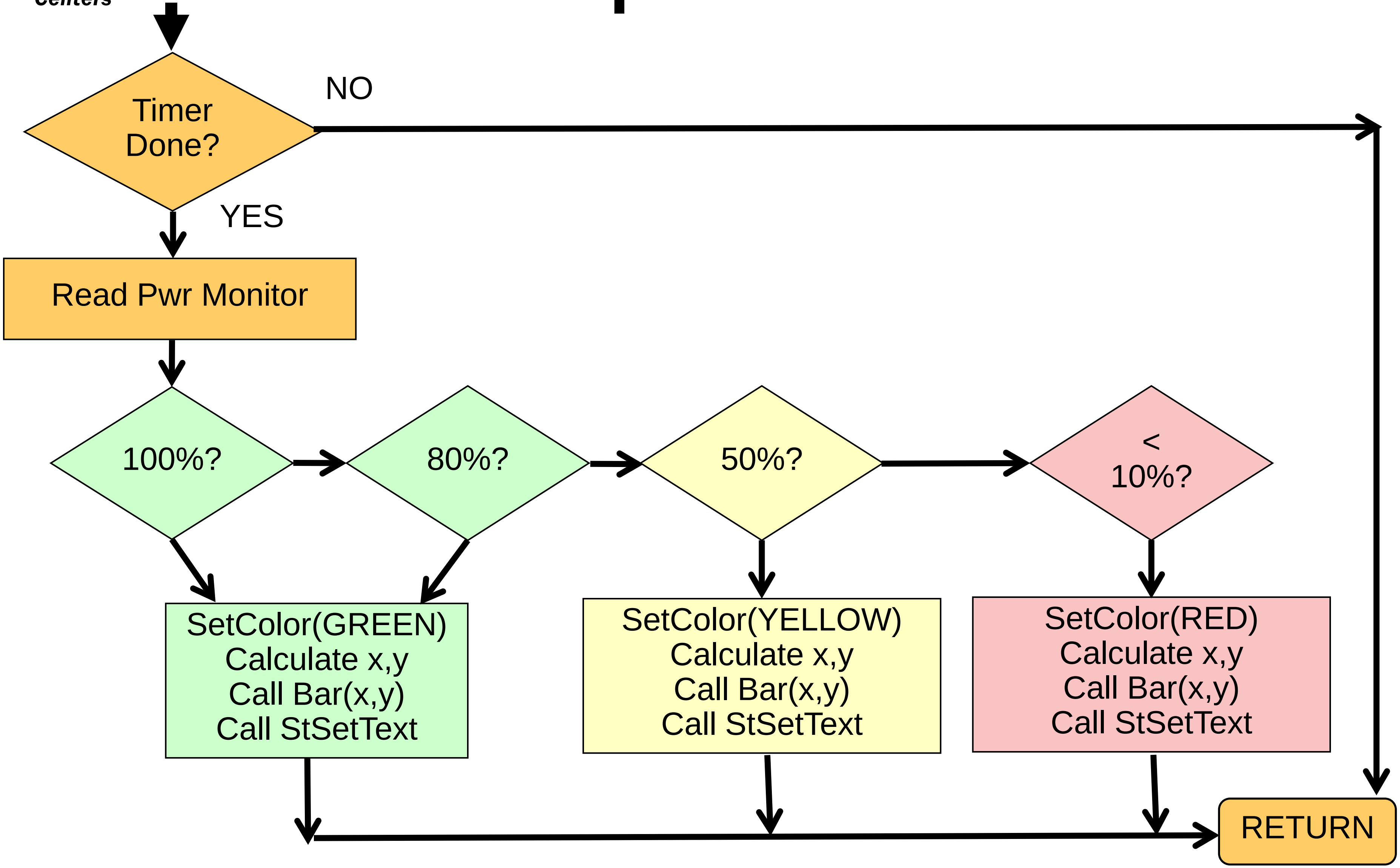


Battery 80%

Battery Critical

- Read from power monitor circuit and update display accordingly

# `GOLDrawCallback()`
# Example – Flow Chart



Timer Done?

NO

YES

Read Pwr Monitor

100%?

80%?

50%?

< 10%?

SetColor(GREEN)
Calculate x,y
Call Bar(x,y)
Call StSetText

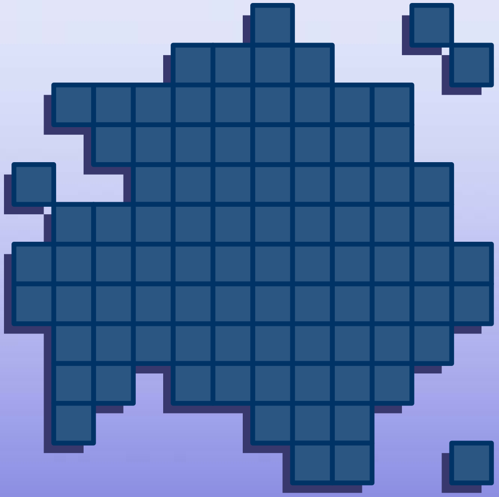
SetColor(YELLOW)
Calculate x,y
Call Bar(x,y)
Call StSetText

SetColor(RED)
Calculate x,y
Call Bar(x,y)
Call StSetText

RETURN

# Tips `GOLDrawCallback()`

- To erase a bar
  - Set the drawing color to the background color
  - Call `Bar()` function with area for one or more bars
    - NOTE: `Bar()` returns true when drawing is complete
- To add a bar
  - Set the drawing color to desired color
  - Call `Bar()` with bar coordinates
- No pointers are passed into `GOLDrawCallback()`
  - Use `GOLFindObject()` to get the address of widget
- Application must set drawing bits for widgets that need to be redrawn
  - When updating widgets, make sure value changed before setting draw bit



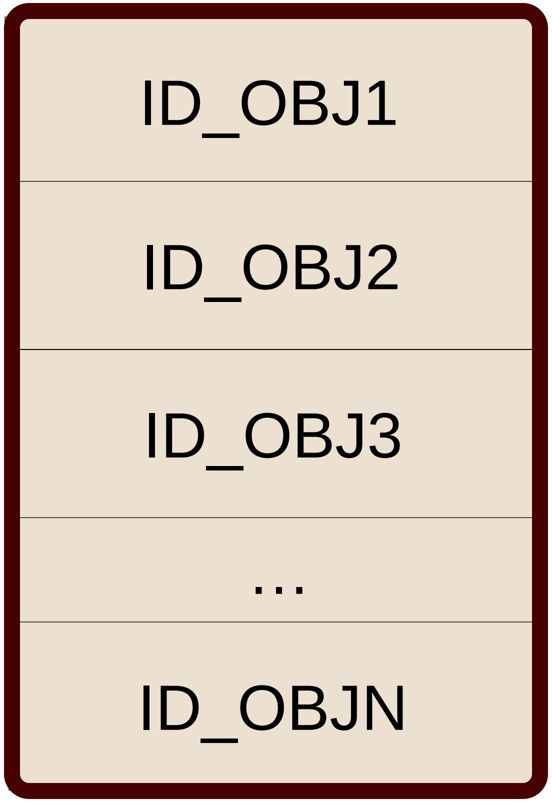
# Microchip Graphics Library

## Putting it all together

## Screen Management

# Creating Widgets

Linked List



ID_OBJ1

ID_OBJ2

ID_OBJ3

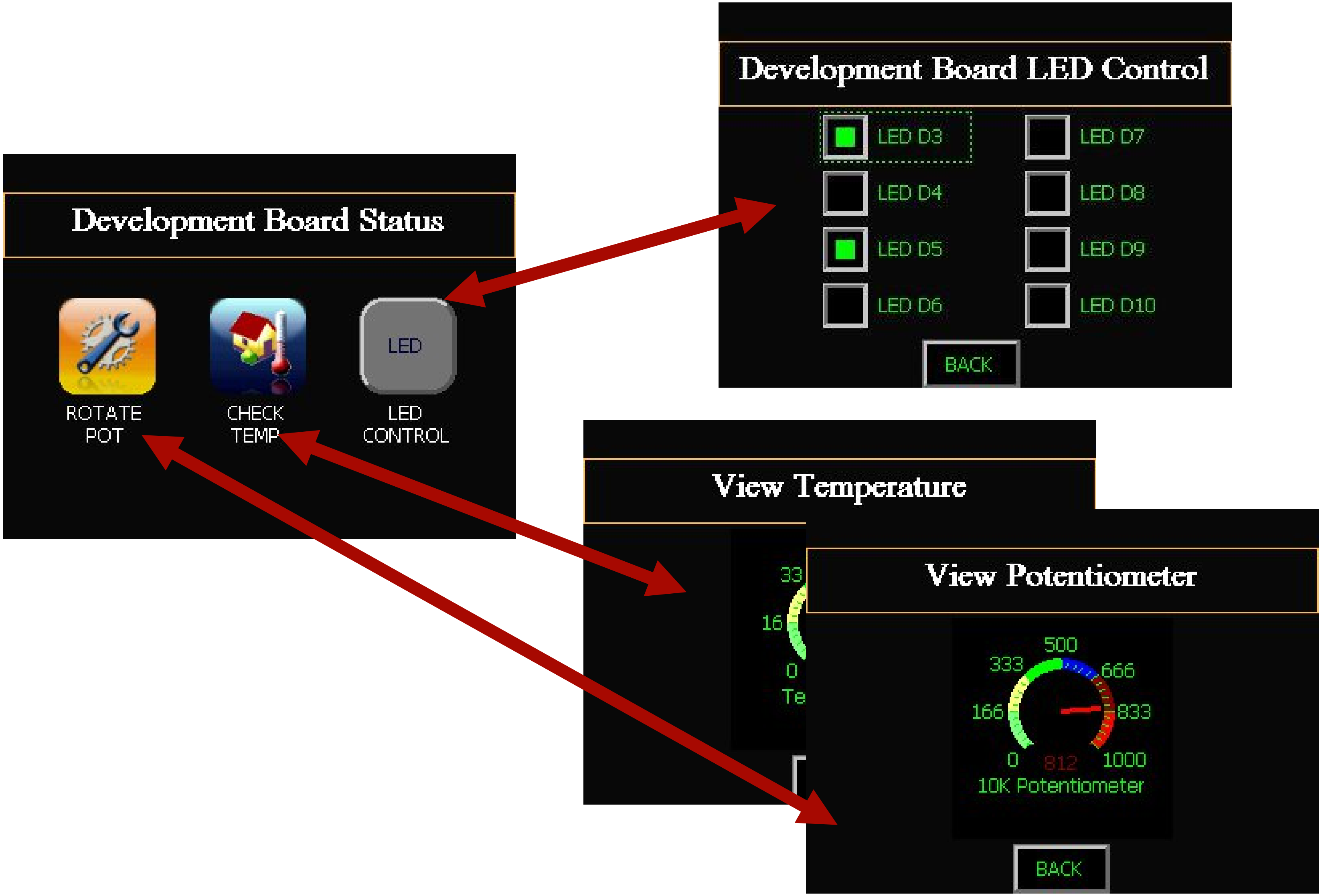
…

ID_OBJN

## Recall…

- `ObjCreate(,,,)`
  - **Populates a structure for the widget**
  - **Adds widget to bottom of the linked list**
- `GOLDraw()`
  - **Parses *ACTIVE* linked list**
  - **Redraws based on state bit settings**

# GUI Applications

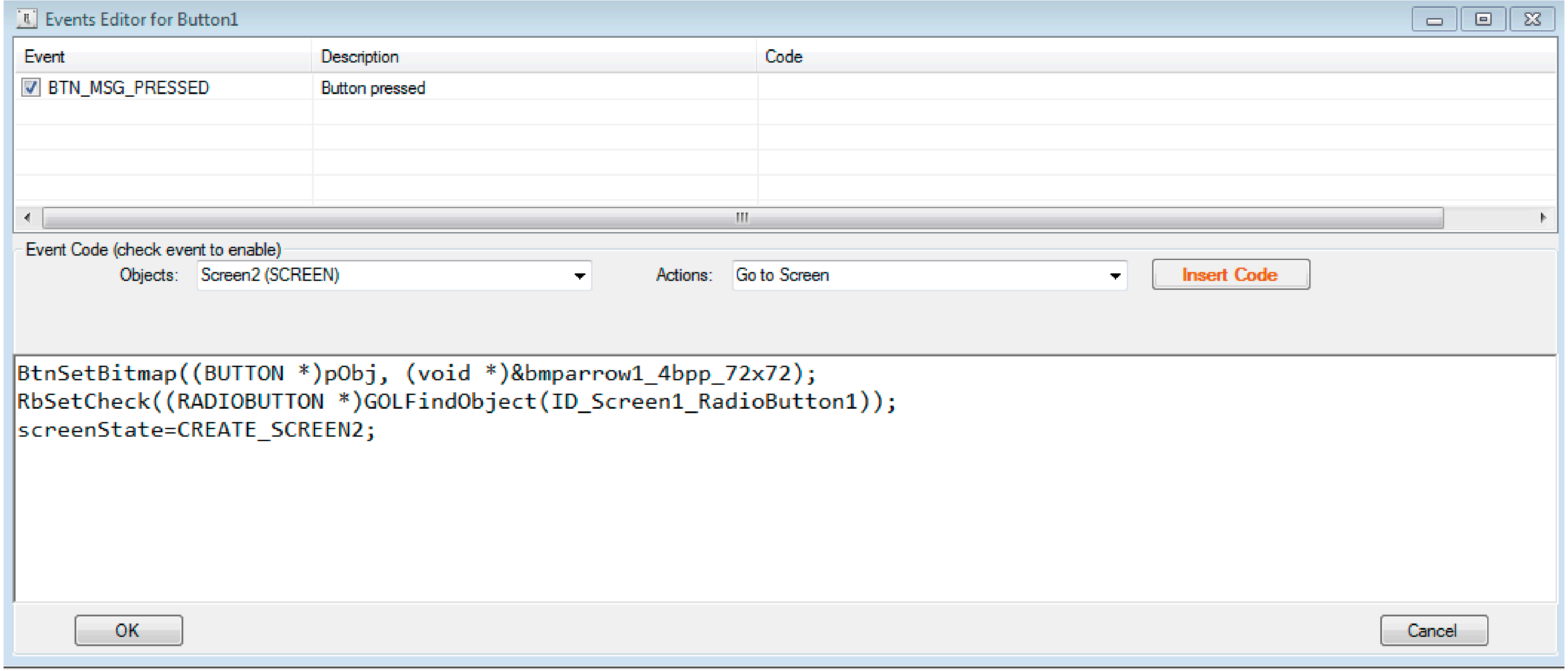


# Screen Management Options Using Multiple Lists

- Create Lists on the Fly
- Used by VGDD
  - Use `GOLFree()` to delete active list
    - Frees heap library uses
    - Does not affect application heap
  - Use `ObjCreate(,,,)` to form new list
  - Minimize heap requirements
- APIs in library to manage screens with multiple lists

Regional Training Centers

# VGDD Event Handling



| Event | Description | Code |
|---|---|---|
| ☑ BTN_MSG_PRESSED | Button pressed | |

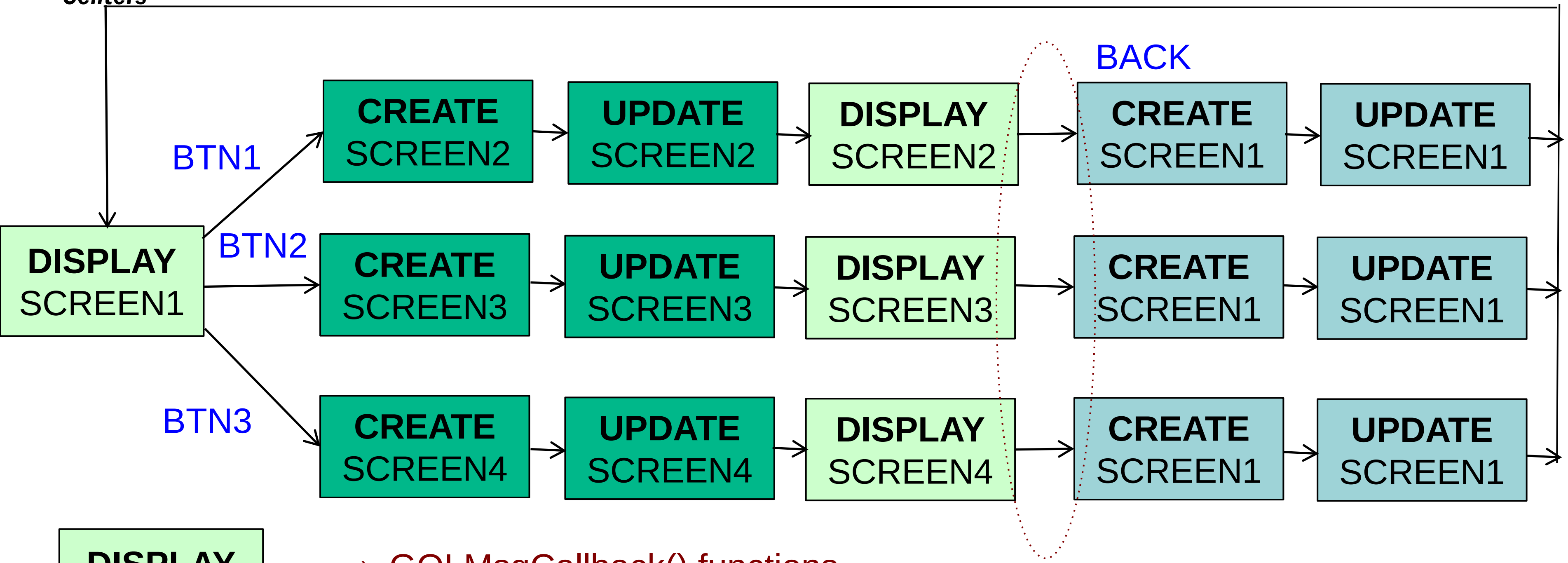
Objects: Screen2 (SCREEN)

Actions: Go to Screen

```
BtnSetBitmap((BUTTON *)pObj, (void *)&bmparrow1_4bpp_72x72);
RbSetCheck((RADIOBUTTON *)GOLFindObject(ID_Screen1_RadioButton1));
screenState=CREATE_SCREEN2;
```

- **Implements screen state machine**
- **Screen state changed in `GOLMsgCallback()`**
- **`GOLFree()` and `CreateScreen()` in `GOLDrawCallback()`**

# VGDD Screen States



BACK

BTN1

BTN2

BTN3

**DISPLAY** SCREEN1

**CREATE** SCREEN2 → **UPDATE** SCREEN2 → **DISPLAY** SCREEN2 → **CREATE** SCREEN1 → **UPDATE** SCREEN1

**CREATE** SCREEN3 → **UPDATE** SCREEN3 → **DISPLAY** SCREEN3 → **CREATE** SCREEN1 → **UPDATE** SCREEN1

**CREATE** SCREEN4 → **UPDATE** SCREEN4 → **DISPLAY** SCREEN4 → **CREATE** SCREEN1 → **UPDATE** SCREEN1

**DISPLAY** SCREENx

⇒ GOLMsgCallback() functions
⇒ User event to change screens

**CREATE** SCREENx

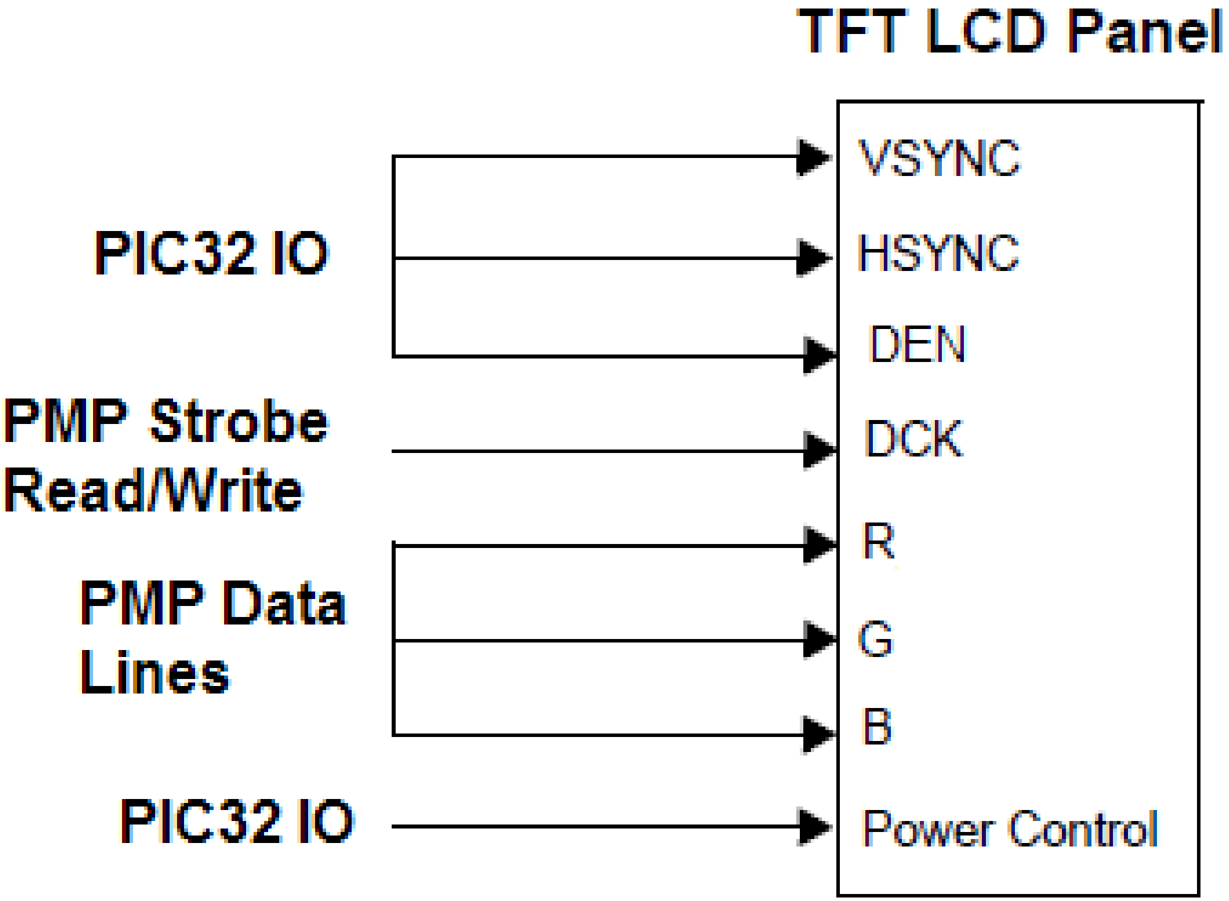
⇒ GOLDrawCallback() functions
⇒ Erase old screen, create new screen

**UPDATE** SCREENx

⇒ GOLDrawCallback() functions
⇒ Primitive drawing

# LCC
# Interfacing to TFT LCD Panels

# TFT LCD Panel:
# LCC Diagram



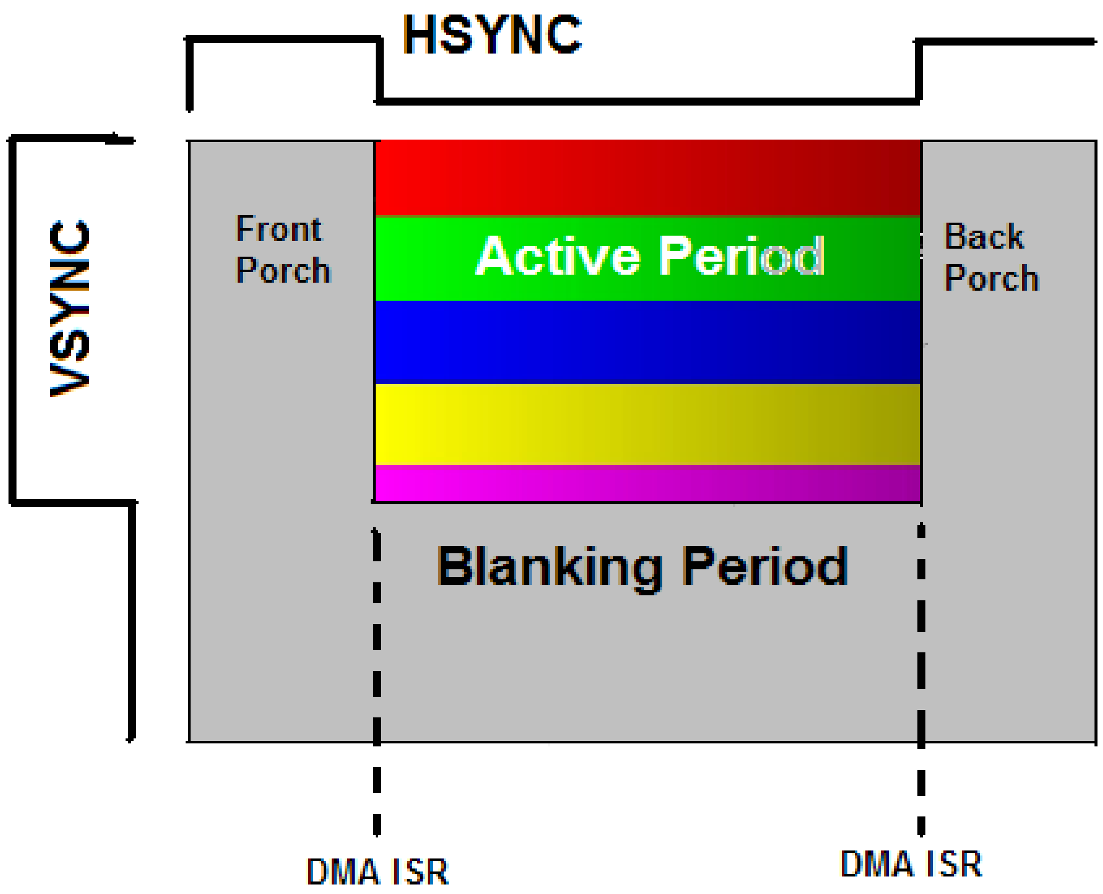
# TFT LCD RGB Data Basics

- Raw color data sent to screen
- Different RGB Color formats
  - 332, 565, 666, 888
- Unused color tied to MSBs

# LCD Timing Basics

- Cycle (Display Period)
  - Active Period + Blanking Period
- Active Period
  - This period displays the actual image
- Blanking Period
  - Sync
    - Used for synchronizing the display to graphics controller
  - Front Porch
    - Time between the end of the active time and the start of the sync pulse
  - Back Porch
    - Time between the end of a sync and the next active time

# LCD Timing

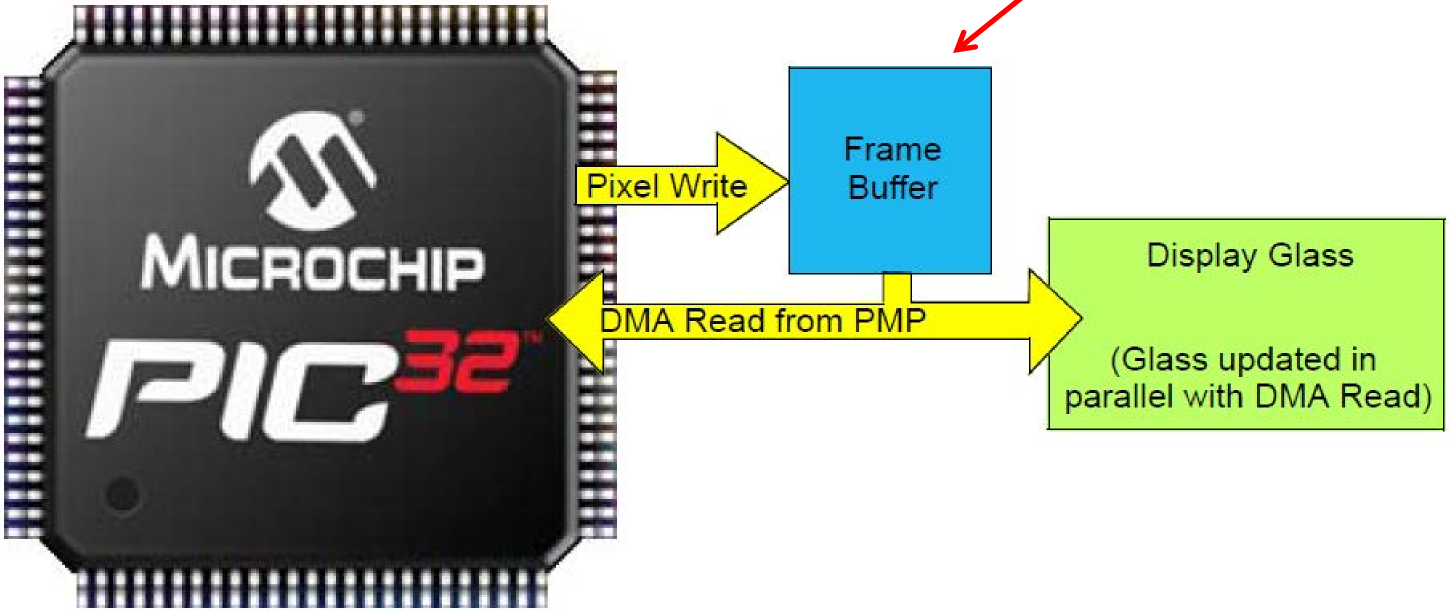

HSYNC
VSYNC
Front Porch
Active Period
Back Porch
Blanking Period
DMA ISR
DMA ISR

Regional Training Centers

# PIC32 Peripherals Used for Graphics

# Controllerless Solution: Diagram

- DMA is used to control LCD
- Most PIC32 with DMA and PMP can be used



Optional External Frame buffer for 16 BPP modes

# Controllerless Solution: DMA Usage

- Continuously reads/writes frame buffer data.
- DMA Transfer Interrupt (ISR) updates timing signals

# Controllerless Solution: PMP Usage

- Read/Write Strobe acts as *Pixel Clock*.
- Data from DMA transmitted through PMP.
- Keeps system synchronized

Regional Training Centers

# Controllerless Tasks

# Controllerless Tasks: Frame Buffer

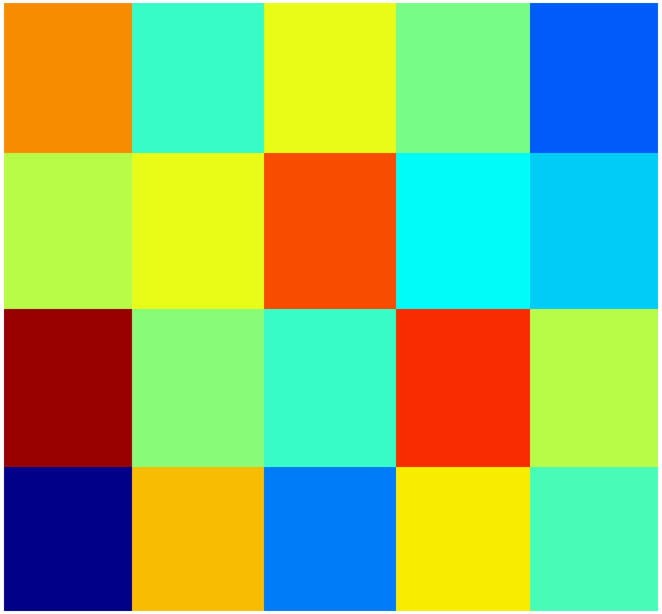
- Frame containing the screen image must be stored somewhere
  - Internal – PIC32 SRAM
  - External – PSRAM on board
- DMA must read/write from this buffer

# Controllerless Tasks: Frame Rendering

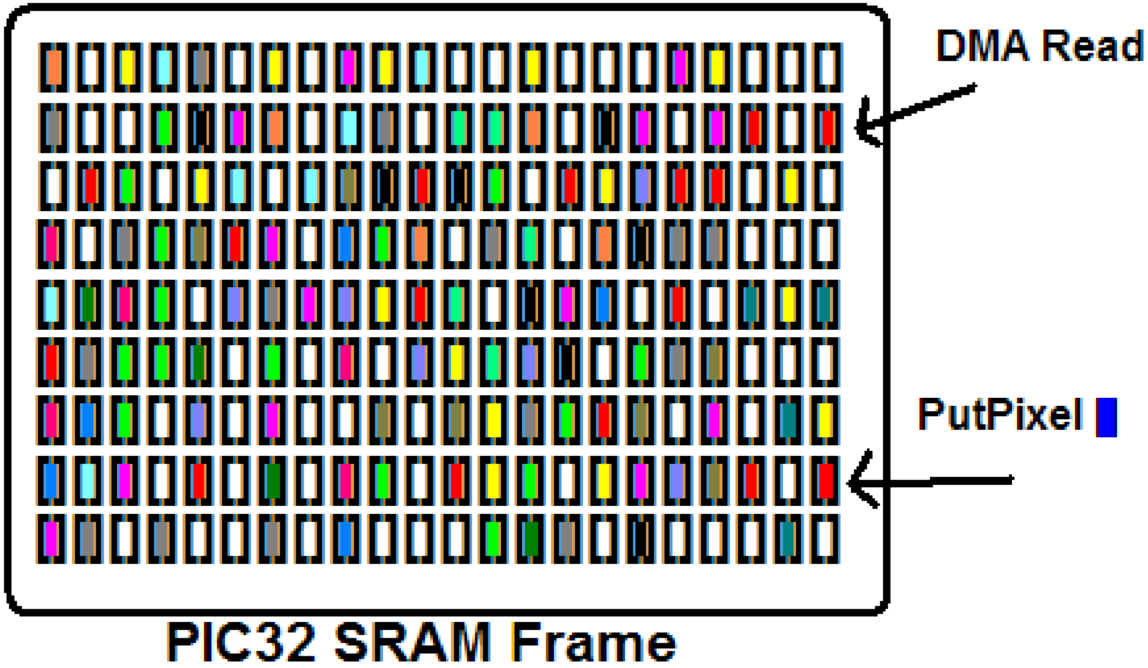
- Usually refresh rates of 50-60 hz needed.
- Support of various resolutions

# Controllerless Tasks: Update Pixels

- Helps to create dynamic interfaces
- Throughput varies depending on resolution
- PutPixel functionality

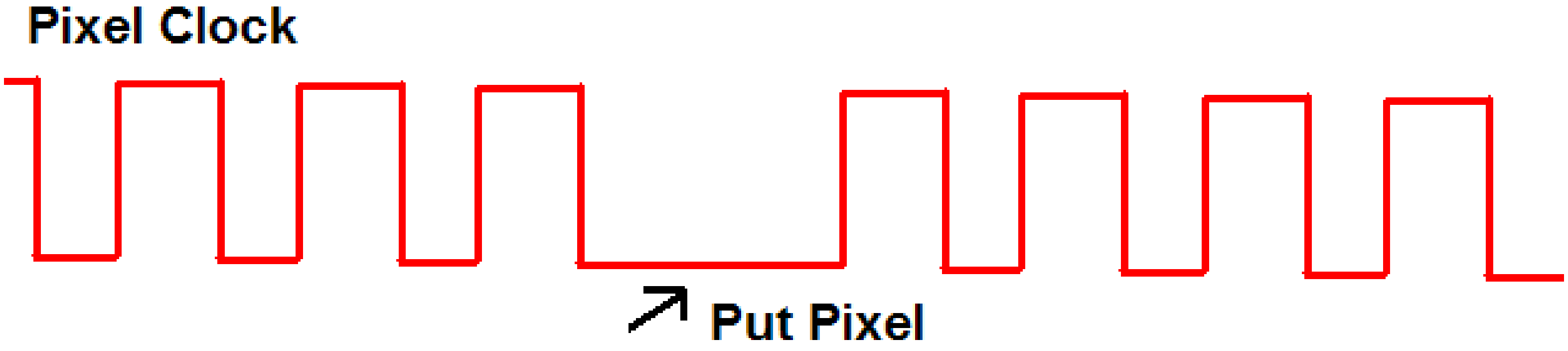


# PutPixel: Internal



- The DMA shares the internal SRAM.
- Pixels are updated in conjunction with DMA. Pixel Updates wait for DMA read.

# PutPixel: External

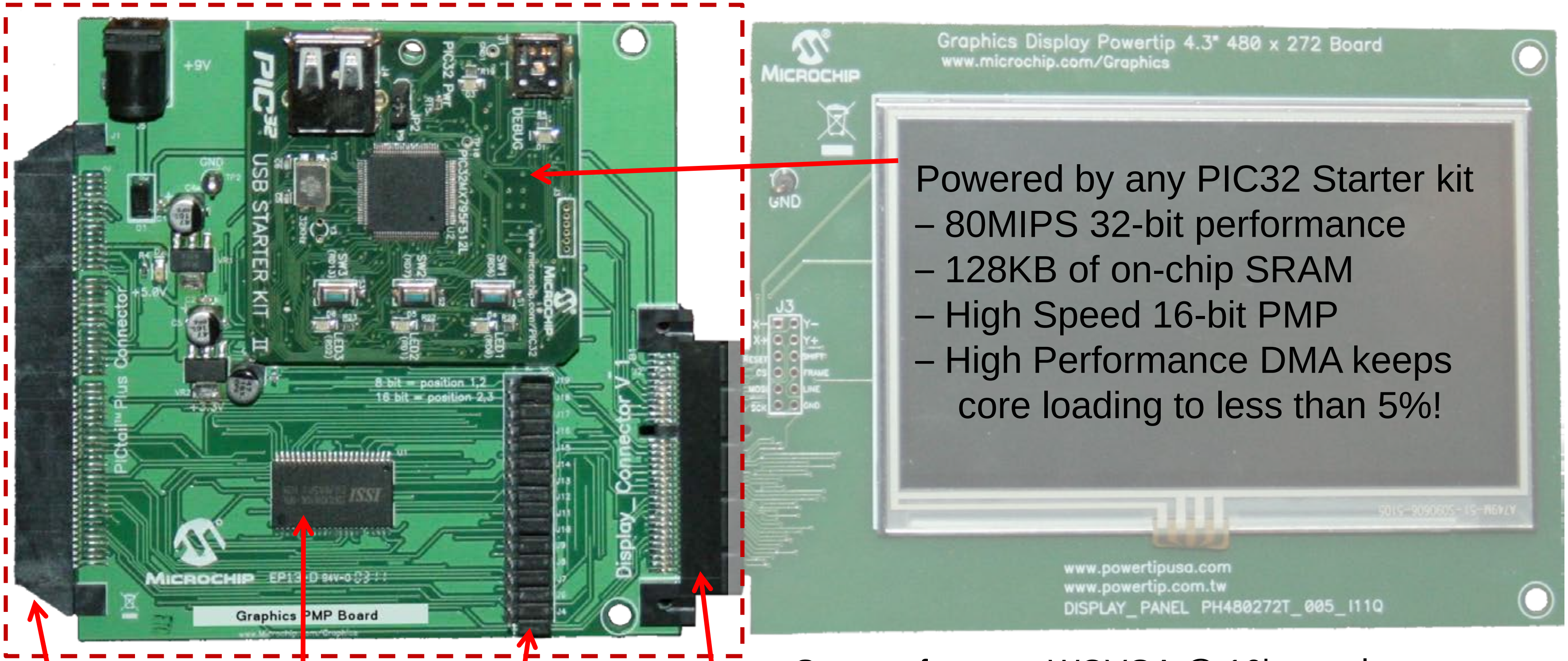


- There is only one PMP.
- DMA (Pixel Clock) suspended during pixel updates.
- Controlled to stabilize screen refresh

# Lab 4 – LCC and Primitive Layer Demo

# LCC (Low-Cost Controllerless) Graphics Board

## Driving Graphics Displays without a Graphics Controller


PIC32 USB STARTER KIT II
PICtail™ Plus Connector
Display_Connector V1
Graphics PMP Board
Graphics Display Powertip 4.3" 480 x 272 Board
www.microchip.com/Graphics
DISPLAY_PANEL PH480272T_005_I11Q

Powered by any PIC32 Starter kit
- 80MIPS 32-bit performance
- 128KB of on-chip SRAM
- High Speed 16-bit PMP
- High Performance DMA keeps core loading to less than 5%!

On Board 512KB SRAM
Can be "jumpered" in or out

Interfaces to Microchip Explorer 16

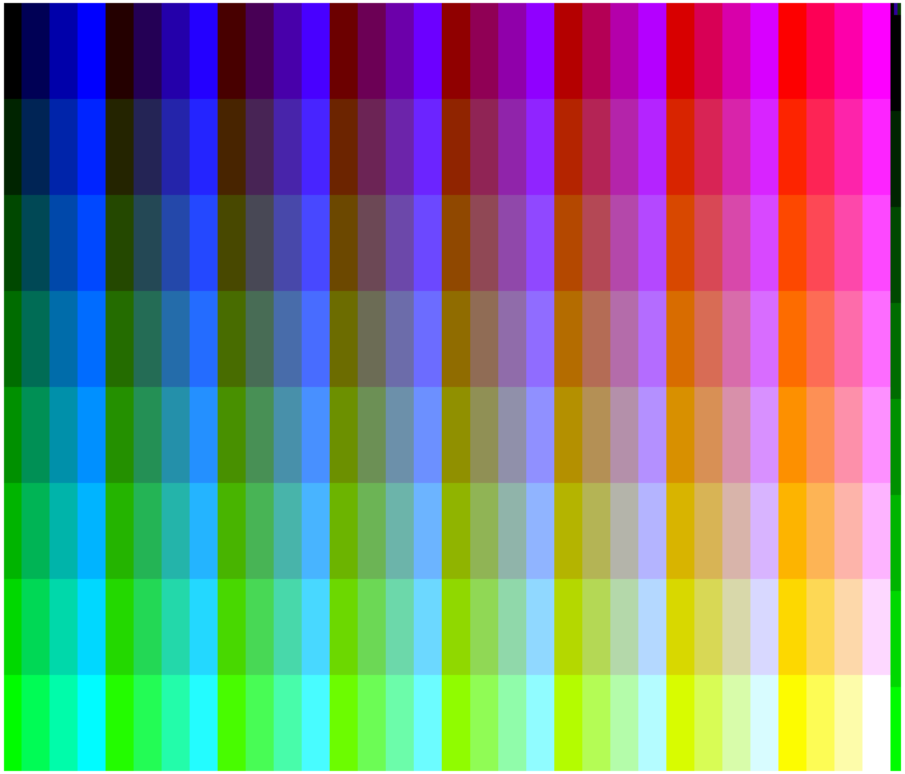
- Support for up to WQVGA @ 16bpp color
- PIC32 alone can drive QVGA @ 8bpp color

Interfaces to Microchip Display modules

PN: AC164144

# LCC Graphics PICtail™: Internal Frame Buffer

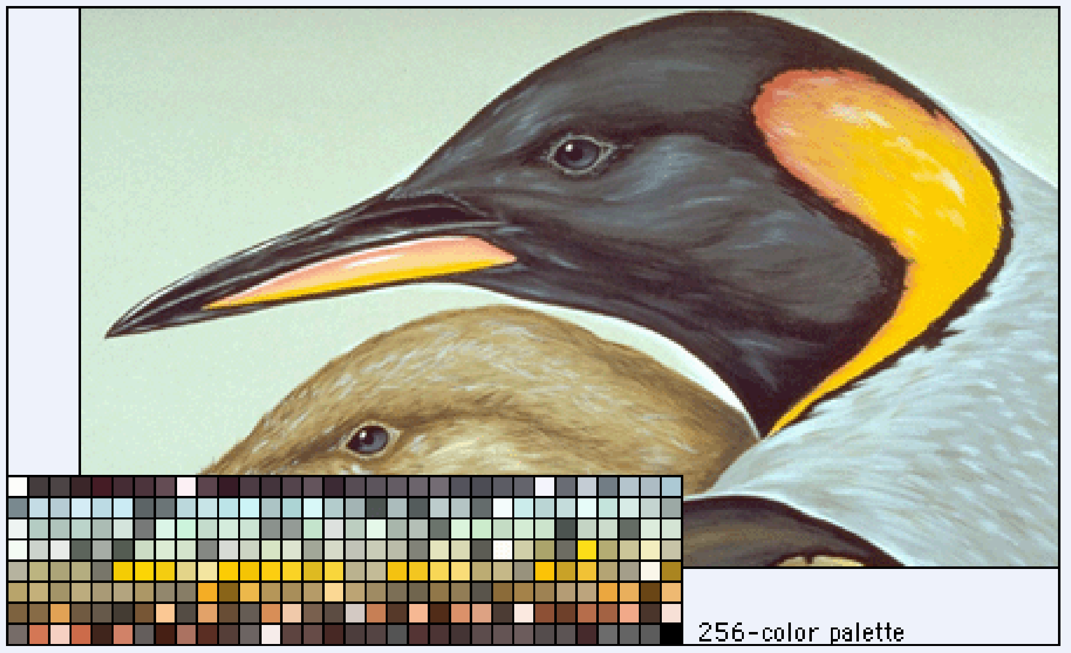
- Supports 8bpp color on QVGA
- No external SRAM
- PIC32 SRAM used for frame buffer
- Color Look-up Table (CLUT)



256 colors available with 8bpp

# LCC Graphics PICtail™: Color Look-up Table

- Applications have themed colors
- CLUT can be used for 16 BPP graphics.
- Color Palette stores color information



# LCC Graphics PICtail™ External Frame Buffer

- Supports 16bpp color on QVGA
- External SRAM on LCC Graphics PICtail™ for frame buffer
- 13MHz pixel clock achieved

16bpp makes many colors available

# Hands-on Lab 4

# Initializing the PIC

```
 #define PMP_CONTROL (PMP_ON | PMP_MUX_OFF | PMP_READ_WRITE_EN|PMP_CS2_EN | PMP_CS2_POL_LO | PMP_WRITE_POL_LO | PCLK_POLARITY)
 #define PMP_MODE     (PMP_DATA_LENGTH | PMP_MODE_MASTER2 |PMP_WAIT_BEG_1 | PMP_WAIT_MID_1 | PMP_WAIT_END_1 )

void ResetDevice(void)
{

   // setup the PMP
    mPMPOpen(PMP_CONTROL, PMP_MODE, PMP_ADDRESS_LINES, PMP_INT_ON);
    PMADDR = 0x0000;

   // Open the desired DMA channel
    DmaChnOpen(1, 0, DMA_OPEN_DEFAULT);

   // set the transfer event control: what event is to start the DMA transfer
    DmaChnSetEventControl(1, DMA_EV_START_IRQ(_TIMER_2_IRQ));

   // set the transfer parameters: source & destination address, source & destination size, number of bytes per event
     DmaChnSetTxfer(1, &GraphicsFrame[0], (void*)&PMDIN, HBackPorch, 1, 2);

     INTSetVectorPriority(INT_VECTOR_DMA(1), INT_PRIORITY_LEVEL_7);                    // set INT controller priority
     INTSetVectorSubPriority(INT_VECTOR_DMA(1), INT_SUB_PRIORITY_LEVEL_3);           // set INT controller sub-priority

     DmaChnSetEvEnableFlags(1, DMA_EV_BLOCK_DONE);               // enable the transfer done interrupt, when all buffer transferred
     INTEnable(INT_SOURCE_DMA(1), INT_ENABLED);                   // enable the chn interrupt in the INT controller

     DCH1CONbits.CHPRI = 0b11;  //DMA channel has highest priority

     // once we configured the DMA channel we can enable it
      DmaChnEnable(1);

      OpenTimer2(T2_ON | T2_SOURCE_INT | T2_PS_1_1, 27);          //Start Timer
}
```

# DMA ISR

```
void __ISR(_DMA1_VECTOR, ipl7) DmaHandler1(void)
{
    static BYTE GraphicsState = 1;
    static short line =0;

     if(GraphicsState ==1)
     {
          DCH1SSIZ =  LINE_LENGTH;
          GraphicsState++;

          if(line++ >= -1)
          {
           VSYNC =1;
           DATA_ENABLE =1;

           DCH1SSA = _VirtToPhys(&GraphicsFrame[line][0]);

                 if(line == (FRAME_HEIGHT))
                 {
                     VSYNC =0;
                     line= -VER_BLANK;
                 }
           }
      }
     else
      {
          HSYNC =0;
          DATA_ENABLE =0;

          //Perform Back Porch Clock Signal
          PMDINSET=1;

          HSYNC = 1;
          DCH1SSIZ =  HBackPorch;
          GraphicsState= 1;
      }

      DCH1INTCLR = 0x08;          //CHBCIF = 0
      IFS1CLR = 0x20000;          //DMA1IF =0
      DCH1CONSET =0x80;           //CHEN =1
 }
```

# Putpixel

```
void PutPixel(short x, short y)
{

     if(_clipRgn)
     {
        if(x < _clipLeft)
           return;
        if(x > _clipRight)
           return;
        if(y < _clipTop)
           return;
        if(y > _clipBottom)
           return;
     }

  GraphicsFrame[(GetMaxX()-x)][y] = _color;
}
```

# Lab Exercise 4

## Purpose

Understand how to send data to an LCD display. After this lab the student should be able to send color data to an LCD Panel and see the data on the screen.

## Procedure

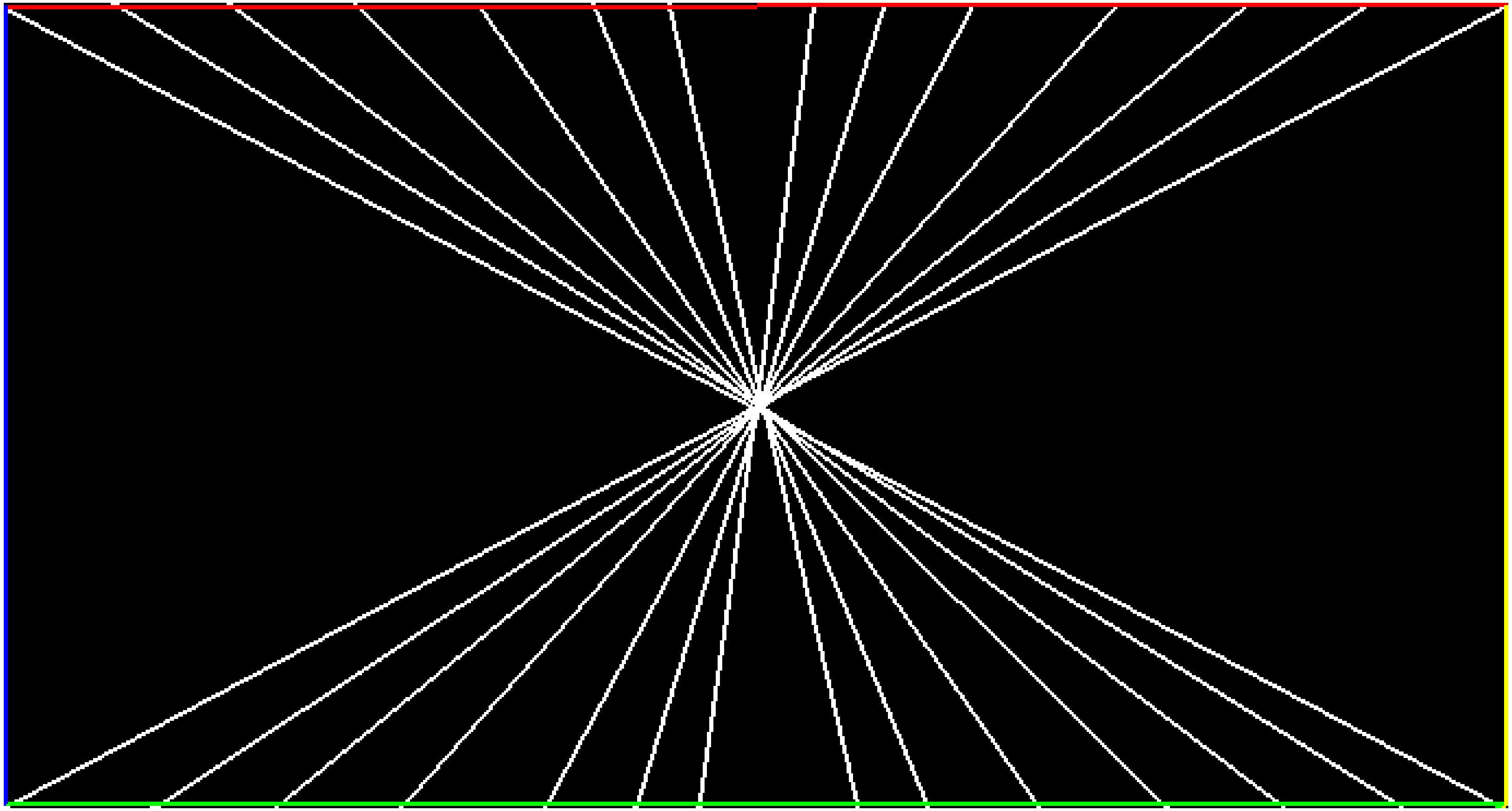
Follow the directions in the lab manual starting on page 1-1

Objectives:

- Use the LCC Graphics board to run the primitive layer demo
- Use microchip graphic library to make changes to the demo

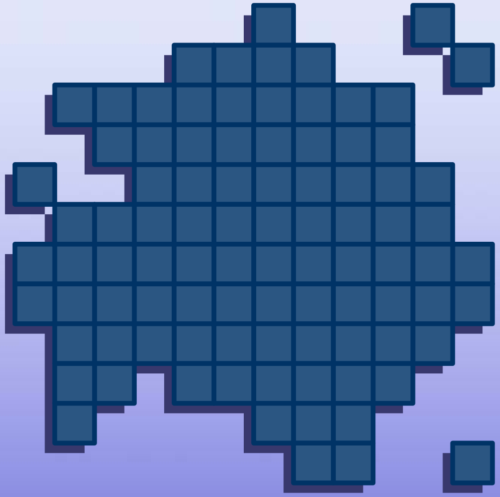
# Lab Exercise 4

## *Expected Results –*



# Lab 4 Summary

- TFT LCD panels can be driven with only a PIC32
- Accomplished using DMA and PMP peripherals
- LCC graphics board available for development



# Summary

# Steps for design with Microchip Graphics Objects Library

- Use a GUI design tool to design screens, import necessary fonts and image files, and establish style schemes
- Configure hardware using **HardwareProfile.h**
- Configure Graphics Objects Layer using **GraphicsConfig.h**
- Place appropriate files in project
- Write application code
  - Use typedefs (see next slide)
  - Implement message structure
  - Remember, allow GOLDraw() to finish parsing linked list before processing messages
  - Add callback functions to customize application response and manage GUI screen states
- Compile, build and run!

# Microchip Graphics Library Types

| Type | Description |
|---|---|
| `XCHAR` | Sets type of character used in library. Types are `char`, `unsigned char`, or `short` depending on setting in GraphicsConfig.h |
| `GFX_COLOR` | Sets type of color used in library. Types are `BYTE`, `WORD` or `DWORD` based on `COLOR_DEPTH` setting in GraphicsConfig.h |
| `FONT_FLASH` | Refers to fonts stored in internal FLASH |
| `FONT_EXTERNAL` | Refers to fonts stored in external memory |
| `IMAGE_FLASH` | Refers to images stored in internal FLASH |
| `IMAGE_EXTERNAL` | Refers to images stored in external memory |
| `GOL_SCHEME` | Refers to style scheme structures used by widgets |
| `GOL_MSG` | Refers to the message structure used in the library's messaging interface |
| `OBJ_HEADER` | Refers to the first member of all widget structures. Specifies ID, location (and dimensions), states and style scheme to be used by widget |
| *`WIDGET`* | Refers to the *widget* structures. See library help for complete listing |

# Words of Wisdom

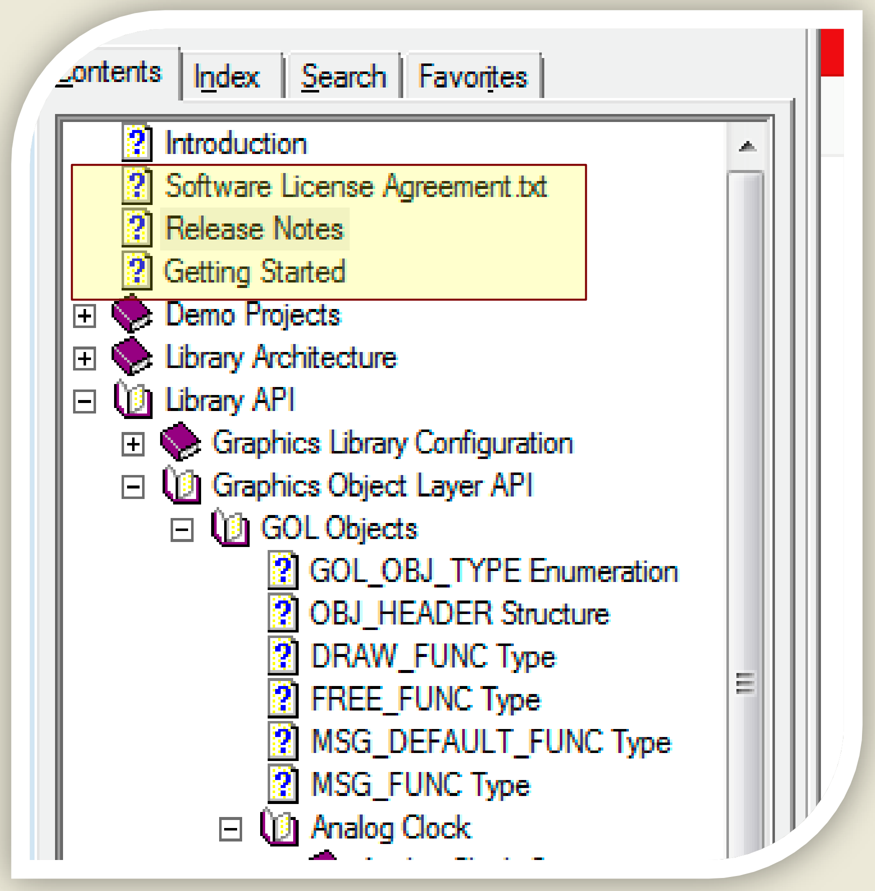
- Avoid heap fragmentation
  - Use GOLFree()
  - Do NOT …
    - Manually remove widgets from active list
  - Do NOT…
    - Modify list until `GOLDraw()` is complete
- Avoid weird drawing effects
  - Do NOT …
    - Modify drawing properties until `GOLDraw()` has completed the drawing sequence
- Watch the list pointers

# Microchip Graphics Library Static Memory Use

- Define and initialize widget structures
- Define and initialize style scheme structures
- Must maintain linked list
- All other concepts remain

Microchip Regional Training Centers

# Microchip Graphics Library Help



- Introduction
- Software License Agreement.txt
- Release Notes
- Getting Started
- Demo Projects
- Library Architecture
- Library API
  - Graphics Library Configuration
  - Graphics Object Layer API
    - GOL Objects
      - GOL_OBJ_TYPE Enumeration
      - OBJ_HEADER Structure
      - DRAW_FUNC Type
      - FREE_FUNC Type
      - MSG_DEFAULT_FUNC Type
      - MSG_FUNC Type
      - Analog Clock

Today's class was taught using v3.04.02 of the Microchip Graphics Library. If you have been using older versions, please refer to the migration notes in the Release Notes section. Much has changed!

..\Microchip\Help

Graphics Library Help.chm

Graphics Library Help.pdf

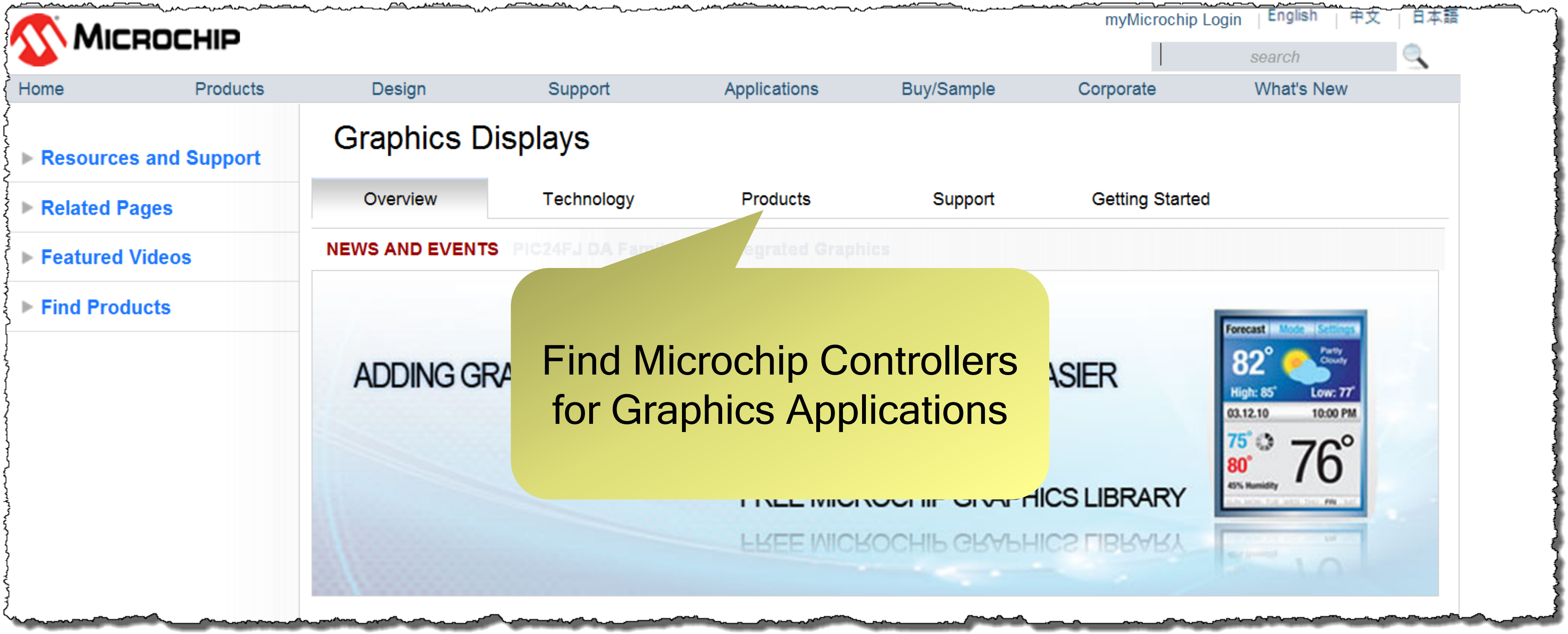
# Summary

Today you learned how to:

- Write programs to display images, fonts, and primitives on LCD panel
- Write programs to display and control widgets on LCD panel
- Graphically create application code to fully utilize Microchip Graphics Library

# Graphics Design Center

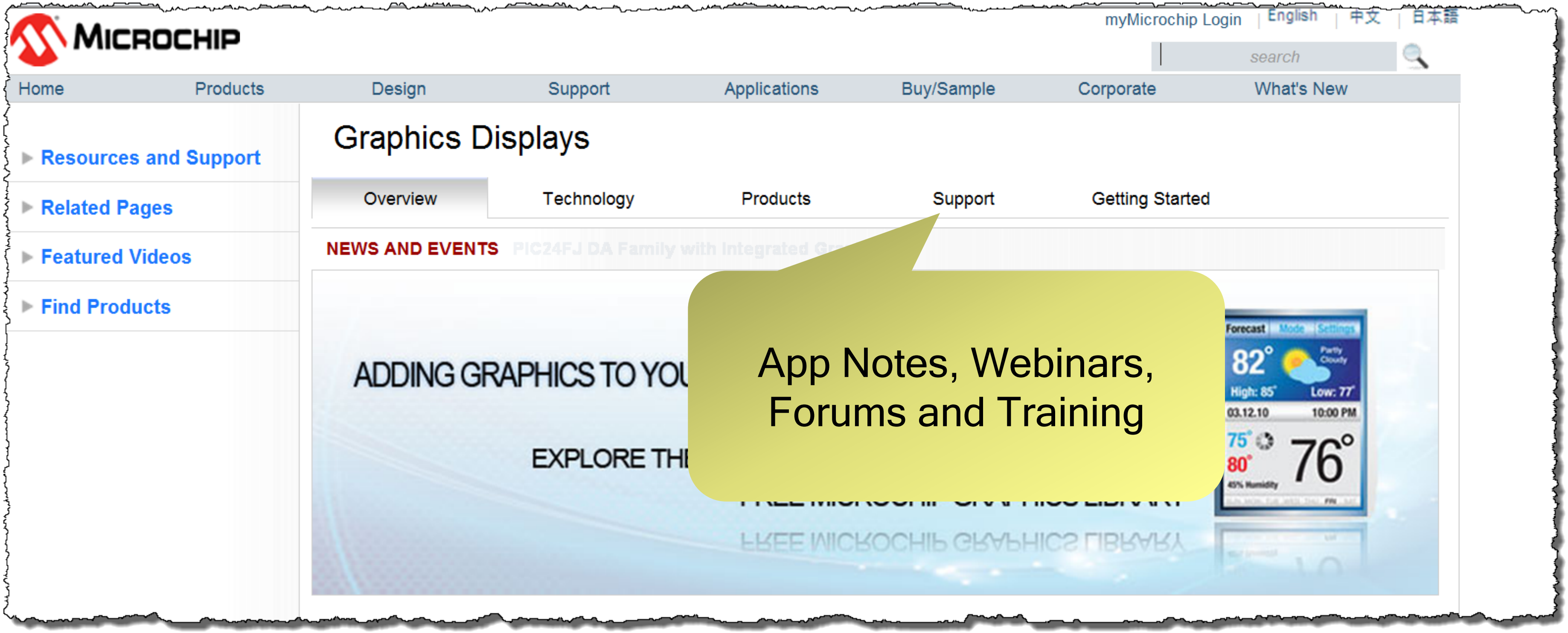
http://www.microchip.com/graphics



Find Microchip Controllers for Graphics Applications

# Graphics Design Center

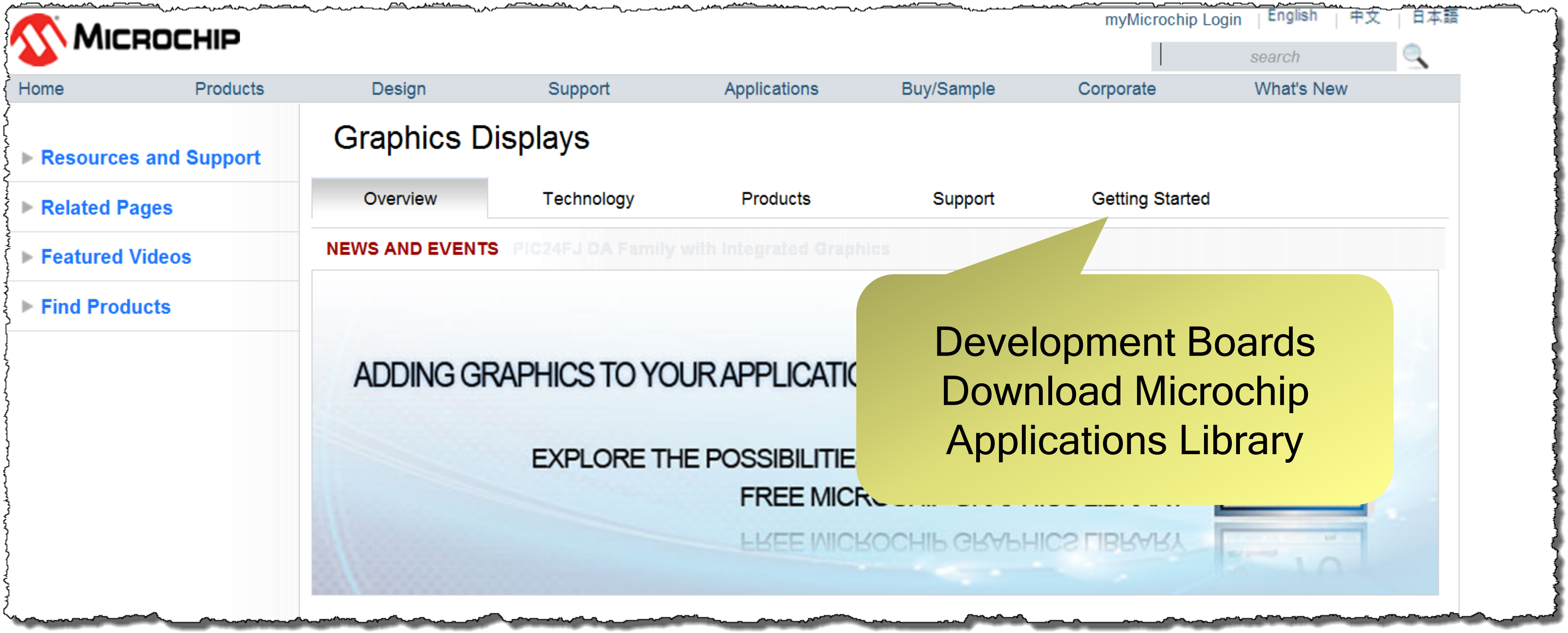
http://www.microchip.com/graphics



App Notes, Webinars, Forums and Training

# Graphics Design Center

http://www.microchip.com/graphics



MICROCHIP

myMicrochip Login | English | 中文 | 日本語

search

Home | Products | Design | Support | Applications | Buy/Sample | Corporate | What's New

- Resources and Support
- Related Pages
- Featured Videos
- Find Products

## Graphics Displays

Overview | Technology | Products | Support | Getting Started

NEWS AND EVENTS PIC24FJ DA Family with Integrated Graphics

ADDING GRAPHICS TO YOUR APPLICATI

EXPLORE THE POSSIBILITIE

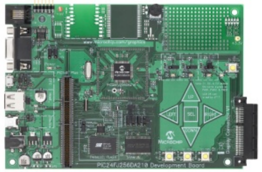
FREE MICR

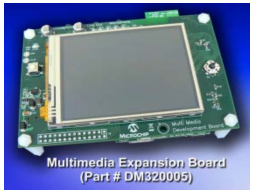
Development Boards
Download Microchip
Applications Library

# Tools

- Microchip Graphics Library v3.04.02
  - http://www.microchip.com/mla
- MPLAB® X IDE
  - http://www.microchip.com/mplabx
- MPLAB 16- and 32-bit Compilers
  - http://www.microchip.com/mplabc
- VGDD
  - http://virtualfab.it/mediawiki/index.php VGDD:Visual_Graphis_Display_Designer

# Tools





- Explorer 16 Development Board (DM240001)
- PIC24FJ256DA210 Development Board(DM240312)
- Multimedia Expansion Board (MEB) (DM320005)
- PIC32 Starter Kits (DM320001, DM320003-2, DM320004*)

**Note: * DM320004 (PIC32 Ethernet Starter Kit) is only compatible with MEB**

# Tools

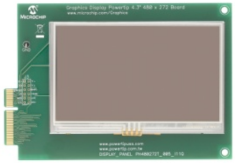
- MPLAB® Starter Kit for PIC24F (DM240011)

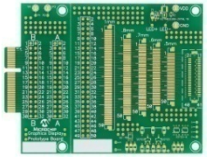
- MPLAB Starter Kit for PIC24H (DM240021)

- Graphics Display Board with 3.2” Display Kit (AC164127-3)

- PIC24FJ256DA210 Development Kit (DV164039)

# Tools

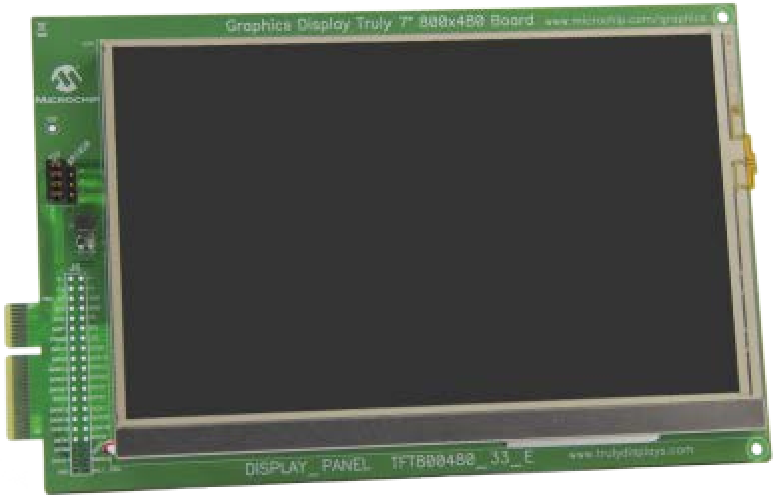




- Graphics LCD Controller PICtail™ Plus SSD1926 Board (AC164127-5)
- Graphics Display Truly 3.2” 240x320 Board (AC164127-4)
- Graphics Display Powertip 4.3” 480x270 Board (AC164127-6)
- Graphics Display Prototype Board (AC164139)

# High Resolution Displays

## Truly 7” WVGA (800x480)


Graphics Display Truly 7" 800x480 Board
www.microchip.com/graphics
DISPLAY_PANEL TFT800480_33_E
www.trulydisplays.com

- **Resistive 4-wire touch screen**
  - **AR1020 Touch Controller**
- **Works with Epson S1D13517 Board**

**Part Number:**
**AC164127-9**

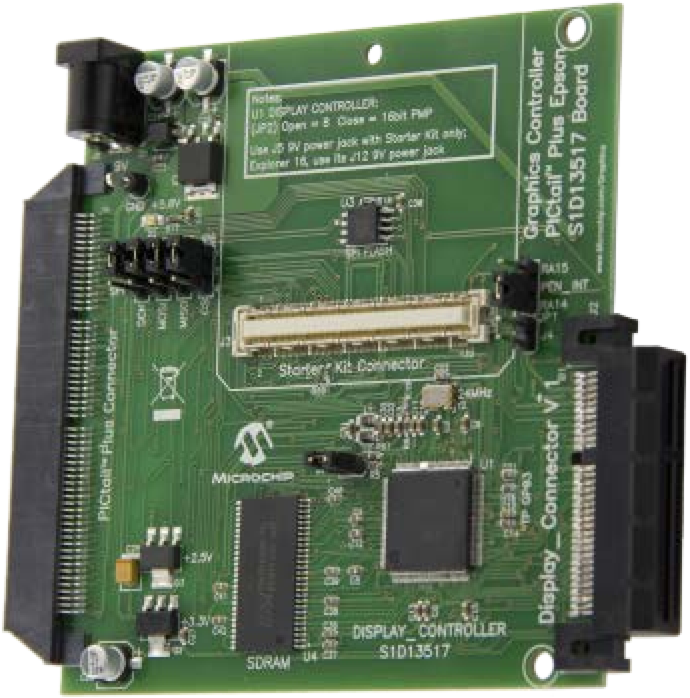
## Truly 5.7” VGA (640x480)

Graphics Display Truly 5.7" 640x480 Board
www.microchip.com/graphics
DISPLAY_PANEL TFT640480_8_E
www.trulydisplays.com

- Resistive 4-wire touch screen
  - AR1020 Touch Controller
- Works with Epson S1D13517 Board and others

Part Number:
AC164127-8

# Graphics Epson S1D13517 Board

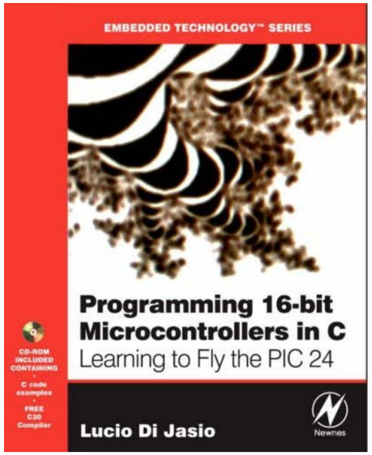

Graphics Controller PICtail™ Plus Epson S1D13517 Board
Starter Kit Connector
Display_Connector V1
PICtail™ Plus Connector
MICROCHIP
SDRAM
DISPLAY_CONTROLLER S1D13517

The Graphics Controller PICtail™ Plus Epson S1D13517 Board allows evaluating Microchip Technology's solution and graphics library for 16- and 32-bit microcontrollers. The Epson S1D13517 offers hardware acceleration for alpha-blending, transparency, animation, multiple buffering, and picture in picture. The kit is compatible with the Explorer 16 development board (DM240001) or one of the PIC32 Starter Boards (DM320001, DM320003).

The features include:

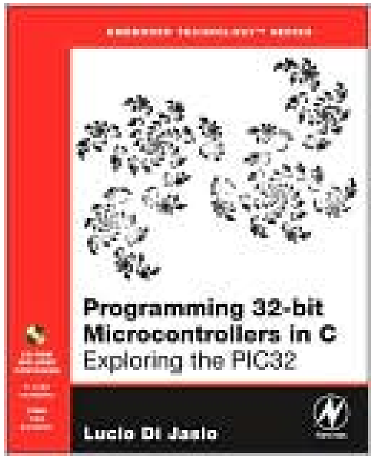
- Graphics display controller Epson S1D13517 supporting 18-bit HR-TFT, and 9/12/18/24 bit TFT interface
- PIC32 Starter Kit Connector
- 16 Megabit (2Mx8) serial flash memory for additional data storage
- Display connector for interfacing with different display boards
- PICtail Plus Interface for connecting to Explorer 16 Development Board
- PIC32 Starter Kit Connector

# Suggested Reading

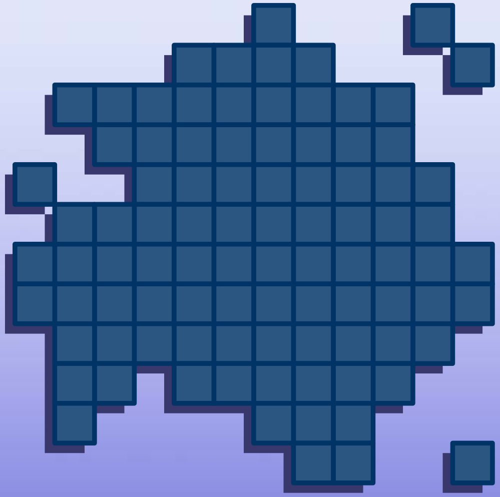

EMBEDDED TECHNOLOGY™ SERIES
Programming 16-bit Microcontrollers in C
Learning to Fly the PIC 24
CD-ROM INCLUDED CONTAINING
C code examples
FREE C30 Compiler
Lucio Di Jasio
Newnes

**<u>Programming 16-bit Microcontrollers in C</u>**
**by Lucio Di Jasio**
**ISBN-10: 0750682922**
**ISBN-13: 978-0750682923**

**http://www.flyingpic24.com**


EMBEDDED TECHNOLOGY™ SERIES
Programming 32-bit Microcontrollers in C
Exploring the PIC32
Lucio Di Jasio

**<u>Programming 32-bit Microcontrollers in C</u>**
**by Lucio Di Jasio**
**ISBN-10: 0750687096**
**ISBN-13: 978-0750687096**



# Thank You!

# Trademarks