



MICROCHIP

Regional Training Centers

HIF 2131A

Designing with Microchip's Graphics Library

*Class Creators: Kristy Seymour and Paolo Tamayo
Microchip Technology Inc.*

Class Objectives

After this class you will be able to:

- **Write programs to display images, fonts, and primitives on an LCD panel**
- **Write programs to display and control widgets on an LCD panel**
- **Create GUI application code to fully utilize the Microchip Graphics Library**



Agenda

- Overview of the Graphics Library
- Graphics Hardware Review
- Graphics Library Primitive Layer
- Using Fonts and Images
- **Lab 1 – GUI Splash Screen**
- Drawing Widgets
- **Lab 2 – Creating a Simple Icon Menu**
- Interfacing the User (Message Interface)
- **Lab 3 – User Interface using Message Callback**
- Advanced Features (Drawing Callback)
- **Lab 4 – User Interface using Draw Callback**
- Putting it All Together (Multiple Screens)
- **Lab 5 – Full Application**
- PIC32 Multimedia Expansion Board
- **Lab 6 – Graphics**



MICROCHIP

Regional Training Centers

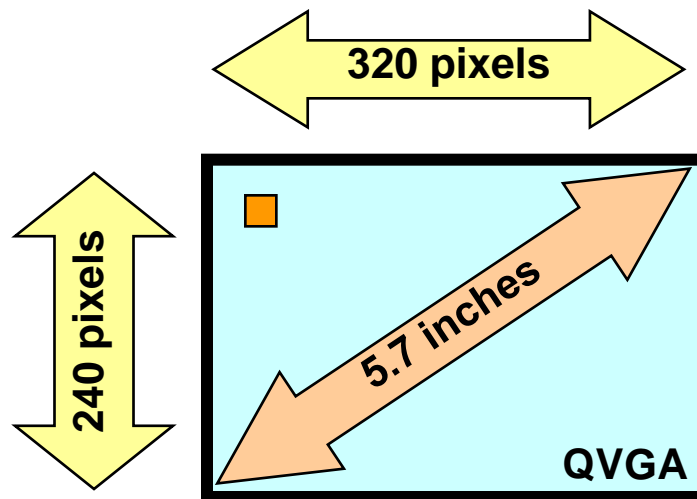
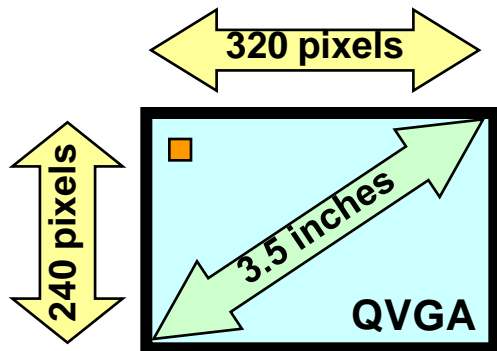
Microchip Graphics Library Overview



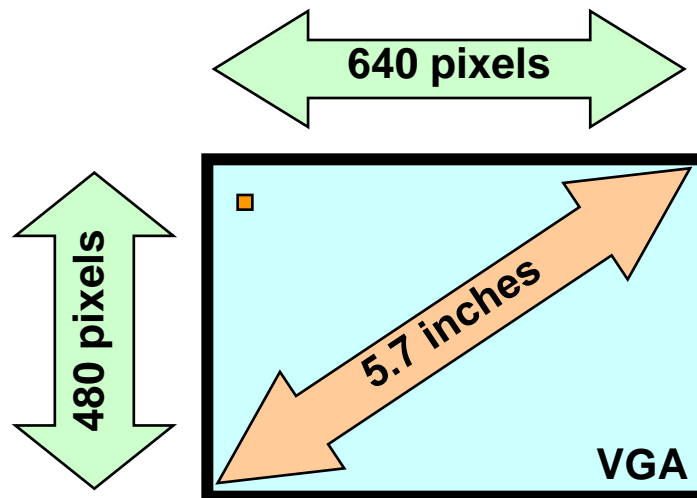
Terminology

- **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)

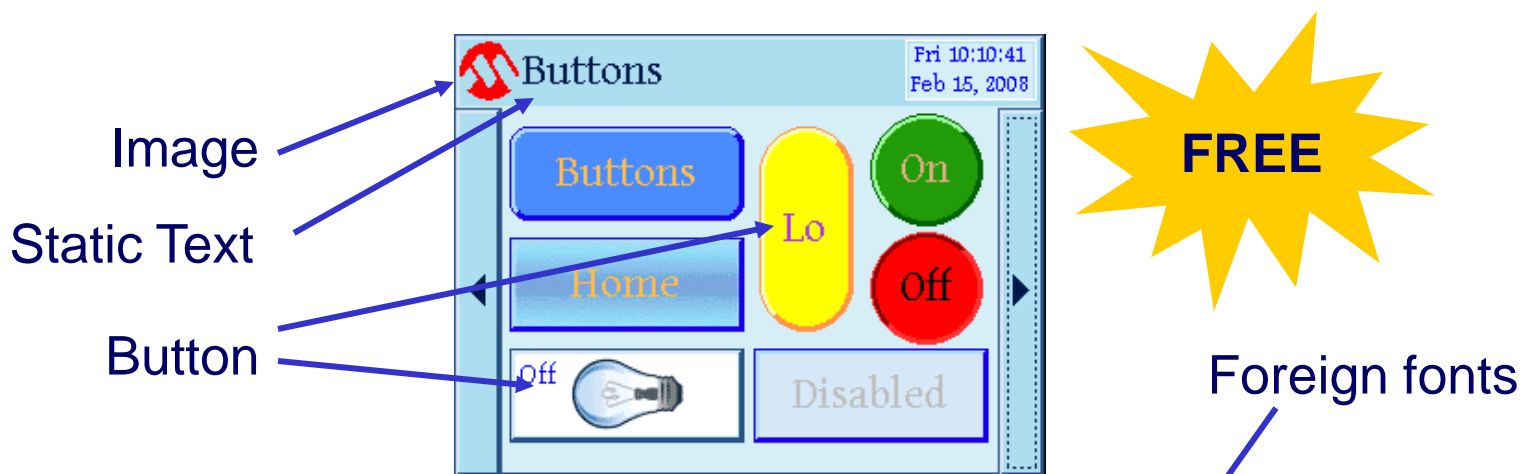
Terminology



- Resolution
- Display Size
- Pixel Size

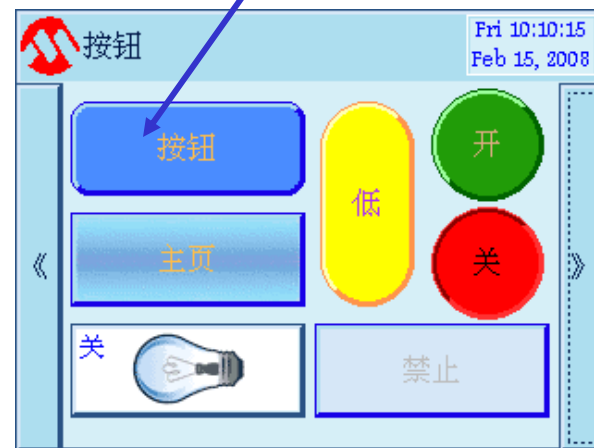


Graphics Library

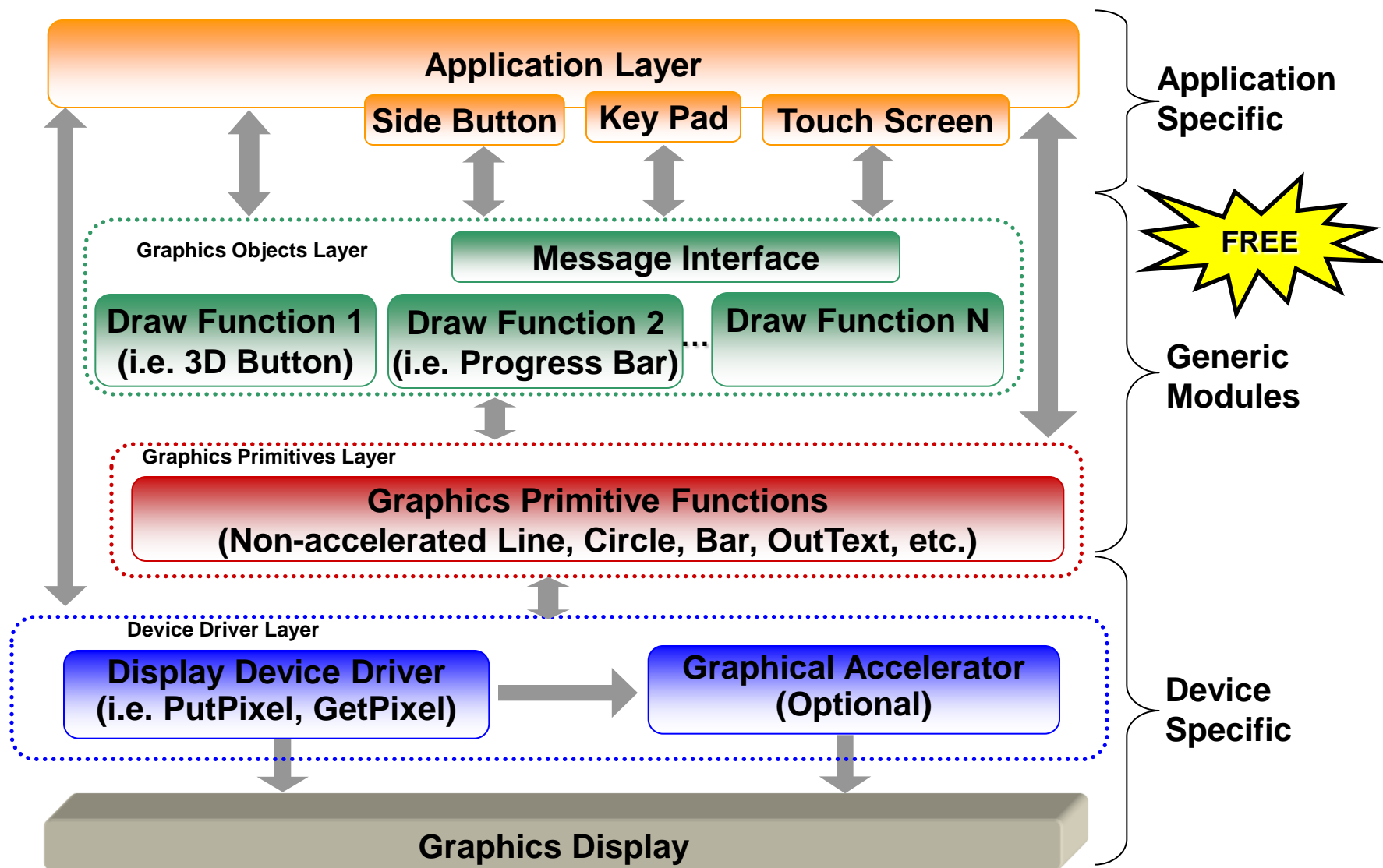


Key advantages:

- Works with 16- and 32-bit PIC® MCUs
- Modular design – compile only what you need!
- Supports multiple user interfaces
- Not dependent on display size or resolution
- Low-cost, full featured development tools
- Free to Microchip customers
 - Source code included
 - Multiple low-level drivers included



Library Overview



Graphics Design Center



Microcontrollers and
Software supporting
Graphics Applications

Home Products Design Support Applications Buy/Sample Corporat

Graphics Displays

Overview **Solutions** Featured Products **Training & Support** Getting Started

Easy and cost-effective graphics solution

- Free Graphics Display Designer (visual design tool)
- Free Graphics Library
- Low cost and full-featured development t
- Full documentation, application notes
- Web seminars and hands-on training

Targeted at graphical interface ap

- TFT, OLED, C-STN, Monochrome display
- Up to VGA (640x480) resolution
- Up to 16 bit per pixel (65K colors)


Industry's broadest portfolio

- New PIC24FJ DA and PIC32MX5
- 16 & 32-bit PIC[®] MCUs
- Up to 100 pins
- Up to 24 direct capacitive touch o
- eXtreme Low Power Technology
- PIC[®] MCUs integrate USB and Ethernet MAC

**Development Kits,
Development Boards,
PICtail? Plus Daughter
Boards**

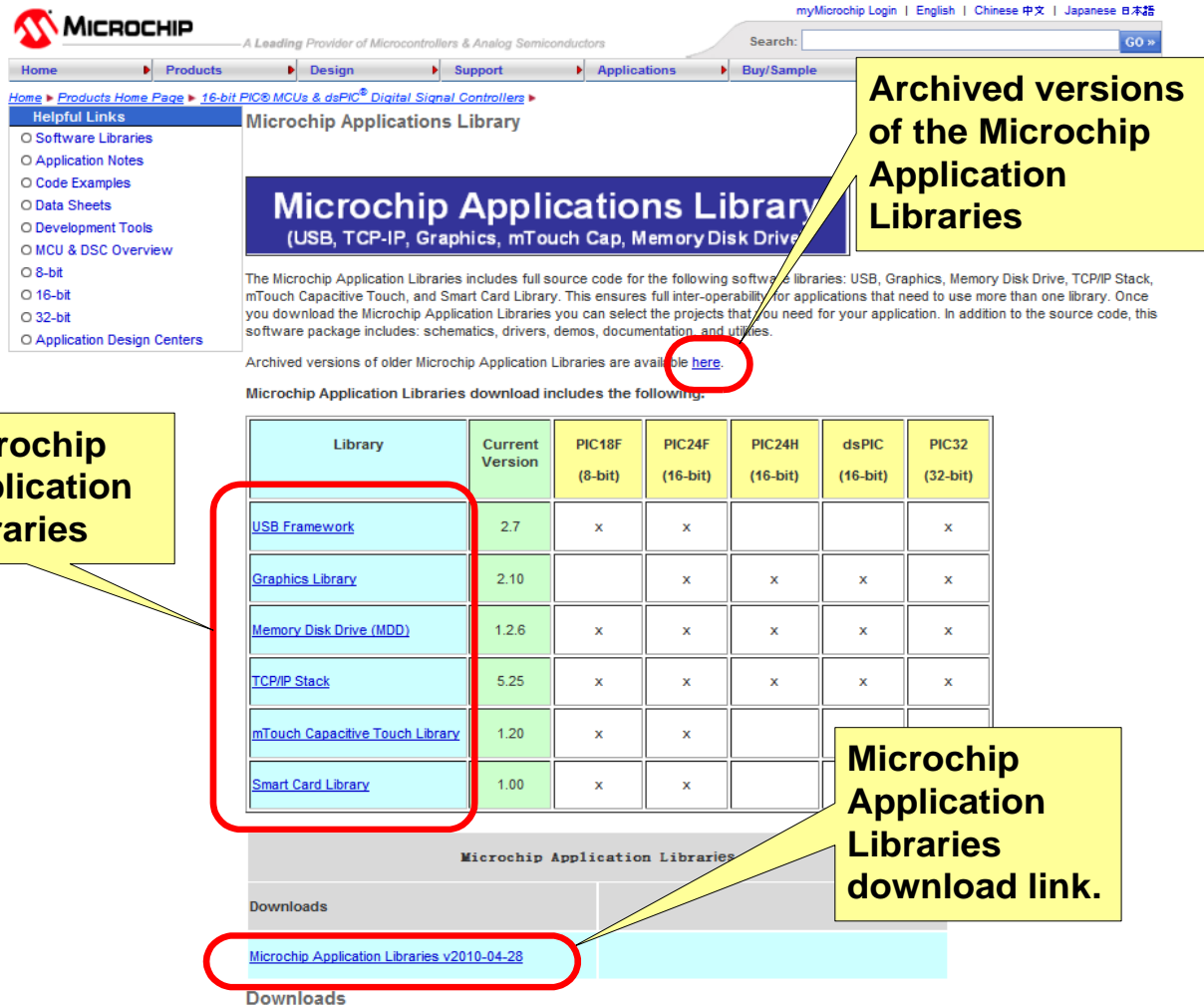
**Training and Design
Support
(Application Notes,
RTC classes, Webinars)**

**Microchip
Graphics Library
download link**



<http://www.microchip.com/graphics>

Microchip Application Libraries



Archived versions of the Microchip Application Libraries

Microchip Application Libraries

The Microchip Application Libraries includes full source code for the following software libraries: USB, Graphics, Memory Disk Drive, TCP/IP Stack, mTouch Capacitive Touch, and Smart Card Library. This ensures full inter-operability for applications that need to use more than one library. Once you download the Microchip Application Libraries you can select the projects that you need for your application. In addition to the source code, this software package includes: schematics, drivers, demos, documentation, and utilities.

Archived versions of older Microchip Application Libraries are available [here](#).

Microchip Application Libraries download includes the following:

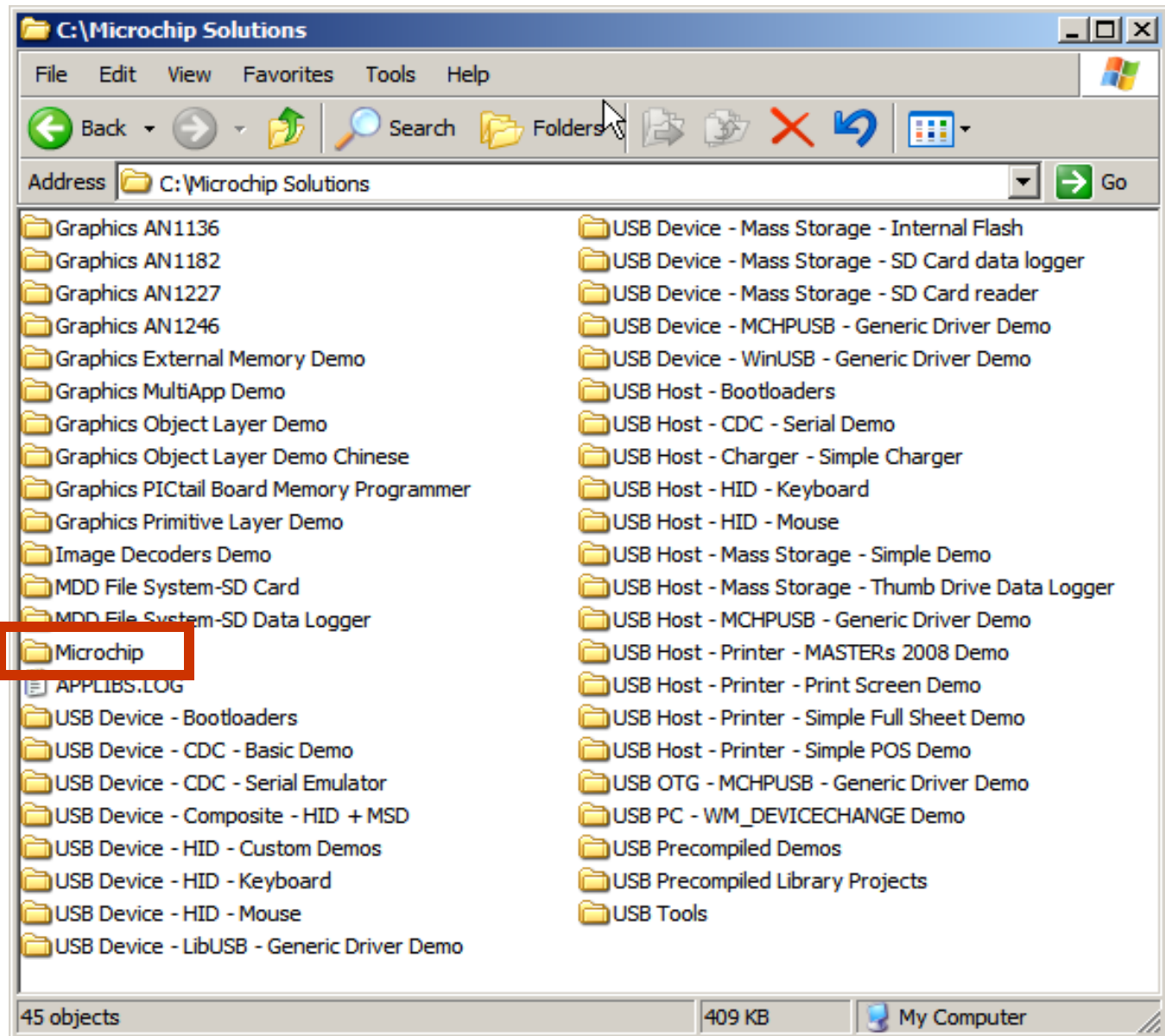
Library	Current Version	PIC18F (8-bit)	PIC24F (16-bit)	PIC24H (16-bit)	dsPIC (16-bit)	PIC32 (32-bit)
USB Framework	2.7	x	x			x
Graphics Library	2.10		x	x	x	x
Memory Disk Drive (MDD)	1.2.6	x	x	x	x	x
TCP/IP Stack	5.25	x	x	x	x	x
mTouch Capacitive Touch Library	1.20	x	x			
Smart Card Library	1.00	x	x			

Microchip Application Libraries download link.

[Microchip Application Libraries v2010-04-28](#)

<http://www.microchip.com/MAL>

Library Directory Structure



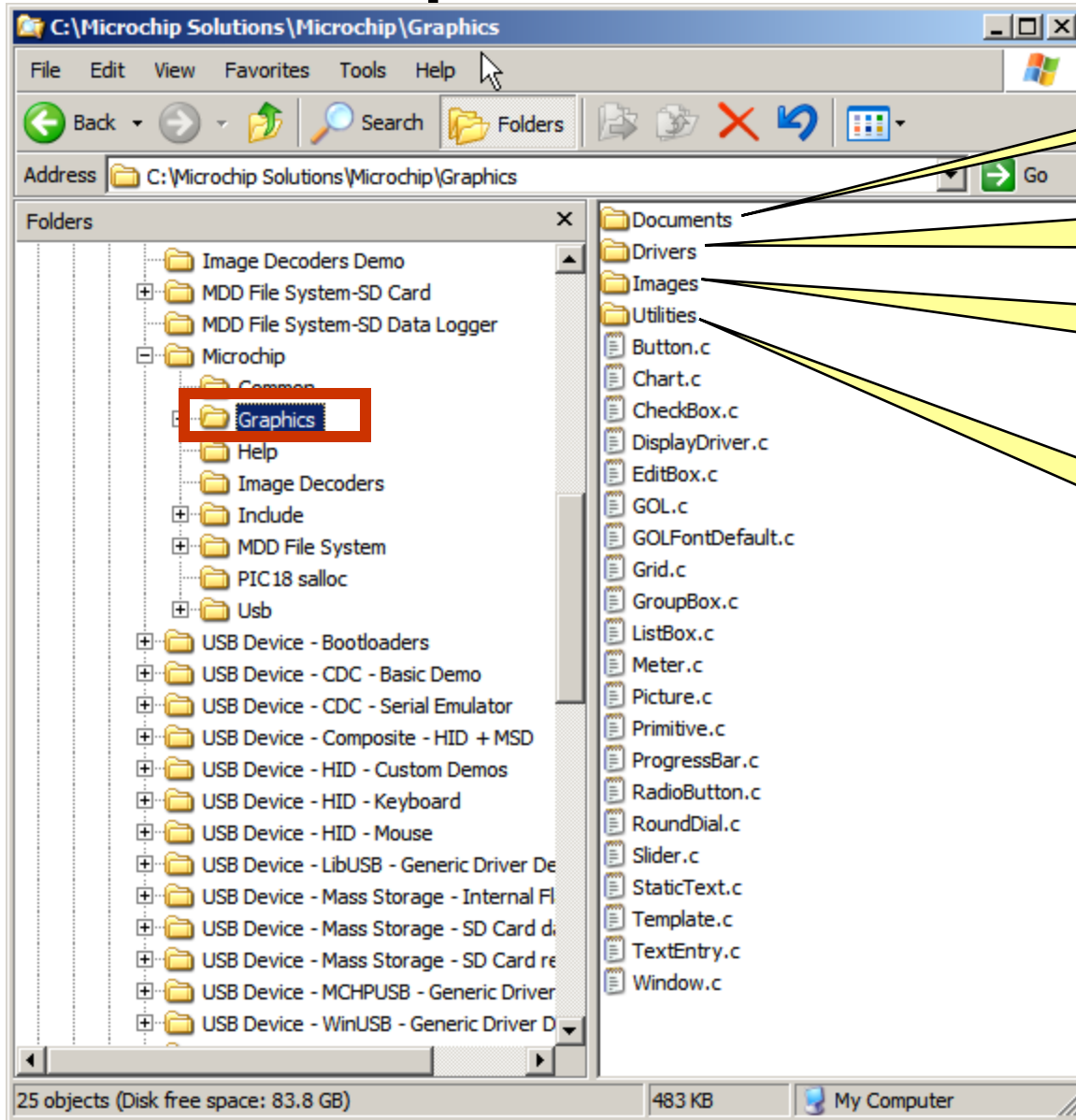


MICROCHIP

Regional Training
Centers

Library Directory Structure

Graphics Source and Tools



Schematics

**Display
Driver
Source Files**

**Demo
Bitmaps and
Icons**

**Graphics
Resource
Converter**

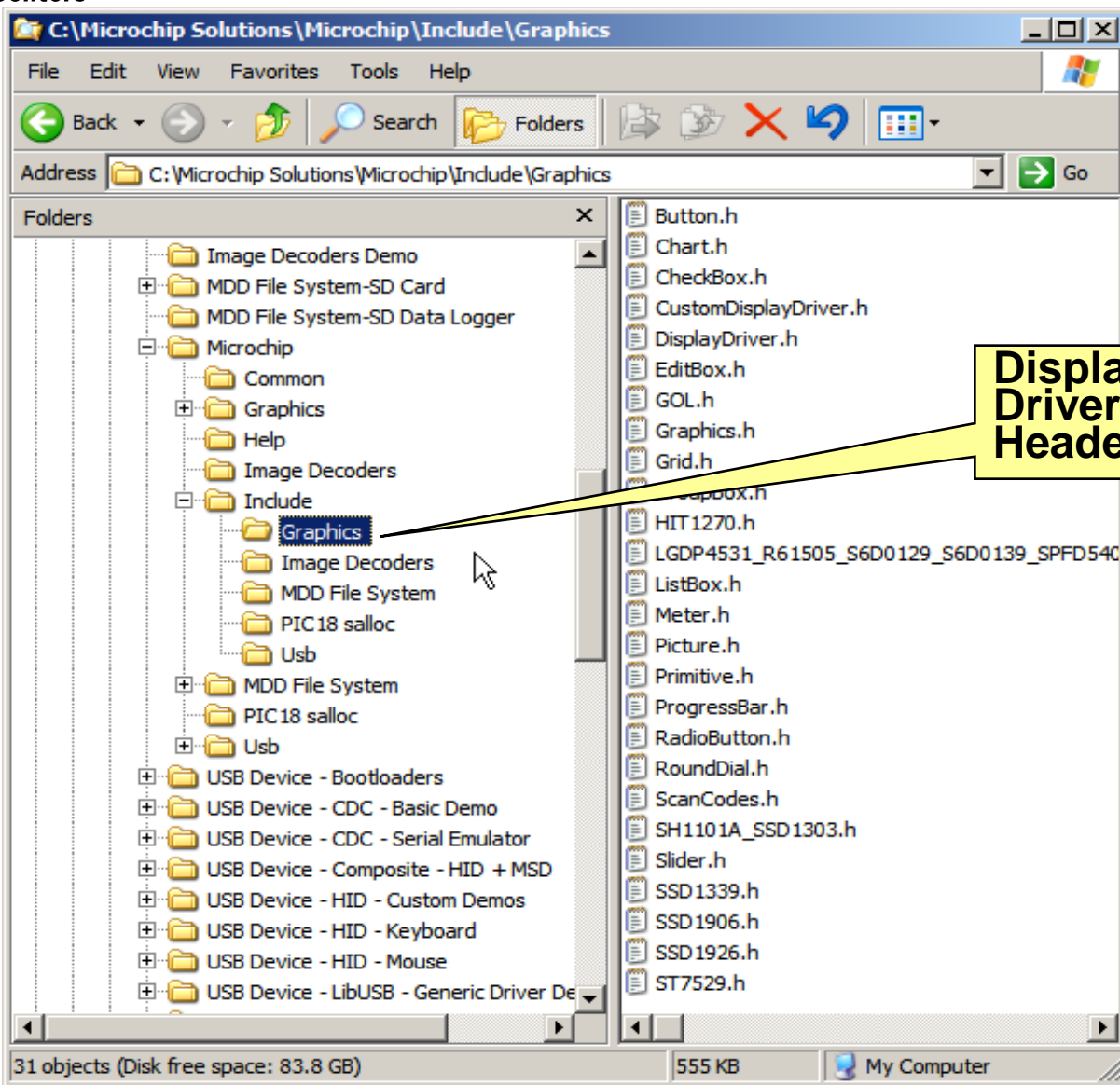


MICROCHIP

Regional Training
Centers

Library Directory Structure

Graphics Header Files



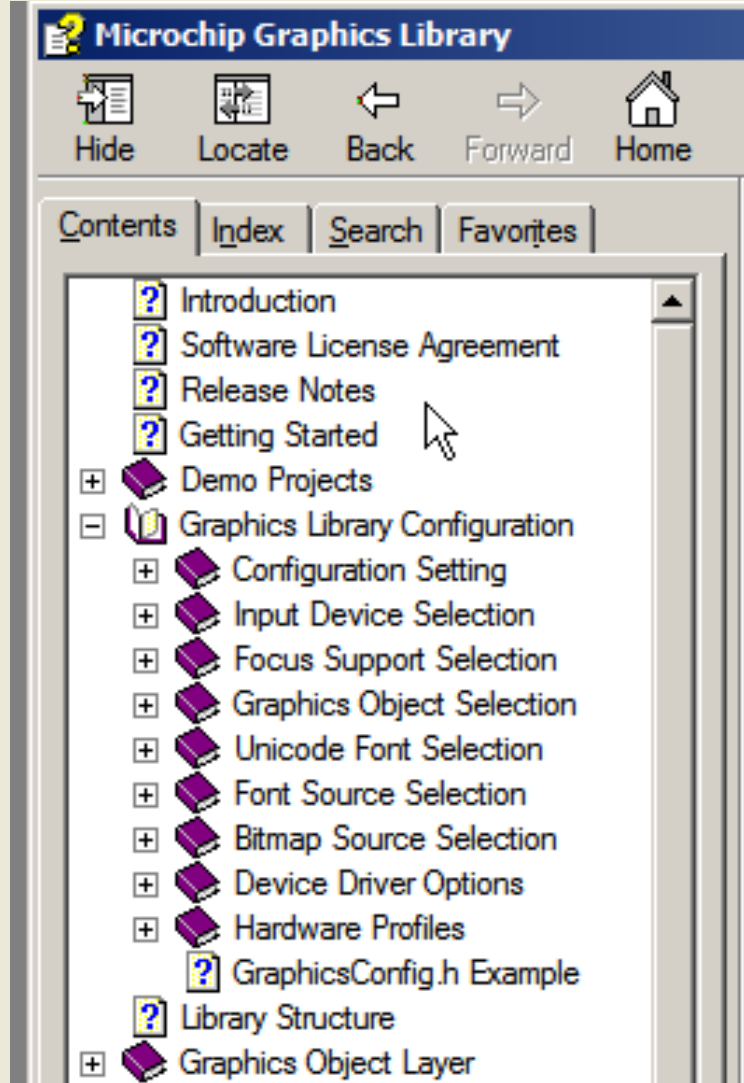
**Display
Driver
Header Files**



MICROCHIP

Regional Training
Centers

Library Help



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

C:\Microchip Solutions\Microchip\Help



Graphics Library Help.htm



MICROCHIP

Regional Training Centers

Getting to Know...
LCD System Hardware



RGB Color Model

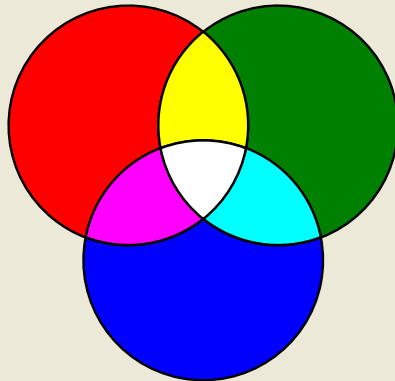
Definition

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

- Each component represents intensity
- www.colorschemer.com has free tool to get RGB

- **16-bit
(65,536 colors)**

- Red: 5 bits
- Green: 6 bits
- Blue: 5 bits
- RGB 565



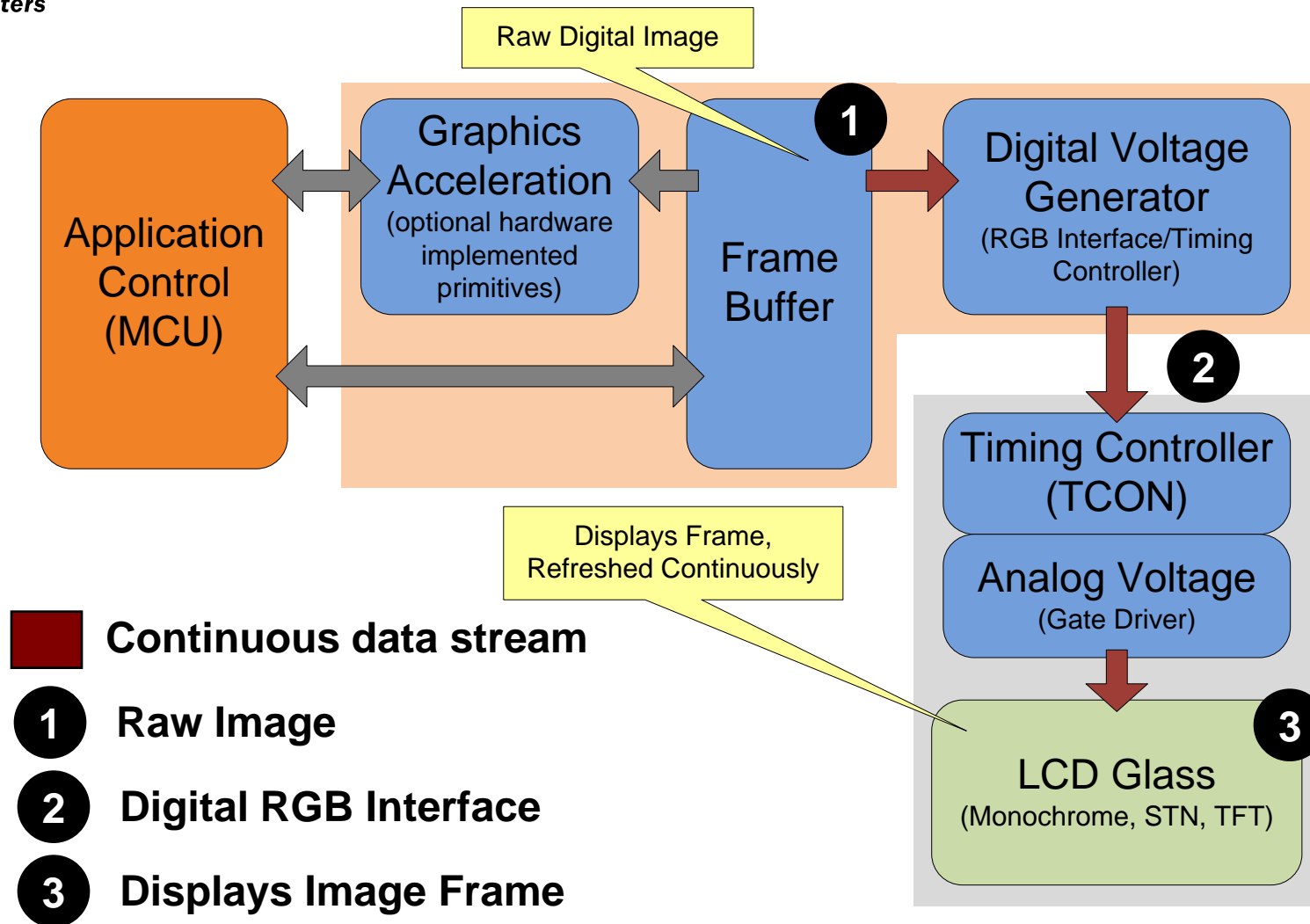
- **18-bit
(262,144 colors)**

- Red: 6 bits
- Green: 6 bits
- Blue: 6 bits
- RGB 666

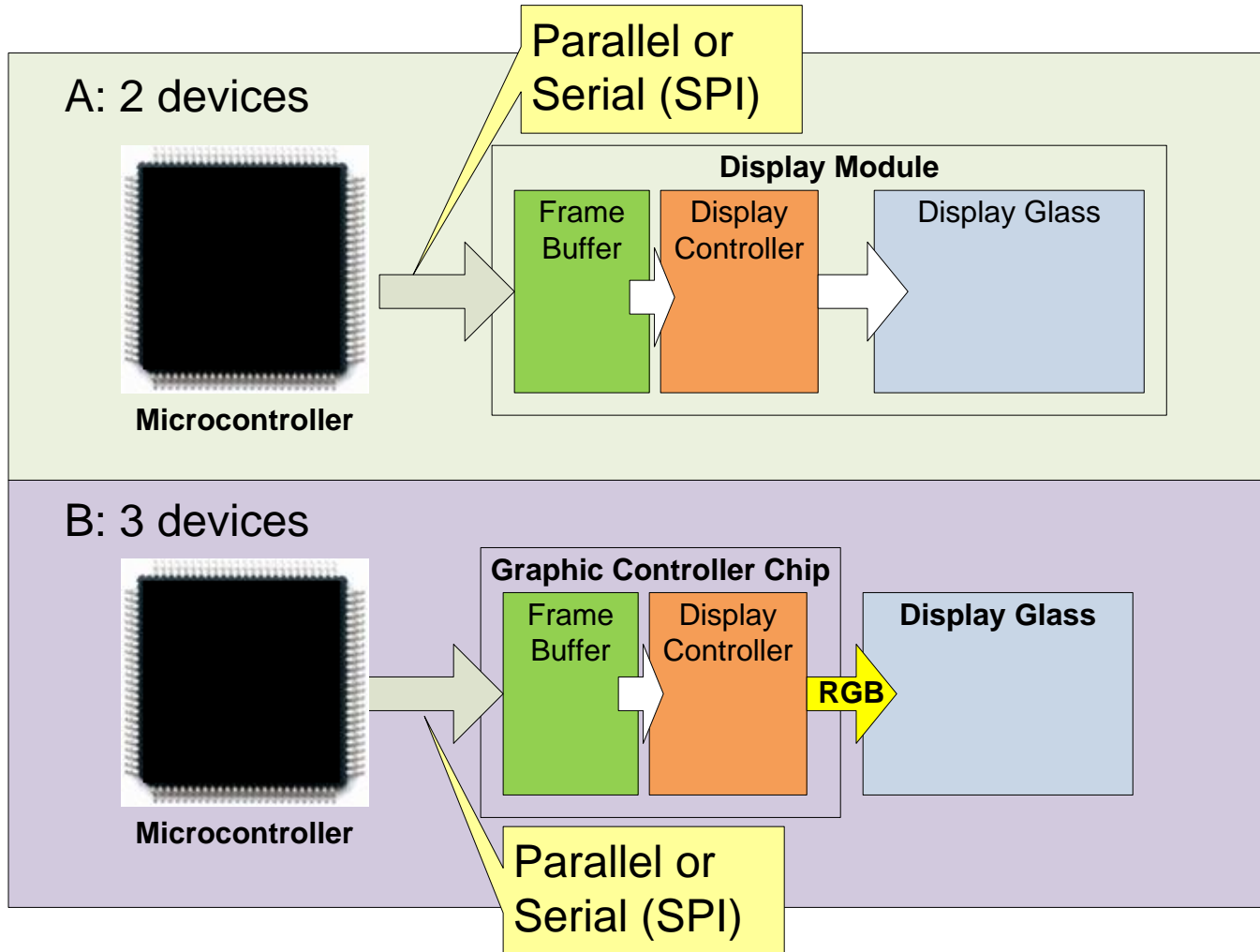
- **24-bit
(1.678M colors)**

- Red: 8 bits
- Green: 8 bits
- Blue: 8 bits
- RGB 888

Graphic LCD System

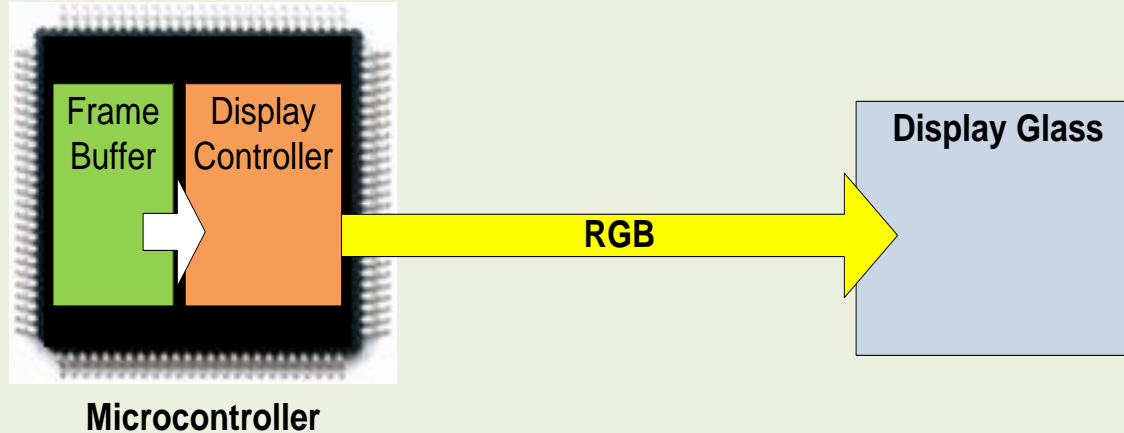


Types of LCD Systems

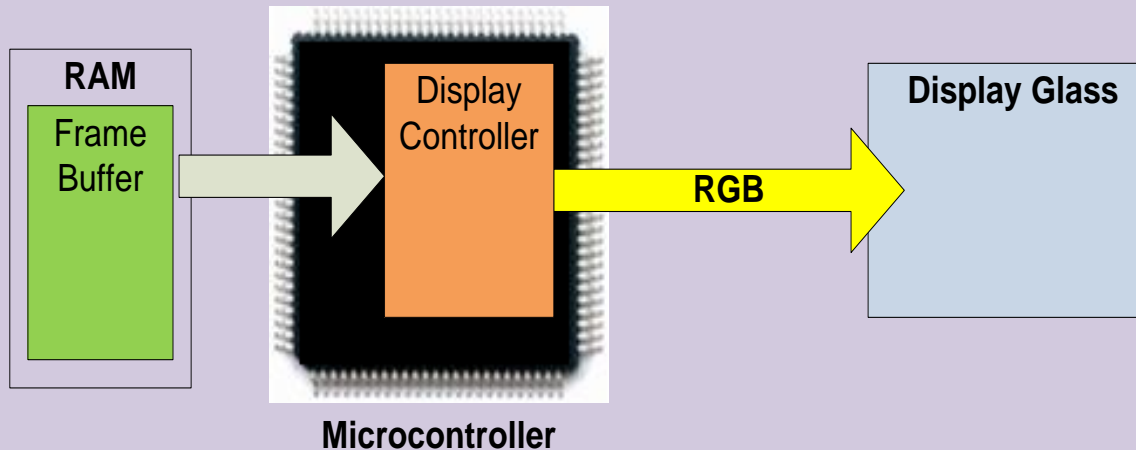


Types of LCD Systems

C: 2 devices



D: 3 devices



Graphics Solutions Drivers

Currently Supplied Driver Code Graphics Controllers (TFT / STN)

Vendor	Part Number
HiTech	HIT1270
LG	LGDP4531
Renesas	R61505U
Orise Tech	SPFD5408A
Samsung	S6D0129/0139
Solomon Systech	SSD1906, SSD1926
Solomon Systech	SSD1339, SSD1289**
Sitronix	ST7529
Ilitek	ILI9320
UltraChip	U1610
Microchip	PIC24FJ256DA210

Currently Supplied Driver Code OLED Controllers

Vendor	Part Number
Sino Wealth	SH1101A
Solomon Systech	SSD1303

Currently Supplied Driver Code Timing Controllers

Vendor	Part Number
HiMax	HX8238, HX8257 HX8247
Solomon Systech	SSD1289**

**** Used both as a Graphics & Timing
Controller**



MICROCHIP

Regional Training Centers

**PIC with Integrated
Graphics Module**

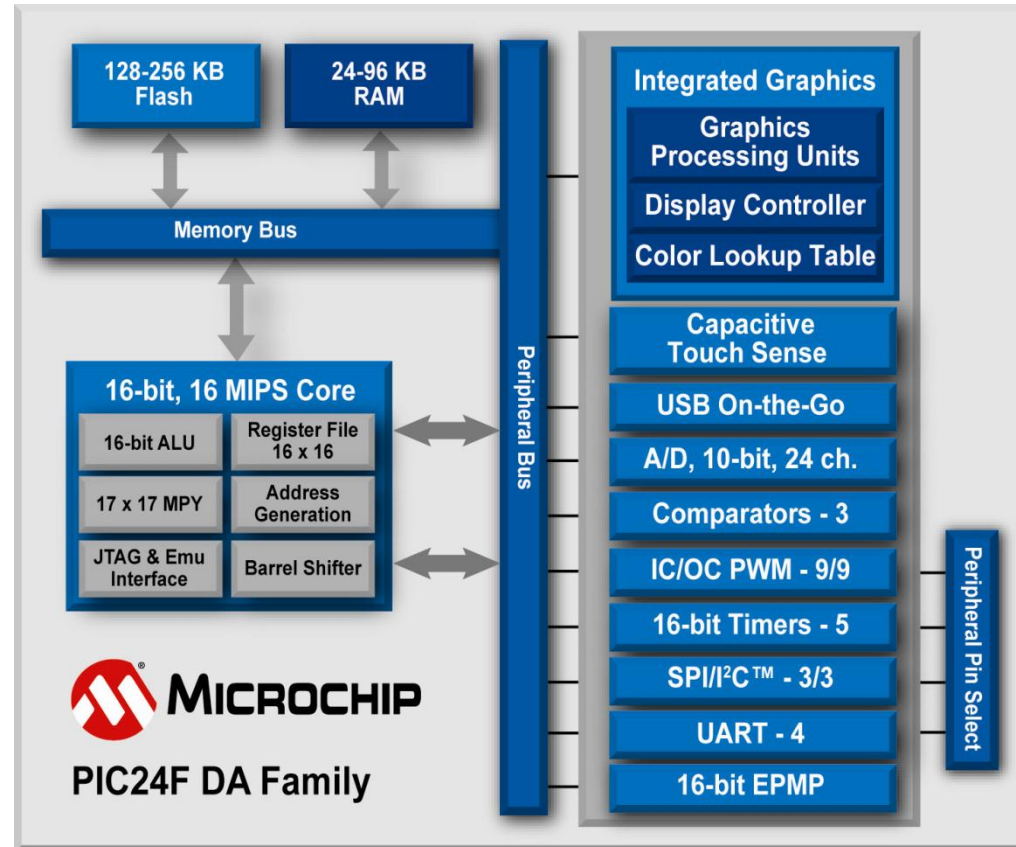
Introducing the PIC24FJ DA Family

Graphics Features

- Graphics Acceleration Units drive performance
- Direct interface to STN, TFT, and OLED displays
- 96 KB RAM and Color Lookup Table enable 256 16-bit colors

Advanced features

- 24 channels of mTouch™ Capacitive Sensing
- Full-speed USB Embedded Host/Device/OTG Module



Available Packages

64-pin QFN, TQFP

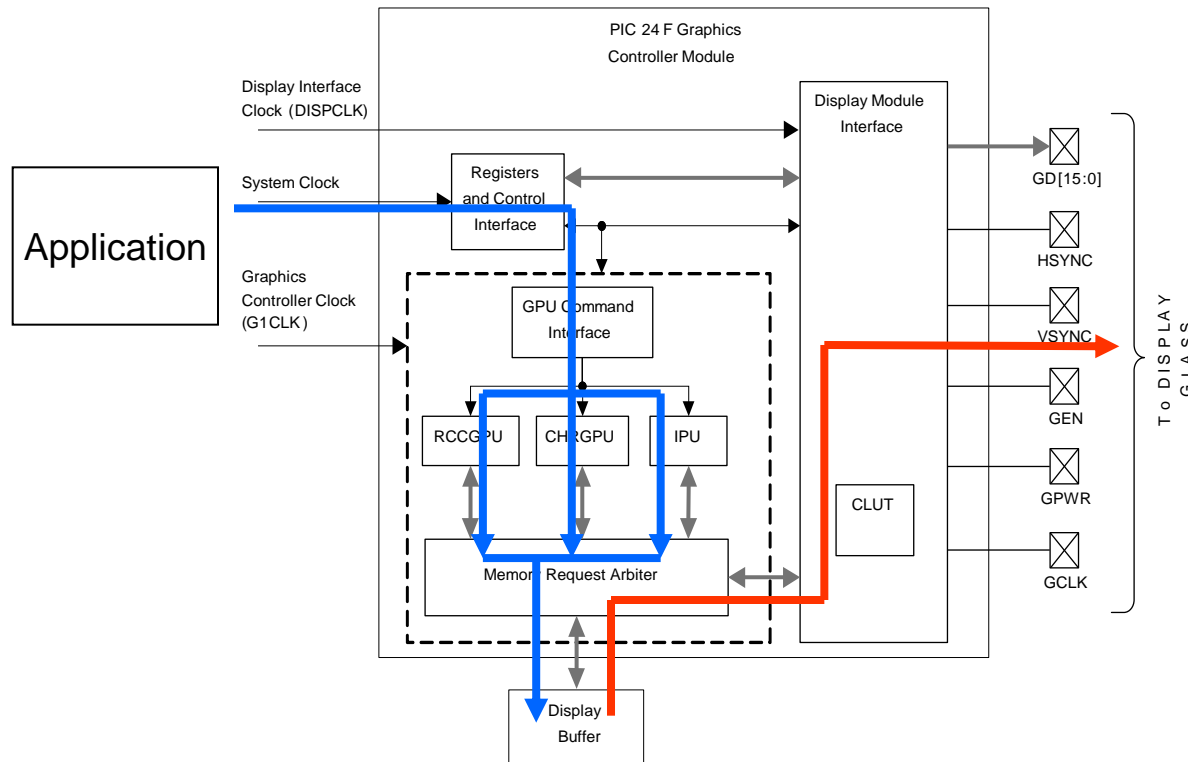
100-pin TQFP

121-pin BGA

MCHP Graphics Module Features

- **Supports ...**
 - TFT (9/12/18/24 bits)
 - Color STN (4/8/16 bits)
 - Mono STN (4/8/16 bits)
- **Programmable synchronization signals**
 - Refresh rate
 - Timing of the vertical and horizontal signals
- **Graphics Processing Units (GPUs)**
 - Character Graphics Processing Unit (CHRGPU)
 - Rectangle Copy Graphics Processing Unit (RCCGPU)
 - Inflate Processing Unit (IPU)
 - Command FIFO to queue rendering commands, free up CPU to do other tasks
- **256 entries Color Look Up Table (CLUT)**
- **Color Depths of 1/2/4/8 and 16 bpp**
- **Programmable Display Resolutions**
 - Up to VGA (640x480) run at 8 bpp

Graphics Acceleration Units and On-Chip Display Controller



VGA @ 8bpp @ 60 Hz
VGA @ 16 bpp @ 30 Hz
WQVGA @ 16 bpp @ 60 Hz
QVGA @ 16 bpp @ 60 Hz






- Application sends commands to the Graphics Processing Units (GPU) to draw.
- While the GPU is drawing, Application can perform other tasks.
- Display controller automatically refreshes the display continuously
 - 4-/8-bit monochrome
 - 4-/8-/16-bit color STN
 - 9-/12-/18-/24-bit parallel RGB

Operation

Memory requirements for common display resolutions

Display Resolution		Color Depth/ Memory Requirement in (Bytes)				
Vertical	Horizontal	Internal Frame Buffer			Internal and/or External Frame Buffer	
		1 bpp (Mono)	2 bpp (4 shades)	4 bpp (16 shades)	8 bpp (256 colors)	16 bpp (65536 colors)
480	272	16,320	32,640	65,280	130,560	261,120
320	240	9,600	19,200*	38,400	76,800	153,600
160	240	4,800	9,600	19,200*	38,400	76,800
160	160	3,200	6,400	12,800	25,600	51,200
128	64	1,024	2,048	4,096	8,192	16,384

(*) May need to switch to DA devices with 96KB RAM

-  - PIC24FJ256DA106/DA110 family – 24K byte RAM
-  - PIC24FJ256DA206/DA210 family – 96K byte RAM
-  - PIC24FJ256DA110 family – 24K byte RAM, AND external SRAM



Color Lookup Table

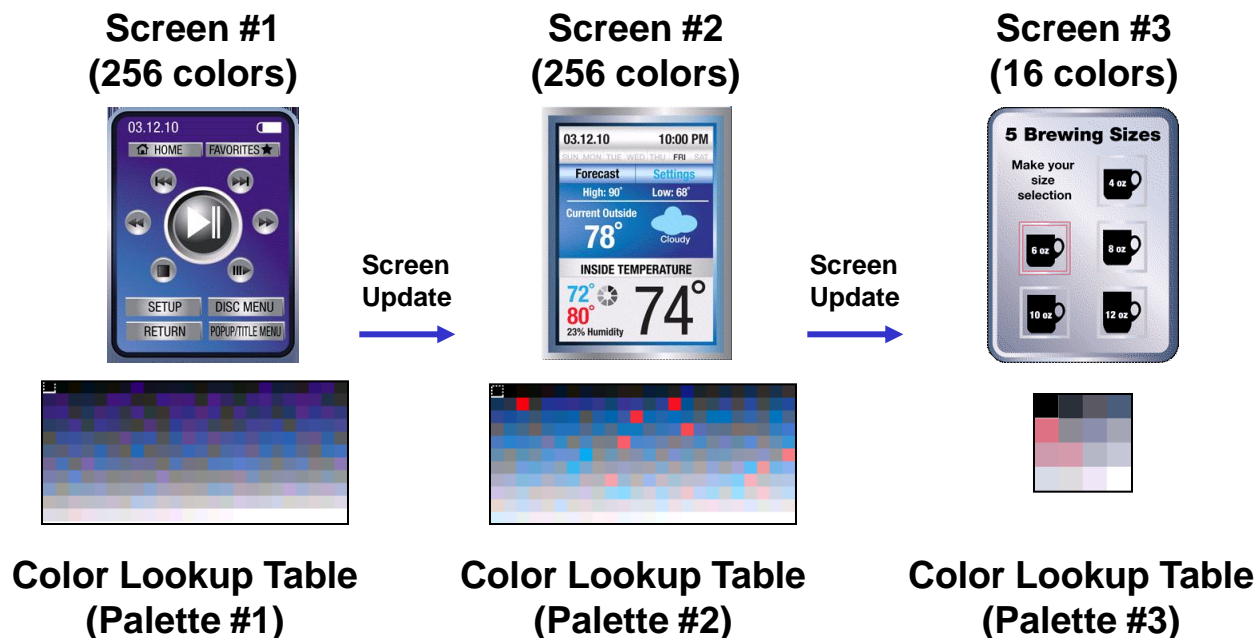
- **256 Entries Color Lookup Table**
- **Each entry can be 16 bits or 4 bits depending on the display type used**
- **TFT/CSTN displays**
 - **Each entry of the table is always 16-bit colors**
 - **Valid number of entries is 256**
- **MSTN**
 - **Each entry is always 4 bits wide or 16 color shades.**
 - **Valid number of entries depends on the display interface**
 - **16/8 – 256 entries**
 - **4 – 16 entries**
 - **2 – 4 entries**
 - **1 – 2 entries**

Table 43-2: RGB and Luminance Output from CLUT for Color and Monochrome Displays

Output Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGB	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0
Luma	Unused												M3	M2	M1	M0

Color Lookup Table

- Applications can perform dynamic switching of different color palettes
- Each palette can utilize a different set of up to 256 colors
- Every screen update may have a different color palette, supporting different set of up to 256 colors





MICROCHIP

Regional Training Centers

**Microchip Graphics
Primitive Layer**

Primitive Layer

- Talks to directly to device driver
- Configuration in **GraphicsConfig.h**:
 - Font image source (internal, external or both)
 - Bitmap source (internal, external or both)
 - Unicode support (AN1182)
- Must include in project:
 - `primitive.c`
 - `primitive.h`

Primitive Functions

■ Setup Functions:

- `InitGraph()` – initialize display
 - Defaults found in `primitive.c`
- `ClearDevice()`
 - Clears screen with current color
 - Puts cursor at (0,0)

Primitive Functions

Drawing Properties

Affect drawing until the next **Set**

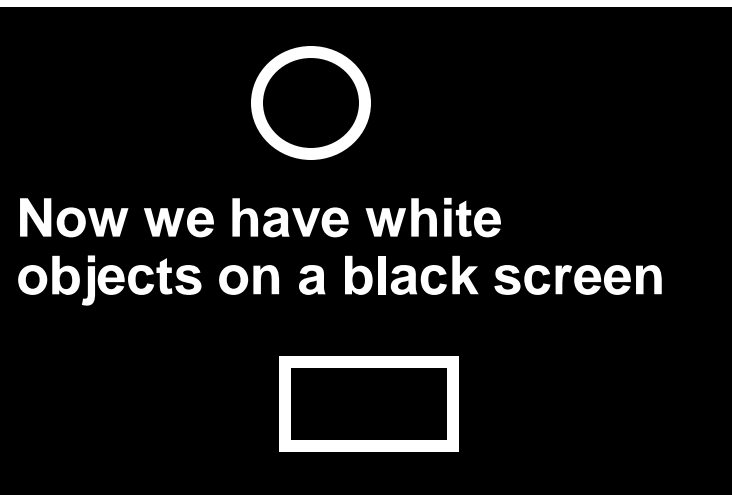
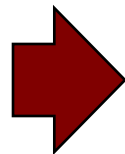
- **SetColor** (*COLOR*)
 - *COLOR* macros located in *driver.h* file
- **SetFont** (&*fontimage*)
 - &*fontimage* – pointer to font image structure
- **SetLineType** (*key*)
 - *SOLID_LINE*
 - *DOTTED_LINE*
 - *DASHED_LINE*
- **SetLineThickness** (*key*)
 - *NORMAL_LINE*
 - *THICK_LINE*

SetColor ()

Create new colors

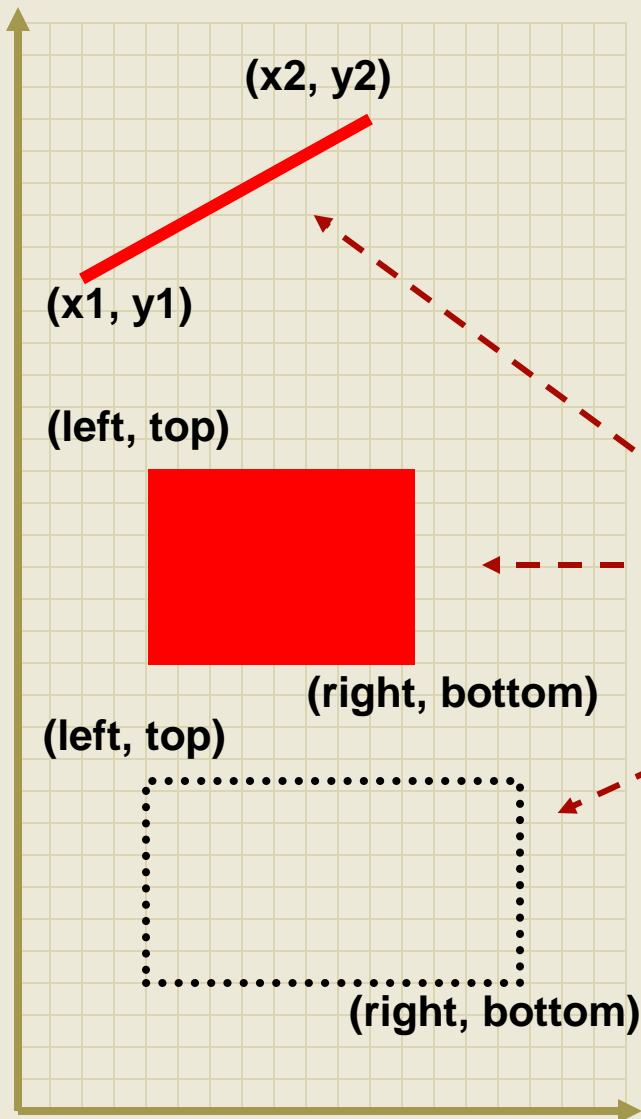
- Use `RGB565Convert (R,G,B)` macro
- Free tool to get RGB at www.colorschemer.com
- **Example:**
 - Set background color to black
 - Set drawing color to white

```
int main(void)
{
...
SetColor (BLACK) ;
ClearDevice () ;
SetColor (WHITE) ;
...
}
```





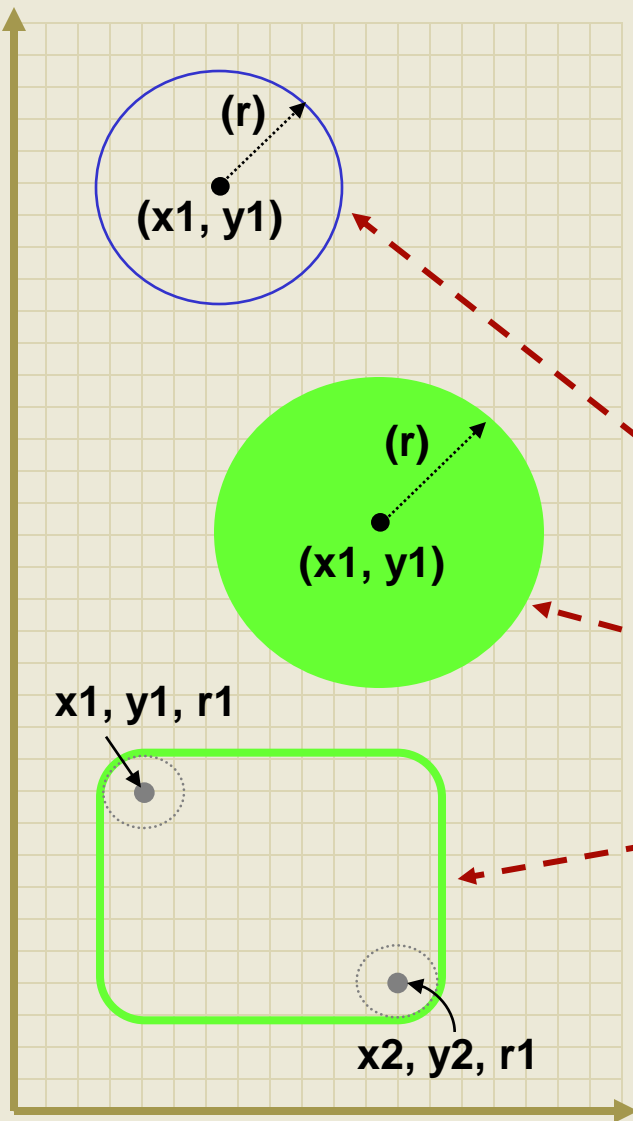
Primitive Drawing Functions



```
int main(void)
{
...
SetColor(BRIGHTRED);
SetLineType(SOLID_LINE);
SetLineThickness(THICK_LINE);
Line(x1, y1, x2, y2);
Bar(left, top, right, bottom);
SetColor(BLACK);
SetLineType(DOTTED_LINE);
Rectangle(left, top, right, bottom);
...
}
```



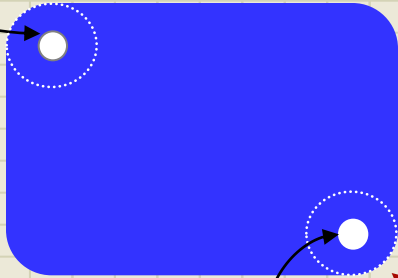
Primitive Drawing Functions



```
int main(void)
{
...
SetColor(BRIGHTBLUE);
SetLineType(SOLID_LINE);
SetLineThickness(NORMAL_LINE);
Circle(x1, y1, r);
SetColor(BRIGHTGREEN);
FillCircle(x1, y1, r);
SetLineThickness(THICK_LINE);
Bevel(x1, y1, x2, y2, r1);
...
}
```

Primitive Drawing Functions

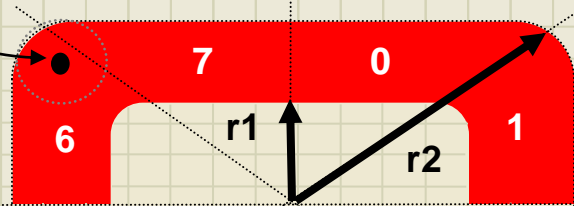
$x1, y1, r1$



$x2, y2, r1$

(xT, yT)

Octant = 0xC3



(xB, yB)

```
int main(void)
{
    ...
    SetColor(BRIGHTBLUE);
    SetLineType(SOLID_LINE);
    SetLineThickness(NORMAL_LINE);
    FillBevel(x1, y1, x2, y2, r1);
    SetColor(BRIGHTRED);
    Arc(xT, yT, xB, yB, r1, r2, Octant);
    ...
}
```



MICROCHIP

Regional Training Centers

**Microchip Graphics
Primitive Layer -- 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.

- **Library supports converted...**
 - True Type and Open Type Fonts
 - Raster (bitmapped) Fonts
- **Font Images ...**
 - May be stored in internal flash or external memory
 - Filtered font images available
- **Unicode support via multi-byte characters**
 - AN1182 -- Fonts in the Microchip Graphics Library

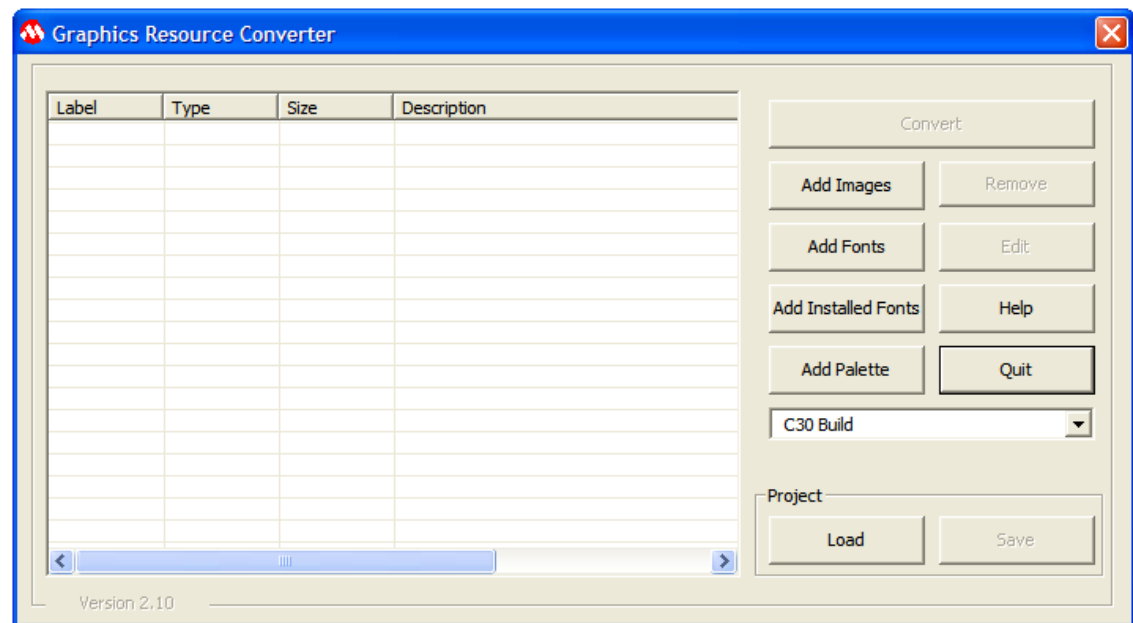
Using Fonts

■ Bitmap and Font Converter

- Free utility provided with the library
- Convert to formats the library understands

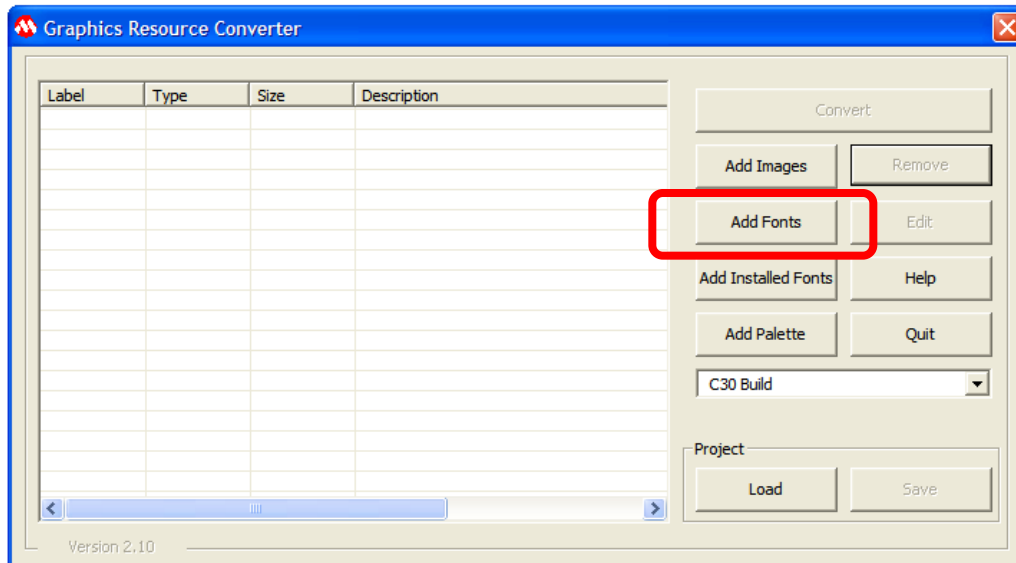
Launch Tool:

Start ▶ Programs ▶ Microchip
▶Graphics Library v2.10
▶Graphics Resource Converter



Font Conversion

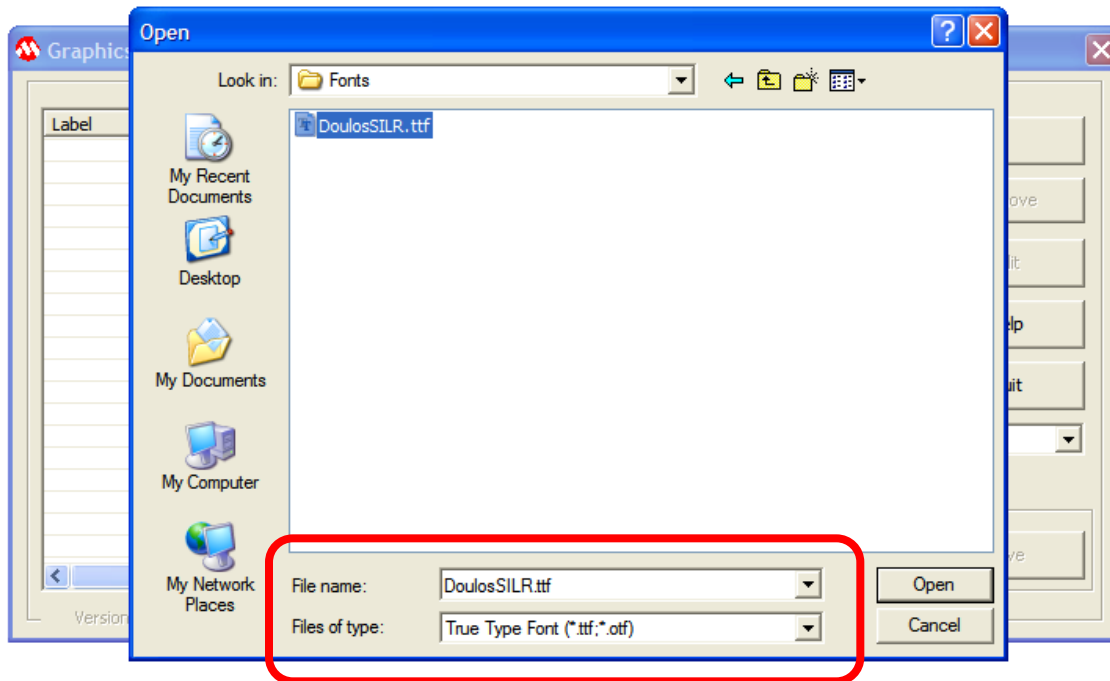
- Import fonts from file
 - True Type (*.ttf)
 - Open Type (*.otf)
 - Raster (*.fnt)



Note: Fonts can also be imported from installed fonts

Using Fonts Example

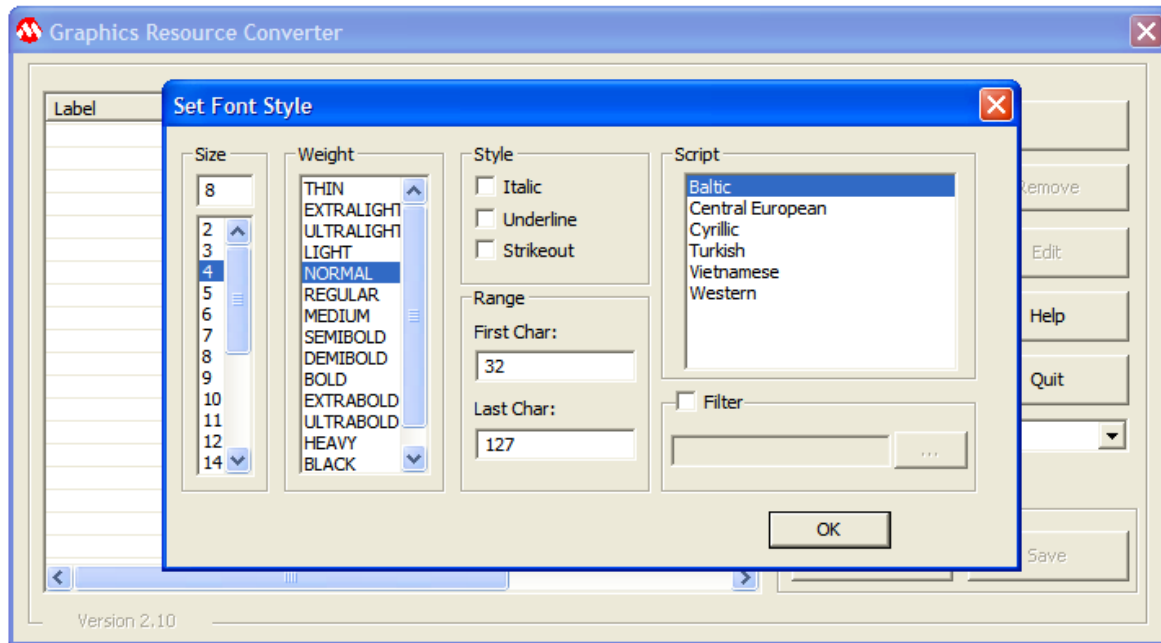
Convert a font ...



- 1) Select Add Fonts
- 2) Navigate to **font folder**
- 3) **Select Font File**
 - Raster -- *.fnt
 - True Type Font -- *.ttf

Using Fonts Example

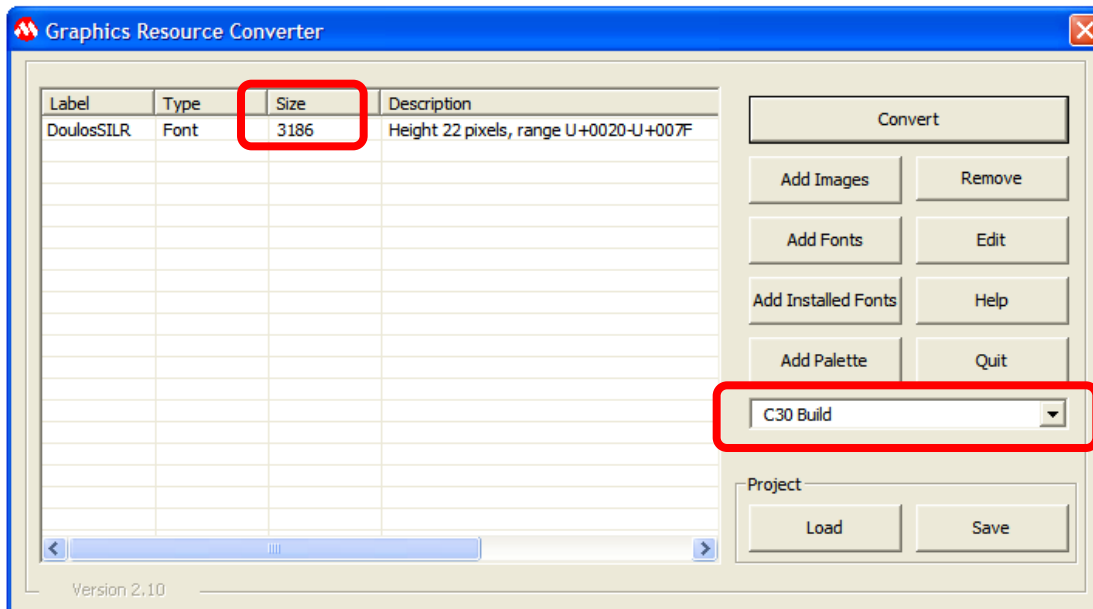
Convert a font ...



- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
 - Raster -- *.fnt or
 - True Type Font -- *.ttf
- 4) Set the Font Style

Using Fonts Example

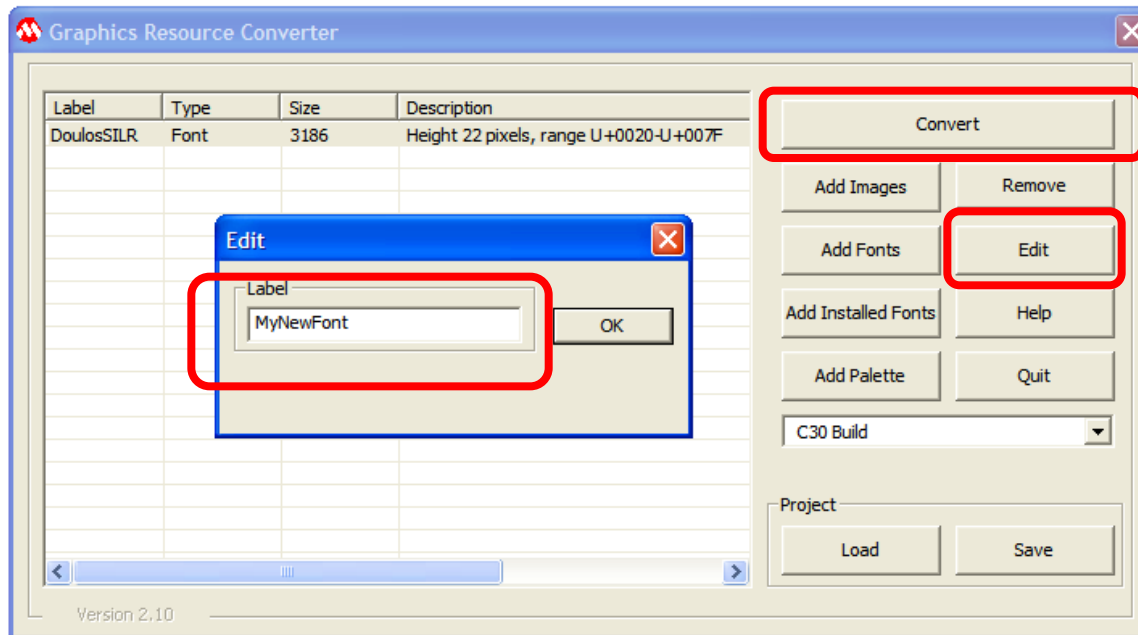
Convert a font ...



- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
- 4) Set the Font Style
- 5) **Note size:**
 - 1) **PIC24 total <32k**
 - 2) **PIC32 application dependent**
- 6) **Choose Build Option:**
 - 1) **C30 Build – PIC24**
 - 2) **C32 Build – PIC32**

Using Fonts Example

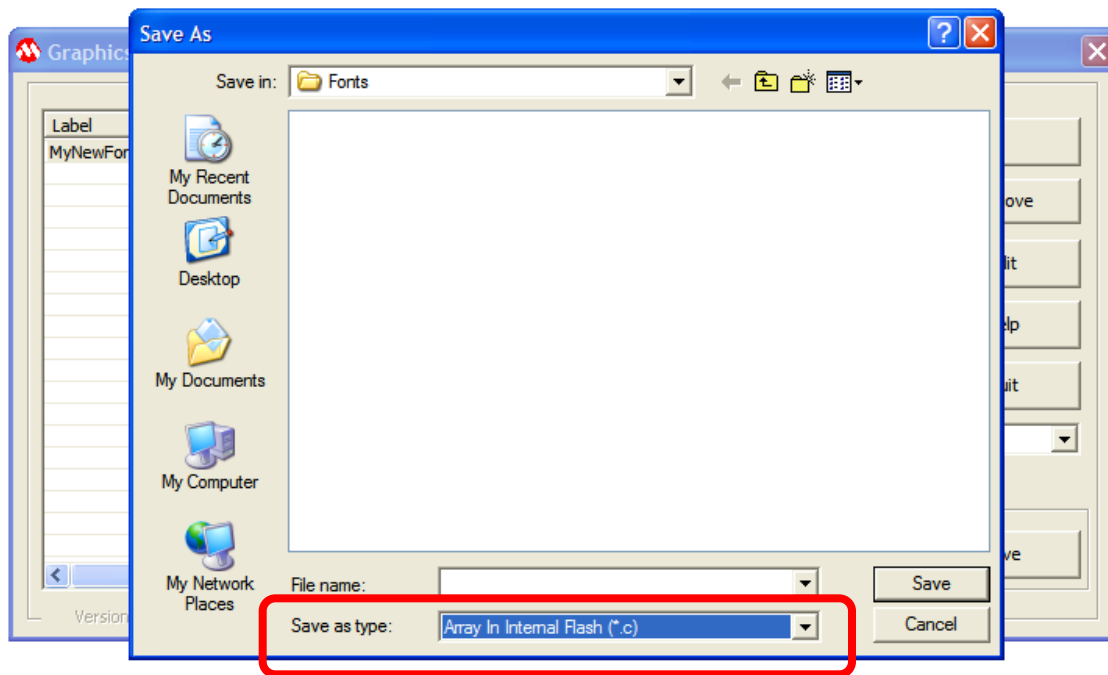
Convert a font ...



- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
- 4) Set the Font Style
- 5) Note size:
 - 1) PIC24 total <32k
 - 2) PIC32 application dependent
- 6) Choose Build Option:
 - 1) C30 Build – PIC24
 - 2) C32 Build – PIC32
- 7) **Name Font Image**
- 8) **Select Convert**

Using Fonts Example

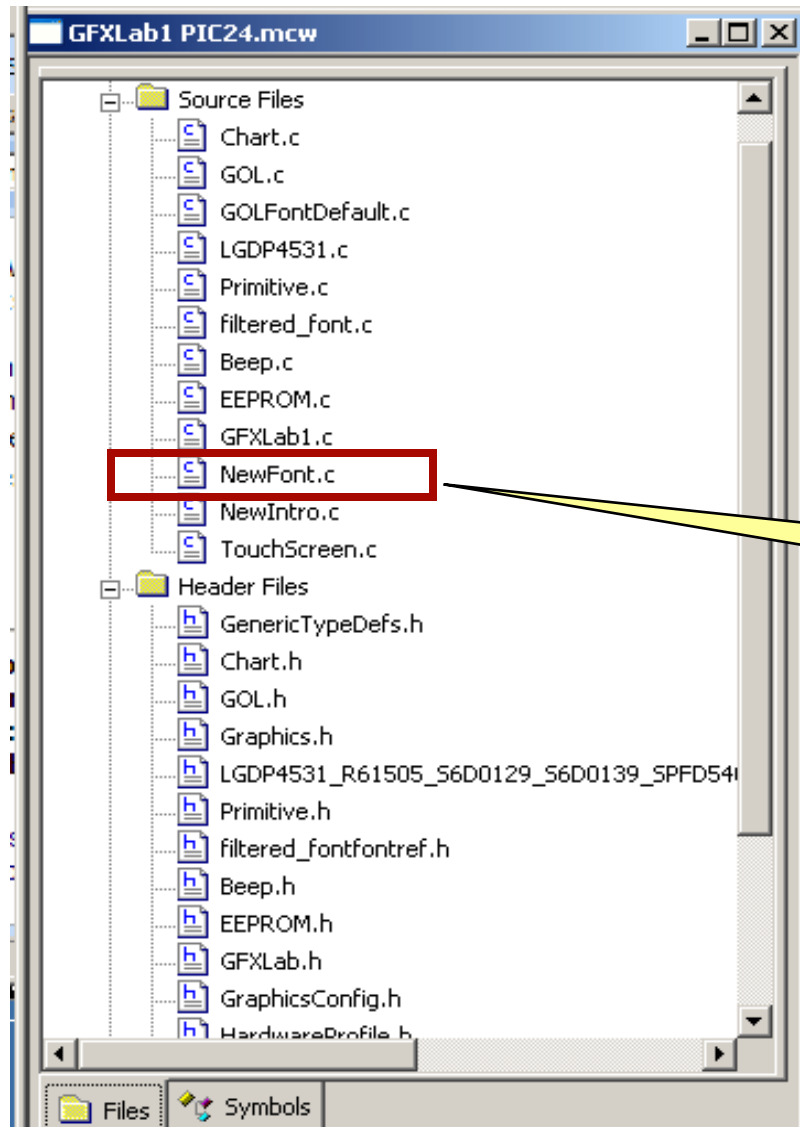
Convert a font ...



- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
- 4) Set the Font Style
- 5) Choose Build Option:
 - 1) C30 Build – PIC24
 - 2) C32 Build – PIC32
- 6) Name Font Image
- 7) Select Convert
- 8) **For internal memory, choose Array In Internal Flash (*.c) type**



Using Fonts Internal Memory

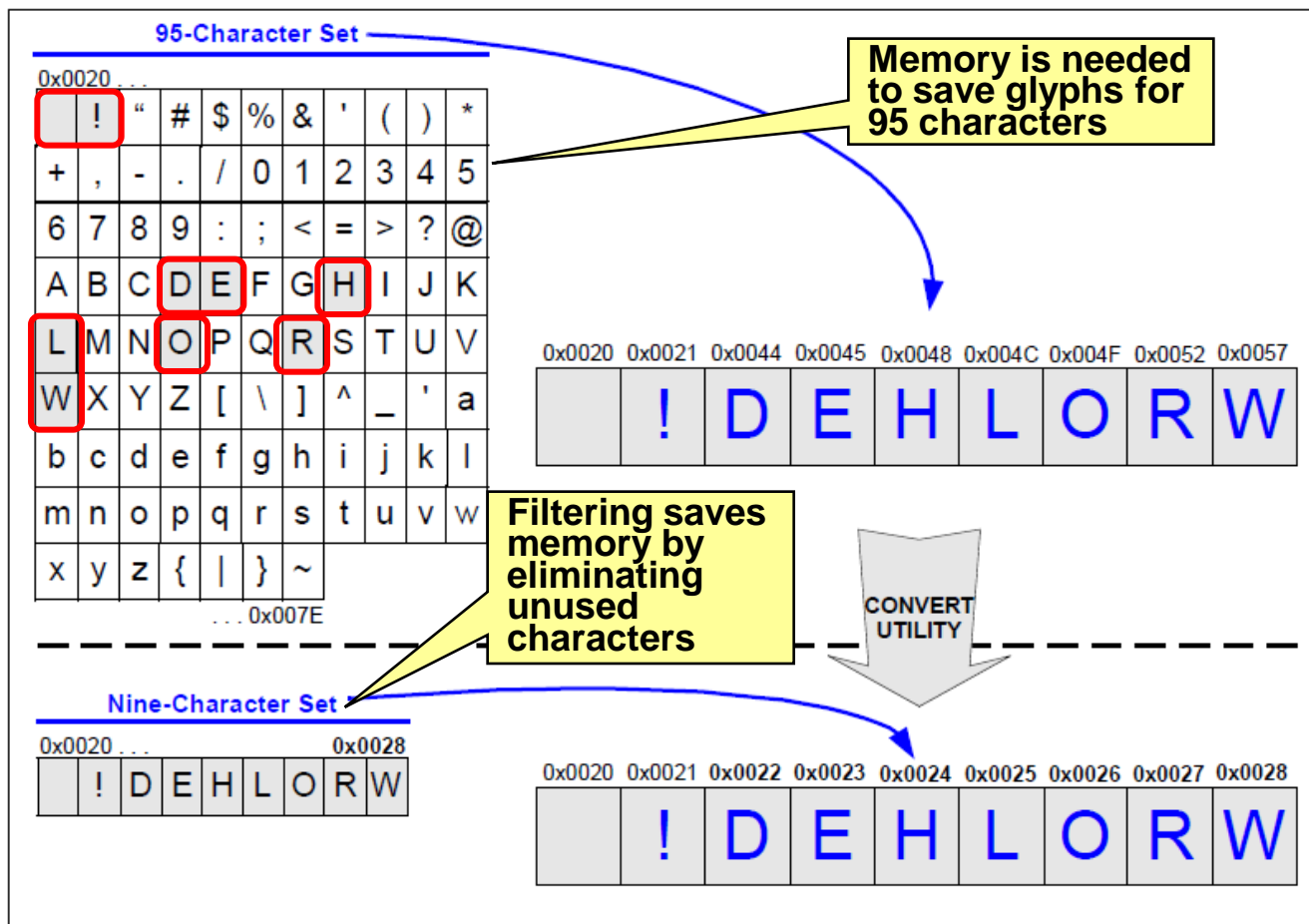


**Add generated
files to project**

Font Filtering

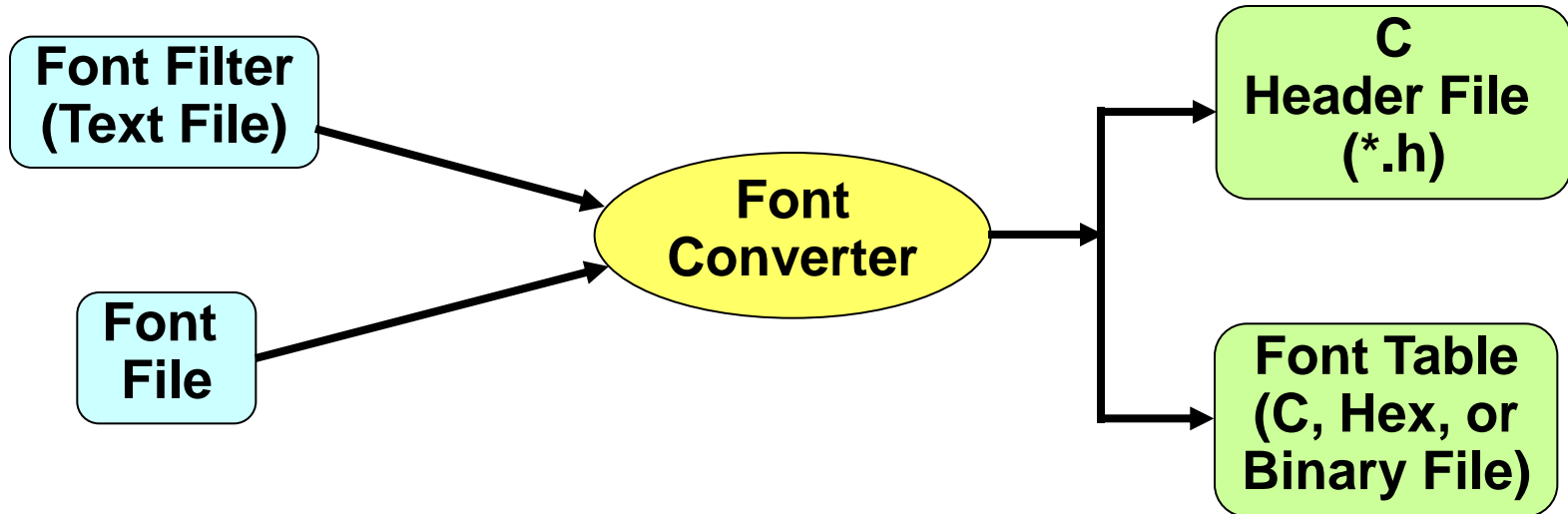
Application will use the string “Hello World!”

Since we only need 9 characters, how can we save flash memory space?



Font Filters

- Used to reduce memory for font image
 - Glyphs are remapped to new character ID
 - Requires filter text file (font filter file)
 - Must include generated reference files in project





Font Filter

Example:

■ Font Filter File (HelloWorld.txt):

```
HelloWorldLabel:Hello World! // My string
```

■ Output reference file:

```
#include "Graphics\Graphics.h"
```

```
// Automatically generated reference arrays for the  
HelloWorld.txt file.
```

```
const XCHAR HelloWorldLabel[] = {  
    0x0022, 0x0025, 0x0026, 0x0026, 0x0027,  
    0x0020, 0x0023, 0x0027, 0x0028, 0x0026,  
    0x0024, 0x0021, 0x0000}; // My string
```

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
- Recreate image to edit strings
 - Can't reference characters directly
- Refer to App Note 1182
(Fonts in the Microchip Graphics Library)



Using Fonts

- **Main.c** – Declare font images

- Structure name match label in conversion tool

```
///////////////////////////////// FONTS AND BITMAPS ///////////////////////////////////
// This font is located in internal flash
extern const FONT_FLASH MyNewFont;

// This font must be stored in external flash memory installed on
// Graphics PICTail Plus board
extern FONT_EXTERNAL externalfont;
```

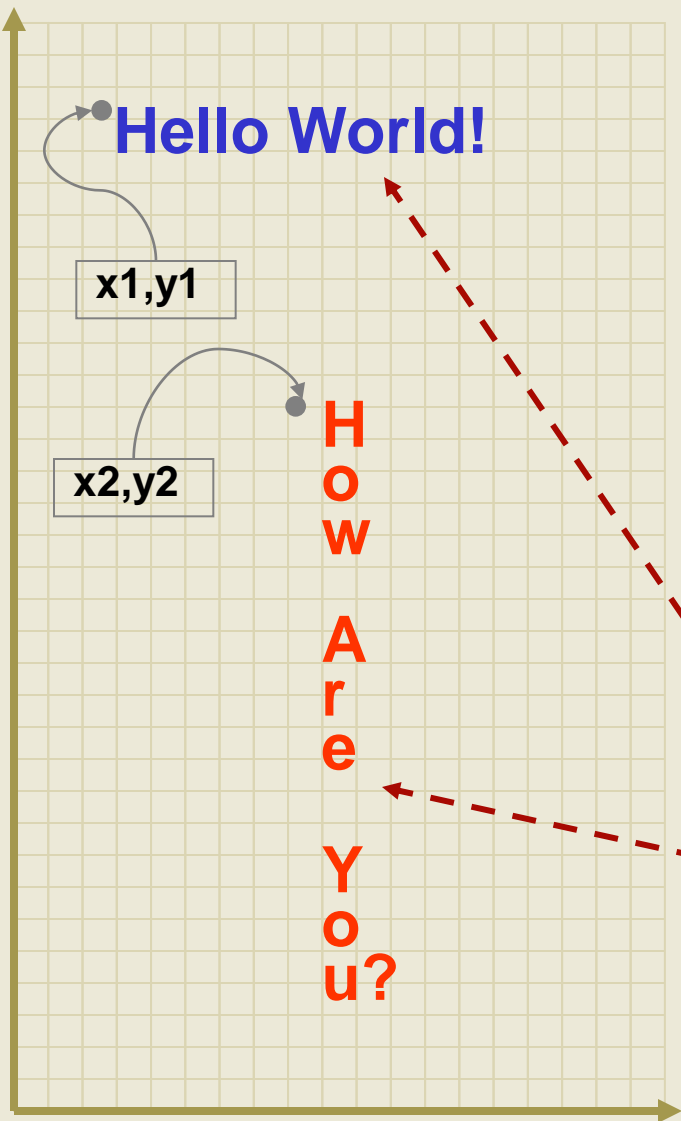
OR

- **NewFont.c** – edit file generated by conversion tool

```
extern const char L11298[] __attribute__((aligned(2)));
//NAME CAN BE CHANGED HERE.
const FONT_FLASH MyNewFont = {0,L11298};
const char L11298[] __attribute__((aligned(2)))
={0x00,0x00,0x20,0x00,0x7F,0x00,0x00,0x23,0x00,0x06,0x88,0x01,0x0
0,0x08,0xAB,0x01,0x00,0x0C,0xCE,0x01,0x00,0x0E,0x14,0x02,0x00,...
```



Using Fonts



```
int main(void)
{
    extern const FONT_FLASH myFont;
    XCHAR myString1[]="Hello World!";
    XCHAR myString2[]="How are you?";
    ...
    SetFont ( (void*) &myFont );
    SetColor (BLUE) ;
    MoveTo (x1, y1) ; //move cursor
    OutText (myString1) ;
    SetColor (BRIGHTRED) ;
    SetFontOrientation (ORIENT_VER) ;
    OutTextXY (x2, y2, myString2) ;
    ...
}
```



MICROCHIP

Regional Training Centers

Microchip Graphics
Primitive Layer – Images / Icons



Images

Definition:

A Bitmap is a dot by dot description of a graphical image in memory. The value of each dot is stored in one or more bits of data as defined by the color depth, or bits per pixel (bpp).



1 bpp



4 bpp



8 bpp



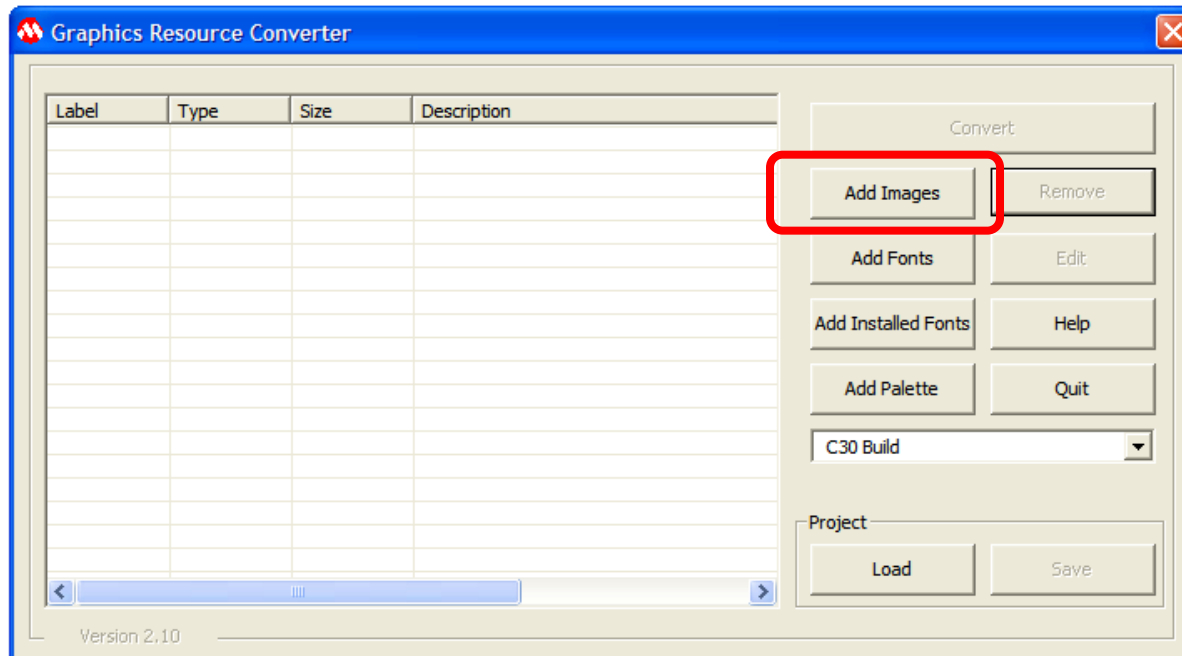
16 bpp

Resolution and Image Buffer Size

	X Resolution	Y Resolution	Pixels	Color Depth (bpp)	Colors	Bytes Required
Monochrome						
1/16 VGA	160	120	19,200	1	2	2,400
1/8 VGA	240	160	38,400	1	2	4,800
QVGA	320	240	76,800	1	2	9,600
Color STN						
1/16 VGA	160	120	19,200	8	256	19,200
1/8 VGA	240	160	38,400	8	256	38,400
QVGA	320	240	76,800	8	256	76,800
Color TFT						
1/16 VGA	160	120	19,200	16	65,536	38,400
1/8 VGA	240	160	38,400	16	65,536	76,800
QVGA	320	240	76,800	16	65,536	153,600
1/16 VGA	160	120	19,200	18	262,144	43,200
1/8 VGA	240	160	38,400	18	262,144	86,400
QVGA	320	240	76,800	18	262,144	172,800

Converting Images

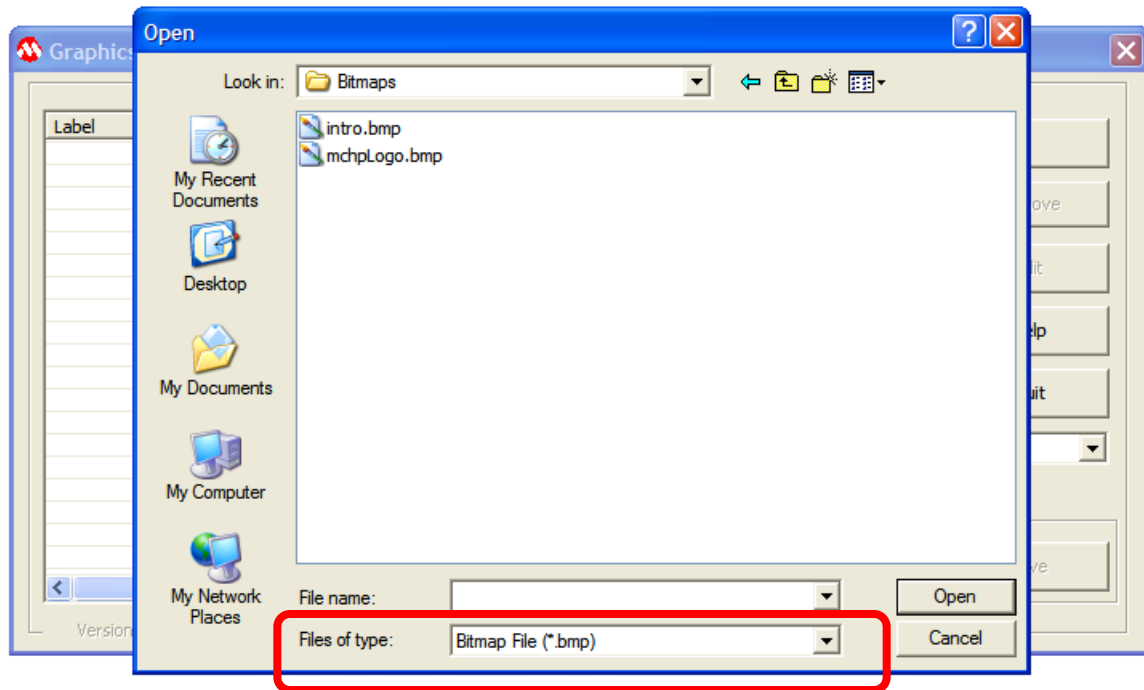
Convert bitmap images...



1) **Select Add Images**

Converting Images

Convert bitmap images...



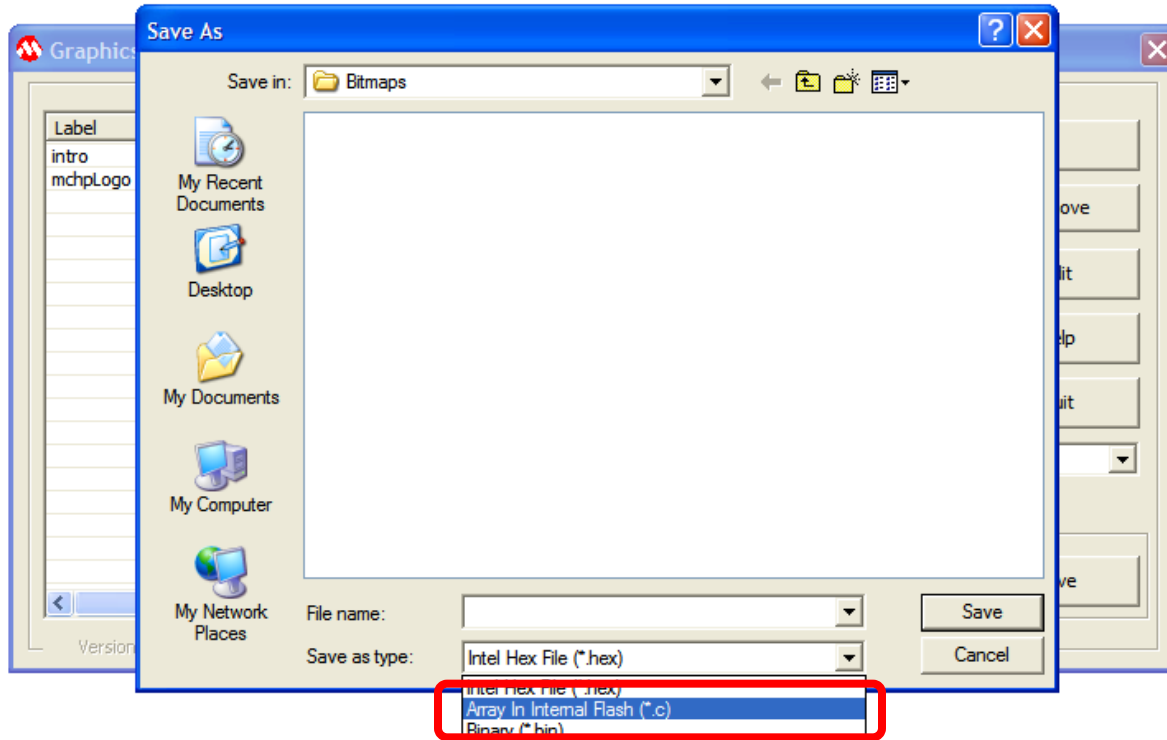
1) **Select Add Images**

NOTE: For the purposes of this class we will use only .bmp formats.



Converting Images

■ Convert bitmap images...



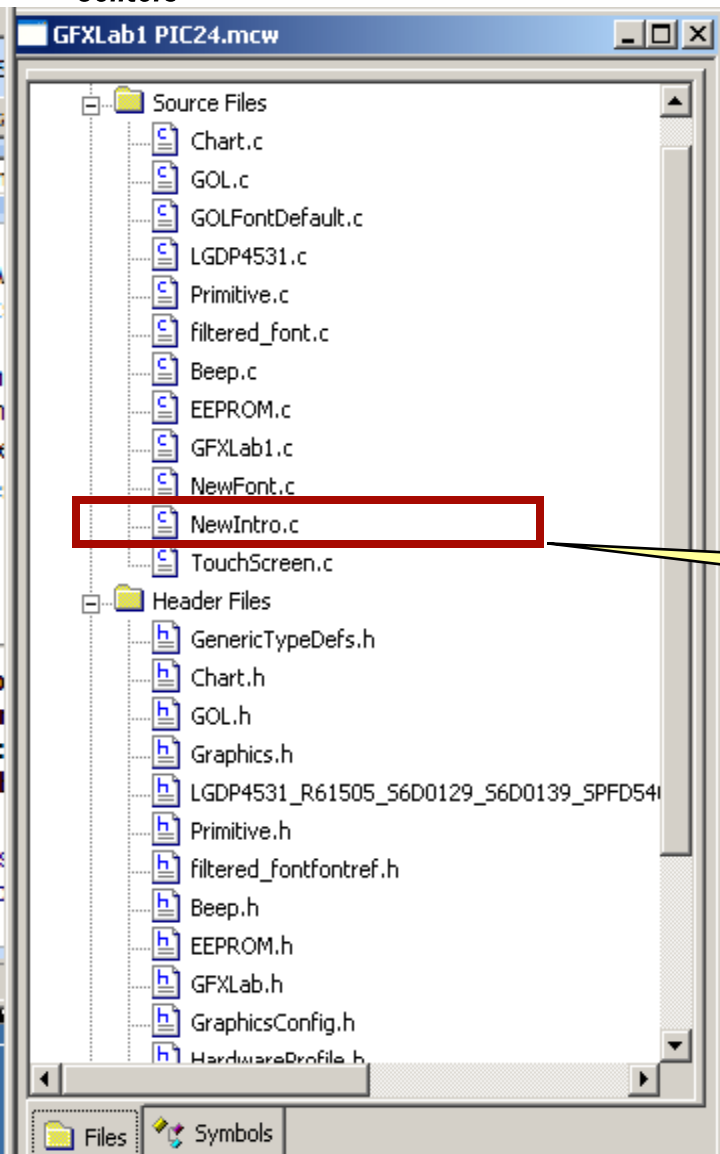
- 1) Select Add Images
- 2) **Convert the file**
- 3) **For external memory, save to .hex**
- 4) **For internal memory, save to .c**



MICROCHIP

Regional Training
Centers

Using Images Internal Memory



**Add generated
files to project**



Using Images

■ **Main.c** – Declare images

■ Structure name match label in conversion tool

```
///////////////////////////////// FONTS AND BITMAPS ///////////////////////////////////
// This font is located in internal flash
extern const BITMAP_FLASH mchpLogo;

// This font must be stored in external flash memory installed on
// Graphics PICTail Plus board
extern BITMAP_EXTERNAL external_bitmap;
```

OR

■ **NewIntro.c** – Edit file generated by conversion tool

```
extern const char L11298[] __attribute__((aligned(2)));
//NAME CAN BE CHANGED HERE.
const struct{short mem; const char* ptr;} mchpLogo =
{0,L11298};
const char L11298[] __attribute__((aligned(2)))
={0x00,0x00,0x20,0x00,0x7F,0x00,0x00,0x23,0x00,0x06,0x88,0x01,0x0
0,0x08,0xAB,0x01,0x00,0x0C,0xCE,0x01,0x00,0x0E,0x14,0x02,0x00,...
```



Using Images



```
int main(void)
{
extern const BITMAP_FLASH image1;
BYTE stretch = IMAGE_NORMAL;
...
X = GetMaxX() - GetImageWidth((void*)&image1);
Y = GetMaxY() - GetImageHeight((void*)&image1);

//put bitmap in the center
PutImage((X >> 1), (Y >> 1), &image1, stretch);
...
}
```



stretch options are:
IMAGE_NORMAL
IMAGE_X2



TIP!

■ Keep x,y coordinates relative using:

■ `GetMaxX()` , `GetMaxY()`

- Return max X or max Y coordinate

- Based on resolution set in
`GraphicsConfig.h`

■ `GetTextWidth(string, &font)` `GetTextHeight(string, &font)`

- Return size of string

- *string* = Desired string

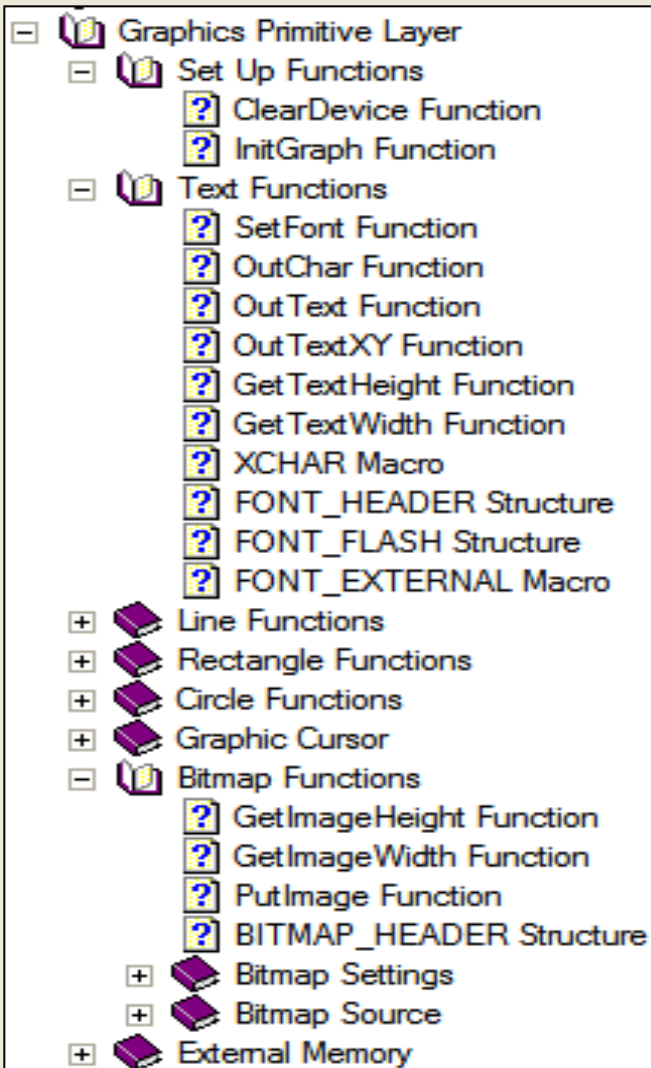
- *&font* = Pointer to font image

■ `GetImageWidth(&bitmap)` `GetImageHeight(&bitmap)`

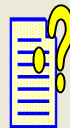
- *&bitmap* = Pointer to bitmap image



Primitive Layer Help



The Primitive Layer section of the help file describes the various APIs to help you with drawing primitives, using fonts, and using images.



Graphics Library Help



MICROCHIP

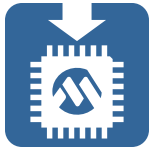
Regional Training Centers

Lab Exercise 1

Create a Splash Screen

Lab Exercise 1

Create a Splash Screen



Purpose

The purpose of this lab is to create and display a splash screen for our application using fonts, bitmaps, and primitive shapes. We will use the Bitmap and Font conversion tool to create our font and bitmap images.



Procedure

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

Objectives:

- **Convert a bitmap and font table using conversion tool**
- **Use primitive functions to:**
 - **Display an image**
 - **Display a text string**
 - **Draw shapes**

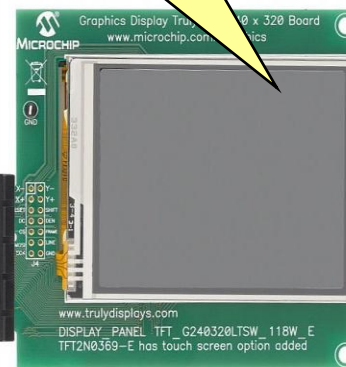
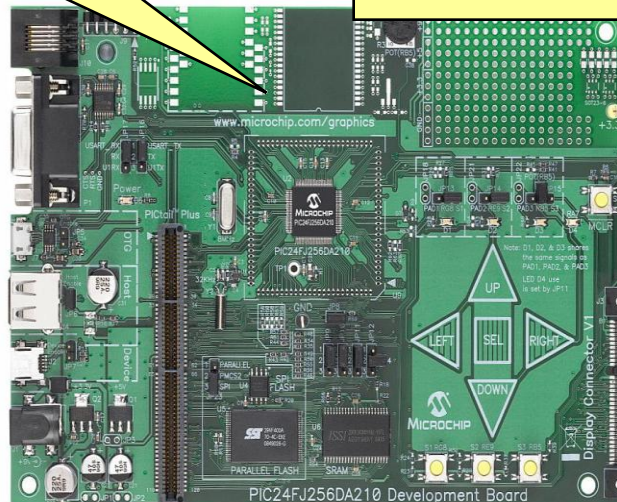
Low Cost Development Tools

**PIC24FJ256DA210
Development Board**

**Graphics Display Truly
3.2 240x320 Board**



OR



- **PIC24FJ256DA210 Development Board (DM240312)**
 - Retail Price: \$89.99
- **Graphics Display Board (AC164127-4)**
 - Truly 3.2" Display
 - 320x240 resolution
 - Retail Price: \$99.99
- **MPLAB ICD 3 In-Circuit Debugger (DV164035)**
 - Retail Price: \$189.99
- **REAL ICE In-Circuit Emulator (DV244005) – This is used in the lab today.**
 - Retail Price: \$499.98



MICROCHIP

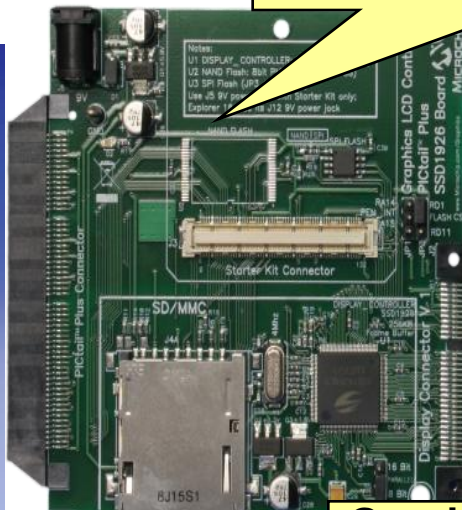
Regional Training
Centers

Low Cost Development Tools

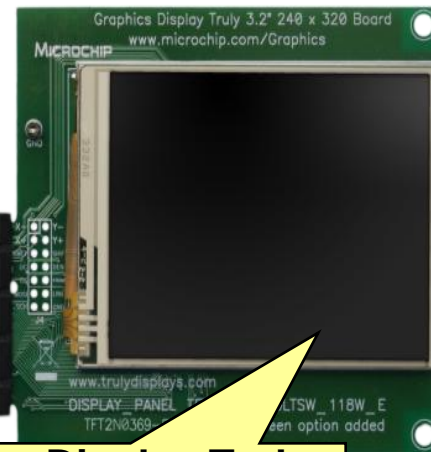
**Explorer 16
Development Board**



**Graphics LCD Controller
PICtail Plus SSD1926 Board**



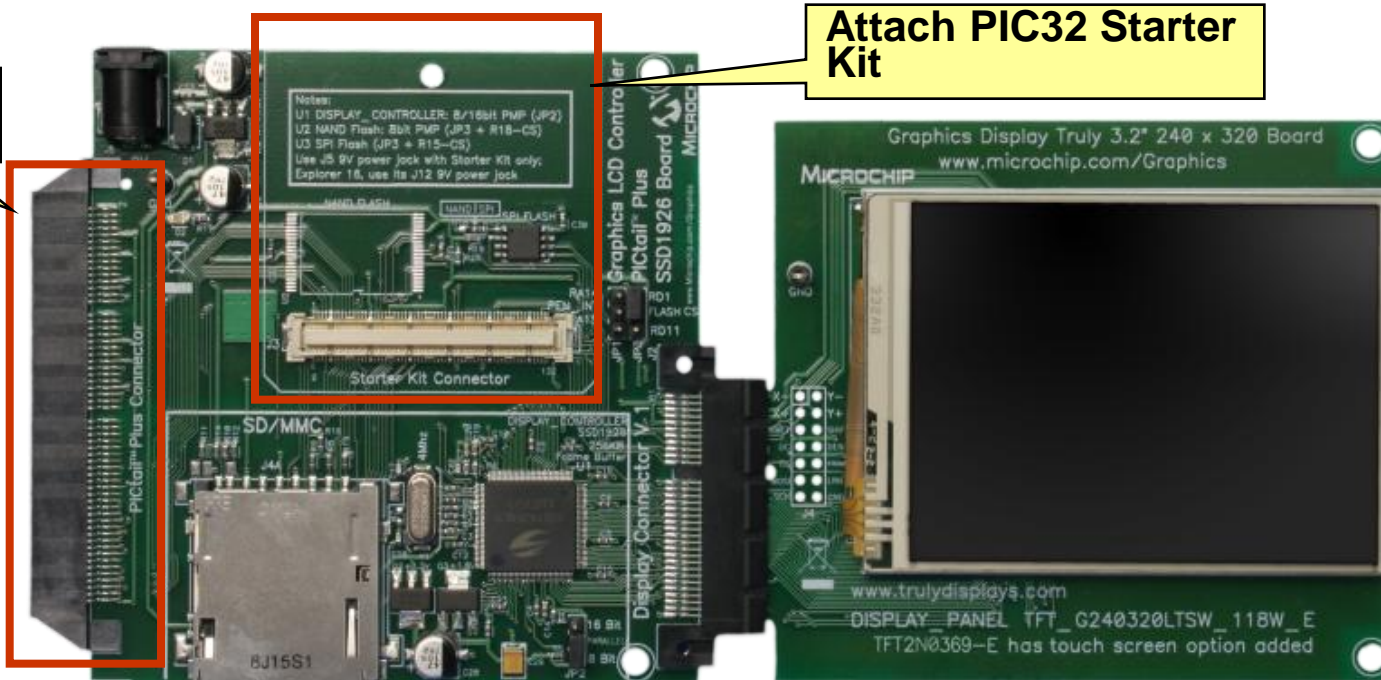
**Graphics Display Truly
3.2 240x320 Board**



- **Explorer 16 Development Board**
 - Retail Price: \$129.99
- **Graphics PICtail Plus Board with 3.2" Display Kit (AC164127-3)**
 - Retail Price: \$154.99

Low Cost Development Tools

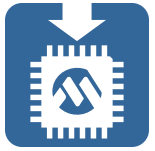
Attach
Explorer 16



- Connect to Explorer 16 board (16- or 32-bit PIMs)
- OR attach PIC32 Starter kit w/ external power
- Graphics Controller Board uses SSD1926
- Separate Display Board uses Truly 3.2" Display
 - 4-wire resistive touch overlay
 - More display boards coming

Lab Exercise 1

Create a Splash Screen



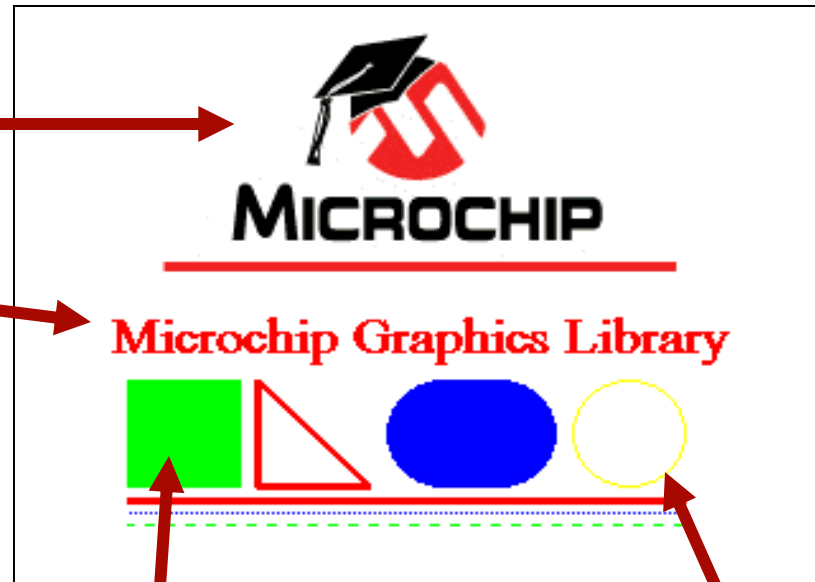
Results

mchpLogo.bmp

Text using
font image

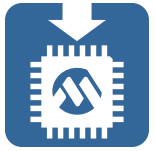
Bar primitive

Circle primitive



Lab Exercise 1

Create a Splash Screen



GFXLab1.c – Display the image (One possible solution) ...

```
void StartScreen(void) {  
    SetColor(WHITE);  
    ClearDevice();
```

Calculate X coordinate



```
    x_image = (GetMaxX() - GetImageWidth((void*) &mchpLogo)) >> 1;
```

```
    PutImage(x_image, 10,
```

X, Y coordinates for left, top image

```
    (void*) &mchpLogo,
```

← Pointer to image

```
    IMAGE_NORMAL);
```

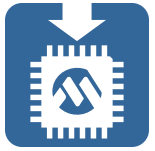
↑ Draw Image

```
... //(Continued on next slide)
```

↙ Image stretch factor

Lab Exercise 1

Create a Splash Screen



GFXLab1.c – Display the text string (One possible solution)...

...

```
XCHAR text[] = "Microchip Graphics Library";  
SetColor(BRIGHTRED);
```

Set active font

```
SetFont((void*) &MyNewFont);
```

Calculate X coordinate

```
x_text = GetMaxX() - GetTextWidth((XCHAR*) text, (void*) &MyNewFont) >> 1;
```

```
OutTextXY(x_text, 120,
```

X,Y coordinates of string start

```
(XCHAR*) text);
```

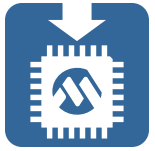
Pointer to text string

Display Text

```
... // Continued on next slide ...
```

Lab Exercise 1

Create a Splash Screen



GFXLab1.c – Draw bar and circle (One possible solution)...

...

```
SetColor(BRIGHTGREEN);
```

```
Bar(44, 155, 88, 199);
```

```
SetLineThickness(NORMAL_LINE);
```

```
SetColor(BRIGHTYELLOW);
```

```
Circle(241, 177, 22);
```

```
... //Drawing other primitives
```

```
} // End StartScreen
```

Draw green bar

Draw yellow circle



MICROCHIP

Regional Training Centers

**Microchip Graphics
Creating Widgets**



MICROCHIP

Regional Training
Centers

Library Widgets

Picture



Buttons

Exit

Scale

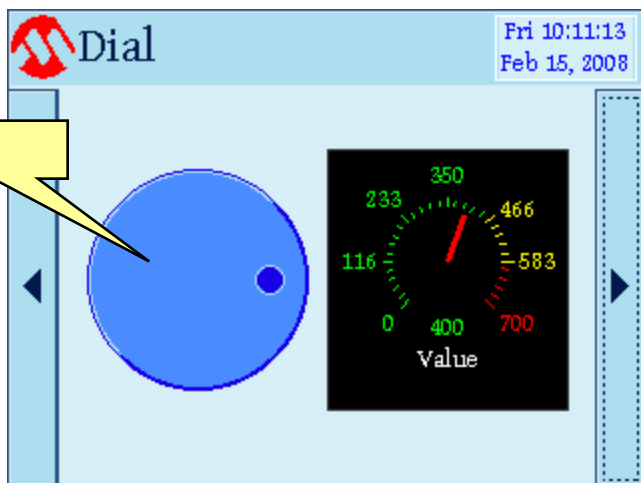


Meter

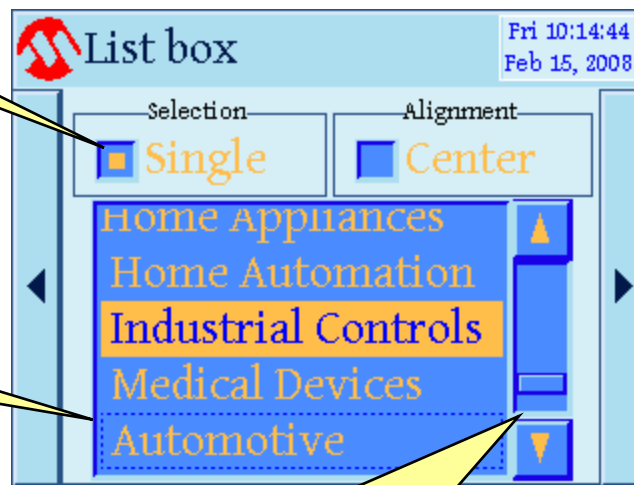


Checkbox

Dial

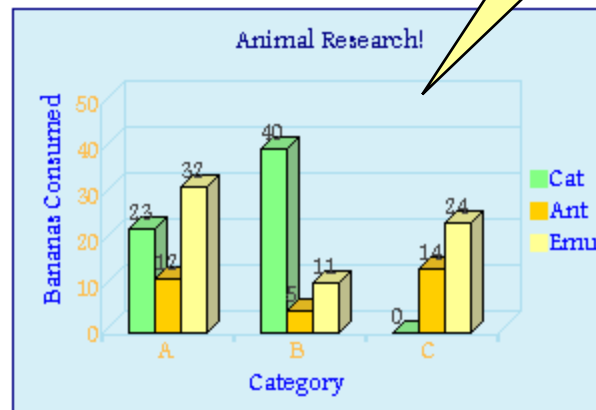


Listbox



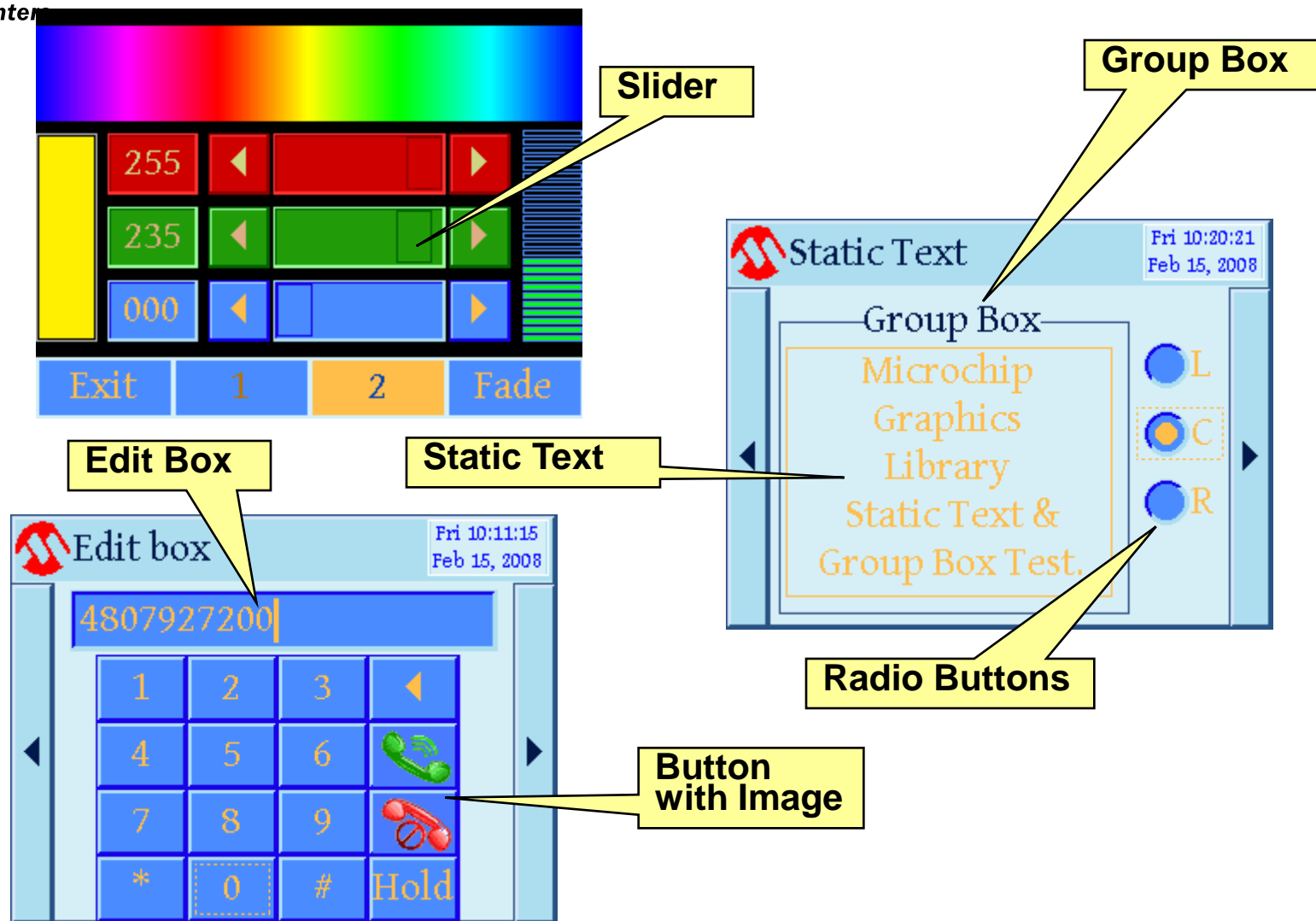
Scrollbar (using
Buttons and Slider)

Chart





Library Widgets



Library Widgets

- New 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	#	



Creating Objects

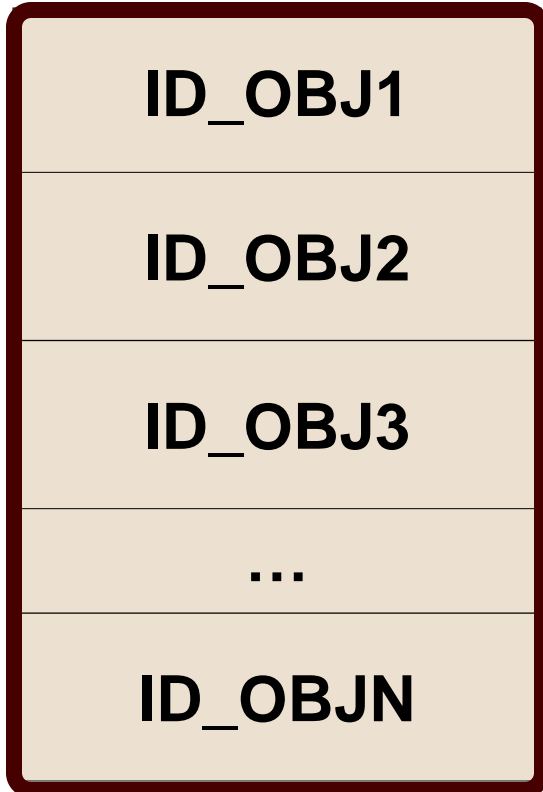
Definition:

The `ObjCreate(,,)` is the function used by the library to create widgets with specified parameters. This function automatically populates the widget's structure, places the widget into a global linked list and returns a pointer to the widget created.

- Each widget has its own create function
- **OBJ** = Widget abbreviation
 - Defined in library help file
 - Example: `BtnCreate(,,)` creates a button
 - Example: `PbCreate(,,)` creates a progress bar
- May create multiple widget instances

Creating Widgets

Linked List



■ `ObjCreate(, , ,)`

- Populates a widget structure
- Adds widget to bottom of the linked list
- Heap required
- More details on using the linked list later...



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



In Application Code ...

■ Example:

```
void CreateButtons(void)
{
...

#define          ID_BTN2          16
...
...
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
...
}
```

Creating Widgets

Common Parameters

- *ID*

- Unique identifier used to form a “Handle”

- *Location*

- Top, left, bottom, right define placement

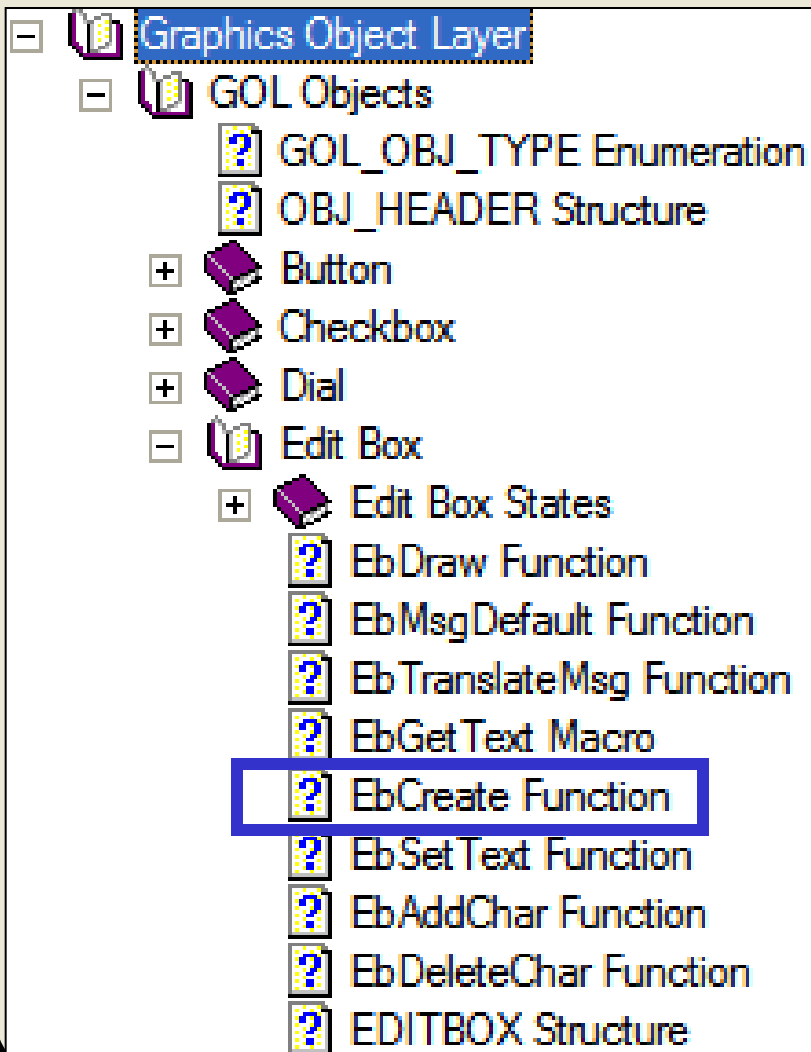
- *Statebits*

- Used to control the widget

- *Style Scheme*

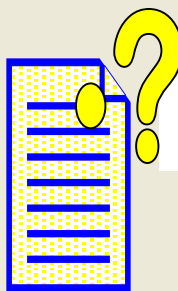
- Defines widget appearance

Widget Help



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

C:\Microchip Solutions\Microchip\Help



Graphics Library Help.htm

Required Resources – PIC24F

Module	Heap (bytes)	RAM (bytes)	ROM (bytes)
Primitives Layer	0	66	3285
GOL	20 (per style scheme)	32	2124
Button	28 (per instance)	8	1131
Check Box	22 (per instance)	2	1008
Radio Button	26 (per instance)	14	1113
Slider/Scroll Bar	32 (per instance)	20	2352
Static Text	22 (per instance)	8	792
Progress Bar	24 (per instance)	12	1074
Picture Control	21 (per instance)	9	687
Group Box	24 (per instance)	8	960
Window	24 (per instance)	2	807
List Box	28 (per instance), 12 (per item)	6	2094
Edit Box	26 (per instance)	2	1008
Meter	40 (per instance)	42	2520
Dial	30 (per instance)	8	1092

Required Resources – PIC32

Module	Heap (bytes)	RAM (bytes)	ROM (bytes)
Primitives Layer	0	53	8868
GOL	24 (per style scheme)	28	5400
Button	44 (per instance)	12	2748
Check Box	36 (per instance)	4	2320
Radio Button	44 (per instance)	20	2632
Slider/Scroll Bar	44 (per instance)	24	5720
Static Text	36 (per instance)	12	1884
Progress Bar	36 (per instance)	16	2452
Picture Control	36 (per instance)	12	1512
Group Box	36 (per instance)	12	2164
Window	40 (per instance)	4	1996
List Box	44 (per instance)	12	2580
Edit Box	40 (per instance)	4	2332
Meter	68 (per instance)	40	6788
Dial	40 (per instance)	12	3228



MICROCHIP

Regional Training Centers

**Microchip Graphics
Widget Styles**



Widget Styles

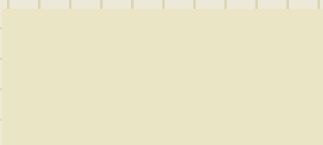
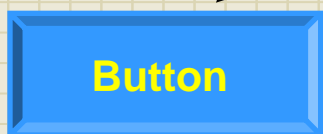
Definition:

The Style Scheme is a structure used by the library to define the appearance of a widget by assigning colors and fonts.

- Multiple schemes may be defined
- Default scheme provided in `GOL.h`
- `GOLCreateScheme()`
 - Creates the scheme structure
 - Returns a pointer to the created structure
- 10 members in the structure
 - Default scheme assigned upon widget creation
 - May change by directly assigning new values
 - All widgets use style scheme member values differently

Style Scheme Structure

Button Example



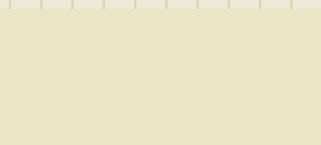
```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = GREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTTAN;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

Style Scheme Usage



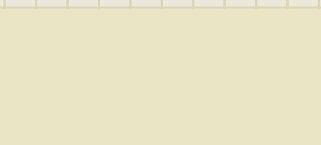
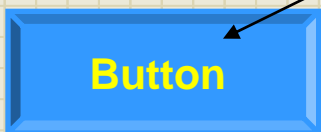
```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = GREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTTAN;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

Style Scheme Usage



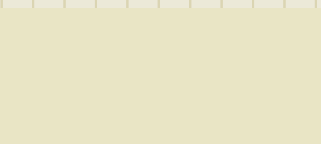
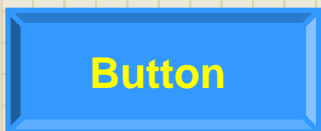
```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = GREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTTAN;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

Style Scheme Usage



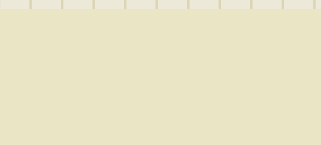
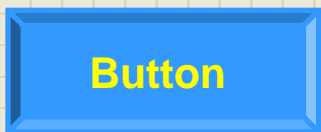
```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = GREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTGREY;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

Style Scheme Usage



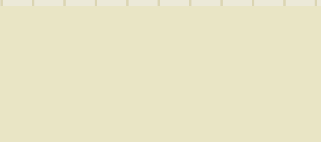
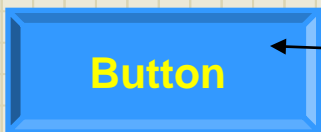
```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = DKGREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTGREY;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

Style Scheme Usage



```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = DKGREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTGREY;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

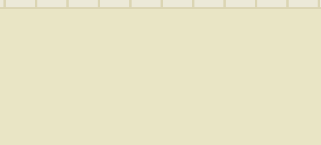
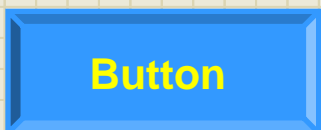
Style Scheme Usage



```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = DKGREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTGREY;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

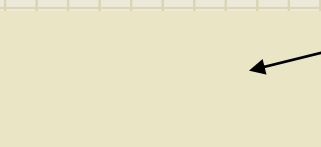



Style Scheme Usage



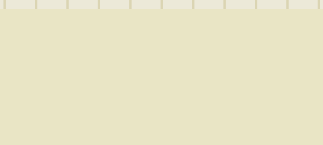
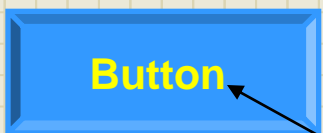
```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = DKGREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTGREY;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

Style Scheme Usage



```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = DKGREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTGREY;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```

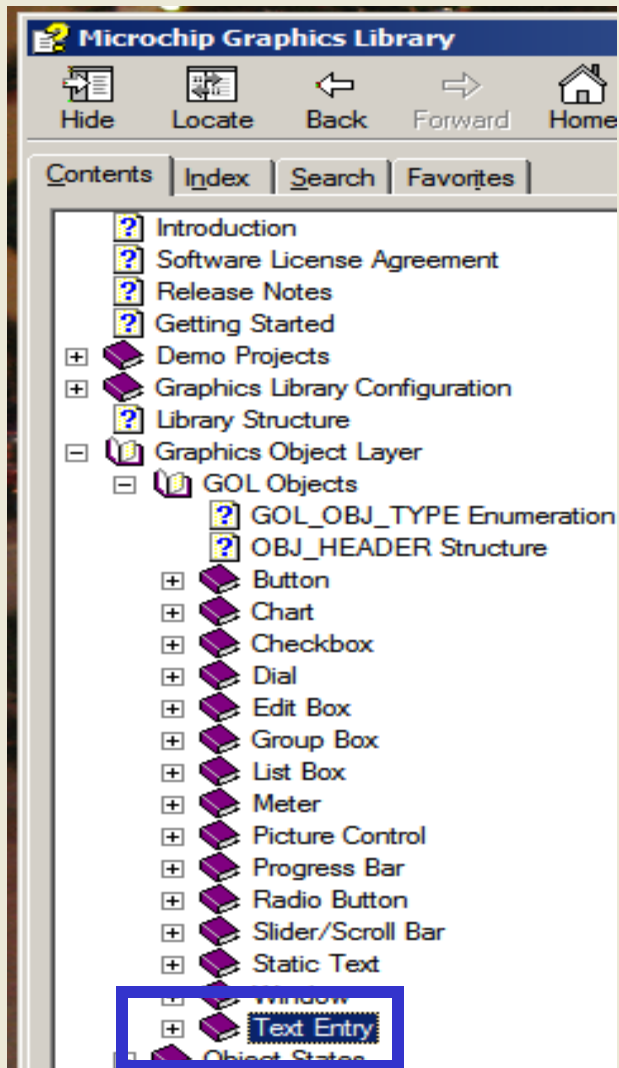
Style Scheme Usage



```
int main(void)
{
...
  GOL_SCHEME      *altScheme;
  altScheme = GOLCreateScheme();
...
  altScheme -> EmbossDkColor = DKBLUE;
  altScheme -> EmbossLtColor = BLUE;
  altScheme -> TextColor0 = WHITE;
  altScheme -> TextColor1 = YELLOW;
  altScheme -> TextColorDisabled = DKGREY;
  altScheme -> Color0 = BRIGHTBLUE;
  altScheme -> Color1 = LTBLUE;
  altScheme -> ColorDisabled = LTGREY;
  altScheme -> CommonBkColor = TAN;
  altScheme -> pFont = GOLFontDefault;
...
}
```



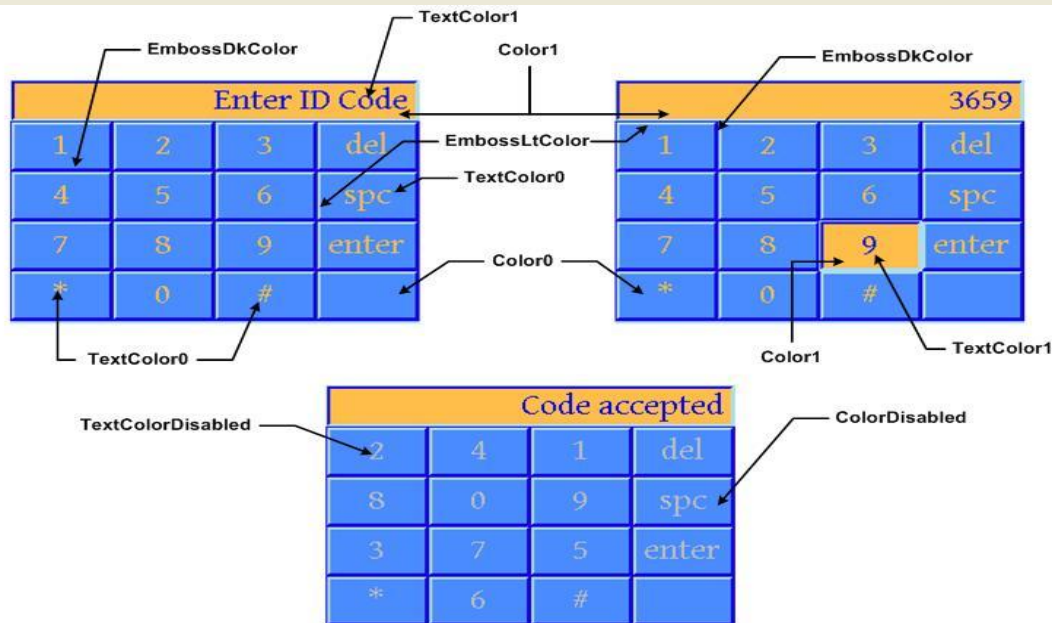
Widget Style Help



The top level of each widget gives a diagram showing how the style scheme members are applied.

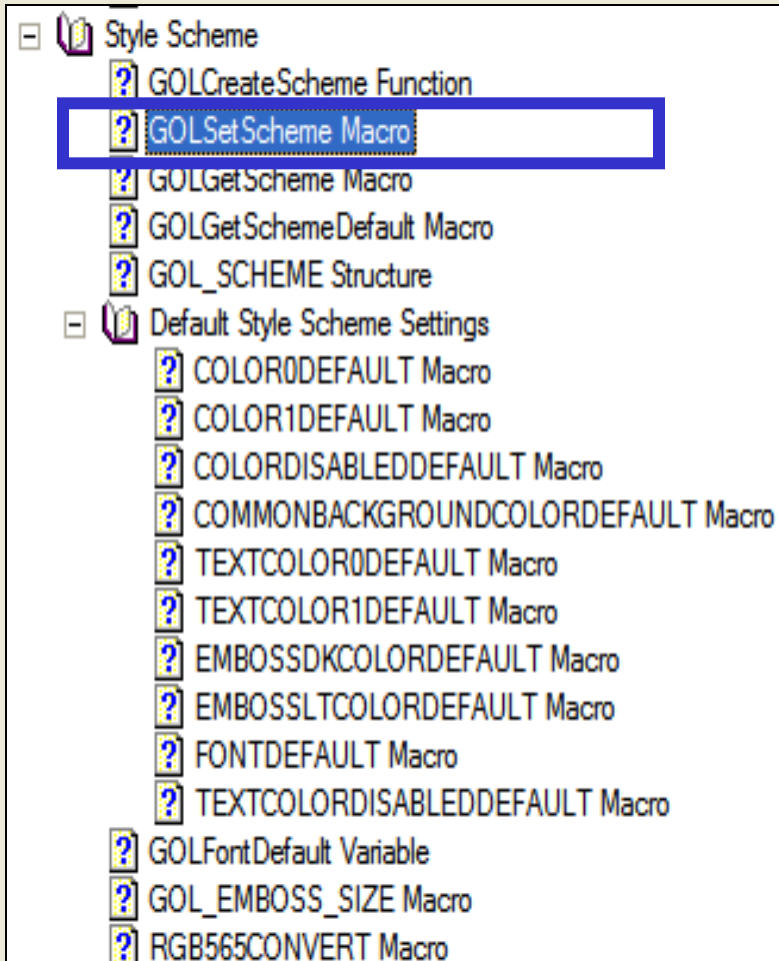


Graphics Library Help.chm





Style Scheme APIs



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



Graphics Library Help.chm



To reassign a widget's style scheme after creation use ...

```
GOLSetScheme(*pObj, *pScheme)
```



MICROCHIP

Regional Training Centers

**Microchip Graphics
Drawing Widgets**

Drawing Widgets

Linked List

ID_OBJ1 ->
state bits

ID_OBJ2 ->
state bits

ID_OBJ3 ->
state bits

...

ID_OBJN ->
state bits

- **GOLDDraw()**
 - No input parameters
 - Parses linked list
 - Checks state field
 - If a drawing bit is set, widget will be rendered
 - Returns TRUE when done



state Field Common Bits

■ Drawing state bits:

- 6 MSB of **state** member of widget structure
- Indicate object needs to be...
 - Hidden
 - Partially Redrawn
 - Fully Redrawn

■ Used by all widgets:

- **OBJ_HIDE**
 - Covers widget area with **CommonBkGnd** color
- **OBJ_DRAW**
 - Redraws entire widget
 - Auto cleared by **GOLDraw()**
- **OBJ_DRAW_FOCUS:**
 - Redraws focus only
 - Auto set by **GOLDraw()**

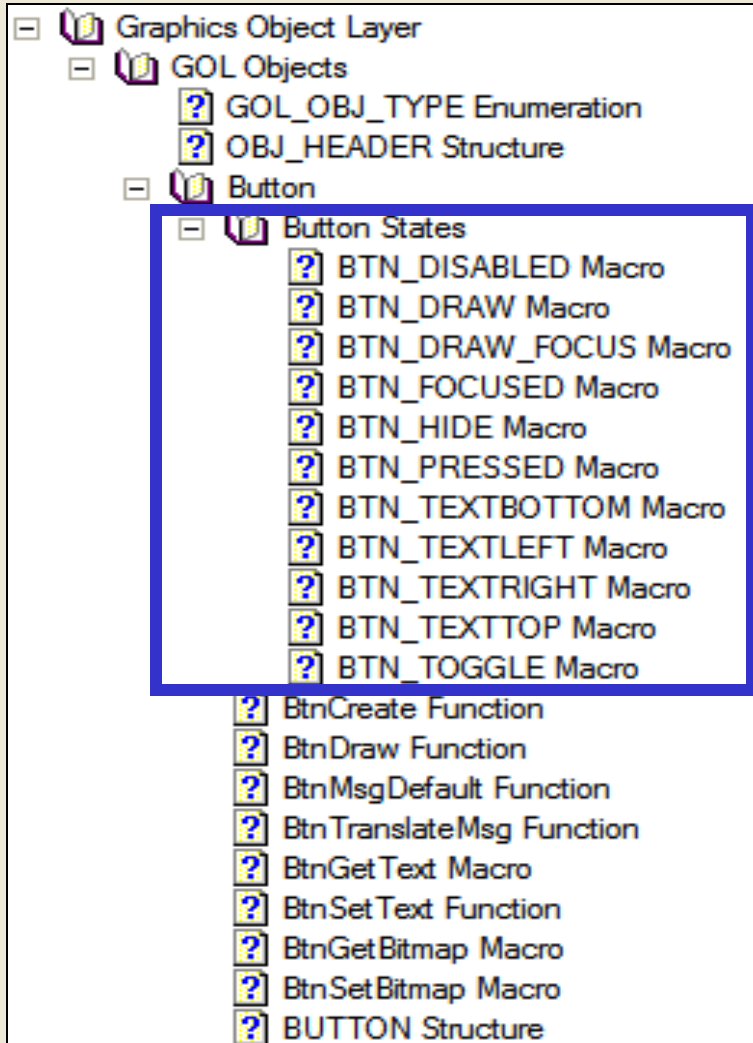


State Macros

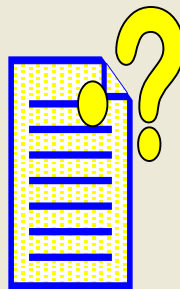
- **SetState (pObj , state)**
 - Set specified state bits
- **ClrState (pObj , state)**
 - Clear specified state bits
- **GetState (pObj , state)**
 - Get a widget's state bits
- **Where...**
 - **pObj** = Widget handle
 - **state** = Value for state field



Help with State Bits



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



Graphics Library Help



Assign a Handle

■ Could assign at creation

```
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 Handle

Assign a Handle Another Way...

- **GOLFindObject (ID)**
 - Returns ADDRESS of the object
 - Avoid mismatch warning by typecasting

```
BUTTON *pBtn  
...  
  
pBtn = (BUTTON*) GOLFindObject (ID_BTN1) ;  
  
SetState (pBtn, BTN_DRAW) ;  
pBtn = (BUTTON*) GOLFindObject (ID_BTN2) ;  
SetState (pBtn, BTN_DRAW) ;
```

Declare the Pointer

Assign Handle

Assign a Handle Another way...

- `GetObjID (*pObj)`
 - Returns ID field of the object

```
#define ID_BTN1  
BUTTON *pBtn;  
...
```

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

Assign Handle

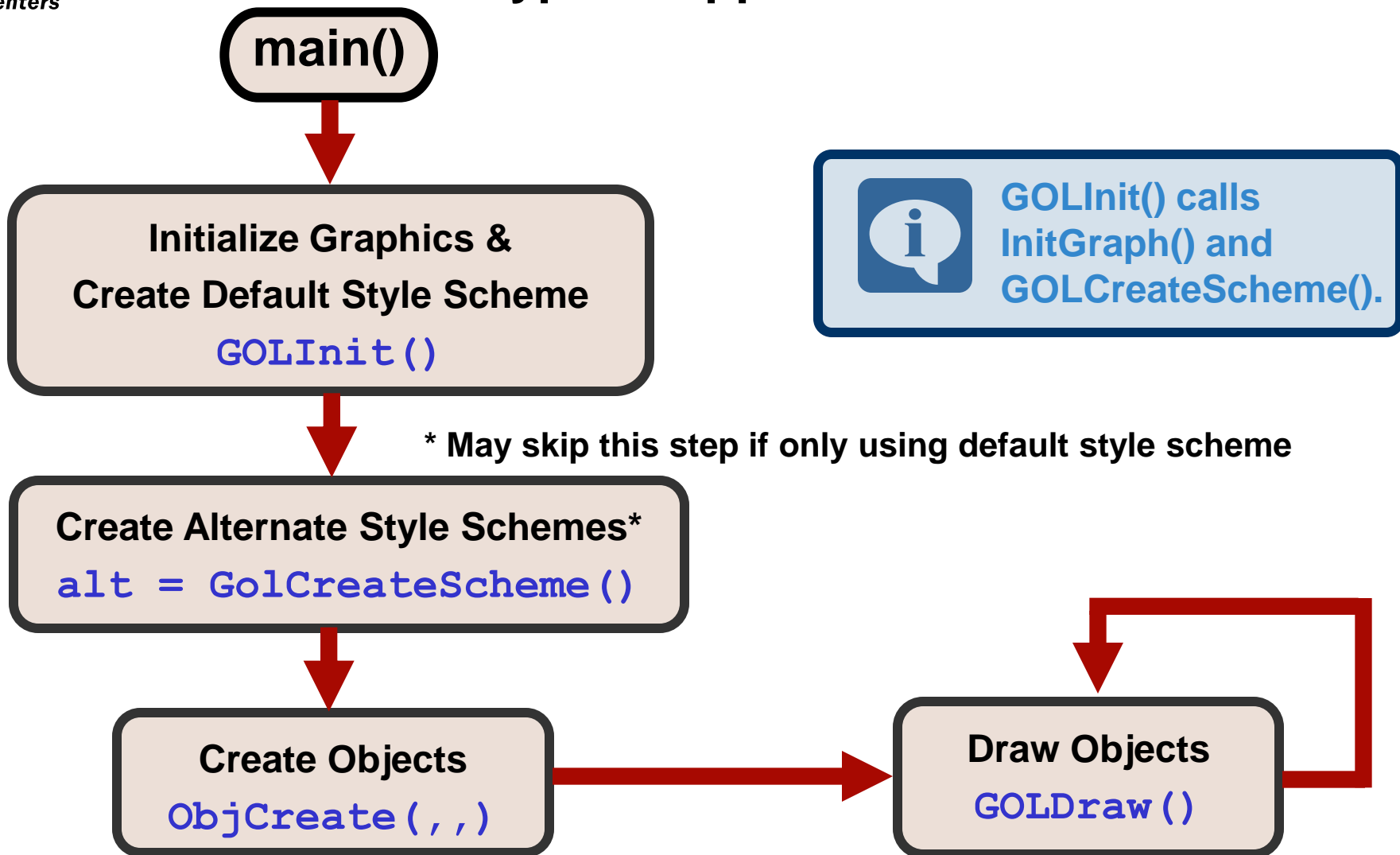


Use Handle



Drawing Example

Typical Application Flow



Drawing Example

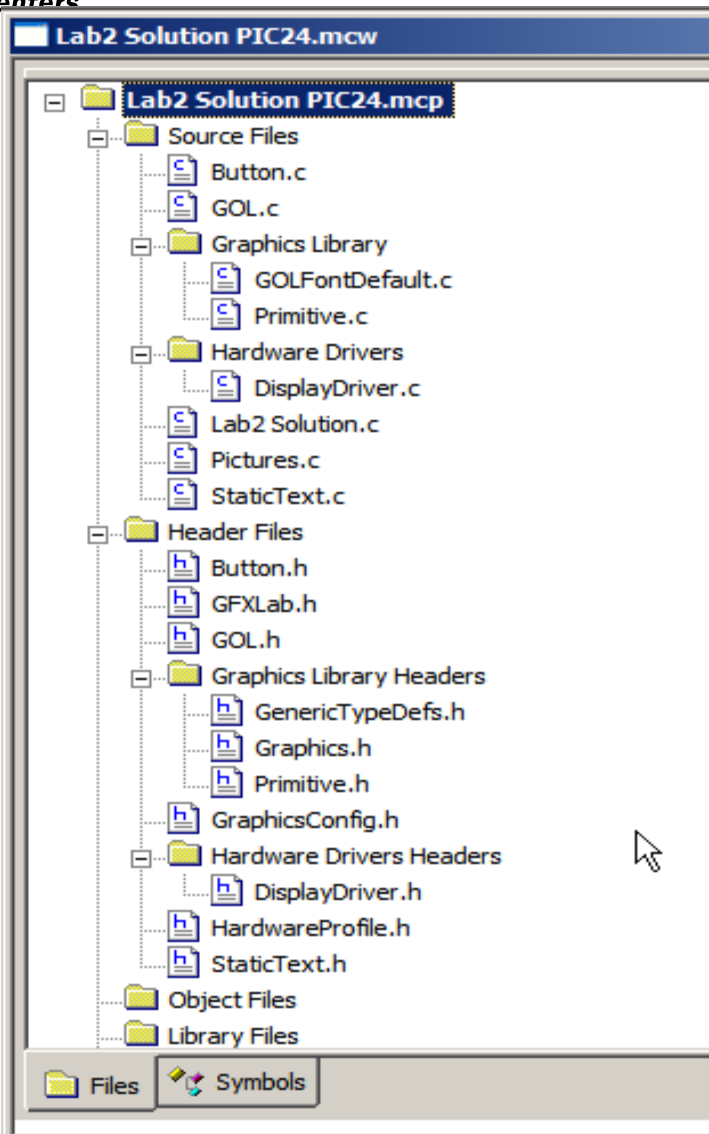


Static Text
Left Aligned
No Frame

- Draw 3 widgets
 - Rectangle button with Text
 - Circle button with Image
 - Static Text



Using Widgets



- **Add required files to the project:**

- GOL.c
- Primitive.c
- *widget.c*
- DisplayDriver.c
- GOL.h
- Graphics.h
- GenericTypeDefs.h
- Primitives.h
- *widget.h*
- DisplayDriver.h
- GraphicsConfig.h
- Plus Font and Bitmap files



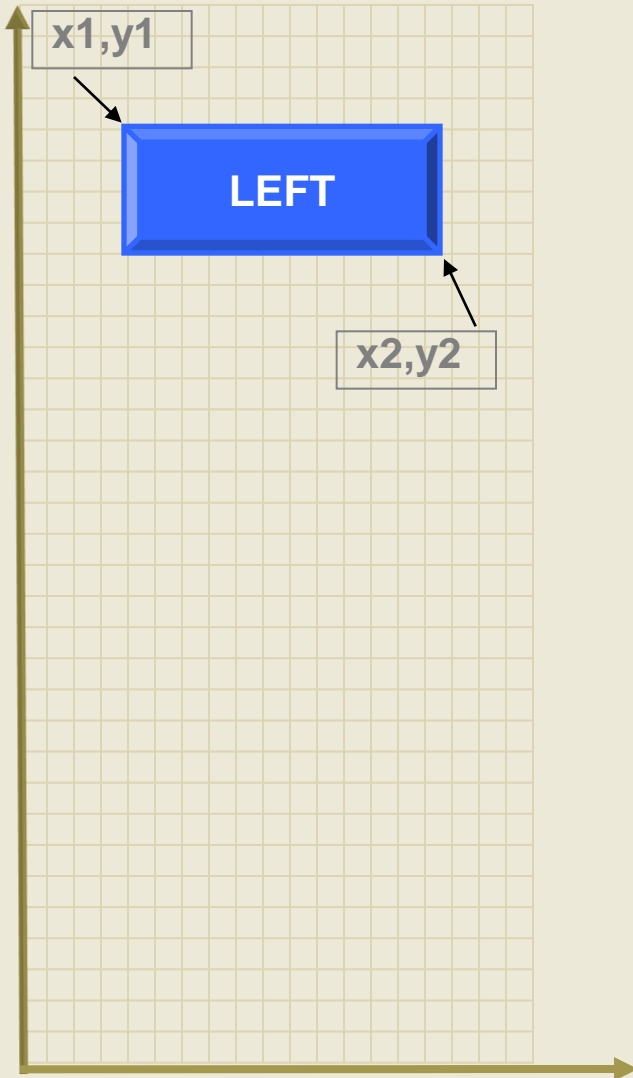
Using Widgets

● Enable Widgets in `GraphicsConfig.h`

```
#define USE_GOL
#define USE_BUTTON           // Enable Button Object.
//#define USE_WINDOW        // Enable Window Object.
//#define USE_CHECKBOX      // Enable Checkbox Object.
//#define USE_RADIOBUTTON   // Enable Radio Button Object.
//#define USE_EDITBOX       // Enable Edit Box Object.
//#define USE_LISTBOX       // Enable List Box Object.
//#define USE_SLIDER        // Enable Slider or Scroll Bar Object.
//#define USE_PROGRESSBAR   // Enable Progress Bar Object.
#define USE_STATICTEXT      // Enable Static Text Object.
//#define USE_PICTURE       // Enable Picture Object.
//#define USE_GROUPBOX      // Enable Group Box Object.
//#define USE_ROUNDDBIAL    // Enable Dial Object.
//#define USE_METER        // Enable Meter Object.
//#define USE_GRID         // Enable Grid Object.
//#define USE_CHART         // Enable Chart Object
//#define USE_CUSTOM        // Enable Custom Control Object
```

Drawing Example

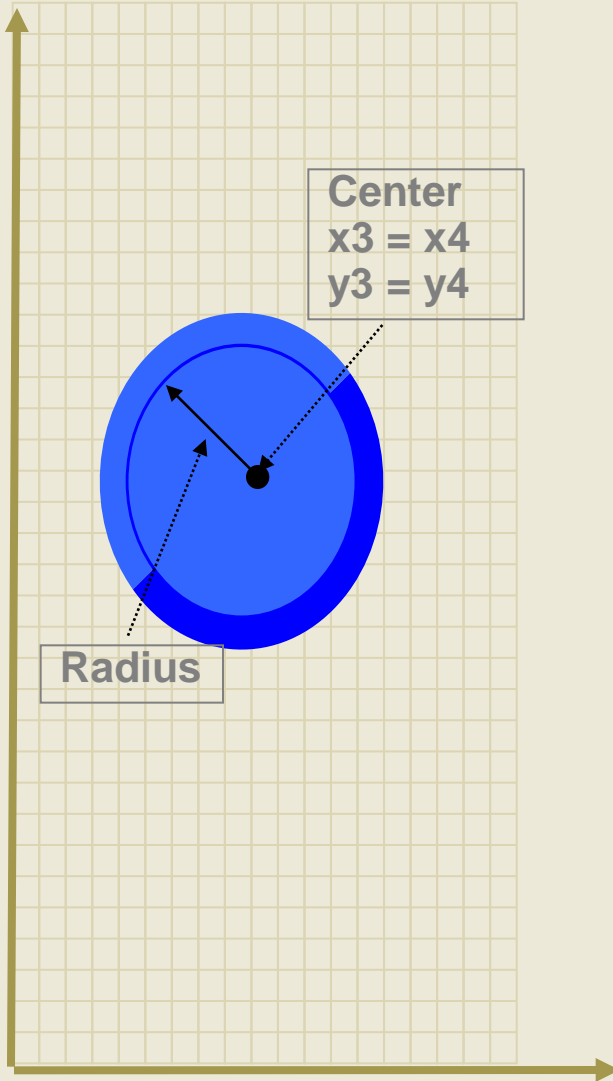
Function to Create Buttons



```
void CreateButtons(void)
{
...
#define ID_BTN1          15
#define ID_BTN2          16
...
altScheme = GOLCreateScheme();
...
BtnCreate(      ID_BTN1, // 1st Button ID
                x1, y1,  // left, top
                x2, y2,  // right, bottom
                0,       // Radius=0
                BTN_DRAW, // Display button
                NULL,     // no bitmap used
                "LEFT",   // use this text
                altScheme); // style scheme
...
}
```

Drawing Example

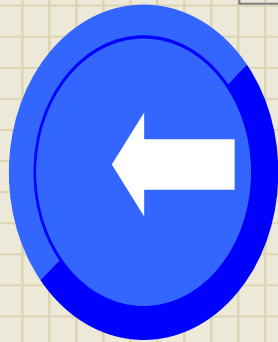
Function to Create Buttons



```
...  
BtnCreate(      ID_BTN2, // 2nd Button ID  
                x3, y3,  // left, top  
                x3, y3,  // right, bottom  
                Radius, // Create Circle  
                BTN_DRAW, // Display button  
                &arrow,  // use this bitmap  
                NULL,    // no text  
                altScheme); // style scheme  
...  
}
```

Drawing Example

Function to Create Buttons



x3 = x4
y3 = y4

...

```
BtnCreate(      ID_BTN2, // 2nd Button ID  
                x3, y3,  // Top, left  
                x4, y4,  // Bottom, Right  
                Radius, // Create Circle  
                BTN_DRAW, // Display button  
                &arrow,  // use this bitmap  
                NULL,    // no text  
                altScheme); // style scheme
```

...

}

Drawing Example

Function to Create Static Text

```
void CreateStatic(void)
{
...
#define    ST_TXT1          17
...
String1 ="Static Text \nLeft Aligned
        \nNo Frame";
StScheme = GOLCreateScheme();
StScheme -> CommonBkColor = BLACK;
StScheme -> TextColor0 = WHITE;
...
StCreate(    ST_TXT1, // Static Text Box
            x5, y5,   // left, top
            x6, y6,   // right, bottom
            ST_DRAW, // Display
            String1,  // Text String
            StScheme); // style scheme
...
```

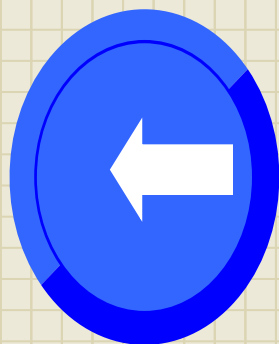
x5,y5

**Static Text
Left Aligned
No Frame**

x6,y6

Drawing Example

Application Code



Static Text
Left Aligned
No Frame

```
int main(void)
{
    ...
    //Initialize Display and
    //Set Default Style Scheme
    GOLInit();

    //Create Widgets
    CreateButtons();
    CreateStatic();
    ...
    //Application Main Loop
    while(1)
    {
        GOLDraw();
    }
    ...
}
```



MICROCHIP

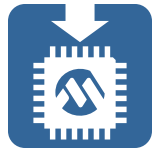
Regional Training Centers

Lab Exercise 2

Create a Menu Screen

Lab Exercise 2

Create a Menu Screen



Purpose

The purpose of this lab is to use the `ObjCreate()`, `GOLCreateScheme()` and `GOLDDraw()` to create and display a menu screen for our application. We will be using button and static text widgets. At this point, we will simply draw the screen ... the buttons will not perform any actions.



Procedure

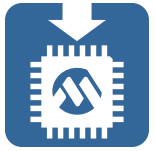
Follow the directions in the lab manual starting on page 2-1

Objectives:

- **Use the `ObjCreate(,,)` function to:**
 - **Create 3 Rounded Buttons (one with text, two with image(no text))**
 - **Create 3 Static Texts to label the Buttons**
- **Use `GOLCreateScheme` to form an alternate style scheme**
- **Use `GOLDDraw()` to render the widgets to the screen**

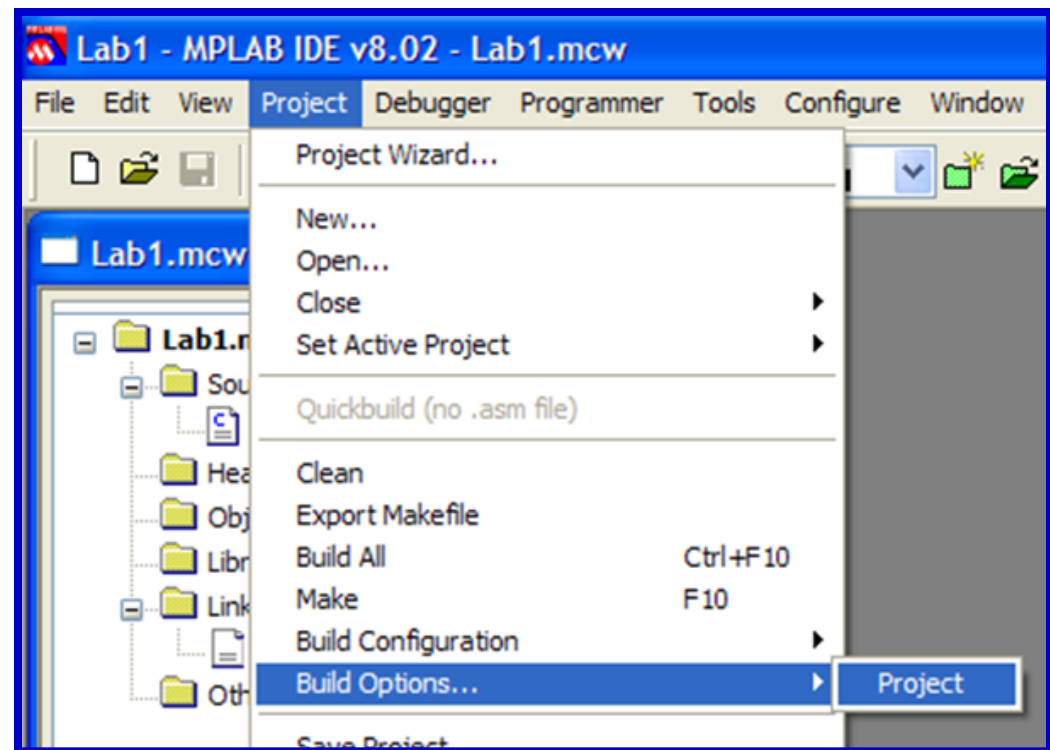
Lab Exercise 2

Allocating Heap



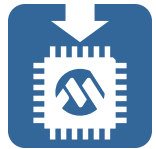
Open the project build options by selecting from the menu:

Project ► Build Options... ► Project



Lab Exercise 2

Allocating Heap



Select the **MPLAB® LINK30** tab at the top of the window.

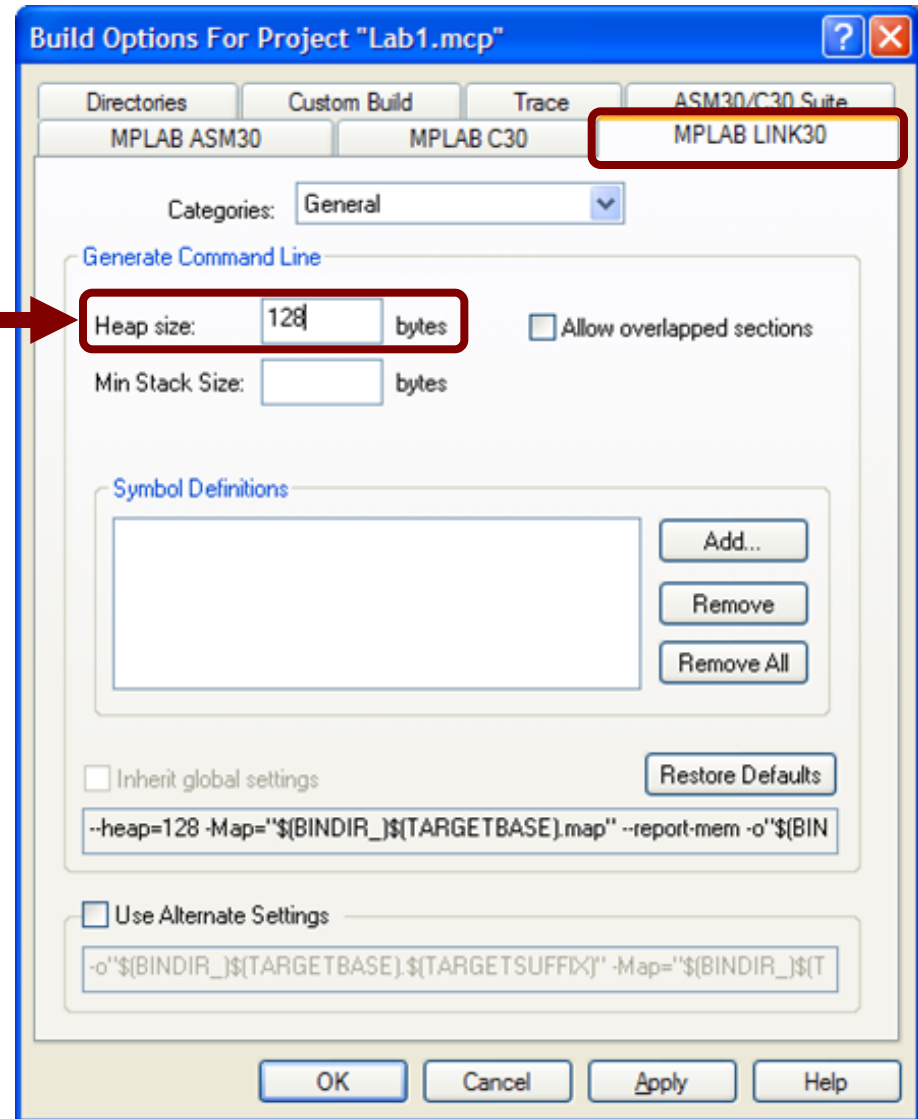
In the **Heap size** box, enter a value of **1500** bytes.



The value of N bytes is based on number of widgets in the list, the number of style schemes used and the application.

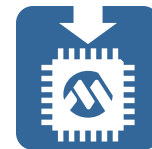
Click

OK

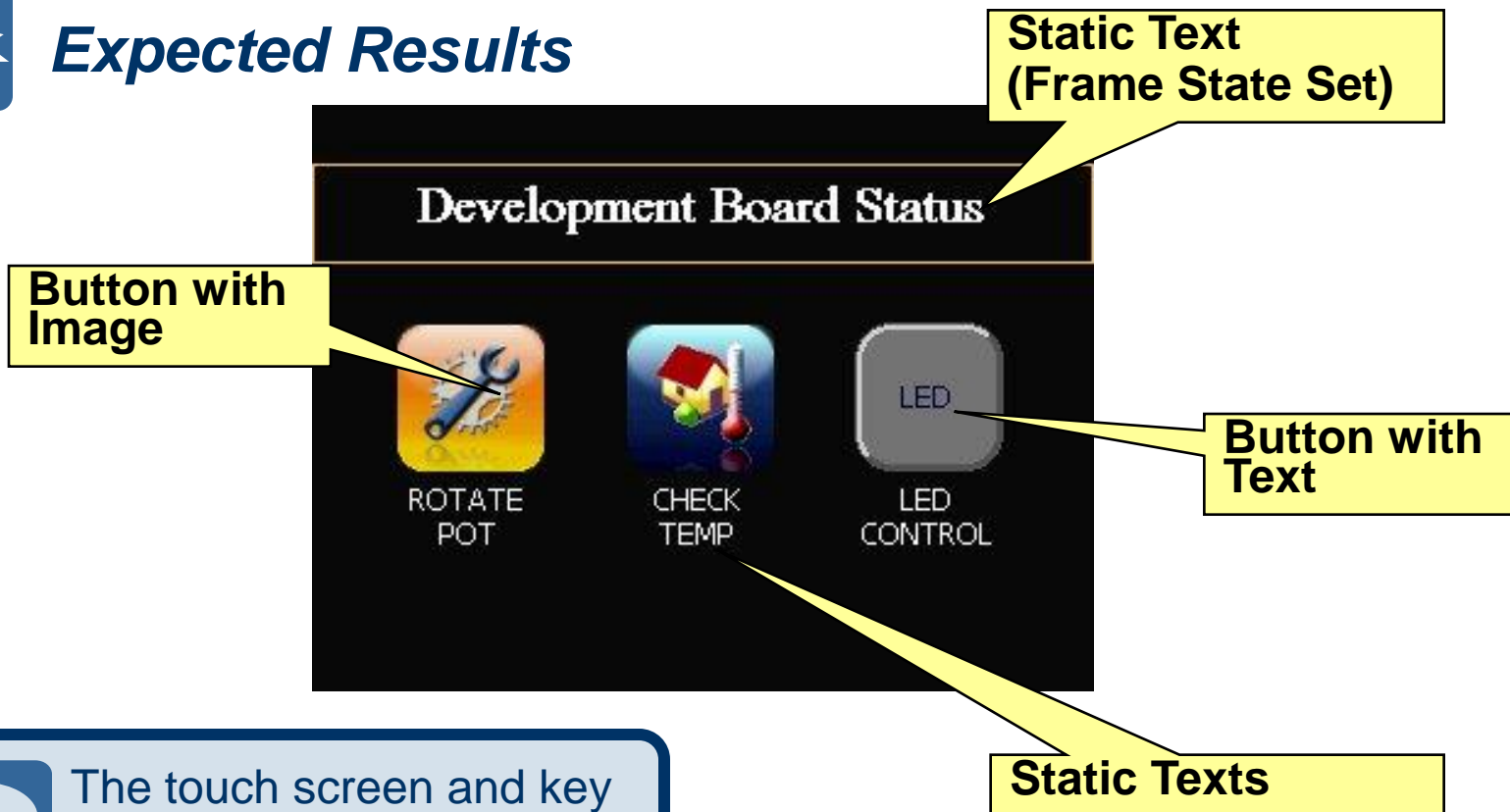


Lab Exercise 2

Create a Menu Screen



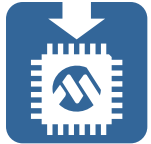
Expected Results



The touch screen and key pad are **DISABLED** for this lab. We will examine their use in the next section and labs.

Lab Exercise 2

Create Buttons

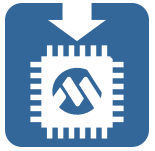


Screens.c – Create button with text and/or Image (One possible solution) ...

```
void CreateMenuScreen(void) {
    ...
    pObj_BUTTON_POT = BtnCreate(
        OBJ_BUTTON_ROTATE_POT,           // Object ID
        POT_STARTX,POT_STARTY,
        POT_STARTX+BTN_WIDTH,
        POT_STARTY+BTN_HEIGHT,           // Button location
        BTN_RADIUS,                      // for rounded corners
        BTN_DRAW,                        // draw the object
        (void *)&Rotate_Icon,           // use an image
        NULL,                            // use no text
        NULL);                           // use default scheme
    ...
    pObj_BUTTON_LED = BtnCreate(
        OBJ_BUTTON_CONTROL_LED,          // Object ID
        LED_STARTX,LED_STARTY,
        LED_STARTX+BTN_WIDTH,
        LED_STARTY+BTN_HEIGHT,           // Button location
        BTN_RADIUS,
        BTN_DRAW,                        // draw the object
        NULL,                            // use no image
        (XCHAR*)Btn_Label,               // use string
        NULL);                           // use default scheme
    ...
}
// (Continued on next slide)...
```

Lab Exercise 2

Create Static Texts

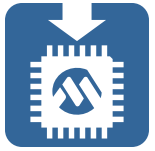


Screens.c – Create static texts (One possible solution) ...

```
void CreateMenuScreen(void) {  
    ...  
    pOBJ_STATICTEXT_POT = StCreate(  
        OBJ_STATICTEXT_POT,           //Static Text ID  
        POT_STARTX,  
        POT_STARTY+BTN_HEIGHT+5,  
        POT_STARTX+BTN_WIDTH,  
        POT_STARTY+(BTN_HEIGHT*2)+5, // Static Text location  
        ST_DRAW|ST_CENTER_ALIGN,      // draw object with text  
                                       // center aligned  
        "ROTATE\nPOT",                 // Text to display  
        labelbox);                    // use labelbox scheme  
    ...  
    // the other two static texts can be implemented in the same way  
}  
// (Continued on next slide)...
```

Lab Exercise 2

Create Style Schemes

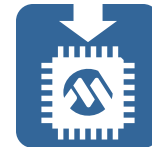


Lab2.c – Edit static texts properties (One possible solution) ...

```
void CreateSchemes(void) {  
    ...  
    // create a style scheme  
  
    statusbox = GOLCreateScheme();  
    labelbox = GOLCreateScheme();  
    flatbuttons = GOLCreateScheme();  
  
    // Set the style scheme to an object  
    GOLSetScheme = (OBJ_HEADER*)pOBJ_BUTTON_LED, flatbuttons);  
  
}  
// (Continued on next slide)...
```

Lab Exercise 2

Main Application Code



Lab2.c – Main application (One possible solution) ...

```
int main(void) {  
    ...  
    // Initialize graphics library and create default  
    // style scheme for GOL  
    GOLInit();  
    ...  
  
    while(1) {  
  
        ///### <--- Lab2 Step 9: Add the function call to  
        render the widgets on the screen ###  
        GOLDraw(); // Draw GOL objects  
  
    }  
}
```

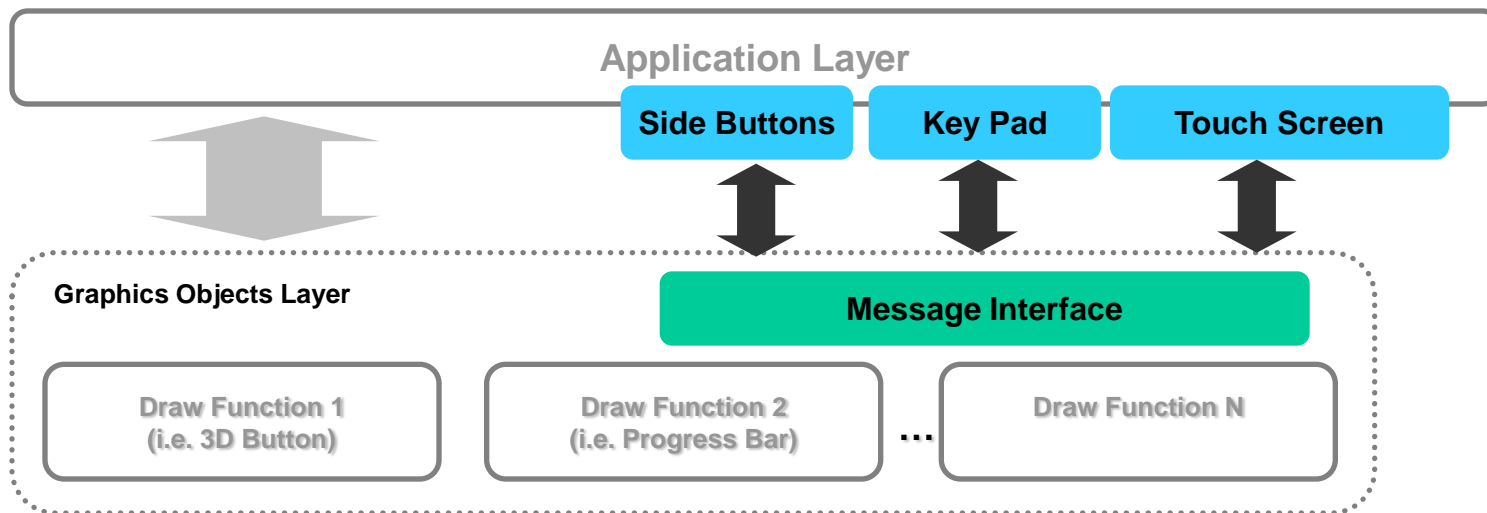


MICROCHIP

Regional Training Centers

**Interfacing the User
Messaging Interface**

Message Interface



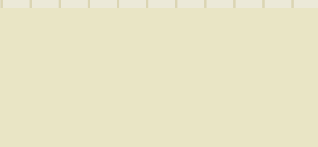
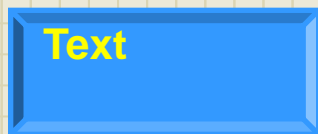
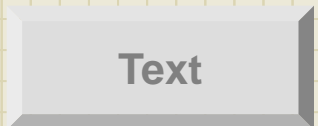
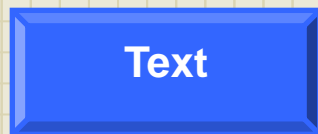
- **Simplifies integration of user input devices**
- **Allow application to efficiently manage widgets**
- **Provides seamless interface for the user**
- **Future support for more devices (e.g., mouse)**

state Field Common Bits

- **Property state bits:**
 - 10 Least Significant Bits of the **state** member of widget structure
 - Define action and appearance
 - Not automatically changed by **GOLDDraw()**
- **Used by SOME widgets:**
 - **OBJ_FOCUSED**: Widget is in focus
- **Used by ALL widgets:**
 - **OBJ_DISABLED**: Widget is turned off
 - All messages will be ignored

State Bits Example

Button Widget



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

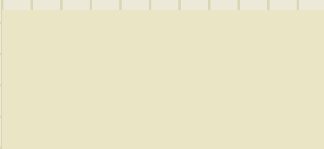
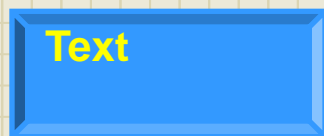
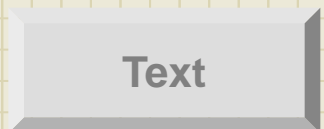
    // Example Button Focus
    SetState(pBtn, BTN_FOCUSED);

    // 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
    SetState(pBtn, BTN_TEXTTOP);
    SetState(pBtn, BTN_TEXTRIGHT);

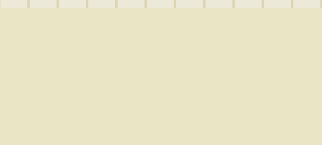
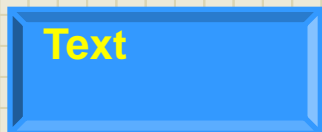
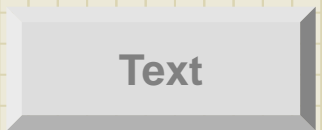
    // Example Button Focus
    SetState(pBtn, BTN_FOCUSED);

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

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



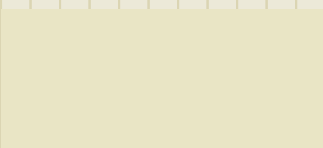
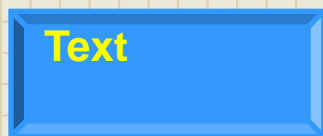
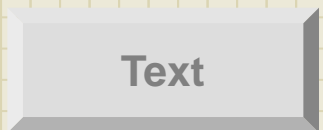
Button States



```
if (GOL_DRAW())  
{  
    // Example Button text alignments  
    SetState(pBtn, BTN_TEXTTOP);  
    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)  
}
```



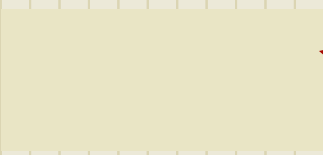
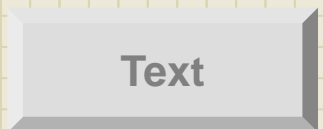
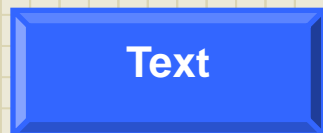
Button States



```
if (GOL_DRAW())  
{  
    // Example Button text alignments  
    SetState(pBtn, BTN_TEXTTOP);  
    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)  
}
```



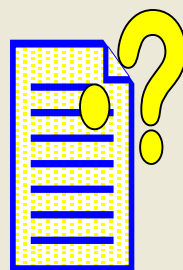
Button States



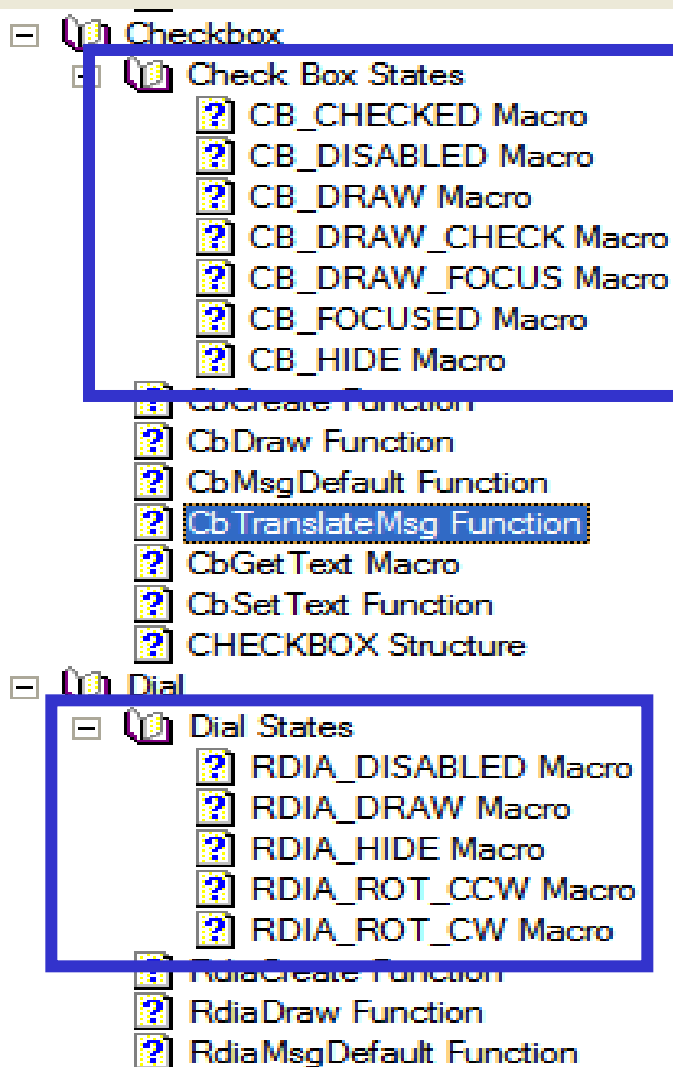
```
if (GOL_DRAW())  
{  
    // Example Button text alignments  
    SetState(pBtn, BTN_TEXTTOP);  
    SetState(pBtn, BTN_TEXTRIGHT);  
  
    // Example Button Focus  
    SetState(pBtn, BTN_FOCUSED);  
  
    // Example Button Action  
    SetState(pBtn, BTN_DISABLED);  
    state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTLEFT;  
    SetState(pBtn, state);  
  
    // Example Hiding Button  
    SetState(pBtn, BTN_HIDE)  
}
```

Help with State Bits

State bit descriptions for all widgets are located in the Graphics Library Help file.



Graphics Library Help.chm



Interfacing the User

Programmer Requirements Part 1:

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



Message Structure

■ Message Structure

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

- **type** = *TYPE_KEYBOARD* or *TYPE_TOUCHSCREEN*
- **param1** and **param2** depend on event and type
 - For touch screen:
 - **param1**: x position
 - **param2**: y position
 - For keypad and side buttons:
 - **param1**: ID of widget receiving message
 - **param2**: depends on widget and event

Message Structure

■ **uiEvent** defines specific user action

■ Touch screens **uiEvent** values

- **EVENT_PRESS**
- **EVENT_RELEASE**
- **EVENT_MOVE**
- **EVENT_INVALID**
 - No touch

■ keypad and side buttons **uiEvent** values

- **EVENT_KEYSCAN**
 - **param2** value varies based on widget
 - Usually a scan code
- **EVENT_CHARCODE** (edit box only)
 - **param2** = character to add
- **EVENT_INVALID**
 - No key pressed

























AT keyboard scan codes are provided in the library help file.

Example

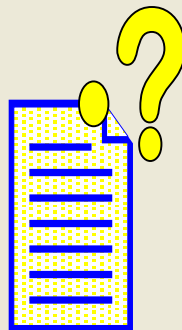
Populate Side Button Message

```
if(S5){//button is pressed
    msg->type      = TYPE_KEYBOARD;
    msg->uiEvent   = EVENT_KEYSCAN;
    msg->param1    = ID;
    msg->param2    = SCAN_CR_PRESSED;
}else{//button is released
    msg->type      = TYPE_KEYBOARD;
    msg->uiEvent   = EVENT_KEYSCAN;
    msg->param1    = ID;
    msg->param2    = SCAN_CR_RELEASED;
}return;
```

Widget Message Help

- [-]  Button
 - [-]  Button States
 -  BTN_DISABLED Macro
 -  BTN_DRAW Macro
 -  BTN_DRAW_FOCUS Macro
 -  BTN_FOCUSED Macro
 -  BTN_HIDE Macro
 -  BTN_PRESSED Macro
 -  BTN_TEXTBOTTOM Macro
 -  BTN_TEXTLEFT Macro
 -  BTN_TEXTRIGHT Macro
 -  BTN_TEXTTOP Macro
 -  BTN_TOGGLE Macro
 -  BtnCreate Function
 -  BtnDraw Function
 -  BtnMsgDefault Function
 -  **BtnTranslateMsg Function**
 -  BtnGetText Macro
 -  BtnSetText Function
 -  BtnGetBitmap Macro
 -  BtnSetBitmap Macro
 -  BUTTON Structure

A table describing the valid input sources, events, and default behaviors can be found in the [ObjTranslateMsg](#) function description for each widget.



Graphics Library Help



MICROCHIP

Regional Training Centers

**Interfacing the User
Widget Actions**

Receiving Messages

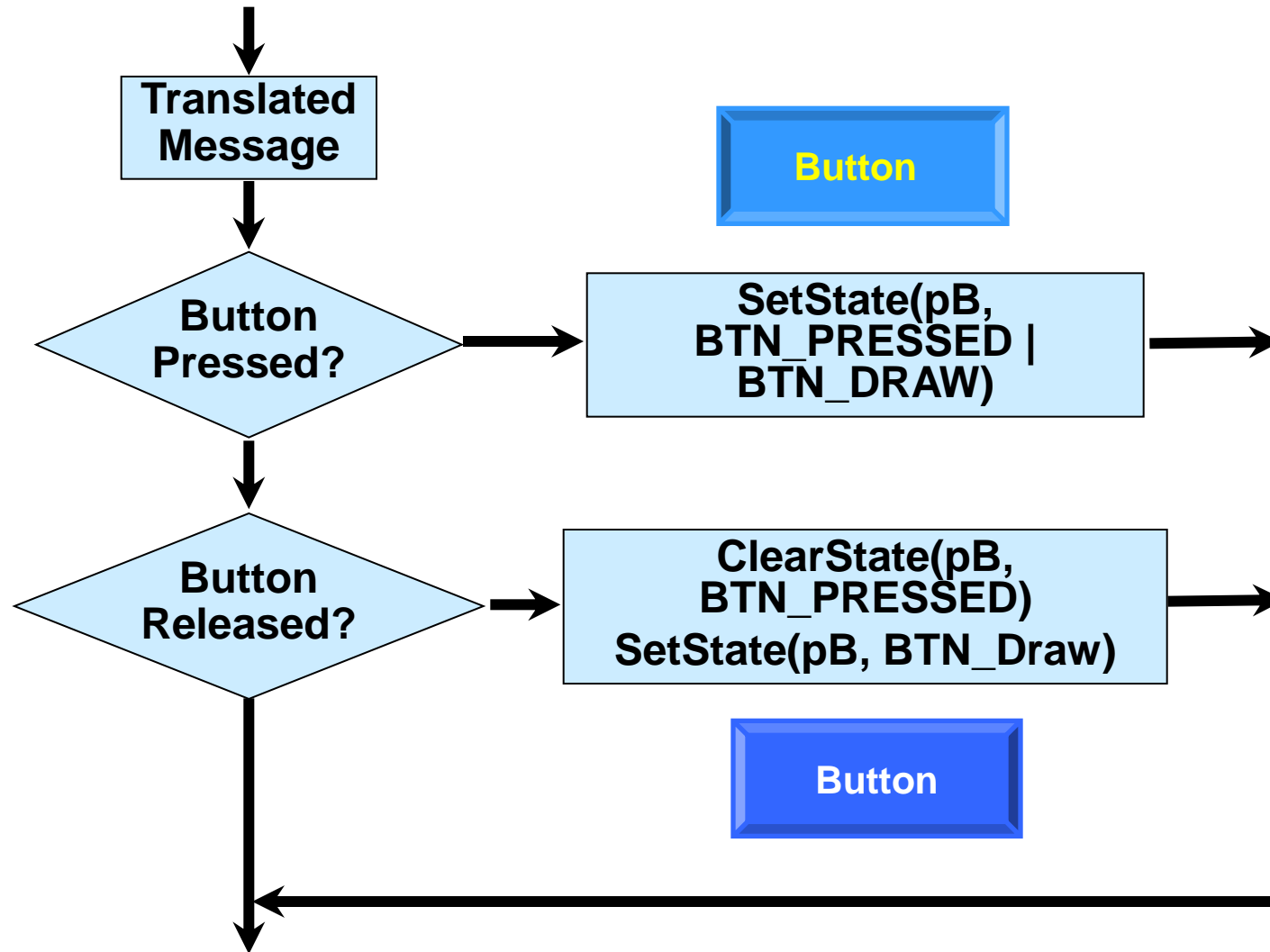
Linked List

ID_OBJ1 -> state bits
ID_OBJ2 -> state bits
ID_OBJ3 -> state bits
...
ID_OBJN -> state bits

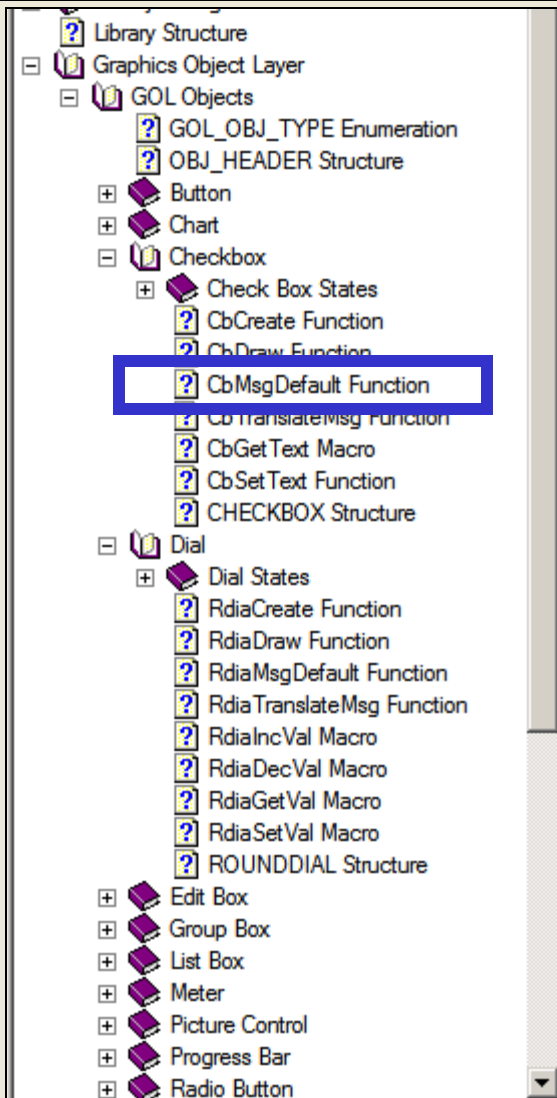
- **GOLMsg (&msg)**
 - Translates message
 - Finds affected widget in linked list
 - Modifies widget statebits
 - Performs some action
 - Returns TRUE when done
 - User must supply pointer to a message structure

Widget Default Actions

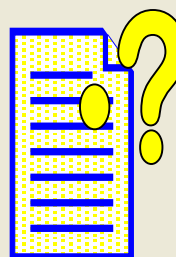
Button



Widget Default Action Help



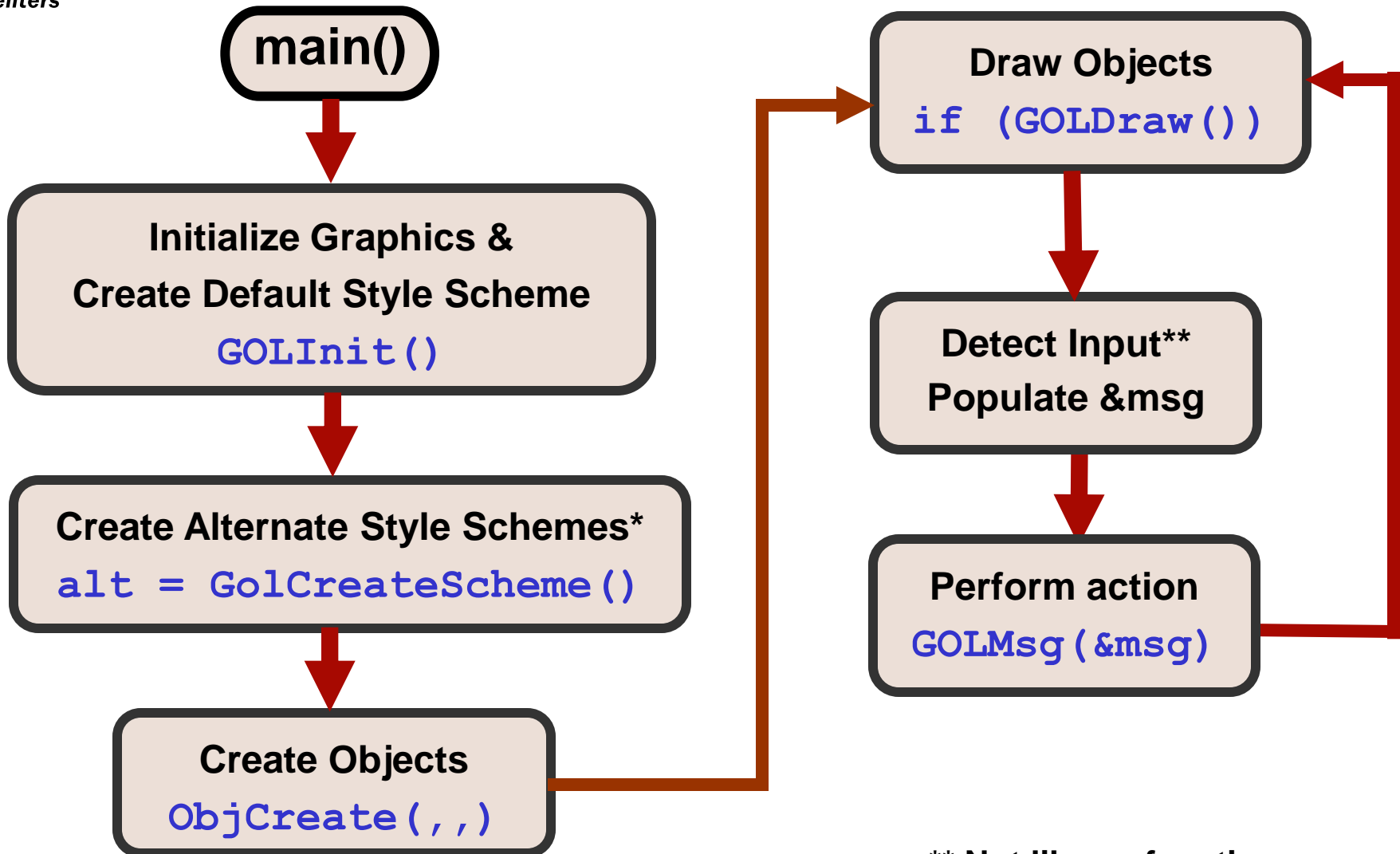
Every widget has it's own default action based on the translated message. These are described in the library help file.



Graphics Library Help

Translated Message	Input Source		Set/Clear State Bit	Description
CB_MSG_CHECKED	Touch	Screen,	Set <u>CB_CHECKED</u>	Check Box will be redrawn in checked state.
CB_MSG_UNCHECKED	Keyboard	Screen,	Clear <u>CB_CHECKED</u>	Check Box will be redrawn in unchecked state.

Typical Application Flow



**** Not library functions**

*** May skip this step if only using default scheme**



MICROCHIP

Regional Training Centers

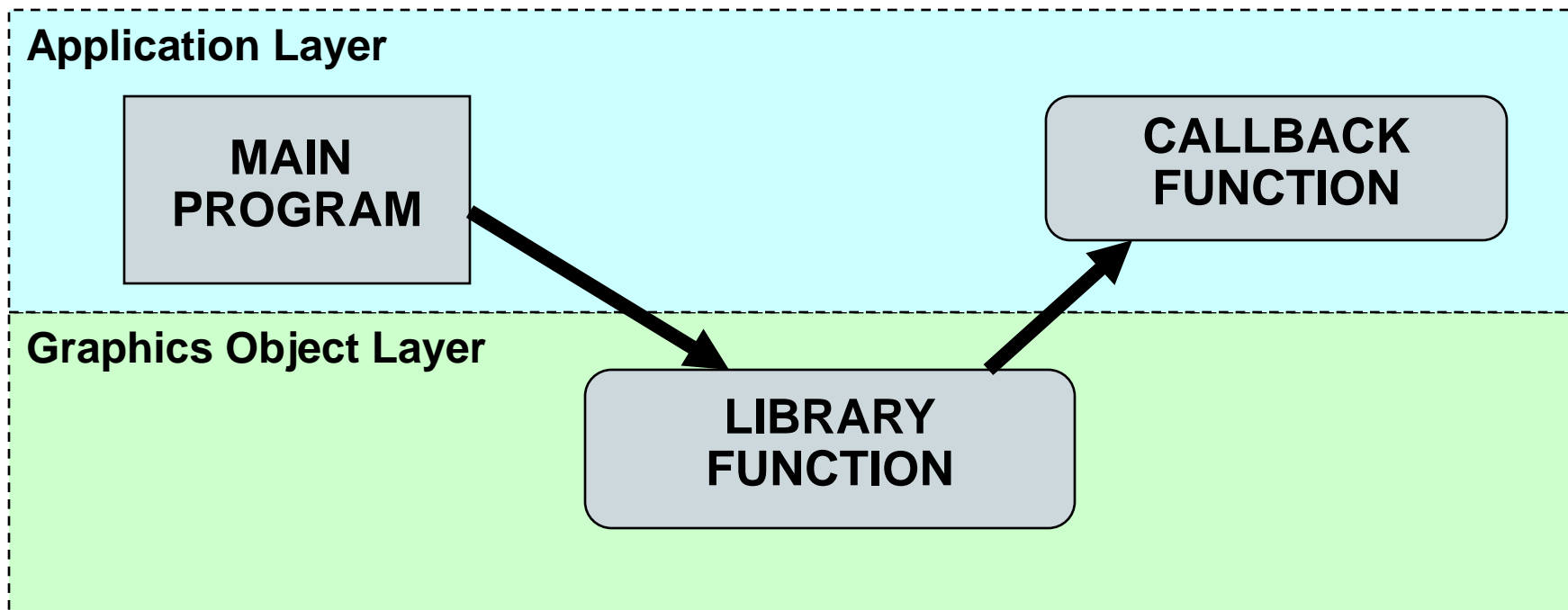
Interfacing the User & System
GOLMsgCallback()

What is a Callback?

Definition:

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

- Used to add system or widget response to user inputs.



Interfacing the User Programmer Requirements Part 2

■ Provide callback functions (REQUIRED)

■ GOLMsgCallback (, , ,)

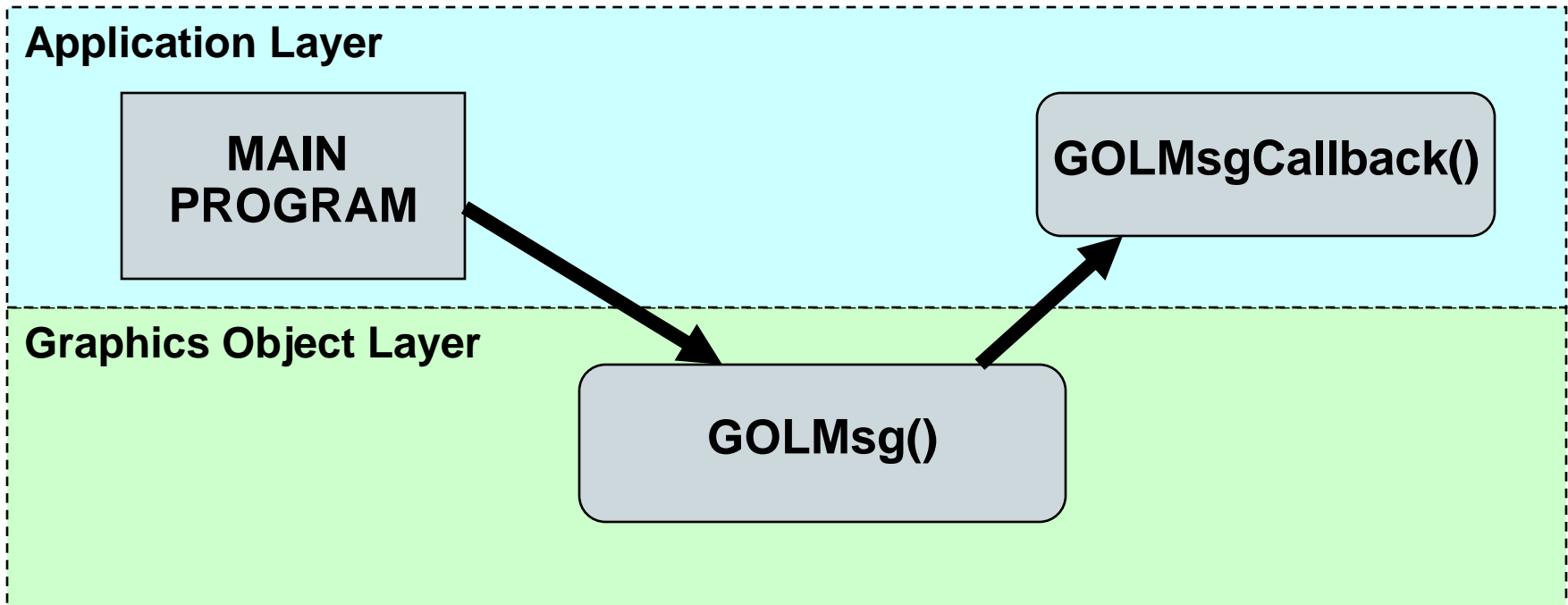
- Uses translated message to perform action
- System/Widget action based on SINGLE events
- Example: Push a button to turn on LED
- Called by **GOLMsg ()**

■ GOLDrawCallback ()

- Does not directly use translated messages
- System/Widget action based on continuous events
- Example: Hold a button to change volume
- Called by **GOLDraw ()**
- ONLY safe place to modify drawing properties

GOLMsgCallback()

- Called by GOLMsg()
- Used to add system or widget response to user inputs
 - Based on non-continuous actions



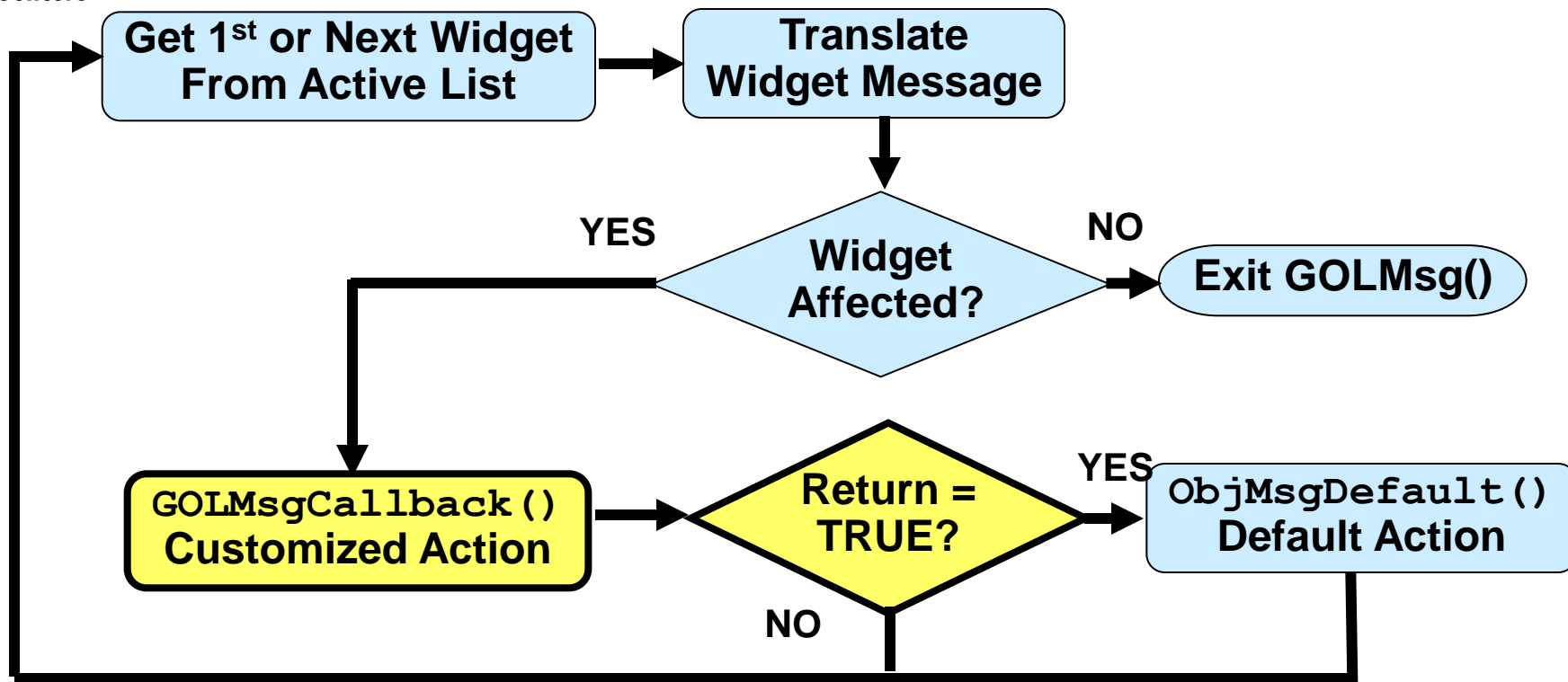
GOLMsgCallback ()

- 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 of the widget
 - pObj: Pointer to the widget
 - pMsg: Pointer to message structure
- Output:
 - TRUE: To perform default actions too
 - '0' : To skip default actions



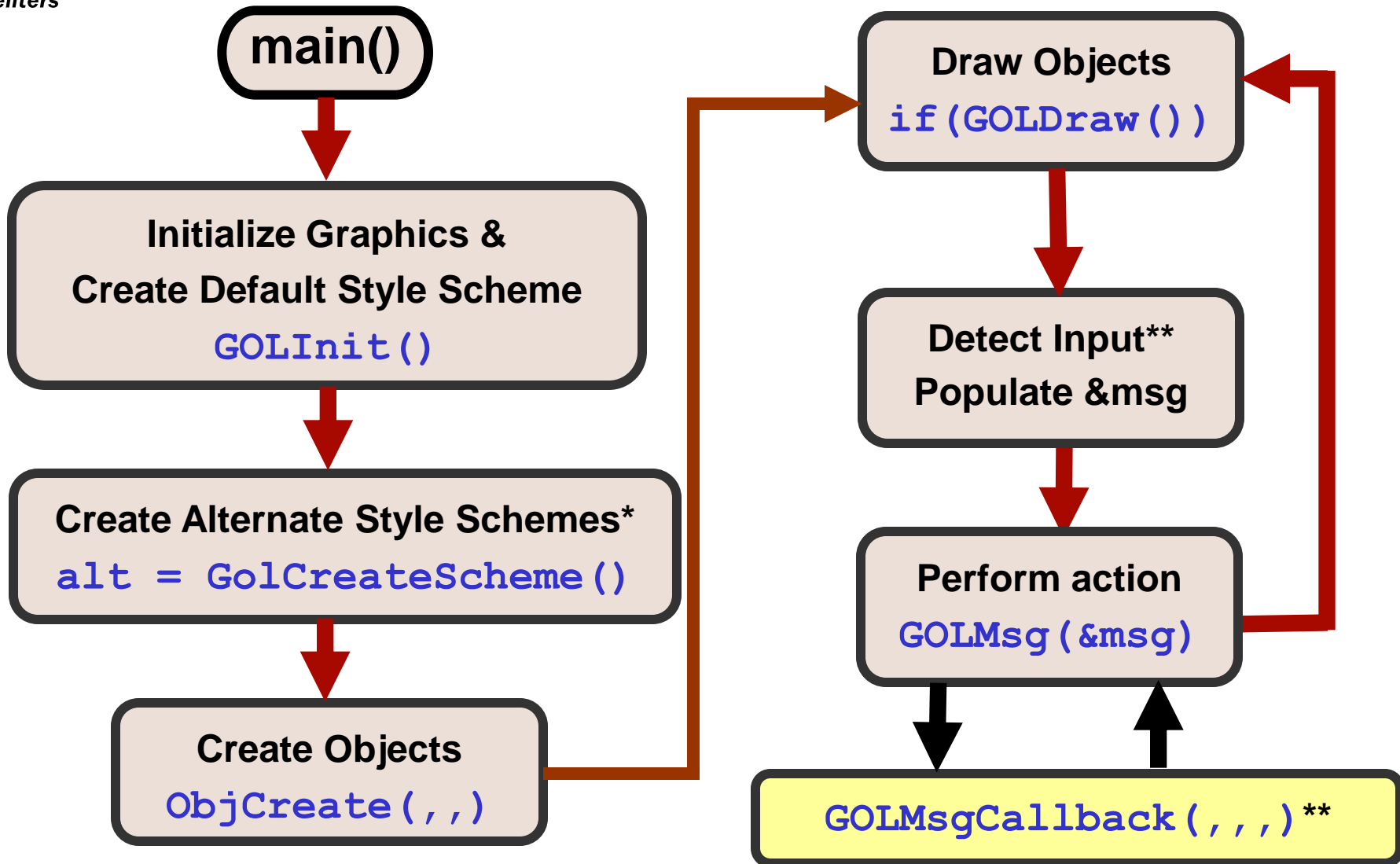
GOLMsg (&msg)

Library Code Flow Chart



- User **MUST** supply GOLMsgCallback()
 - Returning FALSE means no action taken by library
- Do **NOT** call until GOLDraw() completes
 - `if (GOLDraw()) GOLMsg(&msg);`

Typical Application Flow



* May skip this step if only using default scheme

** Application implemented functions



Widget APIs

■ Widgets with Text:

- `ObjSetText(*pObj, *pText)`
- `ObjGetText(*pObj)`
 - Returns a pointer to the text string in use

■ Widgets with Bitmaps:
















- `ObjSetBitmap(*pObj, *pText)`
- `ObjGetBitmap(*pObj)`
 - Returns a pointer to the bitmap in use

■ Widget Specific Actions:

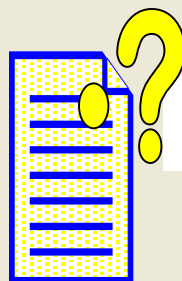
- Examples:
- `PbSetPos(*pObj)` – Increment and decrement a progress bar
- `MtrIncVal(*pObj)`, `MtrDecVal(*pObj)` – Increment and decrement a meter



Widget API Help

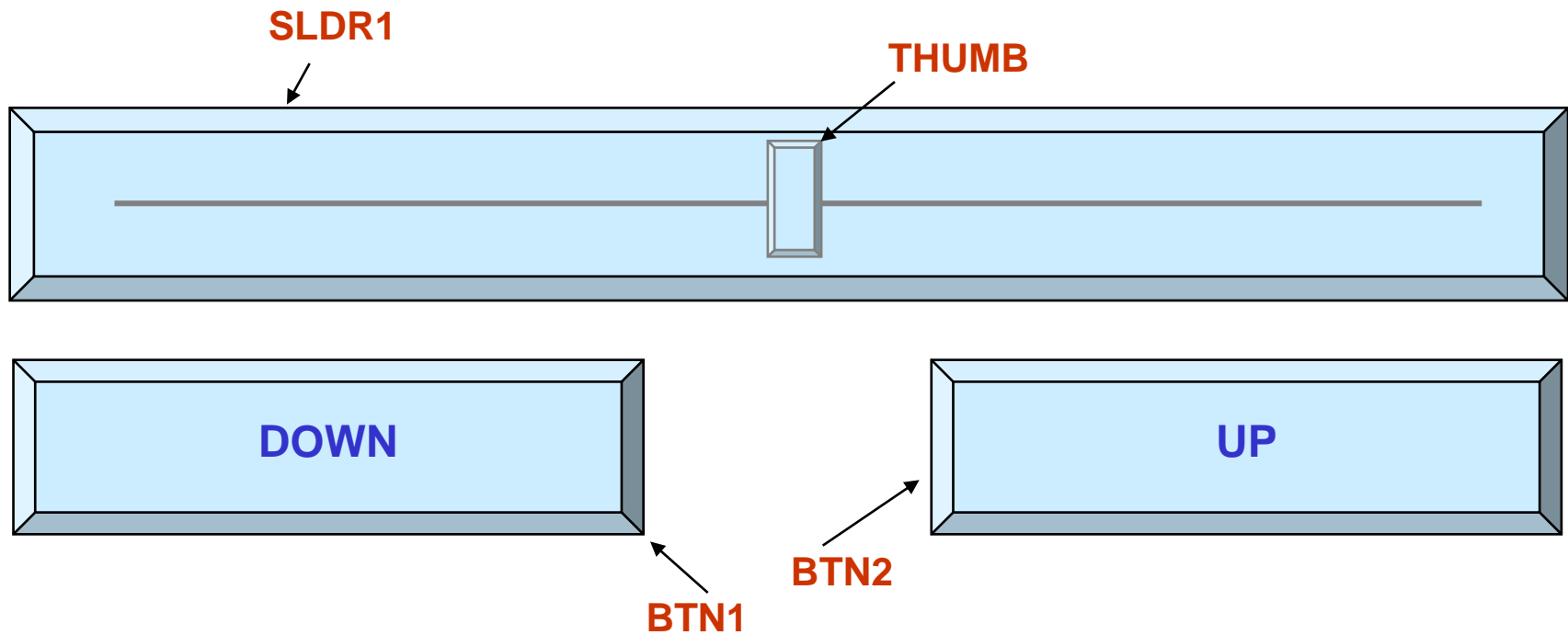
-  Dial
 - +  Dial States
 -  RdiaCreate Function
 -  RdiaDraw Function
 -  RdiaMsgDefault Function
 -  RdiaTranslateMsg Function
 -  RdiaDecVal Macro
 -  RdiaIncVal Macro
 -  RdiaGetVal Macro
 -  RdiaSetVal Macro
 -  ROUND DIAL Structure
 - +  Edit Box
 - +  Group Box
 - +  List Box
 - +  Meter

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

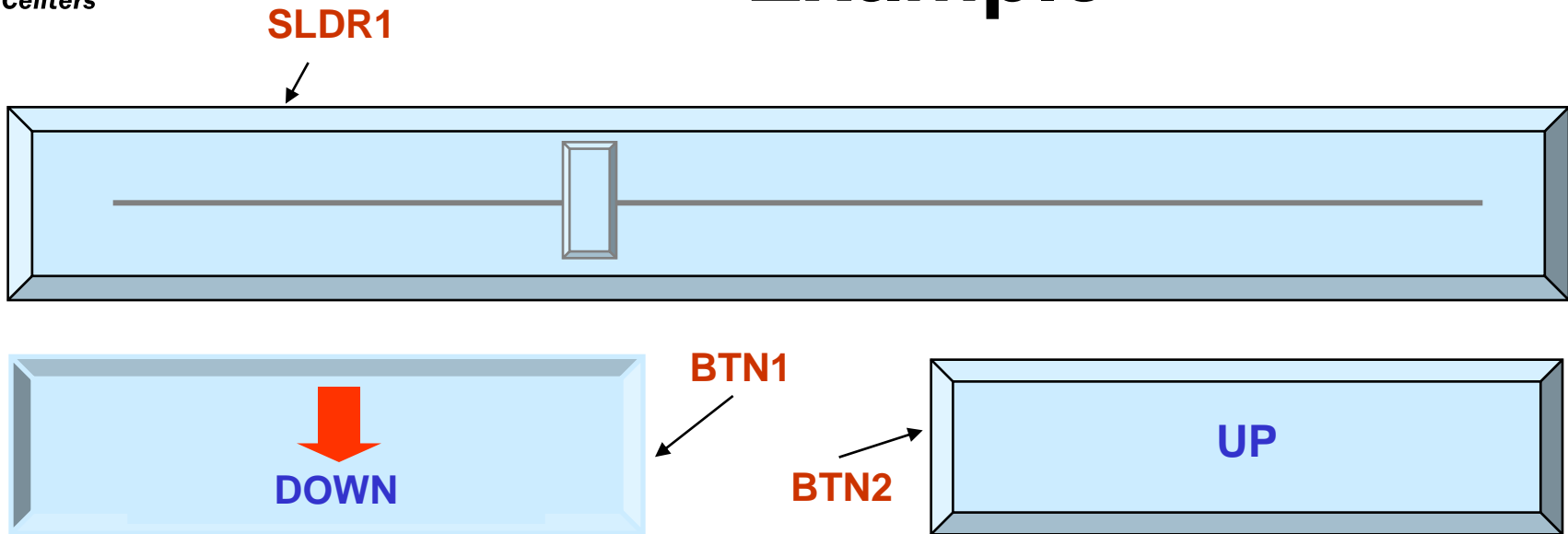


Graphics Library Help.chm

GOLMsgCallback() Example

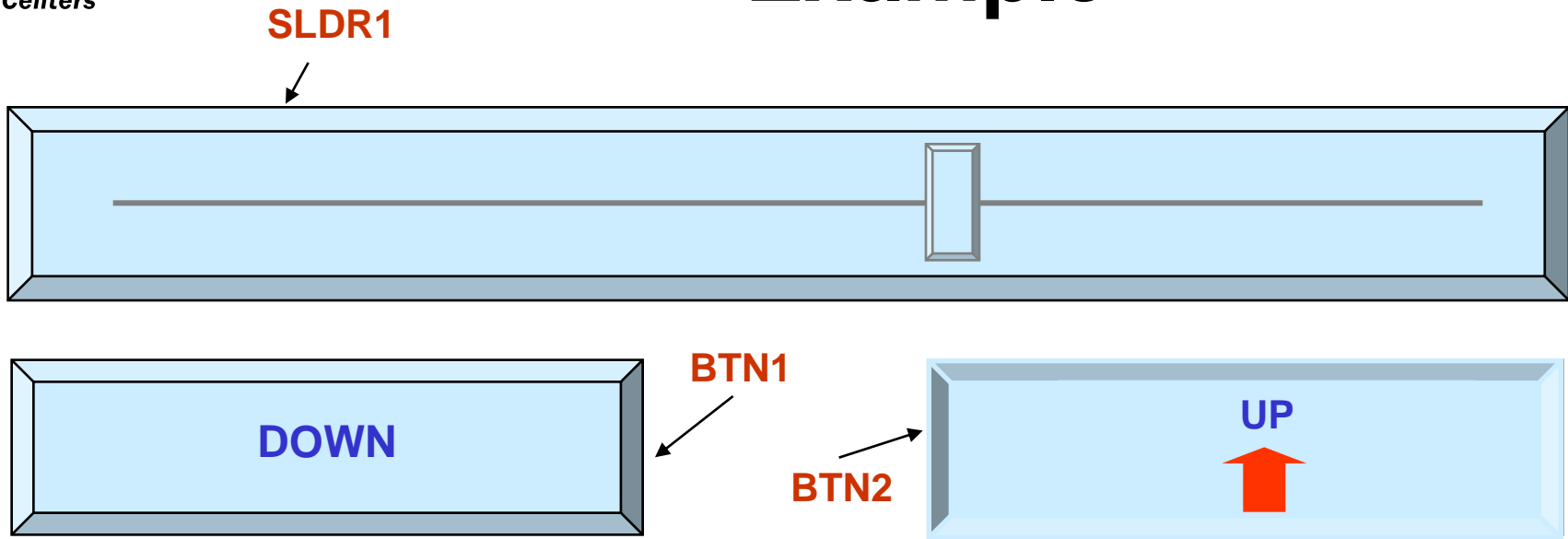


GOLMsgCallback() Example



- **Widget Actions:**
 - Move slider thumb to the left
 - Move BTN1 text down
 - Add down arrow bitmap
- **System Action:**
 - Decrease motor speed

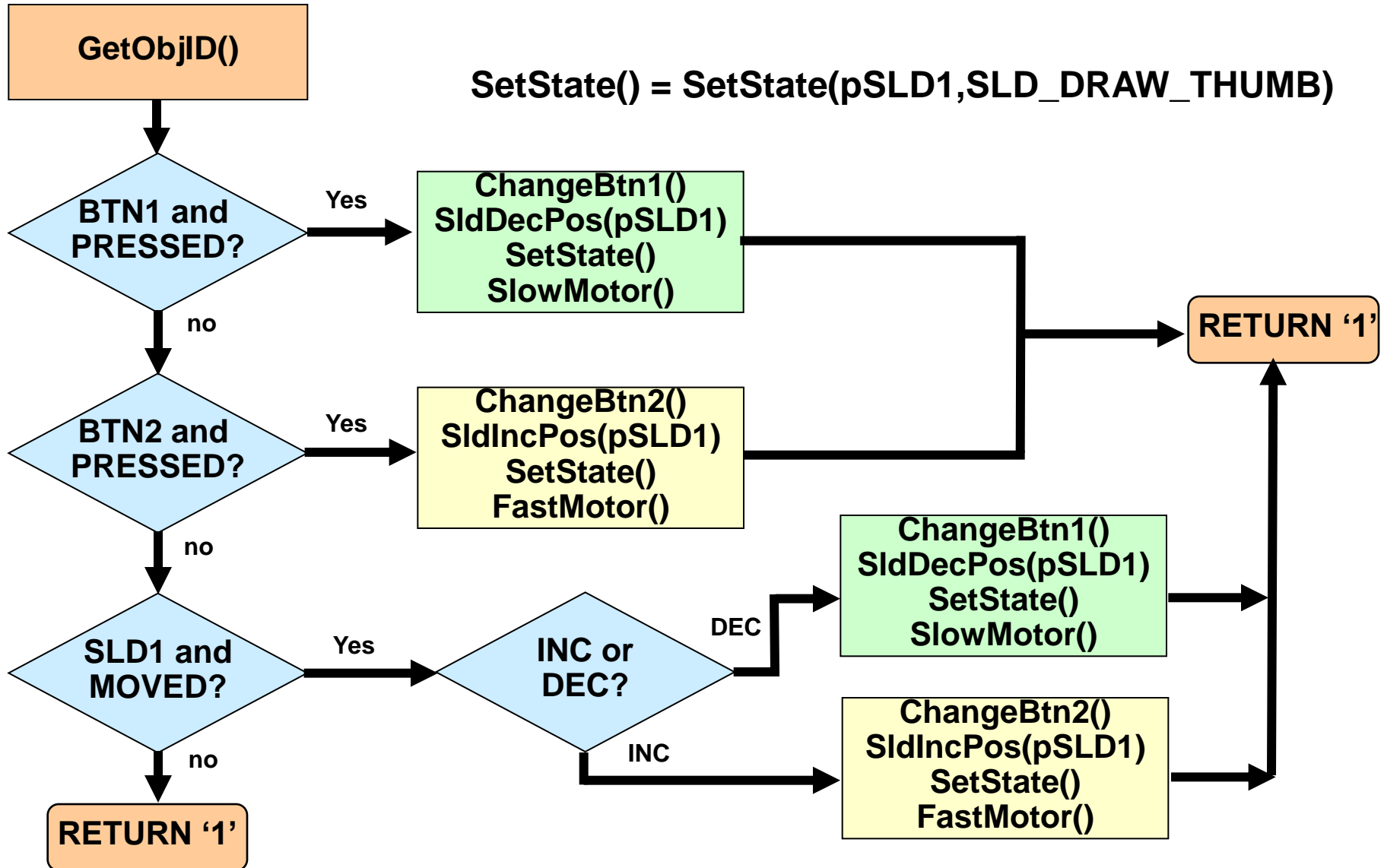
GOLMsgCallback() Example



- **Widget Actions:**
 - Move slider thumb to the right
 - Move BTN2 text up
 - Add up arrow bitmap
- **System Action:**
 - Increase motor speed

GOLMsgCallback () Example

SetState() = SetState(pSLD1,SLD_DRAW_THUMB)





Reminder ...

- `GetObjID (*pObj)`
 - Returns ID field of the object

```
#define ID_BTN1  
BUTTON *pBtn;  
...
```

```
currObjID = GetObjID(pBtn);
```

```
Switch(currObjID){
```

```
    case ID_BTN1: do something useful
```

```
    break;
```

```
...}
```

Assign Handle



Use Handle





GOLMsgCallback()

Example: Code Snippet

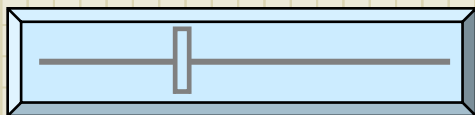
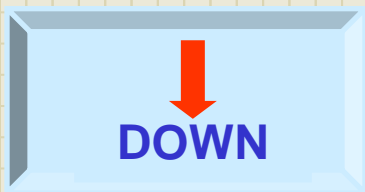


```
WORD GOLMsgCallback (objMsg, pObj, pMsg) {  
...  
objID = GetObjID (pObj) ;  
pSldObj = (*SLIDER) GOLFindObj (SLD1) ;  
    if (objID == BTN1) {  
        if (objMsg == BTN_MSG_PRESSED) {  
            BtnSetBitmap (pObj, &blueDOWNarrow) ;  
            SetState (pBTN1, BTN_TEXTBOTTOM) ;  
            SldDecPos (pSldObj) ;  
            SetState (pSldObj, SLD_DRAW_THUMB) ;  
            SlowMotor () ; }  
        else {  
            BtnSetBitmap (pObj, NULL) ;  
            ClrState (pObj, BTN_TEXTBOTTOM) ; } }  
...  
}
```



GOLMsgCallback()

Example: Code Snippet



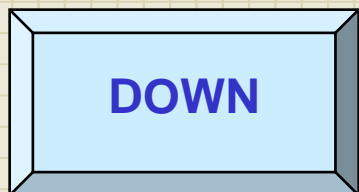
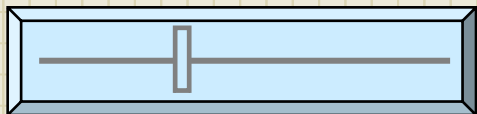
```
WORD GOLMsgCallback(objMsg,pObj,pMsg) {  
...  
objID = GetObjID(pObj);  
pSldObj = (*SLIDER)GOLFindObj(SLD1);  
    if(objID == BTN1){  
        if (objMsg == BTN_MSG_PRESSED){  
            BtnSetBitmap(pObj, &blueDOWNarrow);  
            SetState(pBTN1, BTN_TEXTBOTTOM);  
            SldDecPos(pSldObj);  
            SetState(pSldObj, SLD_DRAW_THUMB);  
            SlowMotor();}  
        else{  
            BtnSetBitmap(pObj, NULL);  
            ClrState(pObj, BTN_TEXTBOTTOM);}}  
...  
}
```



GOLMsgCallback()

Example: Code Snippet

```
WORD GOLMsgCallback(objMsg,pObj,pMsg) {  
...  
objID = GetObjID(pObj);  
pSldObj = (*SLIDER)GOLFindObj(SLD1);  
if(objID == BTN1){  
    if (objMsg == BTN_MSG_PRESSED){  
        BtnSetBitmap(pObj, &blueDOWNarrow);  
        SetState(pBTN1, BTN_TEXTBOTTOM);  
        SldDecPos(pSldObj);  
        SetState(pSldObj, SLD_DRAW_THUMB);  
        SlowMotor();}  
    else{  
        BtnSetBitmap(pObj, NULL);  
        ClrState(pObj, BTN_TEXTBOTTOM);}  
...  
}
```





MICROCHIP

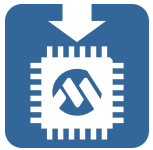
Regional Training Centers

Lab Exercise 3

Interfacing the User

Lab Exercise 3

Cooking Screen



Purpose

The purpose of this lab is to learn to use messaging functions GOLMsg() and GOLMsgCallback() to fully interface the user. We will implement widget control using the state bits and Widget APIs. LEDs will be used to provide system output.



Procedure

Follow the directions in the lab manual starting on page 3-1

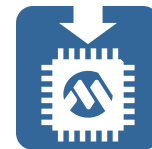
Objectives:

- **Implement a LED control screen:**
 - Static Text to show the latest user action
 - Check boxes to control the LED

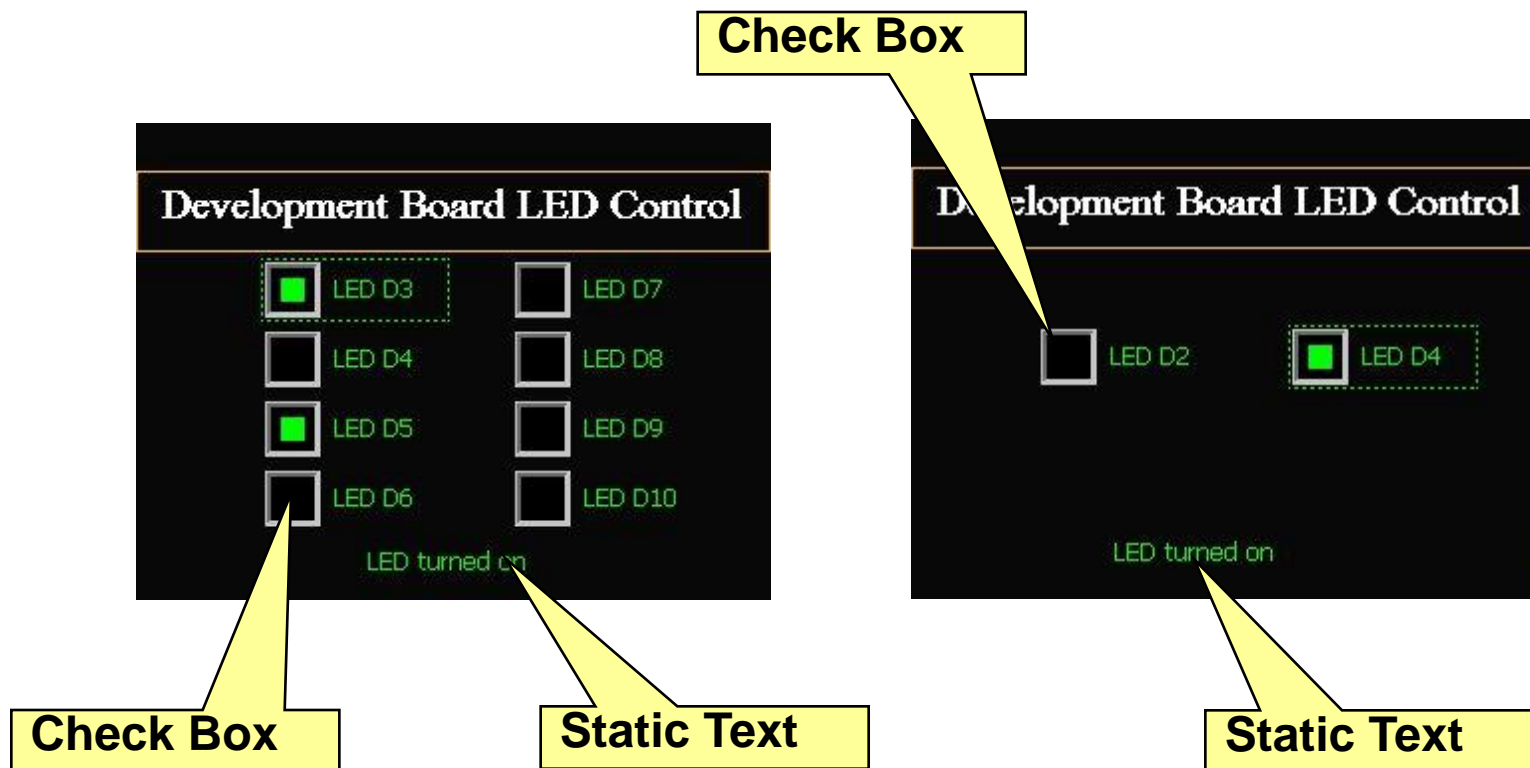


Lab Exercise 3

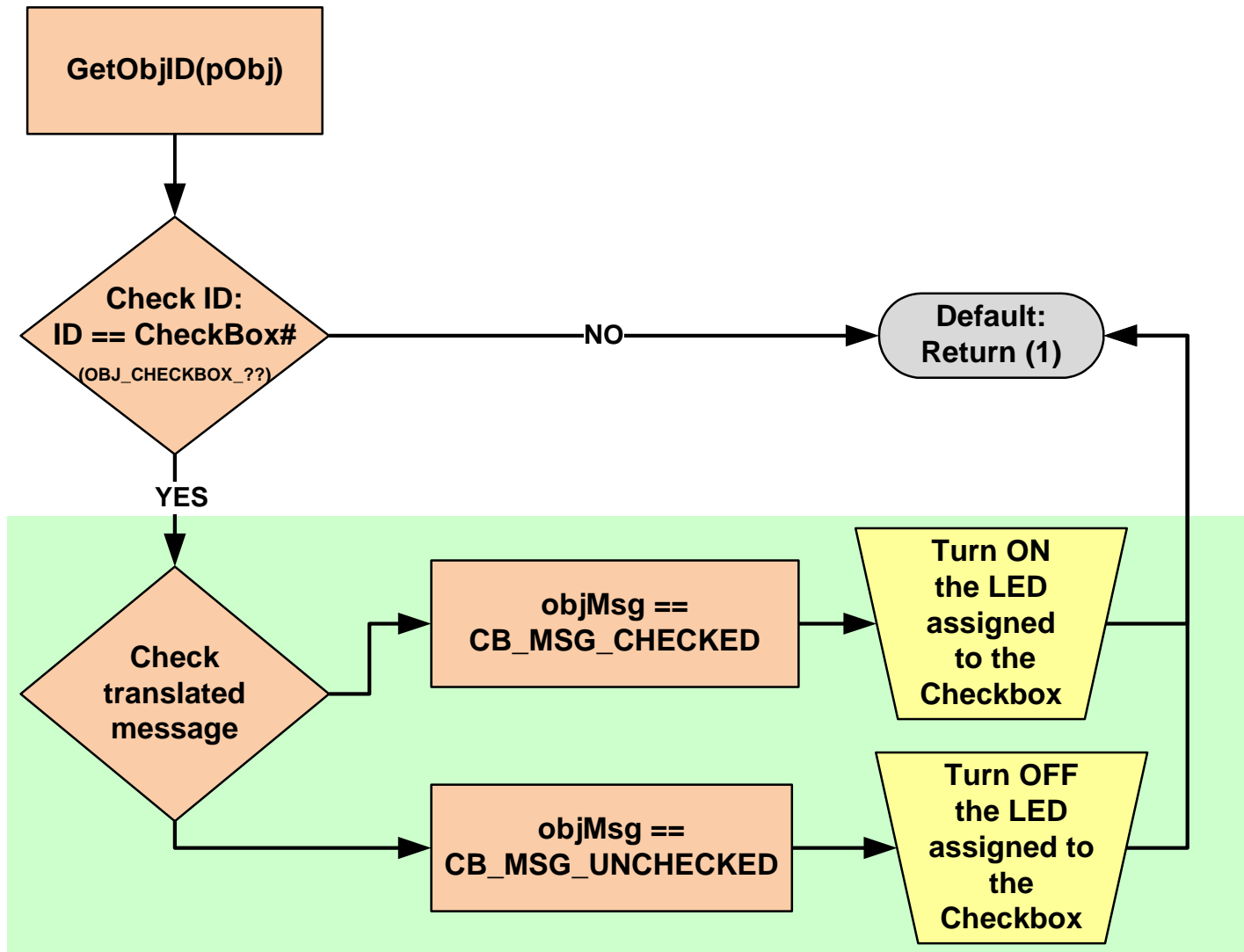
LED Control Screen



Expected Results

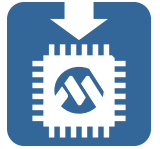


GOLMsgCallback() Flowchart



Lab Exercise 3

Implement `GOLMsgCallback()`



- **Let's examine one possible solution.**
 - **Open MPLAB....**
 - **Open `GOLMsgCallback()`**
 - **Open `Main()`**



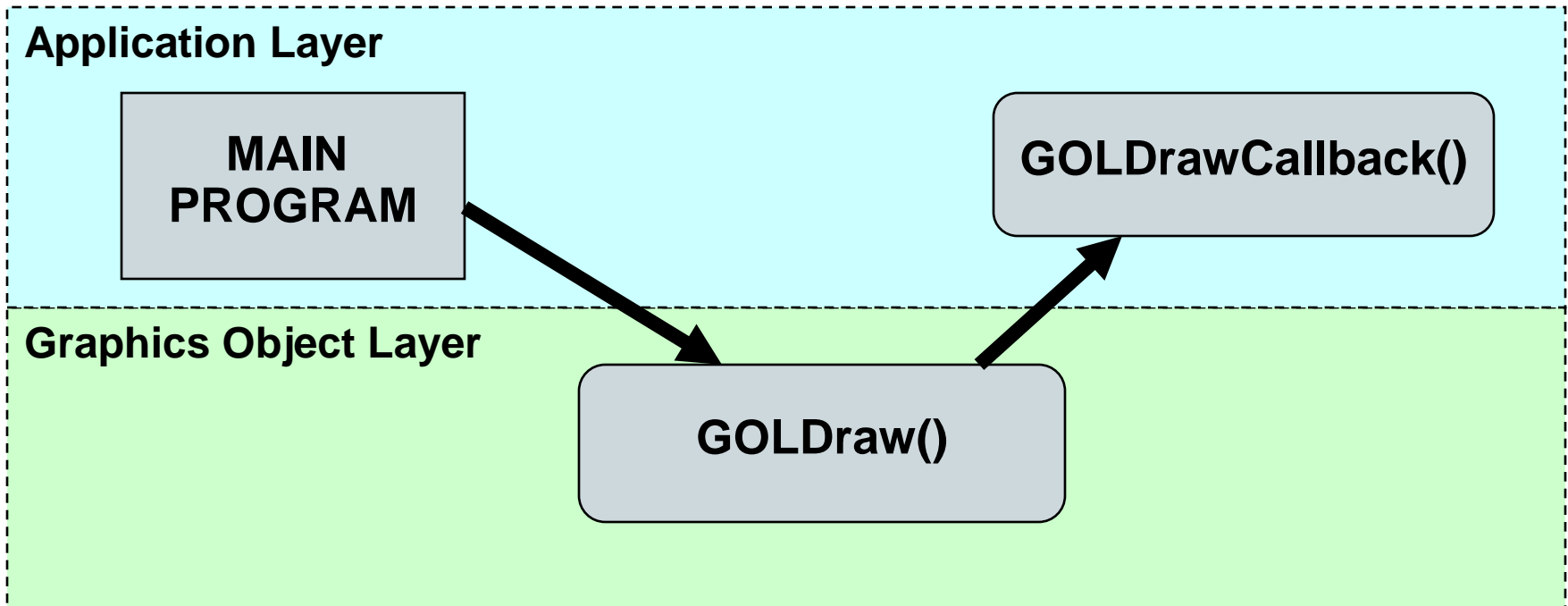
MICROCHIP

Regional Training Centers

Interfacing the User & System
GOLDDrawCallback()

Callback Reminder

- Called by GOLDDraw()
- Used to add system or widget response to user inputs
 - Based on continuous actions (time based events)

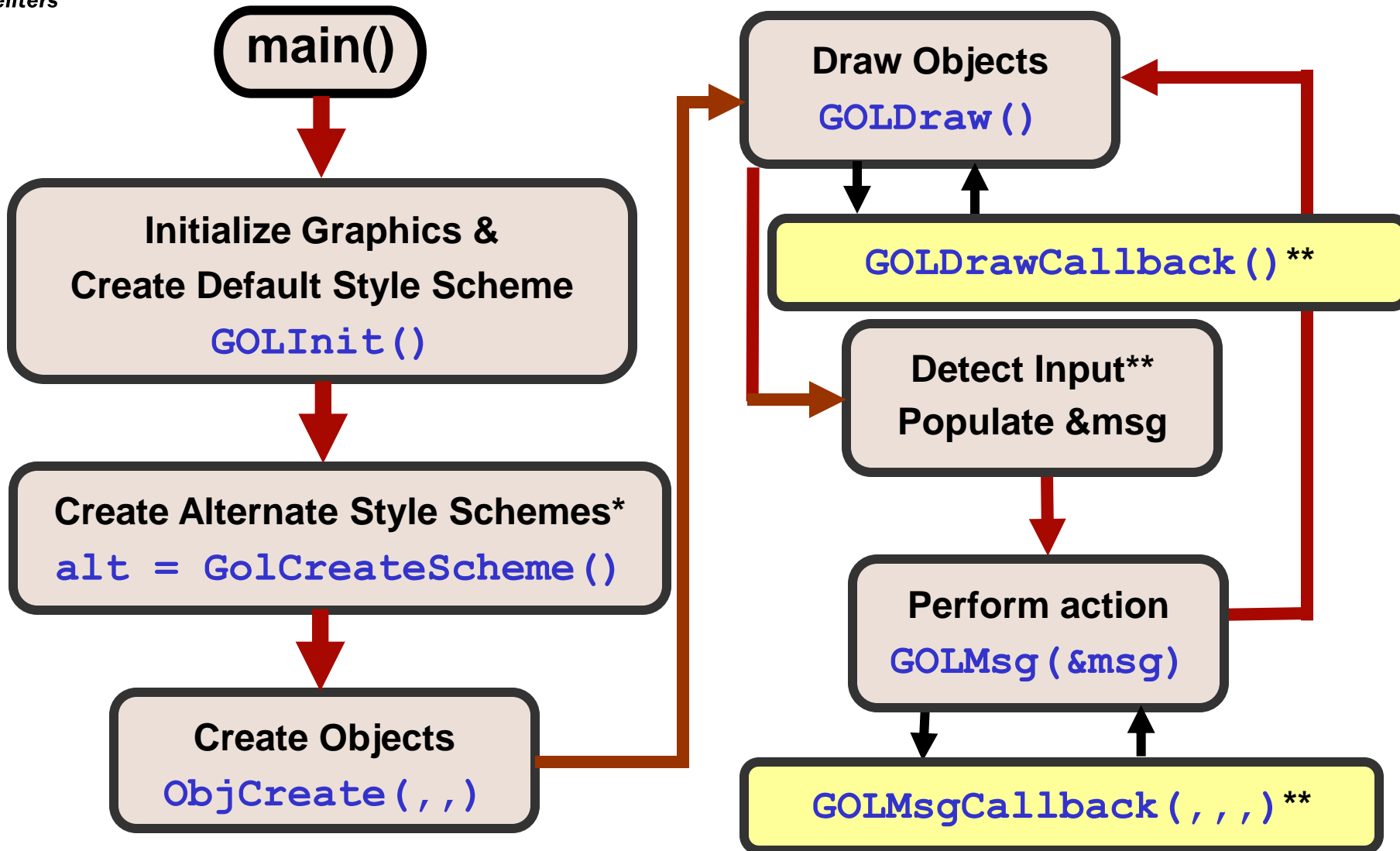


Customized Drawing

GOLDrawCallback ()

- Called by GOLDraw ()
 - Drawing 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: Holding down a button
 - Example: Display a countdown timer
- Only safe place to:
 - Change drawing parameters
 - Modify active linked list

Typical Application Flow

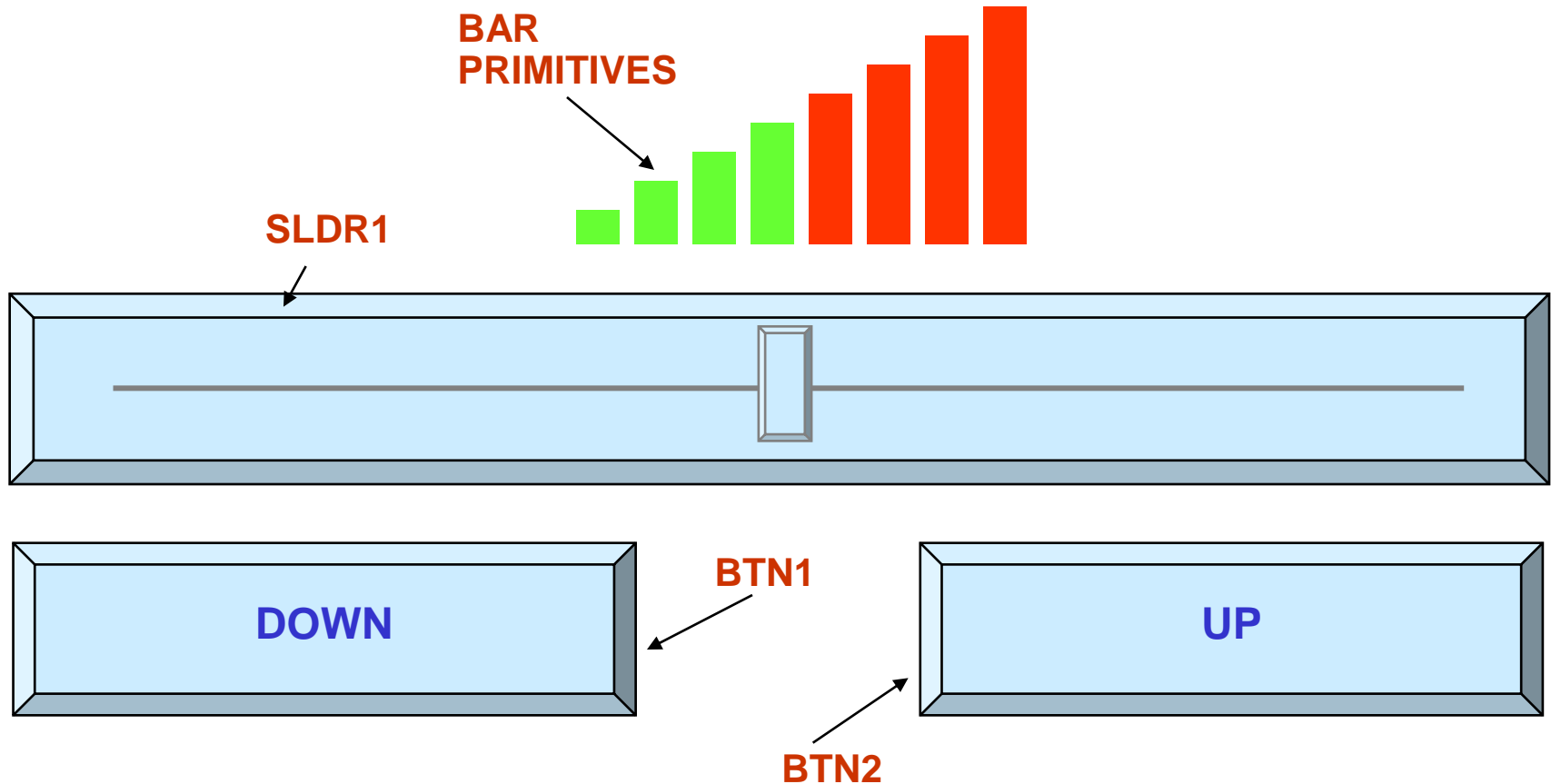


* May skip this step if only using default scheme

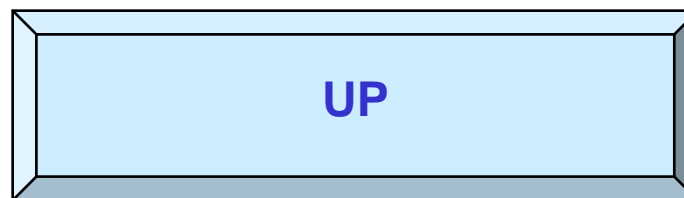
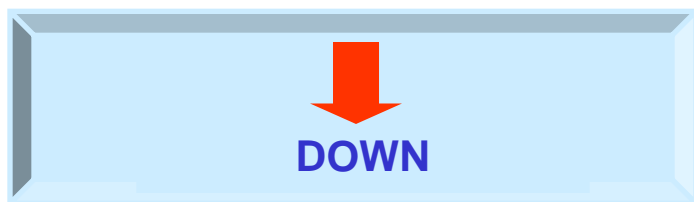
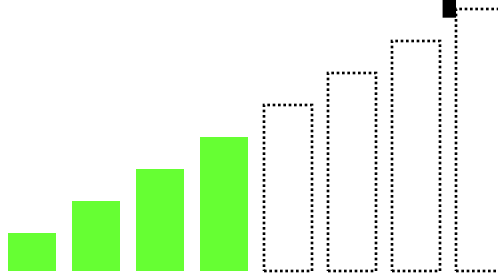
** Not library provided functions

GOLDrawCallback()

Example



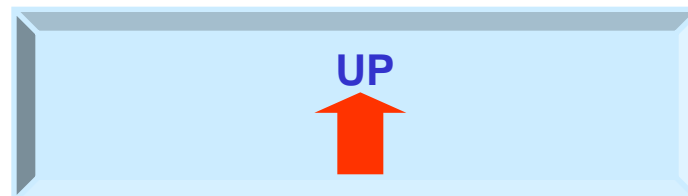
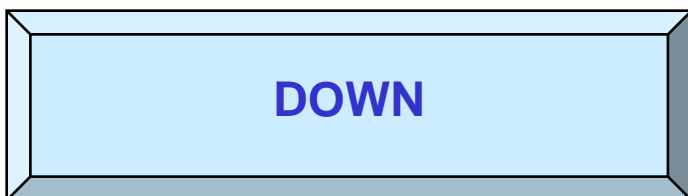
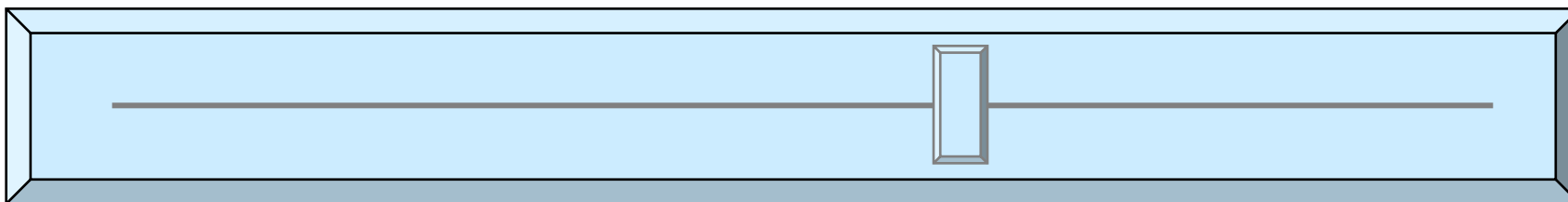
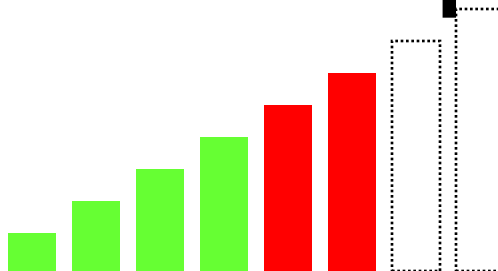
GOLDrawCallback() Example



- **Widget Actions:**
 - Move slider thumb to the left
 - Move BTN1 text down & add image
 - Erase bars
- **System Action:**
 - Decrease volume

GOLDrawCallback()

Example



- **Widget Actions:**
 - Move slider thumb to the right
 - Move BTN2 text up & add image
 - Add bars
- **System Action:**
 - Increase volume

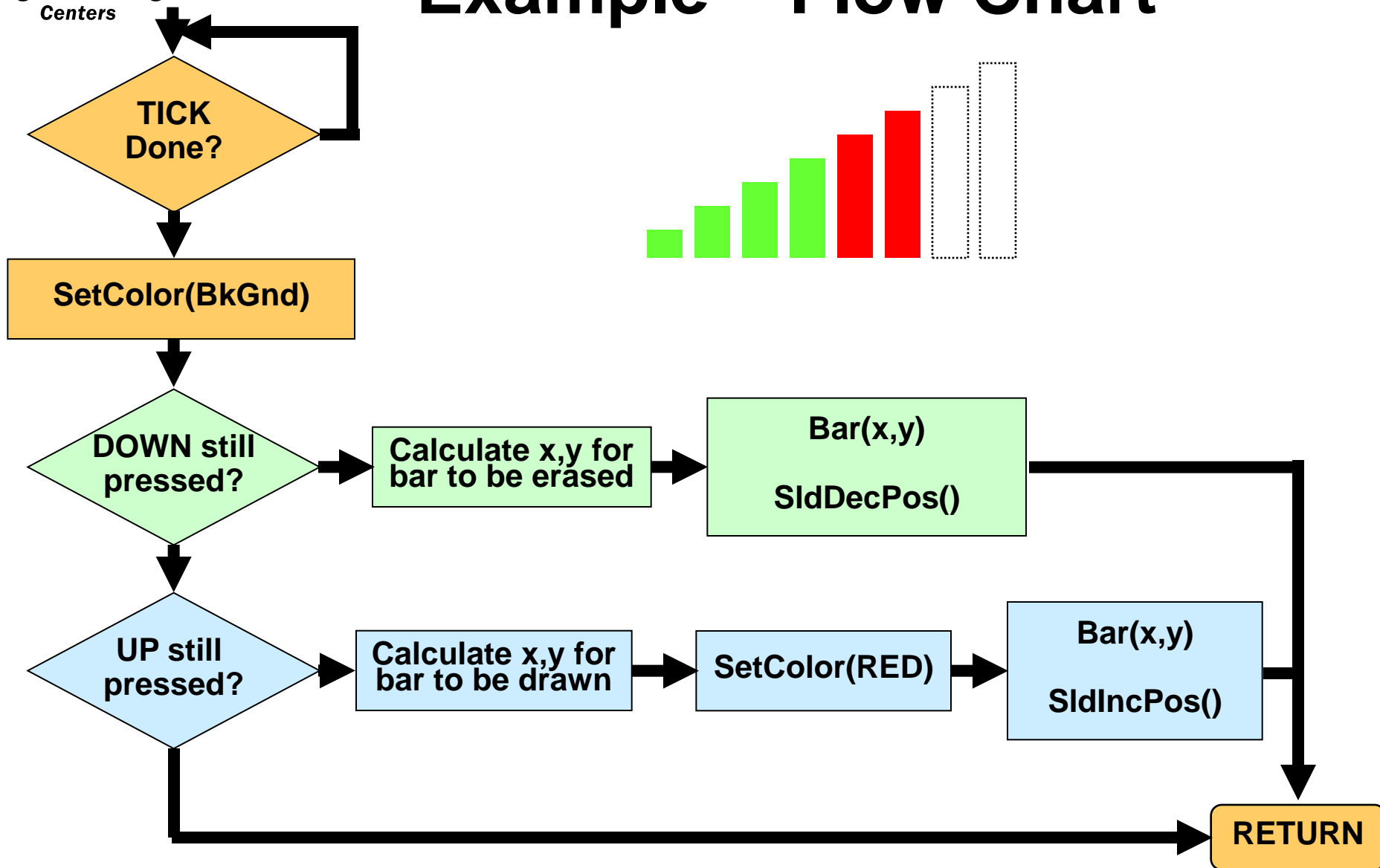
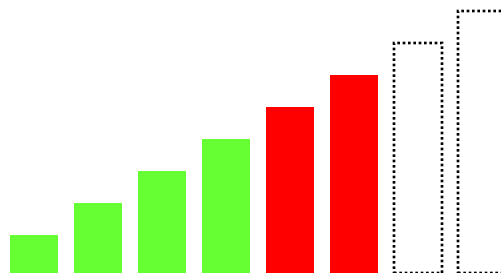


MICROCHIP

Regional Training
Centers

GOLDrawCallback()

Example – Flow Chart





MICROCHIP

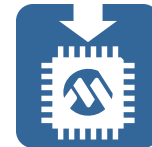
Regional Training Centers

Lab Exercise 4

User Interface Using GOLDrawCallback()

Lab Exercise 4

Potentiometer Screen



Purpose

The purpose of this lab is to learn to use GOLDDrawCallback() functions to update widget and system behavior based on continuous events. We will use the screen we created in lab exercise 3.



Procedure

Follow the directions in the lab manual starting on page 4-1

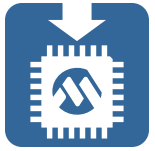
Objectives:

- Monitor the user action on a Potentiometer using a Widget.
 - Use a Meter widget to monitor a potentiometer.

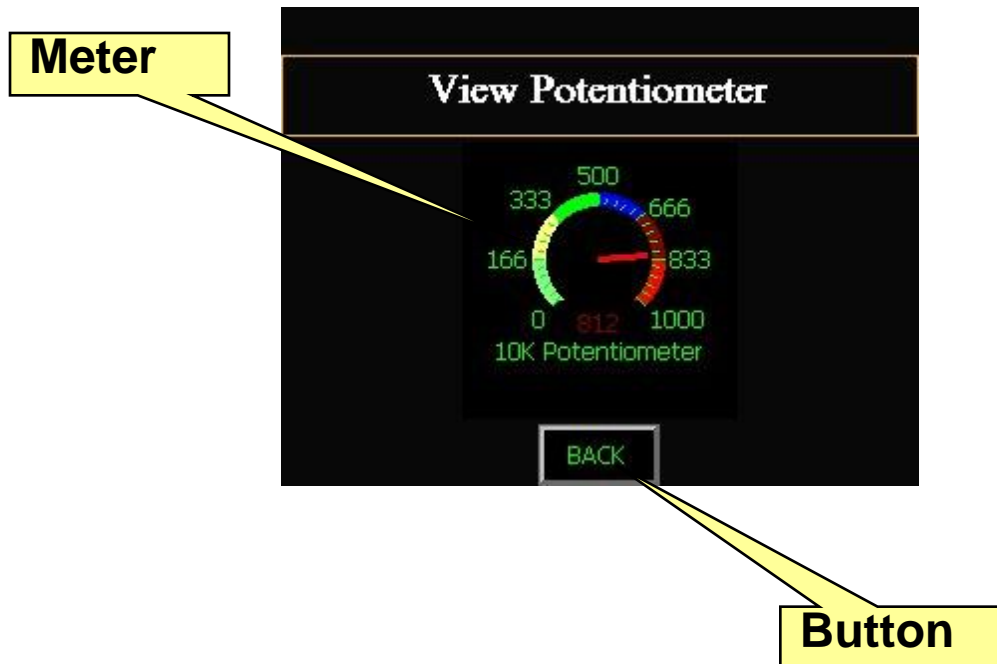


Lab Exercise 4

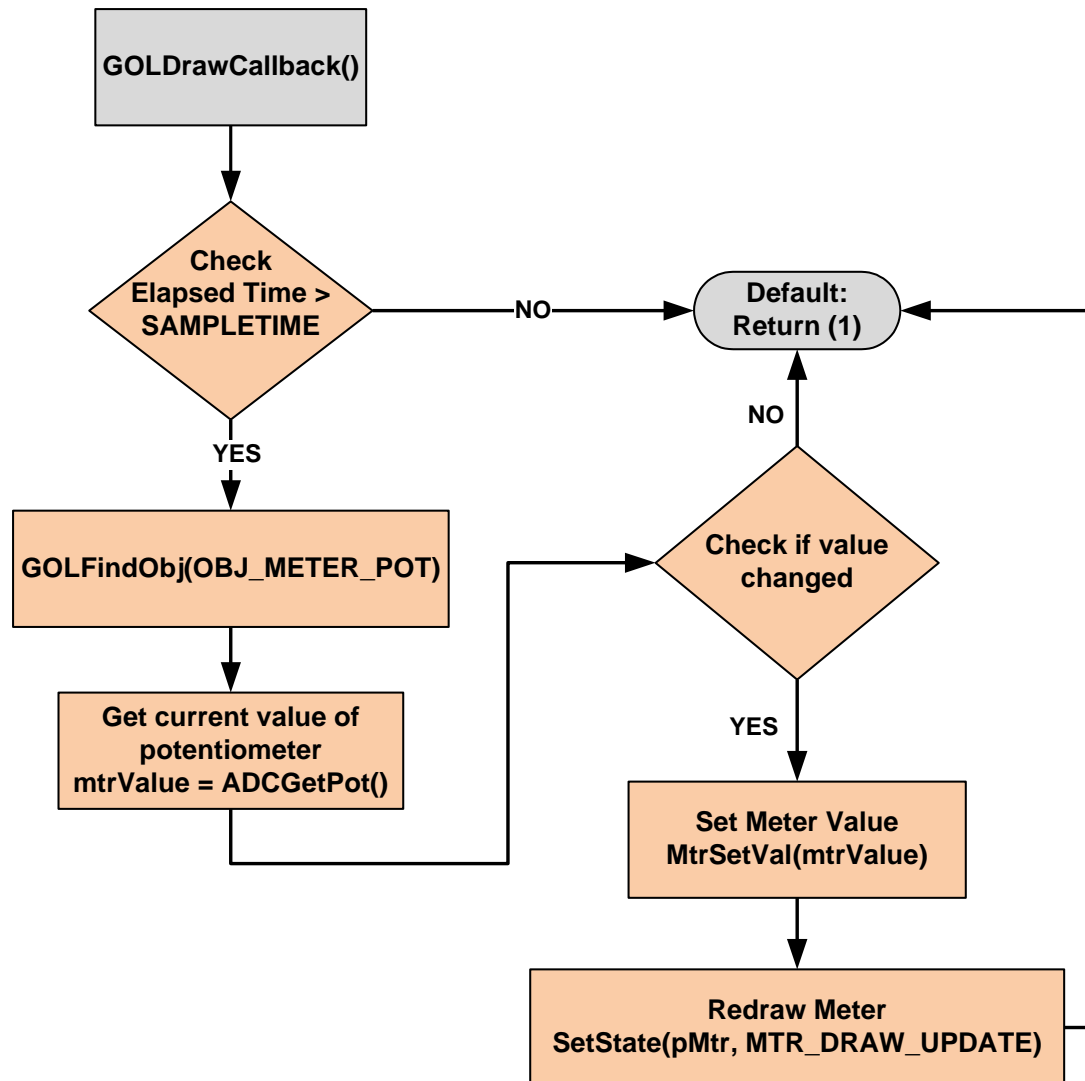
Potentiometer Screen



Expected Results

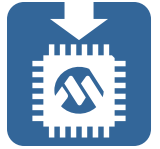


GOLDrawCallback() Flowchart



Lab Exercise 4

Implement `GOLDrawCallback()`



- **Let's examine one possible solution.**
 - **Open MPLAB....**
 - **Open `GOLDrawCallback()`**

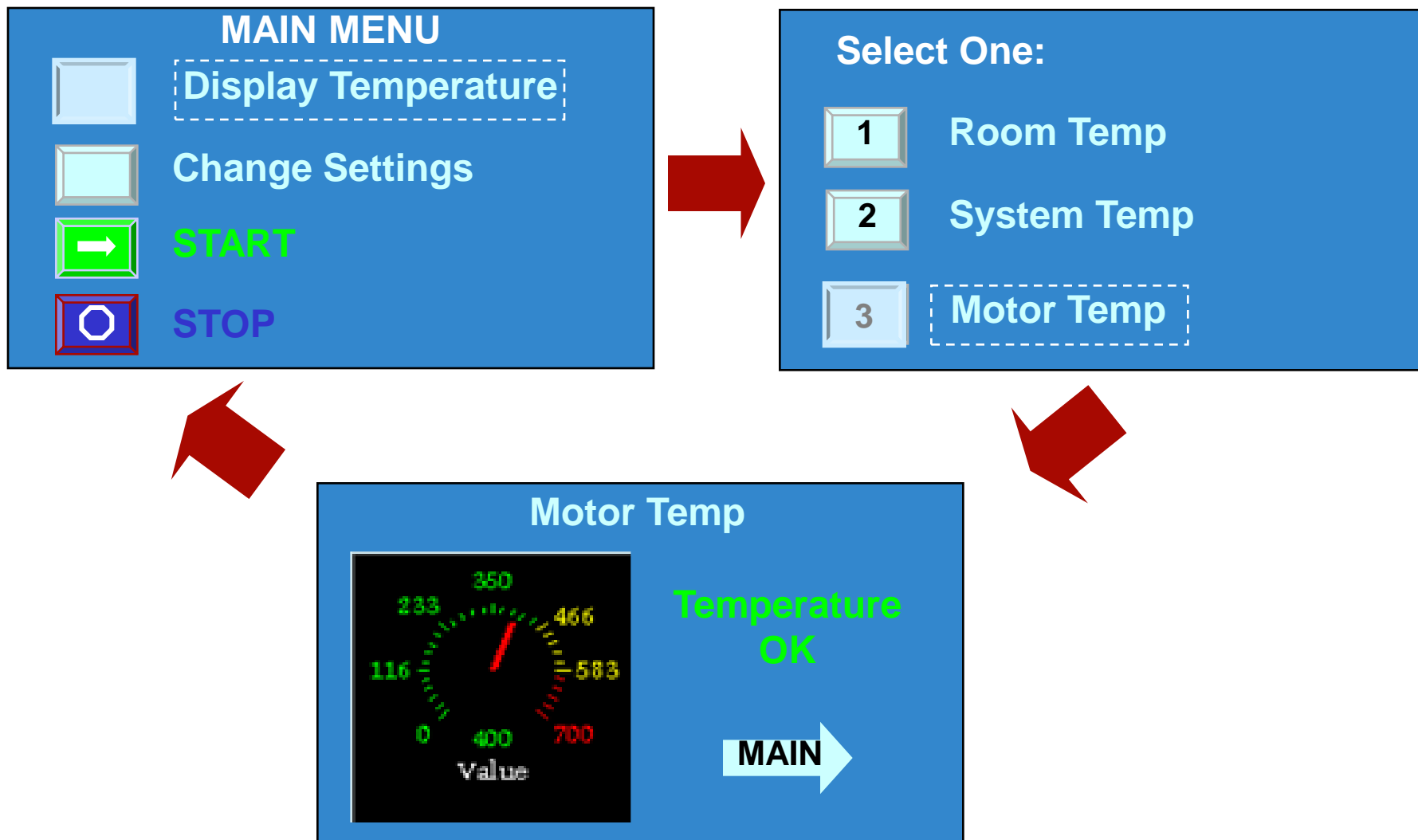


MICROCHIP

Regional Training Centers

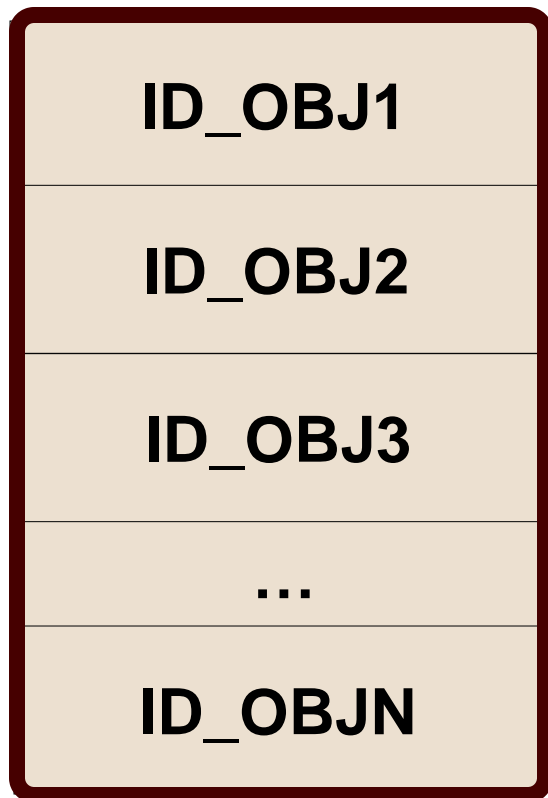
**Putting it All Together
The Full GUI**

GUI Applications



Creating Widgets

Linked List



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

Screen Management Options

Using One List

- Create every widget for every screen
- Hide old widgets and draw new
 - May increase draw time
- OR...
- Draw over old widgets
 - Disable bottom widgets
 - Only draw part of the screen
 - Better draw times

INEFFICIENT

Screen Management Options

Using Multiple Lists

- **Method 1: One list per screen**
 - `ObjCreate(, , ,)` used to form new lists
 - **Use GOL management APIs**
 - `GOLGetList()` – Save active list
 - `GOLNewList()` – Set pointer to null
 - `GOLSetList(pList)` – Switch active list
 - **Must allocate heap for every object on every screen**

Screen Management Options

Using Multiple Lists

```
//List pointer is null here
CreateMainMenu();
pMainMenu = GOLGetList();
GOLNewList(); //List pointer NULL
CreateTempSelect();
pTempSelect = GOLGetList();
GOLNewList(); //List pointer NULL
CreateMotorTemp();
pMotorTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateSystemTemp();
pSystemTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateRoomTemp();
pRoomTemp = GOLGetList();
GOLSetList(pMainMenu);
```

Heap

pMainMenu

pTempSelect

pMotorTemp

pSystemTemp

pRoomTemp



Always use GOLNewList()
to set list pointer to NULL
BEFORE creating a new
list.

Screen Management Options

Using Multiple Lists

```
//List pointer is null here
CreateMainMenu();
pMainMenu = GOLGetList();
GOLNewList(); //List pointer NULL
CreateTempSelect();
pTempSelect = GOLGetList();
GOLNewList(); //List pointer NULL
CreateMotorTemp();
pMotorTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateSystemTemp();
pSystemTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateRoomTemp();
pRoomTemp = GOLGetList();
GOLSetList(pMainMenu);
```

Heap

pMainMenu

pTempSelect

pMotorTemp

pSystemTemp

pRoomTemp



Always use GOLNewList()
to set list pointer to NULL
BEFORE creating a new
list.

Screen Management Options

Using Multiple Lists

```
//List pointer is null here
CreateMainMenu();
pMainMenu = GOLGetList();
GOLNewList(); //List pointer NULL
CreateTempSelect();
pTempSelect = GOLGetList();
GOLNewList(); //List pointer NULL
CreateMotorTemp();
pMotorTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateSystemTemp();
pSystemTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateRoomTemp();
pRoomTemp = GOLGetList();
GOLSetList(pMainMenu);
```

Heap

pMainMenu

pTempSelect

pMotorTemp

pSystemTemp

pRoomTemp



Always use GOLNewList()
to set list pointer to NULL
BEFORE creating a new
list.



Screen Management Options

Using Multiple Lists

```
//List pointer is null here
CreateMainMenu();
pMainMenu = GOLGetList();
GOLNewList(); //List pointer NULL
CreateTempSelect();
pTempSelect = GOLGetList();
GOLNewList(); //List pointer NULL
CreateMotorTemp();
pMotorTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateSystemTemp();
pSystemTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateRoomTemp();
pRoomTemp = GOLGetList();
GOLSetList(pMainMenu);
```

Heap

pMainMenu

pTempSelect

pMotorTemp

pSystemTemp

pRoomTemp



Always use GOLNewList()
to set list pointer to NULL
BEFORE creating a new
list.

Screen Management Options

Using Multiple Lists

```
//List pointer is null here
CreateMainMenu();
pMainMenu = GOLGetList();
GOLNewList(); //List pointer NULL
CreateTempSelect();
pTempSelect = GOLGetList();
GOLNewList(); //List pointer NULL
CreateMotorTemp();
pMotorTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateSystemTemp();
pSystemTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateRoomTemp();
pRoomTemp = GOLGetList();
GOLSetList(pMainMenu);
```

Heap

pMainMenu

pTempSelect

pMotorTemp

pSystemTemp

pRoomTemp



Always use GOLNewList()
to set list pointer to NULL
BEFORE creating a new
list.



Screen Management Options

Using Multiple Lists

```
//List pointer is null here
CreateMainMenu();
pMainMenu = GOLGetList();
GOLNewList(); //List pointer NULL
CreateTempSelect();
pTempSelect = GOLGetList();
GOLNewList(); //List pointer NULL
CreateMotorTemp();
pMotorTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateSystemTemp();
pSystemTemp = GOLGetList();
GOLNewList(); //List pointer NULL
CreateRoomTemp();
pRoomTemp = GOLGetList();
GOLSetList(pMainMenu);
```

Heap

pMainMenu

pTempSelect

pMotorTemp

pSystemTemp

pRoomTemp



Always use GOLNewList()
to set list pointer to NULL
BEFORE creating a new
list.



MICROCHIP

Regional Training
Centers

One List / Screen

Heap Requirements

Motor Temp



Temperature
OK

MAIN

MotorTemp

1 Button	(1 * 28 = 28 bytes)
2 Static Text	(2 * 22 = 44 bytes)
1 Meter	(1 * 40 = 40 bytes)
Total	112 bytes

System Temp



Temperature
OK

MAIN

SystemTemp

1 Button	(1 * 28 = 28 bytes)
2 Static Text	(2 * 22 = 44 bytes)
1 Meter	(1 * 40 = 40 bytes)
Total	112 bytes

Room Temp



Temperature
OK

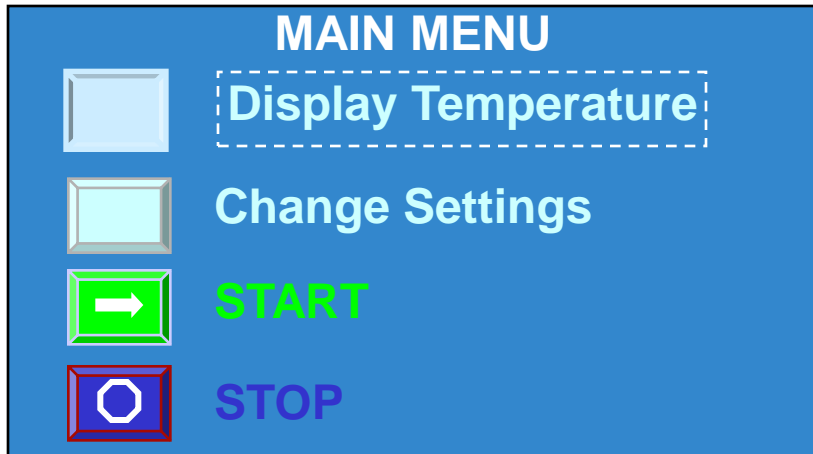
MAIN

RoomTemp

1 Button	(1 * 28 = 28 bytes)
2 Static Text	(2 * 22 = 44 bytes)
1 Meter	(1 * 40 = 40 bytes)
Total	112 bytes

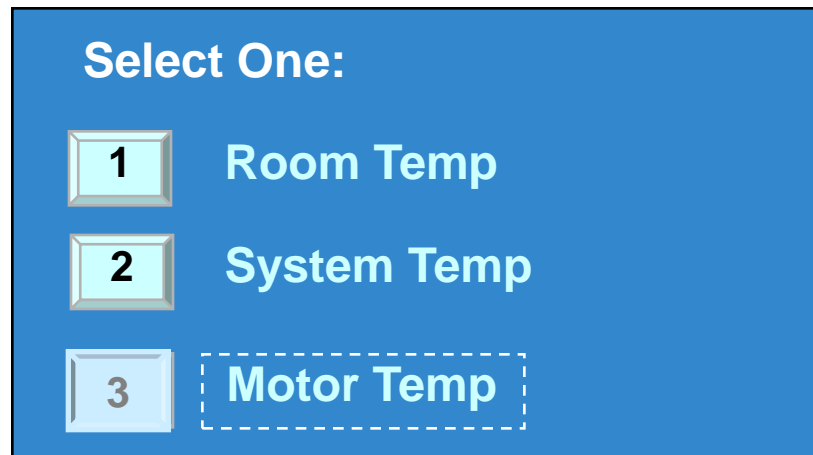
One List / Screen

Heap Requirements



MainMenu

4 Buttons	(4 * 28 = 112 bytes)
5 Static Text	(5 * 22 = 110 bytes)
Total	222 bytes



TempSelect

3 Buttons	(3 * 28 = 84 bytes)
4 Static Text	(4 * 22 = 88 bytes)
Total	172 bytes

Total Heap Required...

5 screens	730 bytes
3 style schemes	60 bytes
Total	790 bytes

Screen Management Options

Using Multiple Lists

- **Method 2: Create Lists on the Fly**
 - Use `GOLFree()` to delete active list
 - Frees heap library uses
 - Does not affect application heap
 - Use `ObjCreate(,,)` to form new list
 - Allocate heap for largest list
 - Example GUI: 282 bytes (including styles)
 - Most memory efficient method

Example

Creating lists “On the Fly”

```
// Code inside a callback function
...
// Determine if screen change button pressed
if(objMsg == BTN_MSG_PRESSED){

    // Delete the active list
    GOLFree();           // library function

    // Create a new list
    CreateMyNewScreen(); // user provided function
    ...
    // do useful things
}
```



MICROCHIP

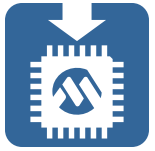
Regional Training Centers

Lab Exercise 5

Putting it all Together

Lab Exercise 5

GUI Application



Purpose

The purpose of this lab is to examine the dynamic switching of screen in detail using only one linked list of widgets.



Procedure

Follow the directions in the lab manual starting on page 5-1

Objectives:

- **Switch from one screen to another:**
 - Dynamically creating one list per screen

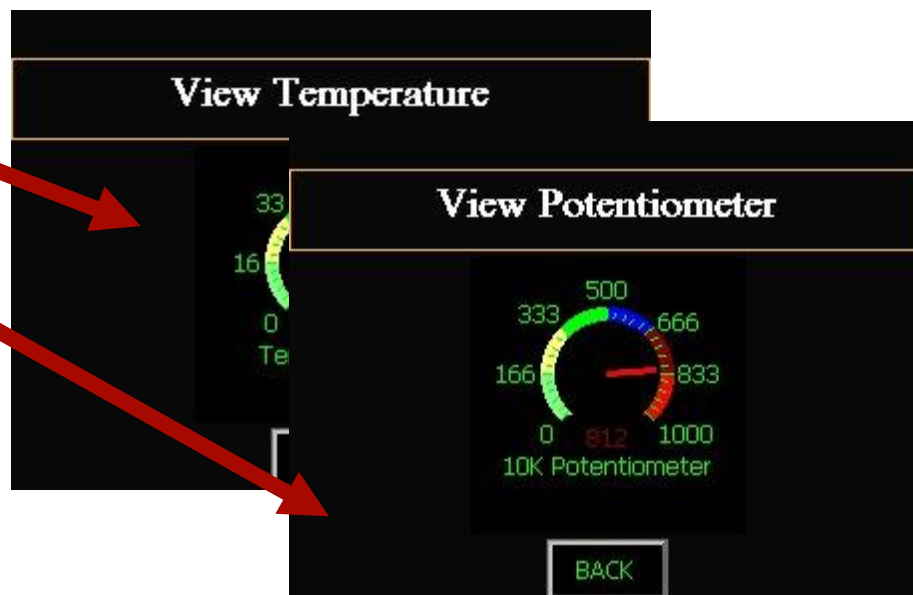
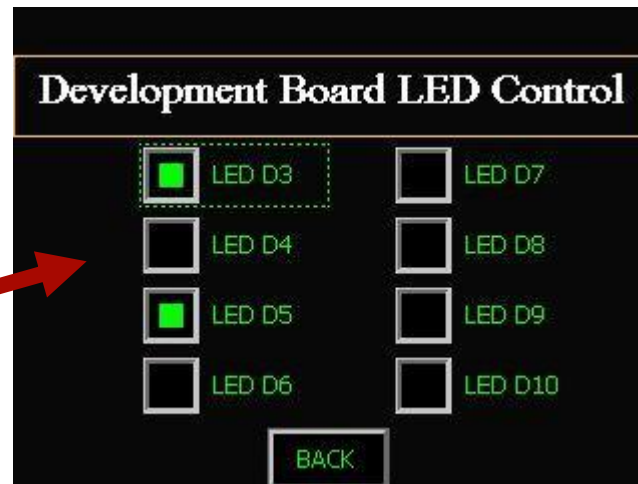


Lab Exercise 5

GUI Application

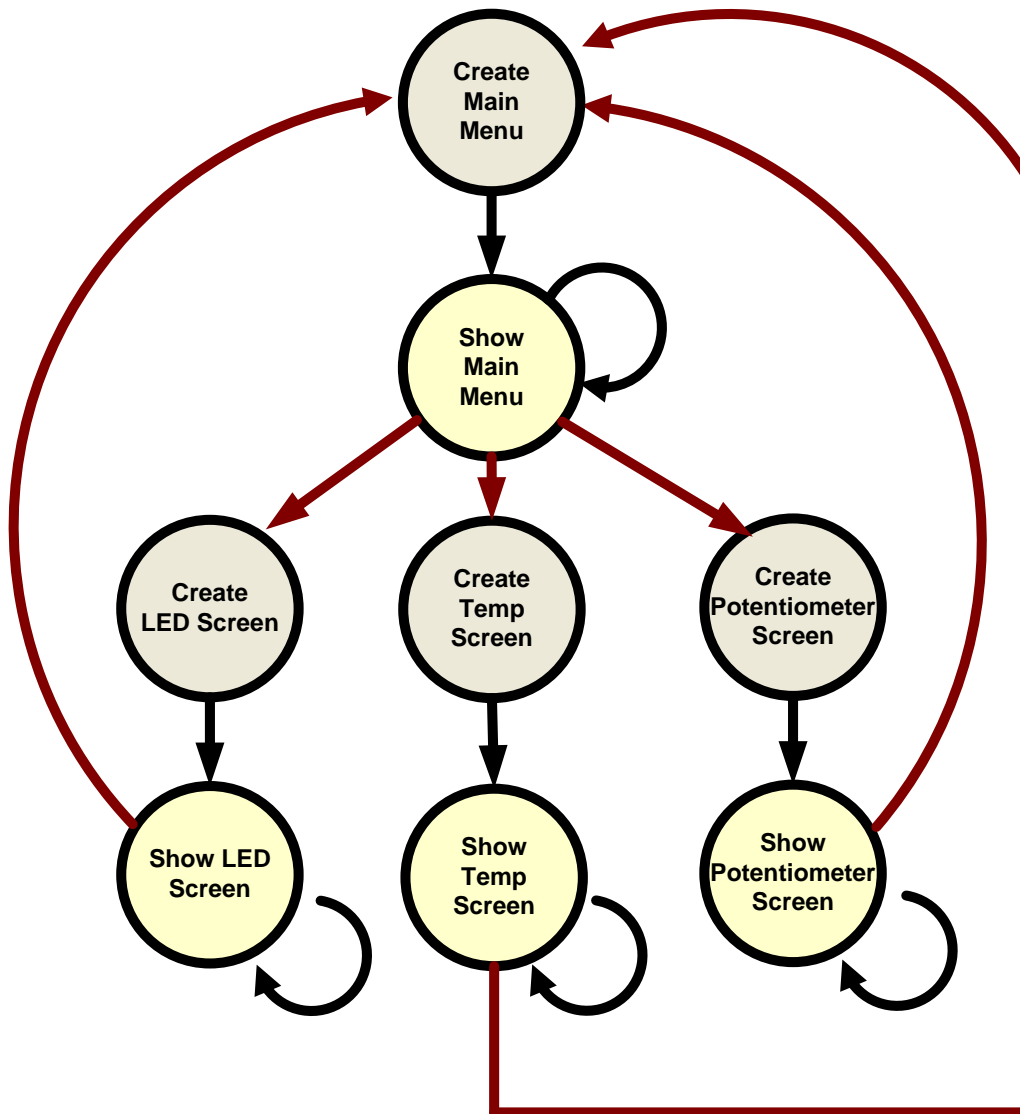


Expected Results



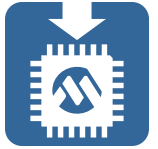


Lab 5 State Diagram



Lab Exercise 5

One List per Screen



- **Let's examine one possible solution.**
 - **Open MPLAB....**
 - **Open CreateLEDScreen ()**
 - **Open CreatePotentiometerScreen()**
 - **Open CreateTemperatureScreen()**



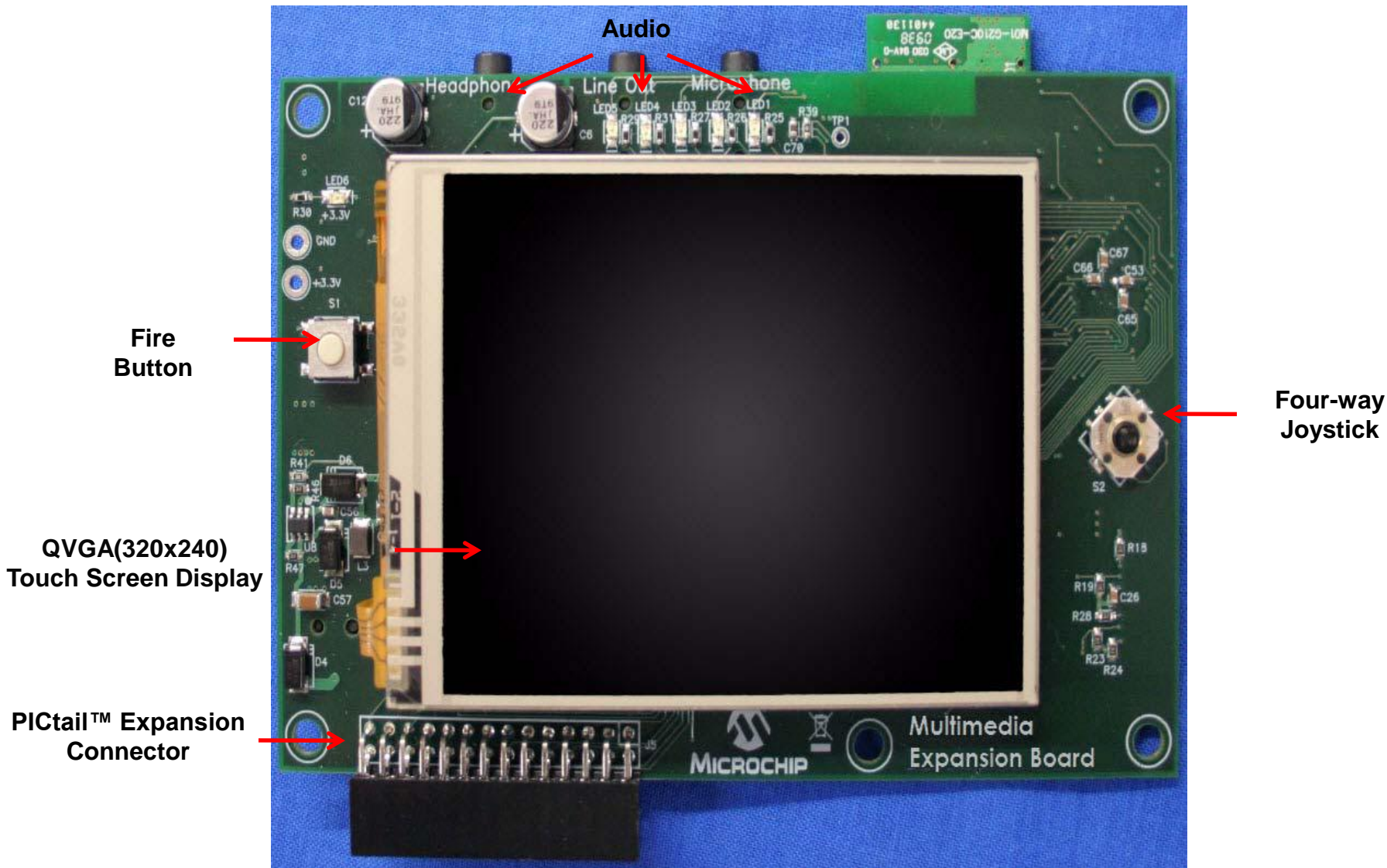
MICROCHIP

Regional Training Centers

**High Impact Graphics Using
Microchip's New Multimedia
Expansion Board with the PIC32**

Features of Microchip's Multimedia Development Board

MDB Front side

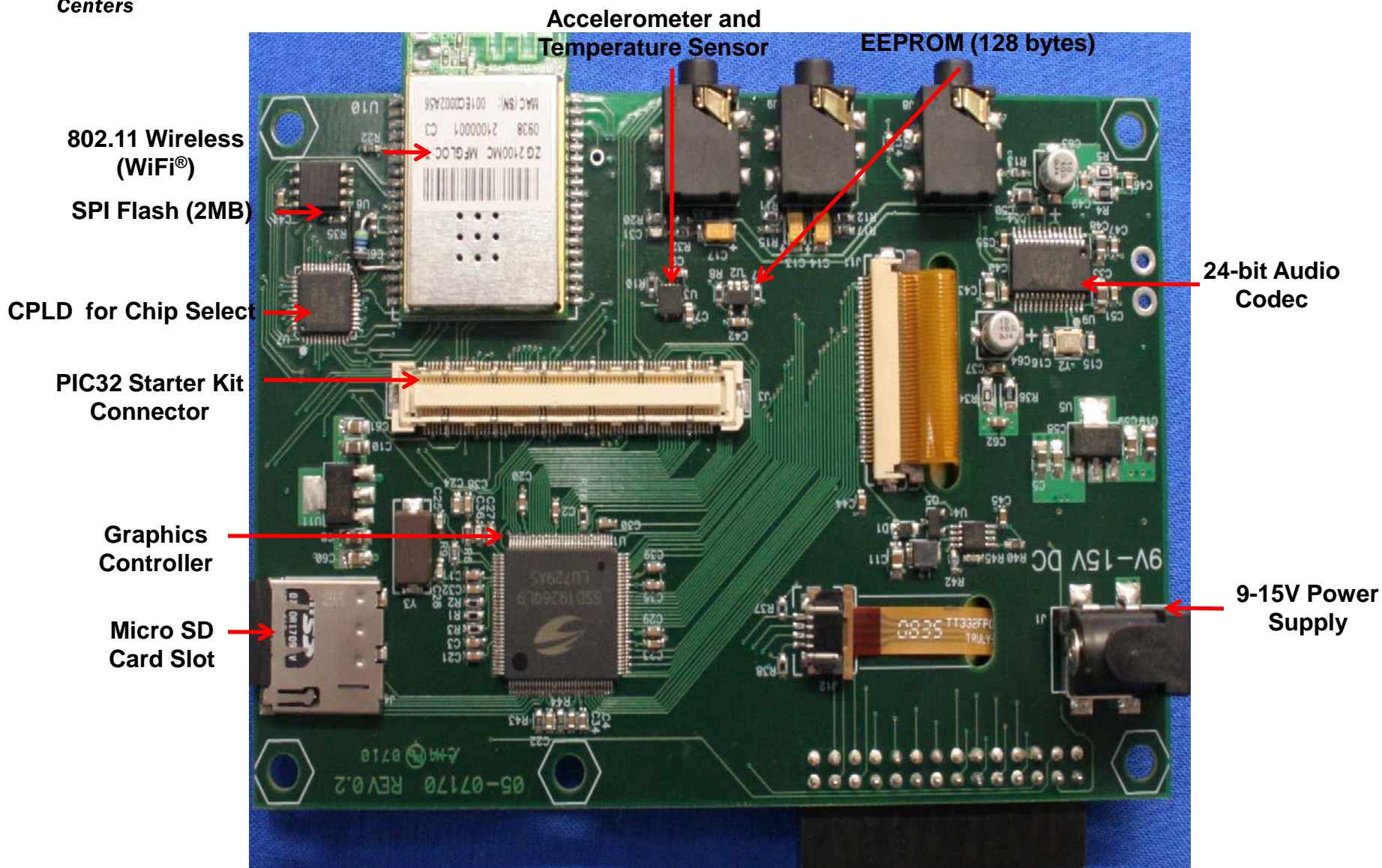




MICROCHIP

Regional Training
Centers

MDB Back Side



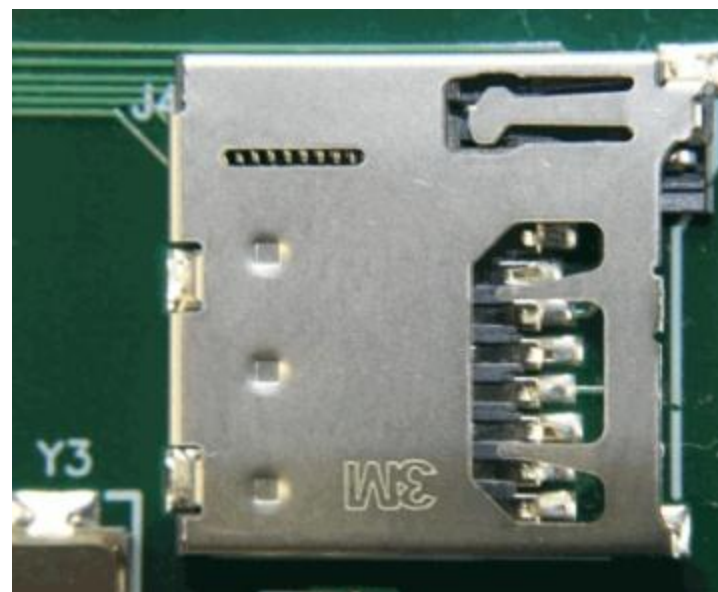
Display and Display Controller

- **Display**
 - 3.2 inch QVGA
 - 320x240 color display
 - 4-wire Touch Screen
- **Display Controller**
 - SSD1926
 - 8 or 16-bit interface (PMP)
 - JPEG Engine
 - 2D Acceleration



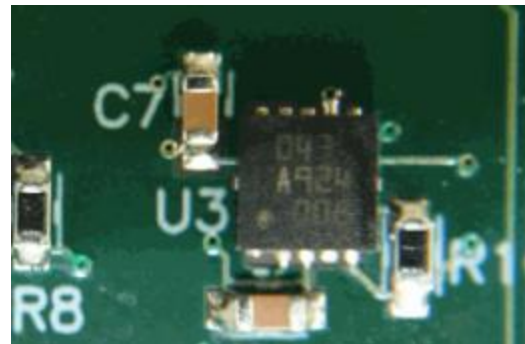
SD Card Slot

- **Micro SD**
- **4-wire interface**
- **Controlled by
SSD1926**



Accelerometer and Temperature Sensor

- **BMA150**
- **I²C™ interface**
- **3-axis acceleration**
- **+/- 2g/4g/8g**
- **-30 to 98 degrees C at 0.5 degree C resolution**



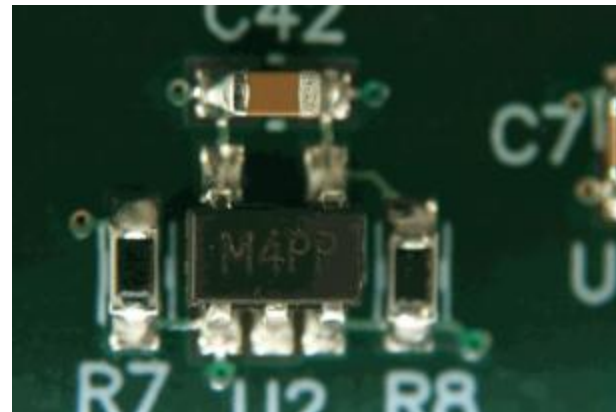
Memory: EEPROM and NOR Flash

■ EEPROM

- 24LC08
- I²C™ interface
- 128 bytes of storage

■ NOR Flash

- SST25VF016
- SPI interface
- 2M Bytes of storage



WiFi®

- **IEEE std 802.11 compliant RF Transceiver**
- **Speed up to 2 Mbps**
- **Integrated PCB Antenna**
- **Range up to 400m**



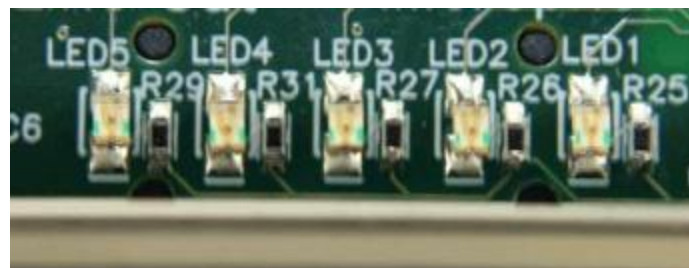
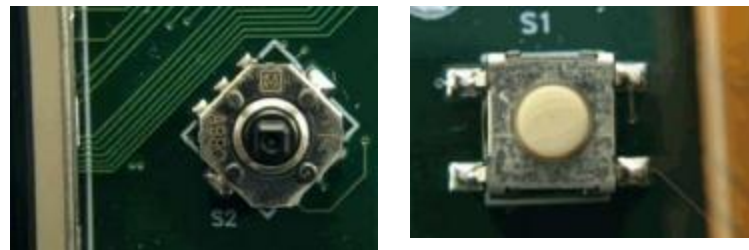
Audio

- **WM8731**
- **24-bit Audio**
- **Headphone and line out outputs**
- **Microphone input**



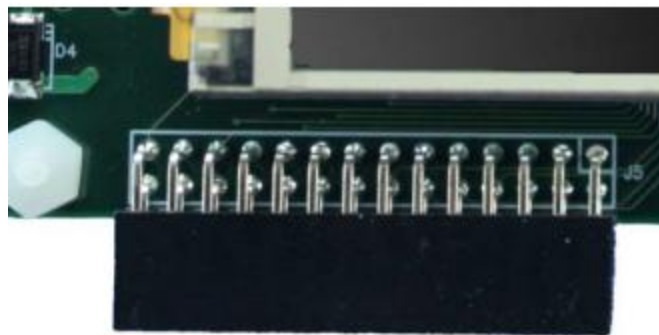
Joystick and LEDs

- 4 direction joystick with fire button
- Inputs on change notice pins
- 5 application controlled LEDs



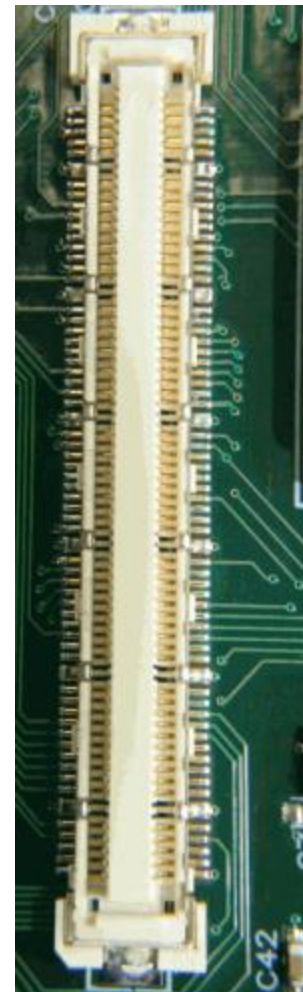
Expansion Slot

- Expansion slot supports some PICtail™ cards
- Developer can design their own cards



Starter Kit Connector

- **Connector found on expandable starter kits**
- **Compatible with PIC32 starter kits**



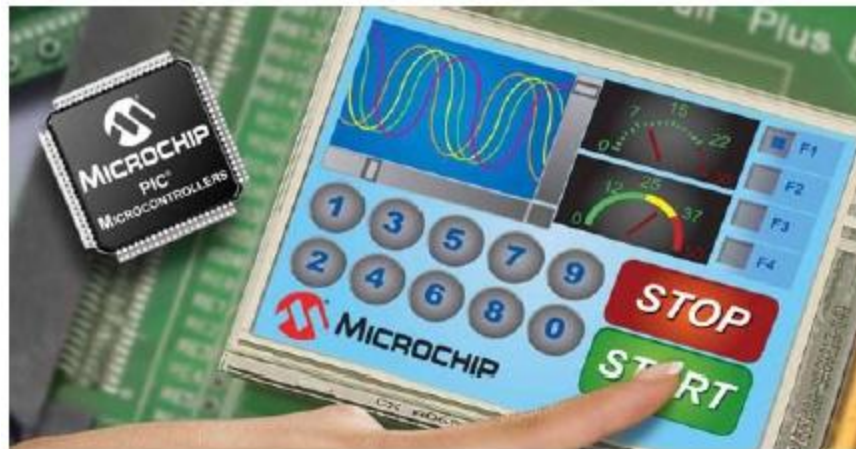
CPLD

- **Complex Programmable Logic Device**
- **Configure display controller bus width**
- **Configure SPI device**
 - SPI Flash, 802.11 or expansion slot
- **Configure SPI channel**
 - SPI2/2A or SPI3A





Integrating Graphics using Microchip's Graphics Solution





Graphics Solution

- **Availability**
- **Where**
- **Graphics library**
- **Integrating into an application**

Availability

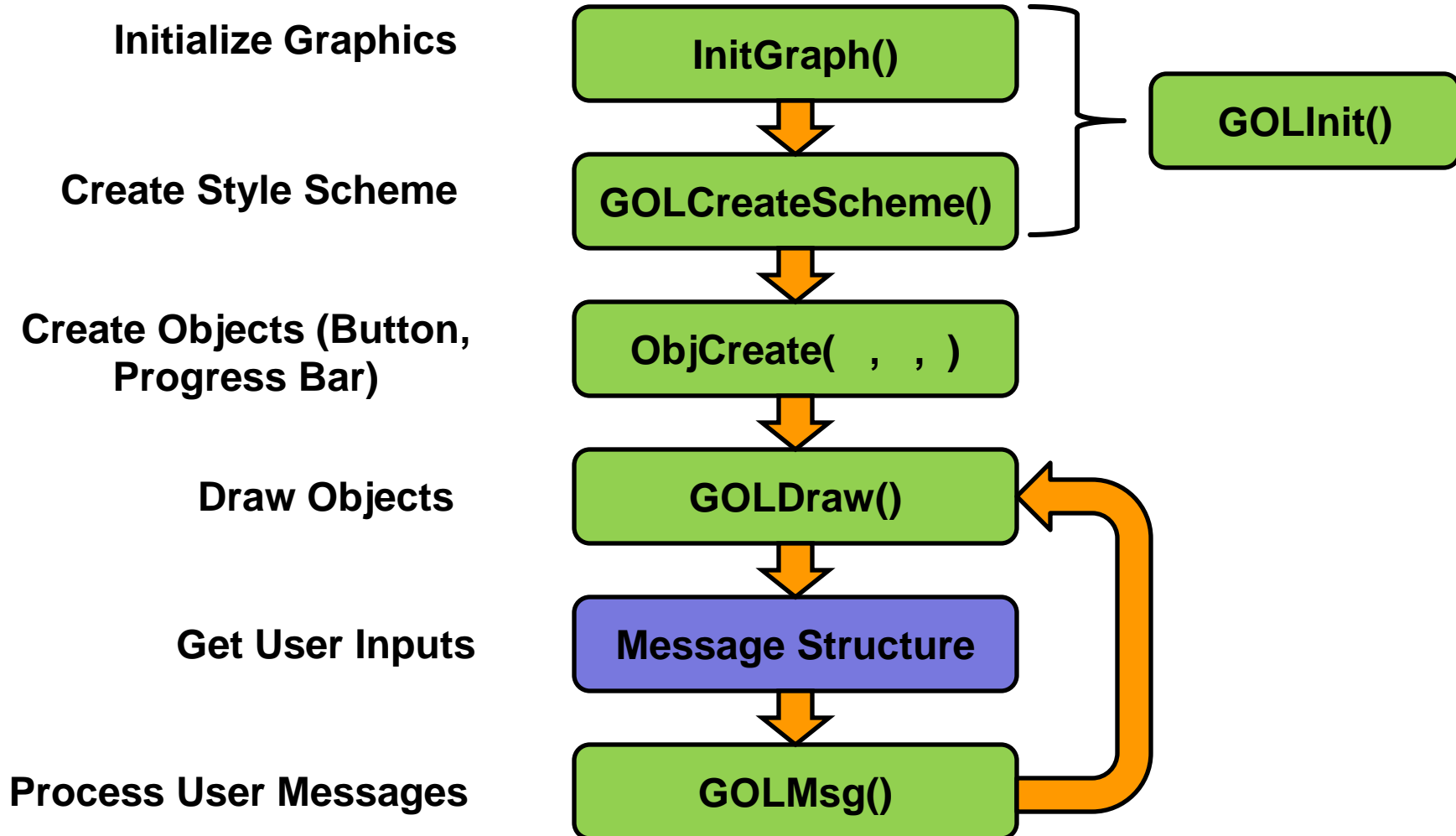
- **Support starting version 2.10 (5/2010)**
- **Twelve demos**
 - **8 or 16-bit PMP data bus**
 - **PIC32 starter kit demo**

Where

- Downloadable with the Microchip Application Libraries (MAL)
 - [www.microchip.com\MAL](http://www.microchip.com/MAL)

Microchip Applications Library	
Downloads	Release Date
Microchip Applications Library	

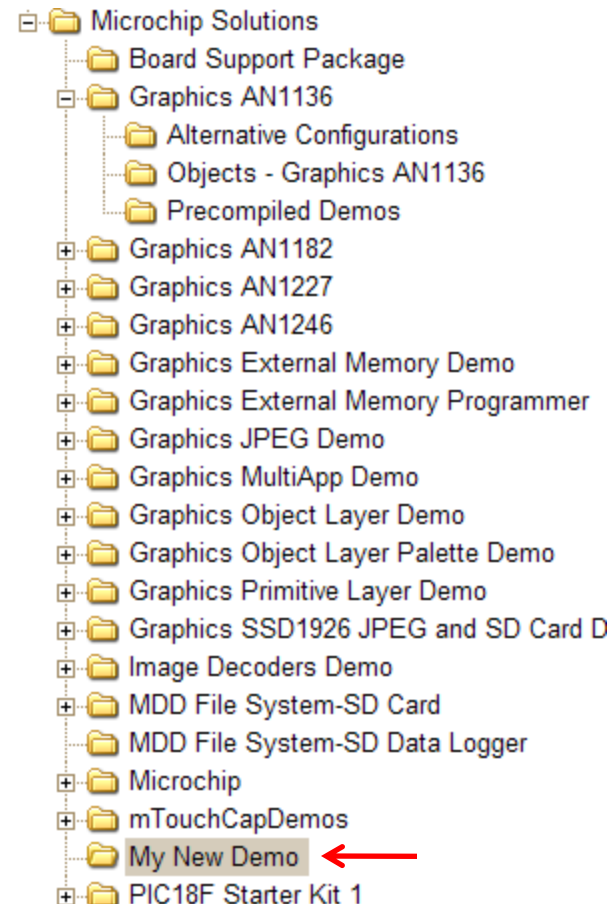
Graphics Library Usage Flow





Integrating

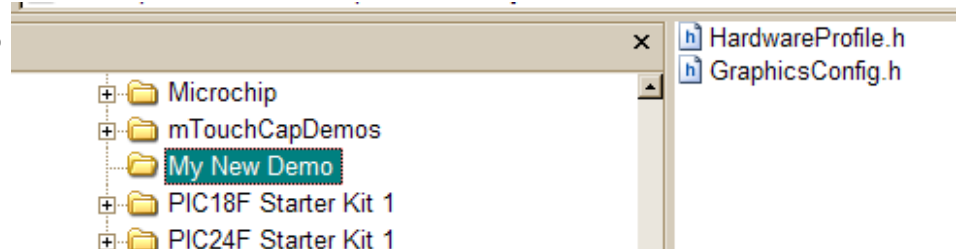
- **Download the current release of the MAL**
- **Create a folder for your demo application**
- **Copy a hardware profile from one of the supported graphics demos**





Integrating

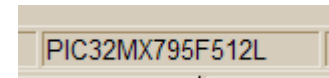
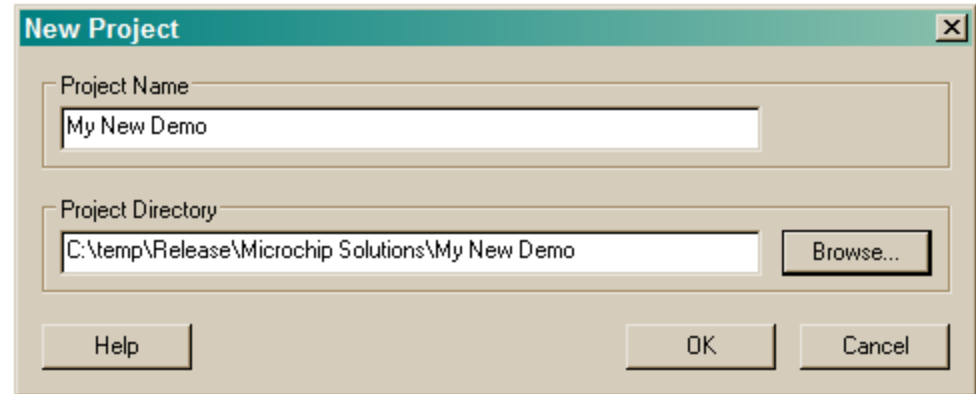
- **Copy a hardware profile from one of the supported graphics demos**
 - Rename the file to HardwareProfile.h
- **Copy from an existing demo or create a Graphics Configuration file**
 - GraphicsConfig



Integrating: MPLAB® IDE Project

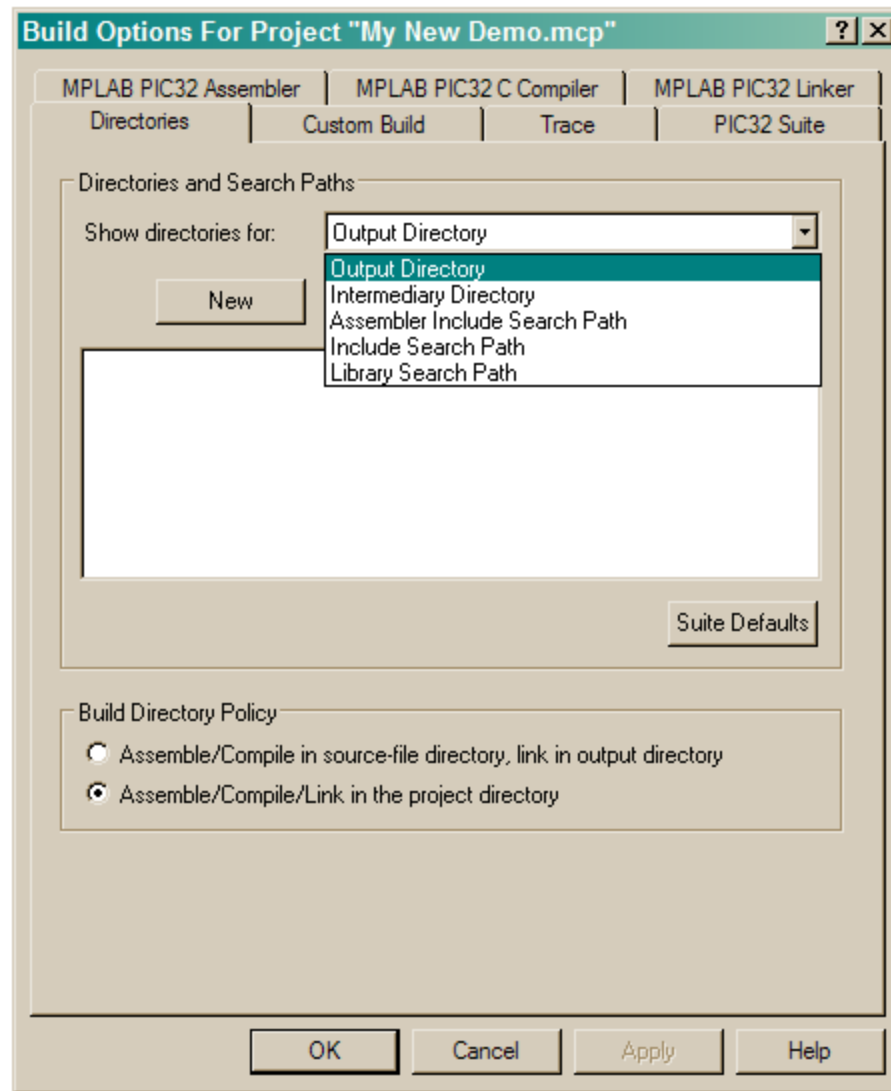
- **Create a new project**
 - **Use the newly created folder**

- **Select the correct device**



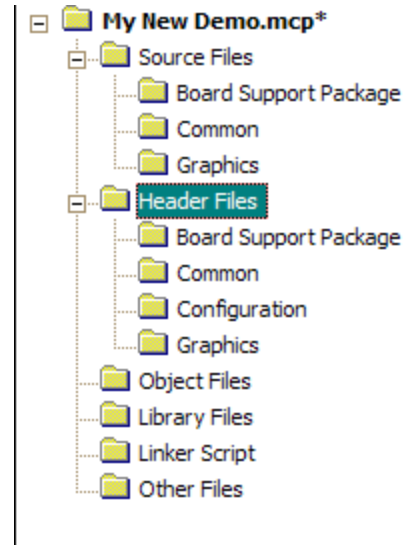
Integrating: Project Build Options

- **Set the Output Directory**
 - .\release
- **Set the Intermediary Directory**
 - .\release
- **Set the Include Search Path**
 - .
 - ..\Board Support Package
 - ..\Microchip\Include



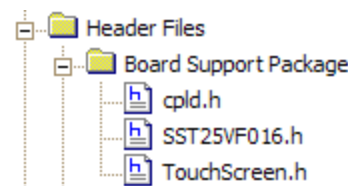
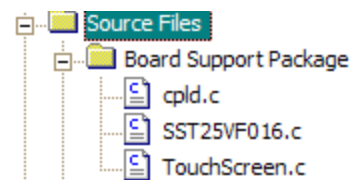
Integrating: Creating Project Sub Folders

- **Source sub-folders**
 - Board Support Package
 - Graphics
 - Common
- **Header sub-folders**
 - Board Support Package
 - Graphics
 - Common
 - Configuration (optional)



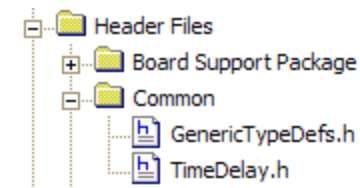
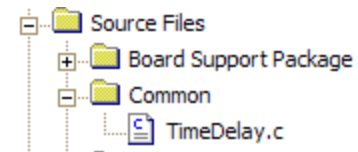
Integrating: Adding Source and Header Files

- **Board Support Package**
 - **CPLD**
 - **Touch screen (optional)**
 - **SST25VF016 (optional, used by touch screen to store calibration data)**



Integrating: Adding Source and Header Files

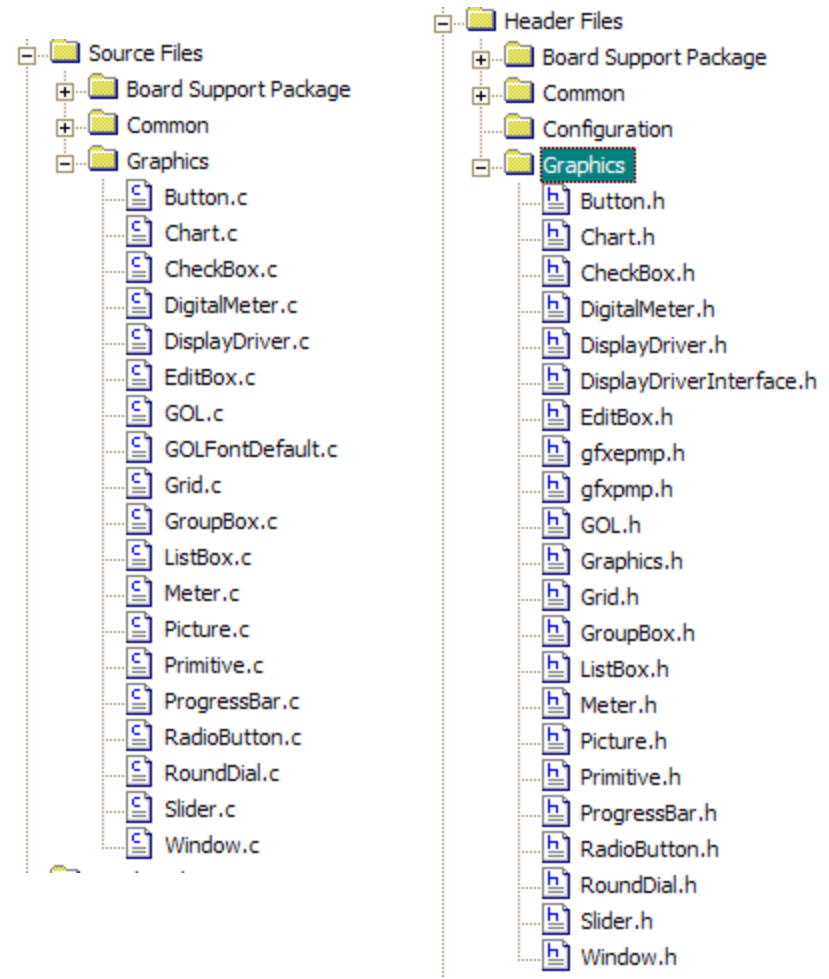
- **Common**
 - **TimeDelay**
 - **GenericTypeDefs (header file only)**



Integrating: Adding Source and Header Files

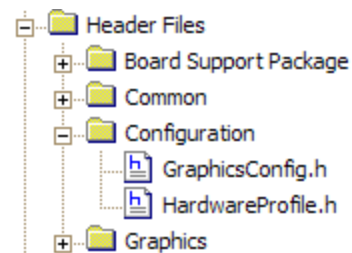
■ Graphics

- DisplayDriver
- Primitive
- Widgets (optional)
 - GOL
 - GOLFontDefault
 - EditBox
 - ...



Integrating: Adding Header Files

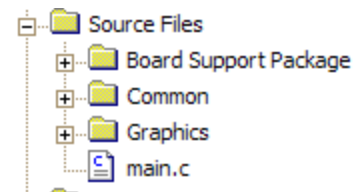
- **Configuration
(optional)**
 - **HardwareProfile**
 - **GraphicsConfig**



Integrating: Application Files

- **Add any other
source and header
files**

- **main**



Integration (Application)

- **The main source file will contain**
 - **Configuration bit settings**
 - **Initialization**
 - Hardware (board specific)
 - Initialization of the Graphics stack
 - Any other initialization
 - **Task calls**
 - Graphics
 - Any other application tasks

Integration: Application Configuration Bits

```
// Configuration bits
// Configure for 80MHz using a 8MHz input
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2
// Use a 80MHz peripheral bus
#pragma config FPBDIV = DIV_1
// No watchdog, Clock switching and Monitoring enabled
#pragma config FWDTEN = OFF, FCKSM = CSECME
// CLKO Output Signal Active, Primary Oscillatory XT, Secondary Oscillator ON, Primary PLL
#pragma config OSCIOFNC = ON, POSCMOD = XT, FSOSCEN = ON, FNOSC = PRIPLL
// Code Protect OFF, Boot write Protect OFF
#pragma config CP = OFF, BWP = OFF, PWP = OFF
```


Integration: Application Initialization

```
GOL_MSG msg;                                // GOL message structure to interact with GOL

INTEnableSystemMultiVectoredInt();
SYSTEMConfigPerformance(GetSystemClock());

CPLDInitialize();
CPLDSetGraphicsConfiguration(GRAPHICS_HW_CONFIG);
CPLDSetSPIFlashConfiguration(SPI_FLASH_CHANNEL);

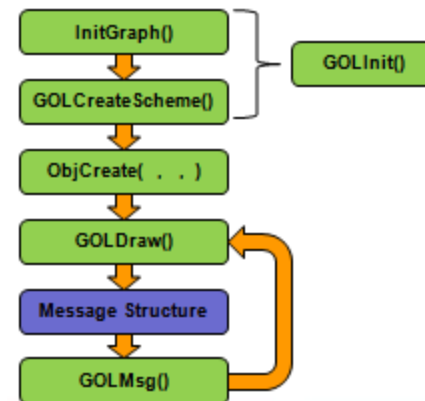
GOLInit();                                  // initialize graphics library &

SST25Init();                                // initialize GFX3 SST25 flash SPI

TouchInit();                                // initialize touch screen

                                           // create default style scheme for GOL
TickInit();                                 // initialize tick counter (for random number generation)
```

Integration: Application Task Calls



```
while(1)
{
    if(GOLDraw())
    {
        TouchGetMsg (&msg);    // Draw GOL object
        GOLMsg (&msg);        // Get message from touch screen
                                // Process message
    }
}
```

Integration: Graphics Stack Callbacks

```
// Function: WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg)
// Input: objMsg - translated message for the object,
//        pObj   - pointer to the object,
//        pMsg   - pointer to the non-translated, raw GOL message
// Output: if the function returns non-zero the message will be processed by default
// Overview: it's a user defined function. GOLMsg() function calls it each
//          time the valid message for the object received
```

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER *pObj, GOL_MSG *pMsg)
{
    return 1;
}
```

```
// Function: WORD GOLDrawCallback()
// Output: if the function returns non-zero the draw control will be passed to GOL
// Overview: it's a user defined function. GOLDraw() function calls it each
//          time when GOL objects drawing is completed. User drawing should be done here.
//          GOL will not change color, line type and clipping region settings while
//          this function returns zero.
```

```
WORD GOLDrawCallback(void)
{
    return 1;
}
```



Integration

- **Project should be ready to integrate graphics into your own application using the Multimedia Expansion Board and PIC32 Starter Kit**

Lab Exercise 6: Graphics



Lab Exercise 6: Graphics

■ Objectives:

- Create 2 Buttons and 1 Progress bar
- Control the progress bar using the button click

■ Description:

- **Creating the graphics objects:**
 - Include graphics library
 - Enable the Objects in GraphicsConfig.h file
 - Initialize and Create the graphics Objects
- **Associate the Touch Screen:**
 - Do the button click action in callback functions

Lab Exercise 6: Instructions

1. **Open the existing graphics demo project framework**
2. **Include the graphics library source code files**
3. **Enable the graphic objects in GraphicsConfig.h file**
4. **Initialize the graphics library**
5. **Start creating the objects**
 - **See the objects' APIs in the Lab Manual**



Follow directions in your Lab Manual.



Lab Exercise 6: Solution

Creating the Objects

```
pProgressBar = PbCreate
(
    ID_PROGRESS_AUDIO,          // ID
    PB_ORIGIN_X + 0,
    PB_ORIGIN_Y + 0,
    PB_ORIGIN_X + 240,
    PB_ORIGIN_Y + 40,          // dimension
    PB_DRAW,                    // will be displayed
    0,                          // position
    100,                        // range
    schemeMasters               // use of schemeMasters
);

pButtonPlay = BtnCreate
(
    ID_BUTTON_PLAY,             // object ID
    PB_ORIGIN_X + 0,
    160,
    PB_ORIGIN_X + 100,
    200,                        // object's dimension
    0,                          // radius of the object
    BTN_DRAW,                   // draw the object
    NULL,                       // no bitmap used
    (XCHAR *) "PLAY",          // use the string "PLAY"
    schemeMasters               // style of the object
);

pButtonStop = BtnCreate
(
    ID_BUTTON_STOP,             // object ID
    PB_ORIGIN_X + 140,
    160,
    PB_ORIGIN_X + 240,
    200,                        // object's dimension
    0,                          // radius of the object
```

GOL Callback Functions

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj)
{
    if(objMsg == BTN_MSG_PRESSED)
    {
        if(GetObjID(pObj) == ID_BUTTON_PLAY)
        {
            TurnLEDOn(LED_3);
            PlayPressed = TRUE;
        }
        else if(GetObjID(pObj) == ID_BUTTON_STOP)
        {
            TurnLEDOff(LED_3);
            PlayPressed = FALSE;
        }
    }
    return 1;
}

WORD GOLDrawCallback()
{
    ProgressBarAction();
    return 1;
}
```


Additional Resources

- www.microchip.com/MEB
- www.microchip.com/MAL
- www.microchip.com/PIC32
- www.microchip.com/GRAPHICS
- www.microchip.com/TCPIP
- www.microchip.com/AUDIO
- www.youtube.com/MicrochipTechnology



MICROCHIP

Regional Training Centers

Summary



Words of Wisdom

- **Avoid heap fragmentation**
 - **Use GOLFree()**
 - **Do NOT ...**
 - **Manually remove widgets from active list**
 - **Do NOT...**
 - **Modify list until `GOLDdraw()` is complete**
- **Avoid weird drawing effects**
 - **Do NOT ...**
 - **Modify drawing properties inside an ISR**
 - **Watch the list pointers**



Summary

Today you learned how to:

- **Apply tips to assist in writing low-level drivers for use with the Graphics Library**
- **Write programs to display images, fonts, and primitives on LCD panel**
- **Write programs to display and control widgets on LCD panel**
- **Create GUI application code to fully utilize Microchip Graphics Library**

Resources



MICROCHIP

Home Products Design Support Applications Buy/Sample Corporat

Graphics Displays

Overview **Solutions** Featured Products **Training & Support** Getting Started

Easy and cost-effective graphics solution

- Free Graphics Display Designer (visual design tool)
- Free Graphics Library
- Low cost and full-featured development tool
- Full documentation, application notes
- Web seminars and hands-on training

Targeted at graphical interface applications

- TFT, OLED, C-STN, Monochrome display
- Up to VGA (640x480) resolution
- Up to 16 bit per pixel (65K colors)

Industry's broadest portfolio

- New PIC24FJ DA and PIC32MX5
- 16 & 32-bit PIC® MCUs
- Up to 100 pins
- Up to 24 direct capacitive touch
- eXtreme Low Power Technology
- PIC® MCUs integrate USB and Ethernet MAC

Microcontrollers and Software supporting Graphics Applications

Development Kits, Development Boards, PICtail? Plus Daughter Boards

Training and Design Support (Application Notes, RTC classes, Webinars)















Microchip Graphics Library download link

The screenshot shows a webpage for Microchip's Graphics Displays. The navigation bar includes links for Home, Products, Design, Support, Applications, Buy/Sample, and Corporat. The main heading is 'Graphics Displays'. Below it, there are tabs for Overview, Solutions, Featured Products, Training & Support, and Getting Started. The 'Solutions' tab is selected. The content area lists several key features and benefits of the graphics solution, including a free design tool, a free graphics library, and low-cost development tools. It also mentions targeted applications like TFT, OLED, and C-STN displays. A sidebar on the right highlights the industry's broadest portfolio, listing various PIC microcontroller models and features. Callout boxes point to specific sections: 'Microcontrollers and Software supporting Graphics Applications' points to the top navigation; 'Development Kits, Development Boards, PICtail? Plus Daughter Boards' points to the 'Getting Started' tab; 'Training and Design Support (Application Notes, RTC classes, Webinars)' points to the 'Training & Support' tab; and 'Microchip Graphics Library download link' points to the 'Free Graphics Library' item in the list.

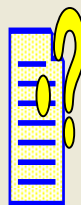
<http://www.microchip.com/graphics>

Resources

Library Help File

- ? Introduction
- ? Software License Agreement
- ? **Release Notes**
- ? Getting Started
-  Library Configuration
 - +  Configuration Setting
 - +  Input Device Selection
 - +  Focus Support Selection
 - +  Graphics Object Selection
 - +  Font Source Selection
 - +  Bitmap Source Selection
 - +  Graphics Mode
 - +  Hardware Profiles
- ? Library Structure
 - +  Graphics Object Layer
 - +  Graphics Primitive Layer
 - +  Device Driver Layer
 - +  Miscellaneous Topics
 - +  Files

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



Graphics Library Help

Tools

- **Microchip Graphics Library v2.10**
 - <http://www.microchip.com/mal>
- **MPLAB[®] IDE (v8.50)**
 - <http://www.microchip.com/mplabide>
- **MPLAB 16- and 32-bit Compilers**
 - <http://www.microchip.com/mplabc>
- **Color Schemer**
 - <http://www.colorschemer.com>



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)**

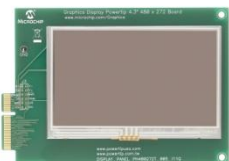


- **Graphics Display Board with 3.2" Display Kit (AC164127-3)**



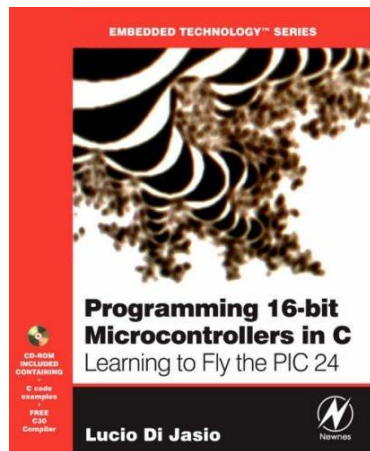
- **PIC24FJ256DA210 Development Kit (DV164039)**

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)**

Suggested Reading



Programming 16-bit Microcontrollers in C by Lucio Di Jasio

ISBN-10: 0750682922

ISBN-13: 978-0750682923

<http://www.flyingpic24.com>



Programming 32-bit Microcontrollers in C by Lucio Di Jasio

ISBN-10: 0750687096

ISBN-13: 978-0750687096



MICROCHIP

Regional Training Centers

PIC Microcontrollers for LCD & Graphics Displays

Displays in Our Daily Life

- Improve Ease of Use
- Adds Value and “Cool” Factor
- Enhances Quality and Capability



Oven Control



Gas Meter



Coffee Maker



Thermostat

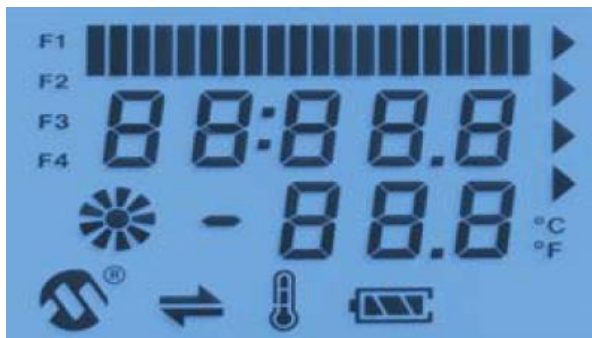


Remote Control



Industrial Control

Choosing a Display Solution



	Segmented Display	Graphical Display
Cost	Lower	Typically higher
Complexity	Easy	More software required
Appearance	Static	Dynamic
Ideal for	Simple functionality	Complex functionality made simple with a good user interface



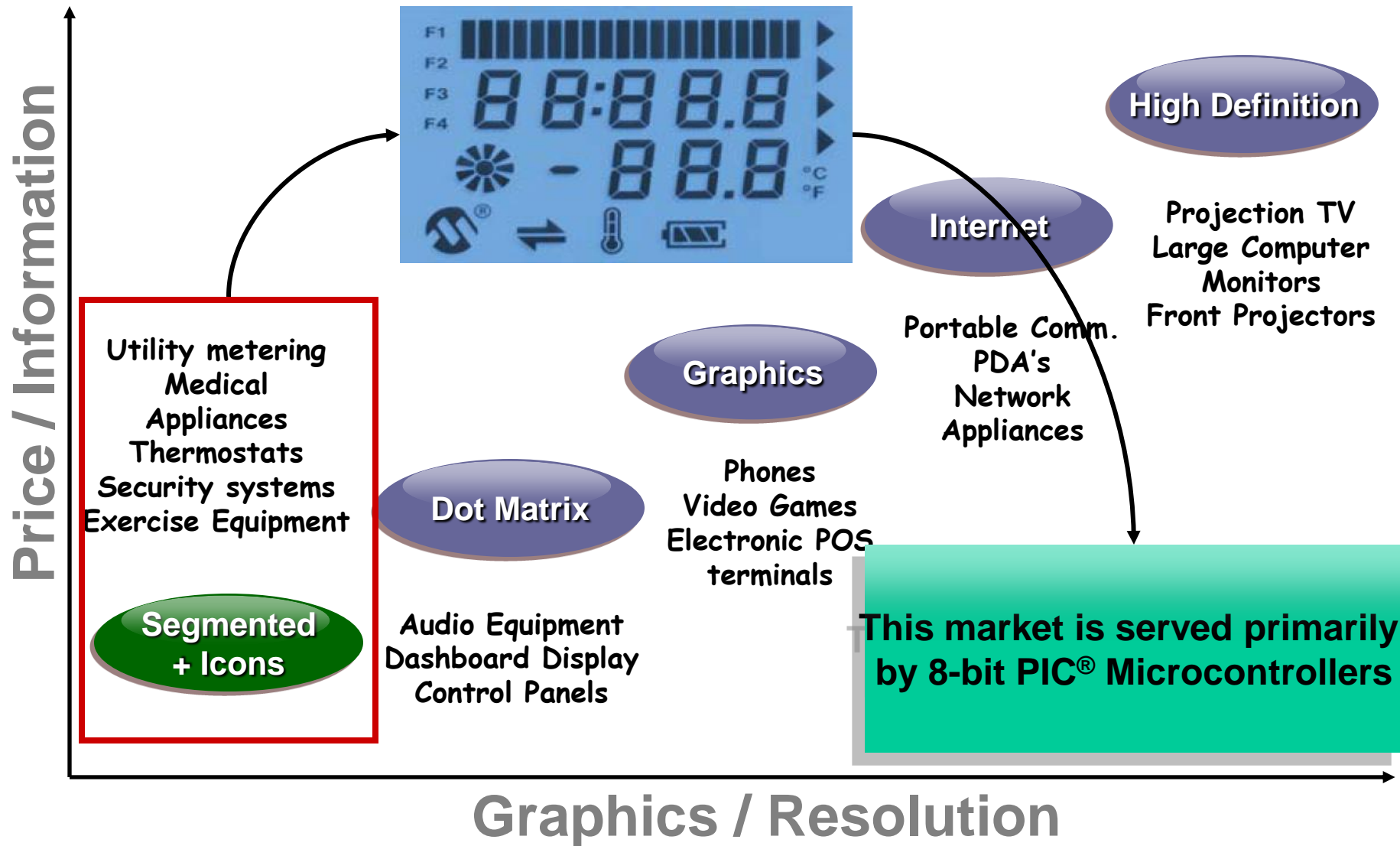
MICROCHIP

Regional Training Centers



**PIC® Microcontrollers For
Segmented LCD Control**

LCD Market Segmentation



LCD Application Examples

- **Medical**

- Pulse Oximeters
- Glucometers
- Digital Thermostats
- Portable Health Monitors



- **Consumer**

- Thermostats
- Security Systems
- In-Home Displays



- **Appliance**

- Coffee Makers
- Ranges
- Refrigerators



- **Industrial**

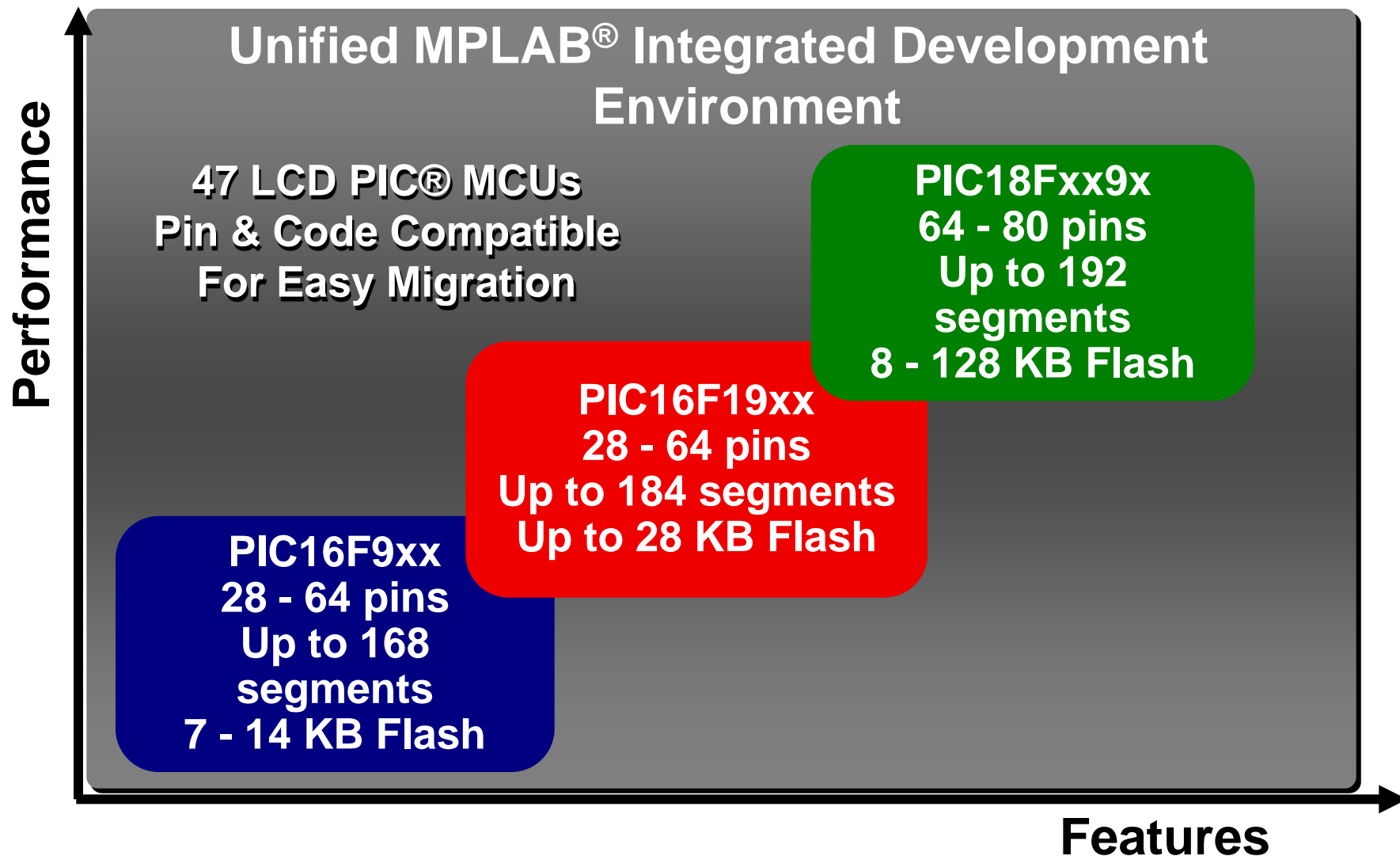
- Electronic Door Locks
- Metering
- HVAC Controls
- Scanners

- **Automotive**

- Alarm Systems
- Remote Keyless Entry
- Dashboard Instrument Clusters
- Center Stack Displays



Segmented LCD Portfolio



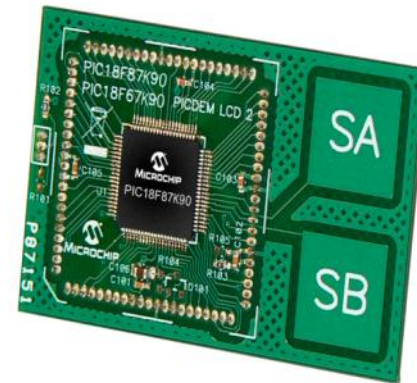


LCD2 Development Board



LCD PICDEM™ LCD2 Demo Board
(Part # DM163030)

**Single LCD Demo Board
Supports ALL LCD PIC MCUs**



***PIC18F87J90 Plug-In Module
(Part # MA180025) \$25***

Other PIMs Support All PIC16 & PIC18 LCD Families

Microchip's PIC[®] MCUs Features & Benefits

Features

Benefits

Results

Directly drive
LCD display



Eliminates separate
display driver chip



Reduced system
components/cost

Drive glass while
microcontroller
in **SLEEP** mode



Power-consumption
conservation



Constant display,
Longer battery life,
Efficient resource use

Optional **clock**
sources



Design efficiency,
Flexible speed control,
Adapt to changes



Fail-safe operation,
Longer battery life

Configurable
segment-driver
pins



Maintain desired
microcontroller features
while driving display



Design flexibility
and reliability

Voltage
waveform generation



Multiple options
for glass selection



Flexible design,
Lower cost

Multiplex
commons/backplanes



Maximize total
number of pixels
per available driver I/O



Drive sophisticated
displays, achieve
differentiation



MICROCHIP

Regional Training Centers



Graphical User Interface Example Applications

- **Consumer**
 - Thermostats
 - Cordless Phones
 - Remote Controls
- **Home Appliance**
 - Coffee Makers
 - Washing Machines
 - Ovens
- **Industrial**
 - Digital Instrument Gauges
 - Storage Controls
 - Remote Terminals
- **Portable Medical**
 - Glucometers
 - Blood-Pressure Monitors
 - Portable ECGs






Memory Requirement

Pixel Resolution / Screen Size

- QVGA = 240 x 320 (~3.2")
- WQVGA = 480 x 272 (~4.3")
- VGA = 640 x 480 (~5.7")
- WVGA = 800 x 480 (~7")

Refresh Rate

- 30 Hz
- 60 Hz

Display Representation	Color Examples	Color Depth (bits per pixel)
Mono 	Black and White	1
Grayscale 	4 shades 16 shades	2 4
Color 	256 colors 65K colors 16 million colors	8 16 24

***Basically, the more shades, more colors, more pixels
and larger screens require more memory!***

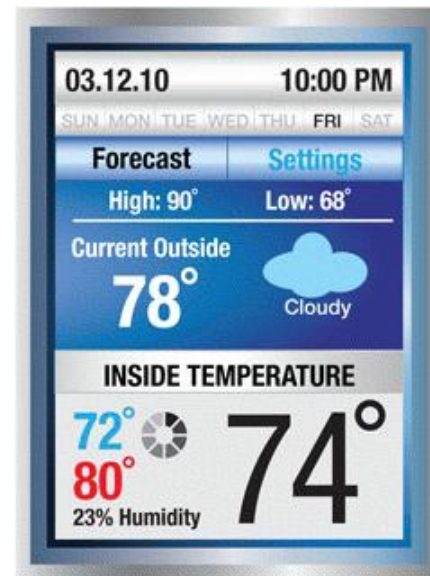
Guess How Many Colors?



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



Frame Buffer Sizes

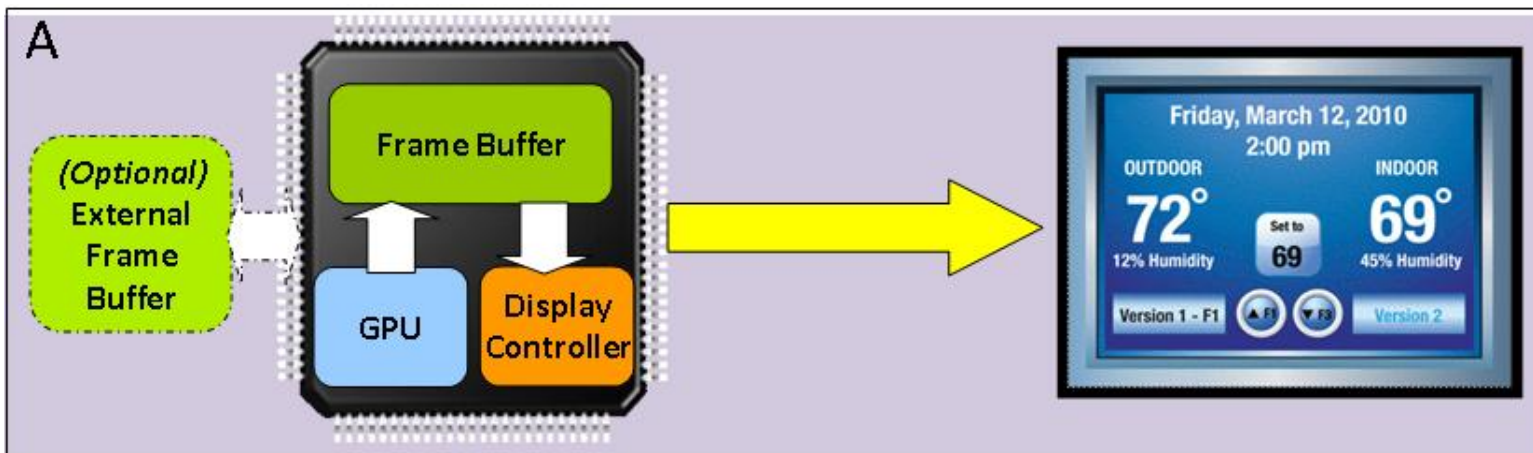
- As the color depth & display resolution increase, the frame buffer grows
- PIC24 options up to 96KB RAM
- PIC32 options up to 128KB RAM
- Use external SRAM for frame buffers > 128KB

Display Resolution		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)
WQVGA	480x272	16,320	32,640	65,280	130,560	261,120
QVGA	240x320	9,600	19,200	38,400	76,800	153,600
HQVGA	160x240	4,800	9,600	19,200	38,400	76,800
QQVGA	160x120	2,400	4,800	9,600	19,200	38,400
Common for OLED	128x64	1,024	2,048	4,096	8,192	16,384

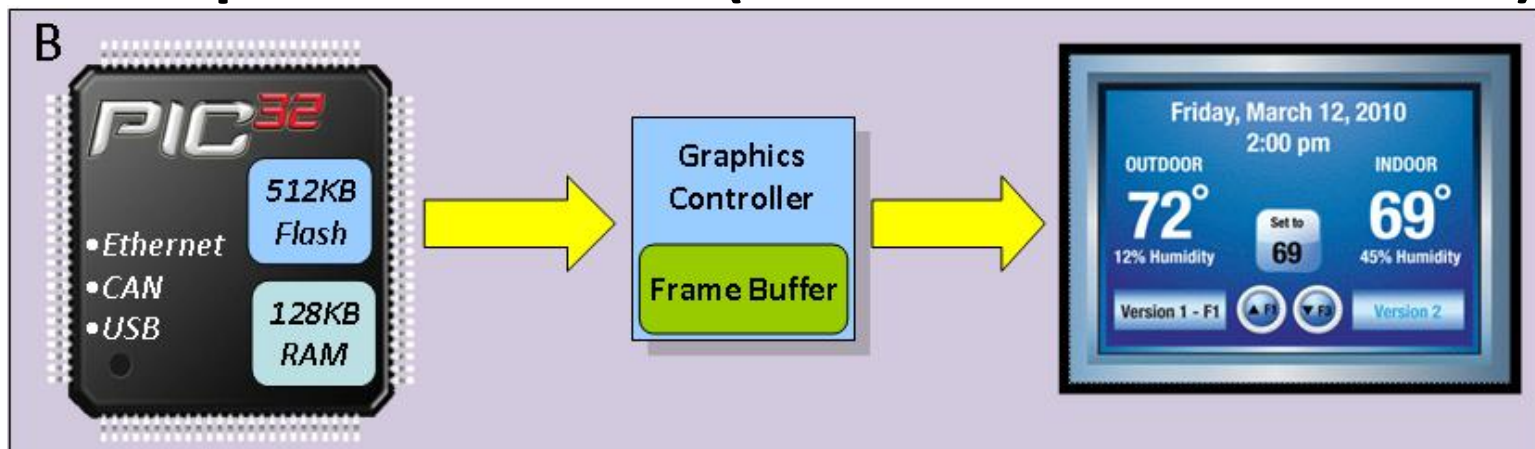
	32K byte RAM
	128K byte RAM
	External SRAM

Choosing a Graphics Subsystem

Low Cost without External Controller (PIC24 “DA” or PIC32)



External Graphics Controller (PIC24 or PIC32 with PMP)





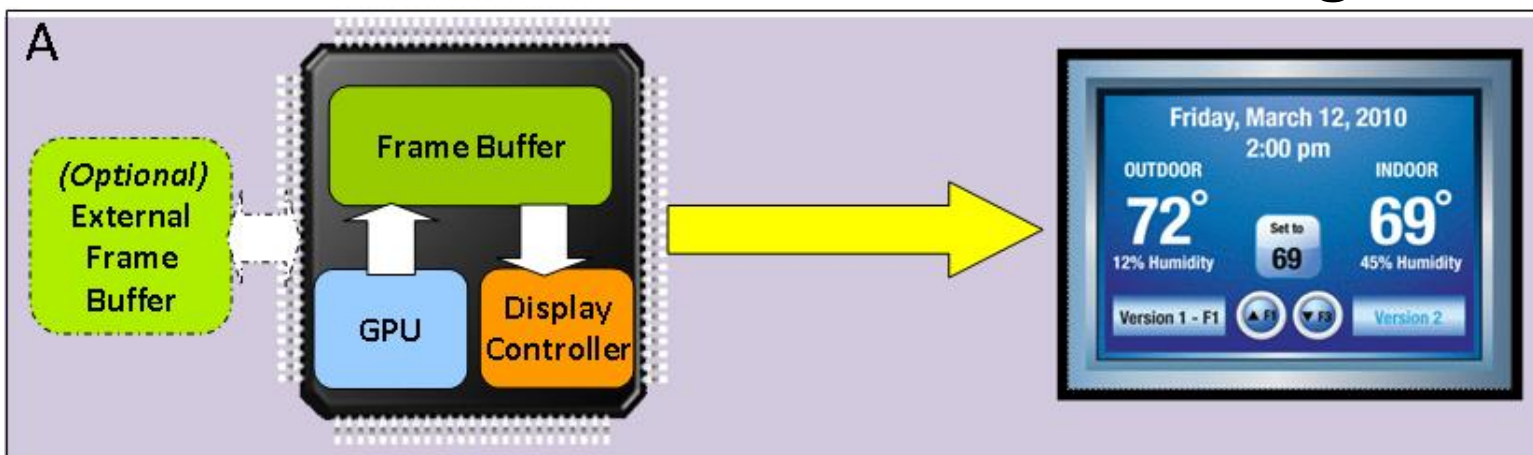
Graphics Portfolio Summary

	PIC24 “DA” Integrated Graphics Controller	PIC32 Controllerless Graphics	External Solomon Graphics Controller SSD1926	External Epson Graphics Controller S1D13517
Display*	WQVGA	WQVGA	WQVGA	WVGA
Graphics	HW acceleration: Rectangles Characters Images	DMA on PIC32 + <5 MIPS	HW acceleration SD card I/F JPEG engine	SDRAM I/F Alpha-blending Picture in picture
Frame Buffer	Color Lookup Table + 96KB on MCU + Ext SRAM	128KB on MCU + Ext SRAM	256KB on Solomon Controller	Ext SDRAM
Core MIPS	16	80	-	-
Power	Better	Good	Good	Good
Cost	\$	\$	\$\$	\$\$\$

* Max size at 16bpp, 60 Hz

Low Cost Graphics

For smaller screens, lowest cost, most integration



Both can support
up to 8bpp QVGA
240x320 with
internal frame
buffer!

External SRAM
can be used for
larger frame
buffers

PIC24F "DA" Family

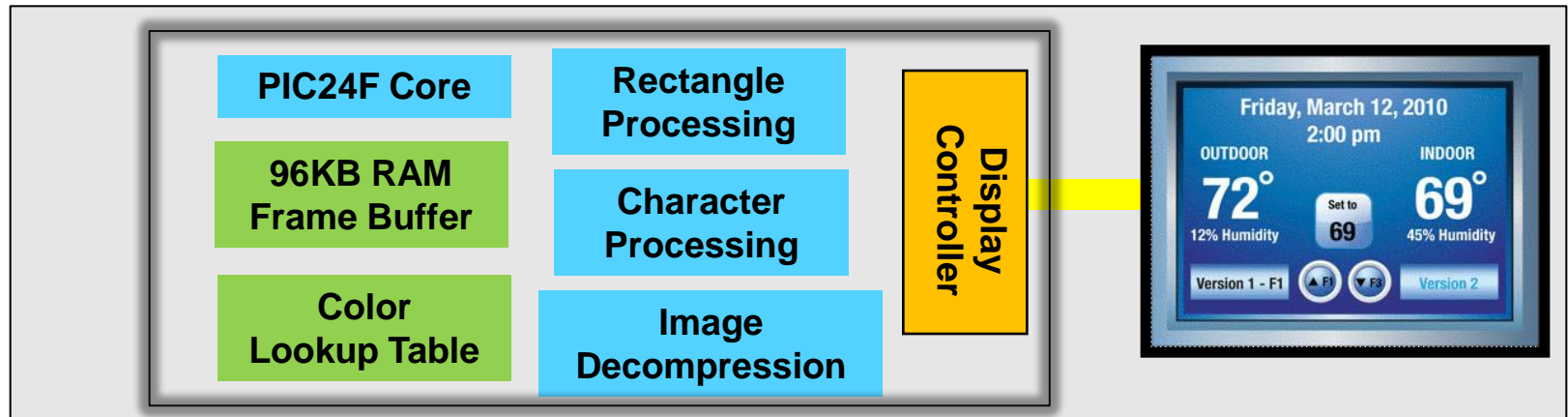
Hardware Acceleration
Integrated Controller
Lowest Power
Lowest Cost

PIC32

Controllerless

<5 MIPS + DMA
32-bit Performance
Low Cost
Integrated connectivity & touch

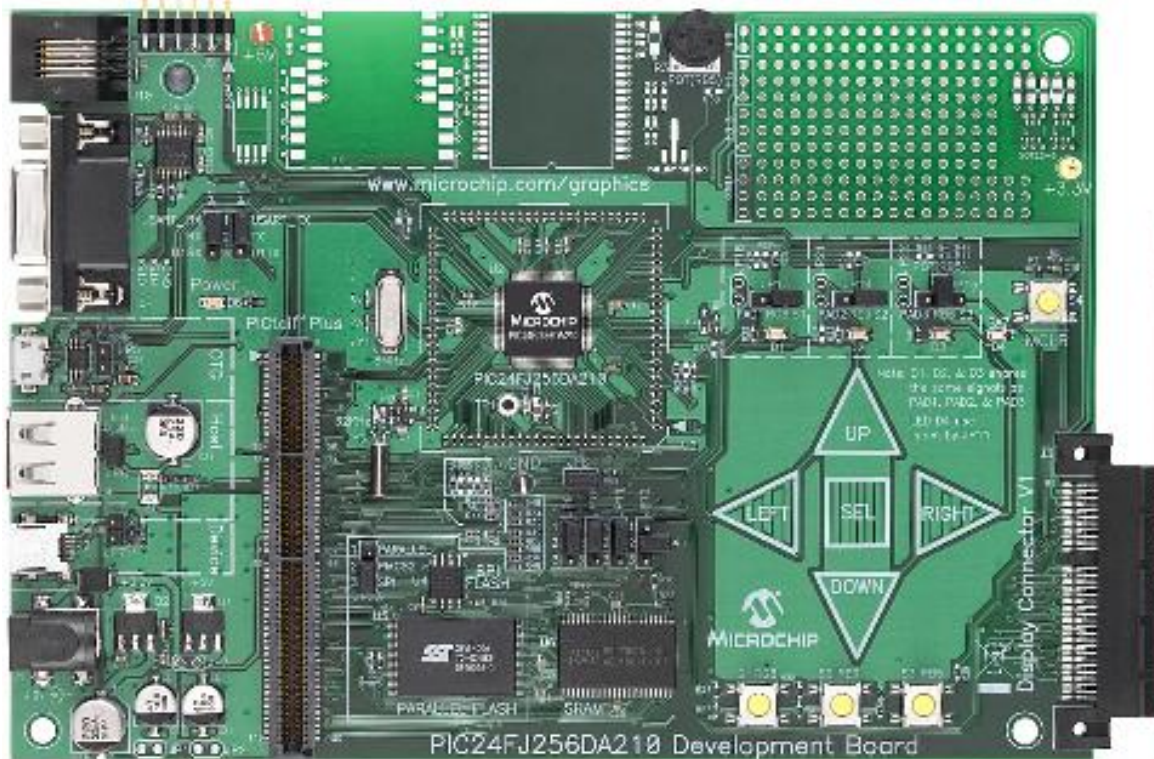
Lowest Cost Graphics - PIC24FJ DA Family



- Process and render graphics without utilizing any MCU MIPS
- Easy to use Graphics Processing Units for hardware acceleration
 - Rectangle Processing
 - Move and copy rectangles with smooth and fast memory to memory transfers
 - Enables scrolling and animation
 - Image Decompression – show compressed graphical images with 0 MIPS
 - Character Processing – render text without CPU intervention
 - Integrated display controller with a clean, low-jitter clock
- Color Lookup Table – support different color palettes for each screen
- Cost savings using internal 96KB RAM for QVGA 8bpp
 - External memory interface option for higher resolution

PIC24FJ256DA210 Development Board

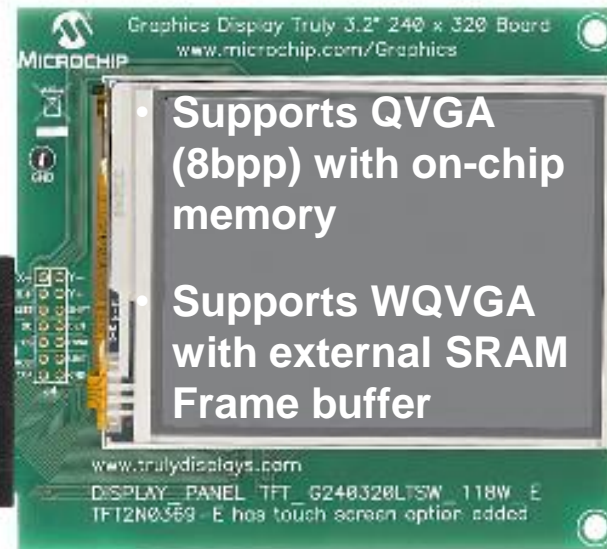
Part Number: DM240312



PICtail™ Plus
daughter board
expansion slot

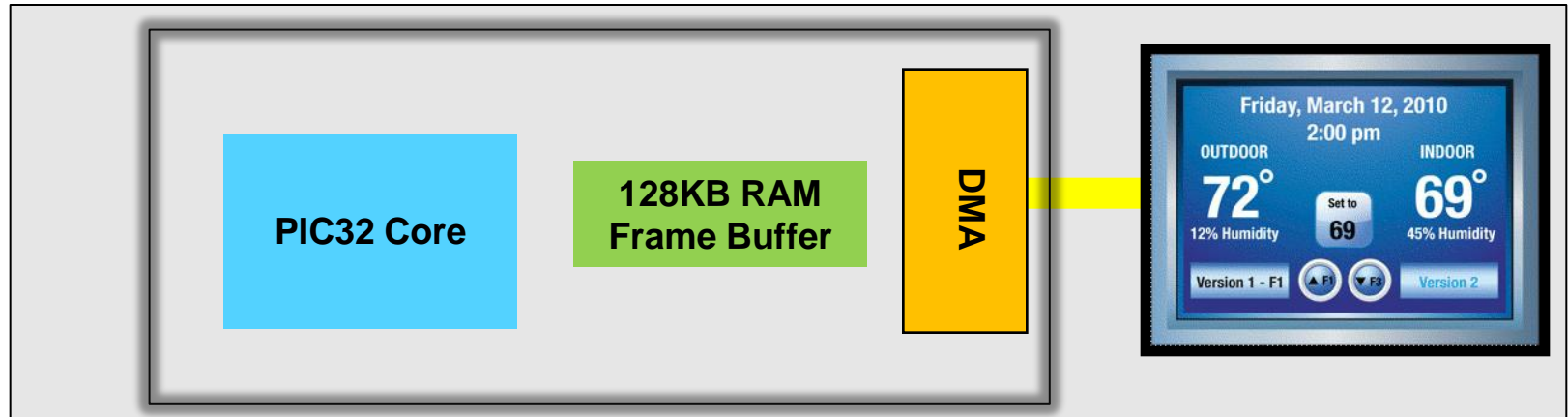
Additional Flash and
RAM to store more
objects and fonts

Connect to one of our
available TFT displays or
choose your own



- Supports QVGA (8bpp) with on-chip memory
- Supports WQVGA with external SRAM Frame buffer

Low Cost Controllerless (LCC) Graphics PIC32 Family

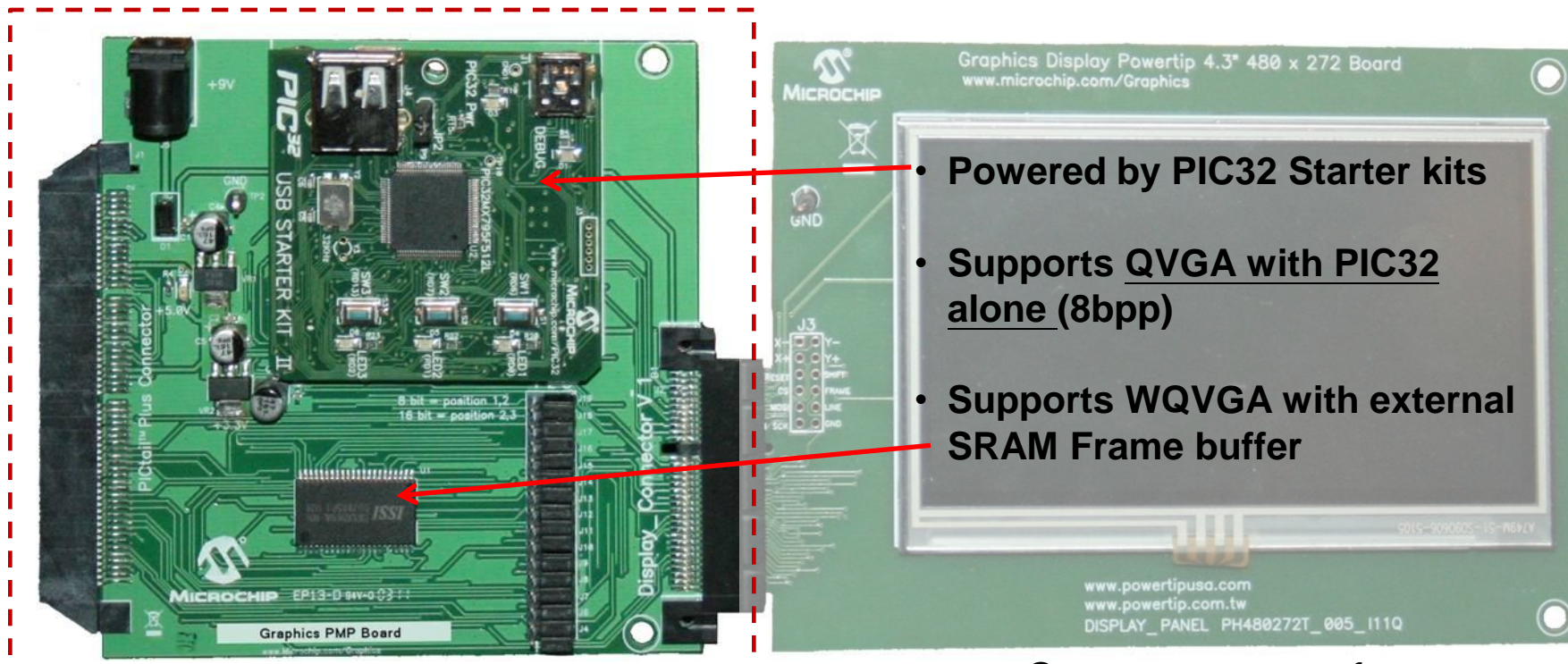


- Uses <5 MIPS & DMA to render graphics and drive display without a dedicated graphics controller
 - Direct interface to STN, TFT displays
 - Supports QVGA 8bpp with room to spare in 128KB RAM
 - Supports WQVGA with external SRAM Frame Buffer
 - PMP Interface to external memory for frame buffers >128KB
- PIC32 Core with 80 MIPS for high performance
- Connectivity - USB, Ethernet & CAN
- Integrated mTouch



LCC (Low-Cost Controllerless) Graphics PiCTail Plus Board

Driving Graphics Displays **WITHOUT** a Graphics Controller



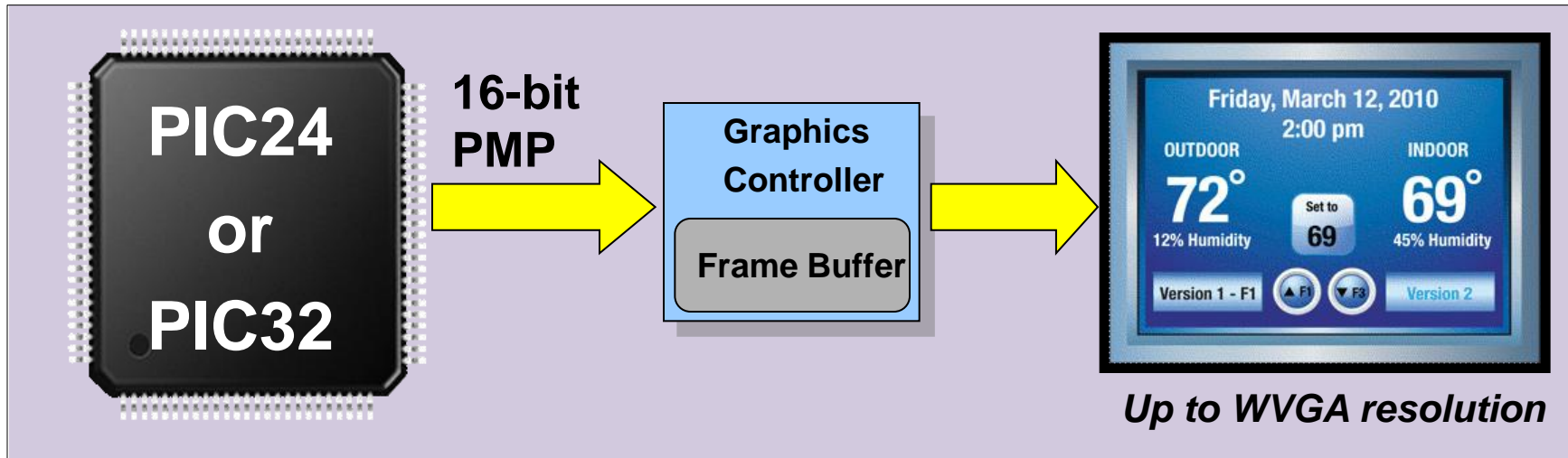
Part Number: AC164144

Available May 2011

Connect to one of our
available TFT displays or
choose your own

External Graphics Controllers: Solomon or Epson

For larger screens, more colors, advanced graphics



Works with any PIC24 or PIC32 with Parallel Master Port (PMP)

**PIC24 or PIC32 +
Solomon SSD1926
Graphics Controller**
Includes 256KB Frame Buffer
SD Card Interface
JPEG Decode Engine
WQVGA (480x272)

**PIC24 or PIC32 +
Epson S1D13517
Graphics Controller**
Includes SDRAM I/F
Alpha blending
Picture in picture
WVGA (800 x 480) at 24bpp

Solomon SSD1926 Graphics Controller

Hardware Graphics Acceleration

SD Card Interface

JPEG Decode Engine

256KB RAM on SSD1926

Supported Interfaces

- 4/8 bit STN
- 4/8 bit CSTN
- 18-bit HR-TFT
- 9/12/18/24 bit TFT

PICtail Plus Board Includes:

- 16 Megabit (2Mx8) serial flash memory for additional data storage
- Interface with different display boards
- Explorer 16 Development Board Connector
- PIC32 Starter Kit Connector



**Graphics LCD Controller PICtail
Plus SSD1926 Board
AC164127-5**

Epson S1D13517 Graphics Controller

- **Advanced Graphics Capabilities**
 - Up to 24bpp
 - Supports larger screen sizes up to 7" (WVGA)
 - Alpha blending
 - Picture in picture
 - Transparency
- **Supports up to WVGA (800 x 480) at 24bpp**
- **SDRAM Interface for less expensive external memory**
- **18/24-bit TFT Interfaces**
- **PICtail Plus Board Includes:**
 - 128 Megabit SDRAM frame buffer
 - 64 Megabit serial Flash memory
 - Interface with different display boards
 - Explorer 16 Development Board Connector
 - PIC32 Starter Kit Connector



**Graphics Controller PICtail
Plus Epson S1D13517 Board
AC164127-7**

Available April 2011

Summary of Graphics Solutions

	Graphics Performance	Frame Buffer (KB)	Max 16bpp Size (60Hz)	Integration	Sleep Power	Cost
Low Cost Options - No External Graphics Controller						
PIC24FJ256DA210 Lowest Cost Lowest Power	HW acceleration: Rectangles, Characters, Images	Color Lookup Table + 96 + Ext. SRAM	WQVGA	Touch USB	Better	\$
PIC32 Low-Cost Controllerless 32-bit Performance Integration	<5 MIPS & DMA on PIC32	128 + Ext. SRAM	WQVGA	Touch USB Ethernet CAN	Good	\$
External Solomon Graphics Controller – More Graphics Flexibility						
PIC24 + Solomon SSD1926	Controller: HW acceleration, SD card I/F, JPEG engine	256 on SSD	WQVGA	Touch USB	Good	\$\$
PIC32 + Solomon SSD1926	Controller: HW acceleration, SD card I/F, JPEG engine	256 on SSD	WQVGA	Touch USB Ethernet CAN	Good	\$\$
External Epson Graphics Controller – More Graphics Capability and Resolution						
PIC24 + Epson S1D13517	Controller: SDRAM I/F, Alpha- blending, Picture in picture	External SDRAM	VGA	Touch USB	Good	\$\$\$
PIC32 + Epson S1D13517	Controller: SDRAM I/F, Alpha- blending, Picture in picture	External SDRAM	WVGA	Touch USB Ethernet CAN	Good	\$\$\$

FREE Graphics Software Tools



Graphics Library

Pre-made graphics

Multiple fonts and languages

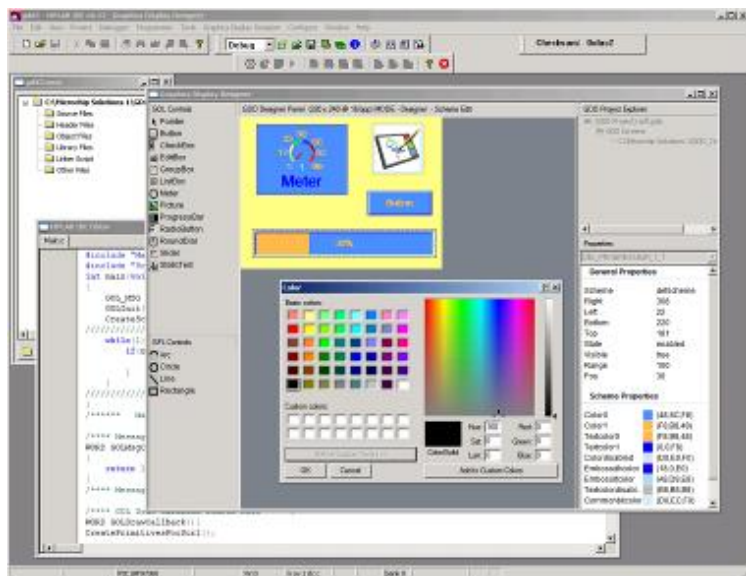
For 16-bit and 32-bit PIC® MCUs

Graphics Display Designer

Visual Design Tool

Provides GUI design wizard

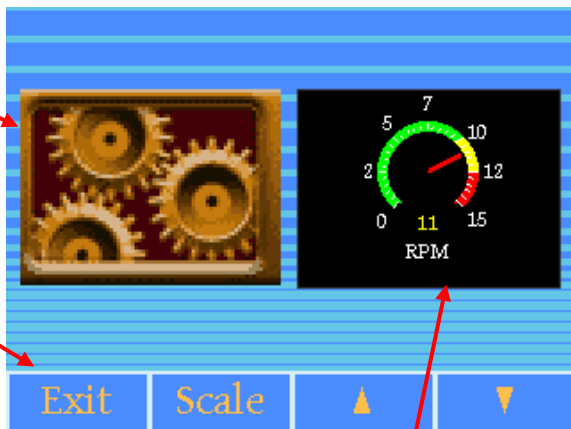
Works with Graphics Library



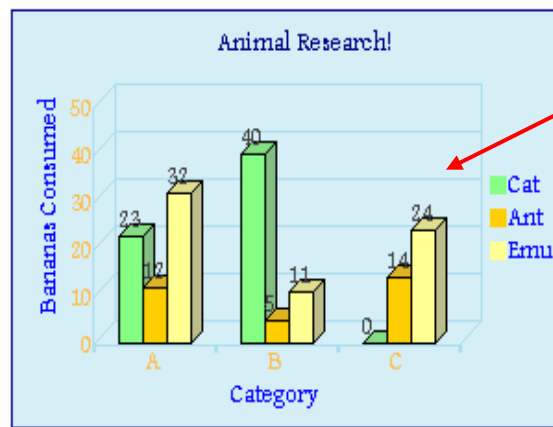
Graphics Library Support

Picture

Buttons



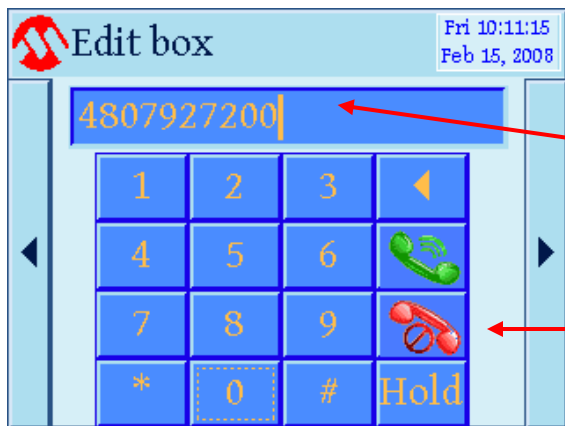
Meter



Chart

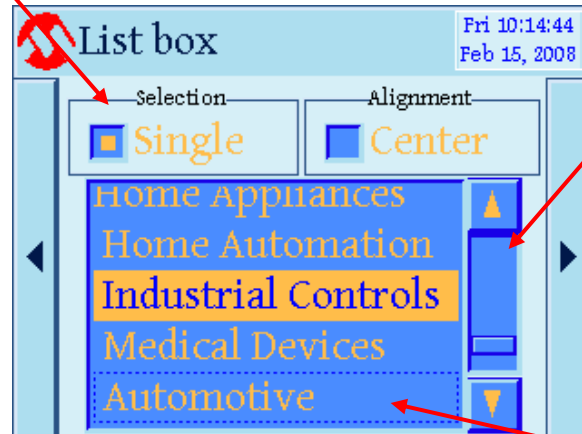
Checkbox

Scroll Bar



Edit Box

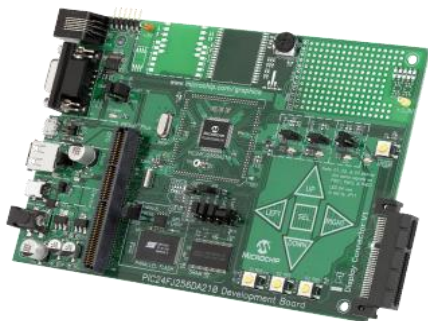
**Buttons
w/ Image**



List Box

Graphics Development Boards & PICtails

Low Cost Solutions No External Graphics Controller



**PIC24FJ256DA210 Board
(DM240312)**



**PIC32 Low-Cost Controllerless
Graphics PICtail
(AC164144)**

**Available
May 2011**

Solutions with External Graphics Controllers



**Graphics LCD
Controller PICtail Plus
SSD1926 Board
(AC164127-5)**



**Graphics Controller
PICtail Plus Epson
S1D13517 Board
(AC164127-7)**



**PIC32 Multimedia Expansion Board
Includes Solomon SSD1926 Controller
(DM320005)**

**Available
April 2011**



Graphics Development Tools

Family	Tools Recommended
PIC24 “DA” Family	PIC24FJ256DA210 Board (DM240312) + Display Board
PIC32 “LCC” Graphics	PIC32 Starter Kit (DM320001 or DM320003) + LCC Graphics Board (AC164144) + Display Board
PIC24 + Solomon SSD1926	Explorer 16 (DM240001)+ Solomon GFX Board (AC164127-5) + Display Board
PIC32 + Solomon SSD1926	PIC32 Starter Kit (DM320001 or DM320003) + Multimedia Expansion Board (DM320005)
PIC24 + Epson S1D13517	Explorer 16 (DM240001)+ Epson GFX Board (AC164127-7)+ Display Board
PIC32 + Epson S1D13517	PIC32 Starter Kit (DM320001 or DM320003) + Epson GFX Board (AC164127-7)+ Display Board

Display Board Options

**Available
June 2011**

**Available
April 2011**



	QVGA 3.2" Graphics Display Truly 240x320 Board AC164127-4	WQVGA 4.3" Graphics Display Powertip 480x272 Board AC164127-6	VGA 5.7" Graphics Display Truly 640x480 Board AC164127-8	WVGA 7" Graphics Display Truly 800x480 Board AC164127-9	Prototype Boards Connect your glass AC164139
	PIC24 "DA" Family	Yes	Yes		Yes
	PIC32 "LCC" Graphics	Yes	Yes		Yes
	PIC24 + Solomon SSD1926	Yes	Yes		Yes
	PIC32 + Solomon SSD1926	Yes	Yes		Yes
	PIC24 + Epson S1D13517	Yes	Yes	Yes	Yes
	PIC32 + Epson S1D13517	Yes	Yes	Yes	Yes

Recommendations based on 16bpp, 60Hz performance on PIC MCU and LCD controller
Actual displays are capable of more than 16bpp, 60Hz

Graphics PIC[®] MCUs Features & Benefits

Features

Benefits

Results

Free Software
Graphics Library



Includes buttons, charts,
checkboxes, scroll bars, list
boxes, buttons, images



Saves money and
Faster time to market

Free
Graphics Designer GUI



Easy to design screens
with visual design tools



Reduced design
complexity

Options with
Hardware
Acceleration



Faster rendering of lines,
rectangles, characters and
Images without using MIPS



Plenty of MIPS left
for other functions

Controllerless Options



Eliminates separate
graphics controller chip



Lower system cost

Support
for External
Graphics Controllers



Support for various display
sizes and advanced graphics



Flexible range of
solutions

Broad Portfolio
of Graphics MCUs

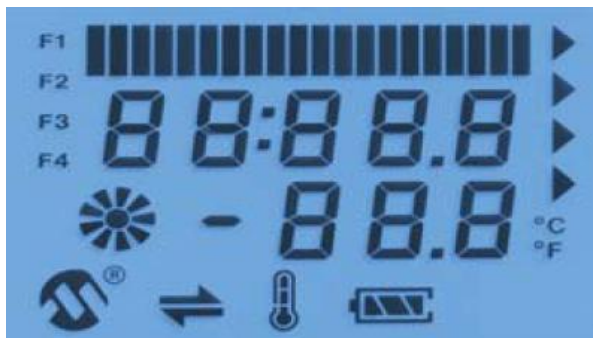


Easy platform migration
with a rich set of integrated
peripherals and memory



Easy migration and
reduced BOM cost

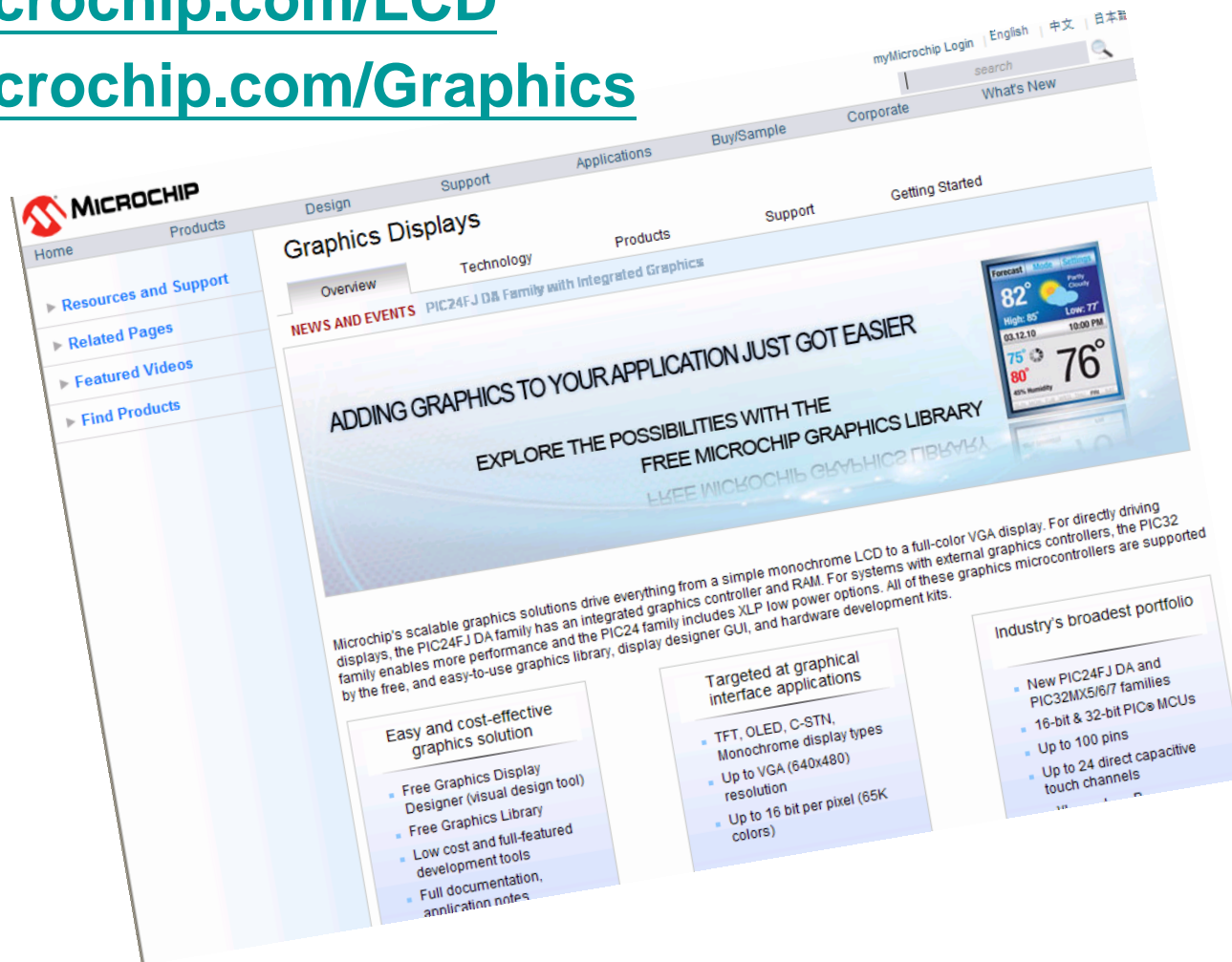
Display Solutions



- **More than just the MCU**
 - Drivers
 - Software
 - Design Tools
 - Development Boards

For Additional Information

- www.microchip.com/LCD
- www.microchip.com/Graphics



Training Available

- **Embedded Designers Forum featuring Smart Energy**
- **Web Seminar: Introduction to the PIC24FJ256DA210 Graphics PIC® MCU Family**
- **Web Seminar: Microchip Graphics for Human Interface Applications**
- **Web Seminar: Microchip Graphics Display Library Architecture**
- **Web Seminar: How Does a Graphics LCD Work?**
- **Web Seminar: Graphics LCD System and PIC24 Interface**
- **Regional Training Center – HIF2131A**
- **MASTERS Conferences**



MICROCHIP

Regional Training Centers

THANK YOU!!



MICROCHIP

Regional Training Centers

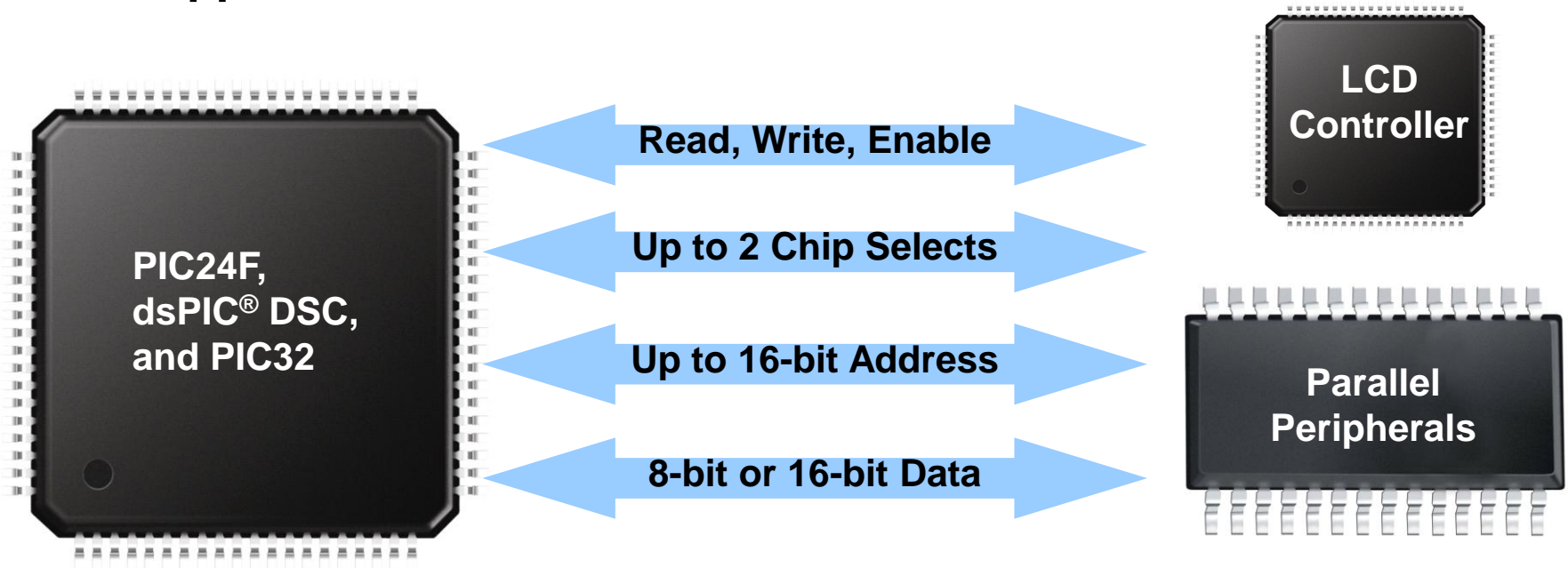
APPENDIX A


Parallel Master Port Initialization

Writing a Driver

Parallel Master Port – PMP

- `ResetDevice()` Initializes PMP
- PMP Supports I80 or M68 interfaces



 Use DMA on PIC32 or dsPIC33 to increase PMP throughput.





Writing a Driver

Parallel Master Port – PMP

- Configure PMP for LCD System Interface
 - PMP Mode Register (PMMODE)
 - PMP Control Register (PMCON)
- Example from `ResetDevice()` in `LGDP4531.c` driver...

MODE1:MODE0

Mode = 1 for M68

Mode = 2 for I80

```
// PMP setup
PMMODE = 0;
PMAEN = 0;
PMCON = 0;
PMMODEbits.MODE = 2;
PMMODEbits.WAITB = 0;
PMMODEbits.WAITM = 1;
PMMODEbits.WAITE = 0;
PMCONbits.CSF = 0;
PMCONbits.PTRDEN = 1;
PMCONbits.PTWREN = 1;
PMCONbits.PMPEN = 1;
```



Writing a Driver

Parallel Master Port – PMP

- Configure PMP for LCD System Interface
 - PMP Mode Register (PMMODE)
 - PMP Control Register (PMCON)
- Example from `ResetDevice()` in `LGDP4531.c` driver...

WAITB:
Data Setup to R/W

WAITM:
Read to Byte Enable Strobe

WAITE:
Data Hold after Enable/Strobe

```
// PMP setup
PMMODE = 0;
PMAEN = 0;
PMCON = 0;
PMMODEbits.MODE = 2;
PMMODEbits.WAITB = 0;
PMMODEbits.WAITM = 1;
PMMODEbits.WAITE = 0;
PMCONbits.CSF = 0;
PMCONbits.PTRDEN = 1;
PMCONbits.PTWREN = 1;
PMCONbits.PMPEN = 1;
```

Writing a Driver

Parallel Master Port – PMP

- Wait state calculations...
 - Dependent on driver IC WR and RD strobe min times

$$\text{Wait States} = (T_{\text{MIN}} / T_{\text{CY}}) - 1$$

For PIC 24F running at 16 MIPS:

$$T_{\text{CY}} = 62.5 \text{ ns}$$

From Driver IC datasheet

$$T_{\text{WRMIN}} = 25 \text{ ns} \Rightarrow 0 \text{ Wait States}$$

$$T_{\text{RDMIN}} = 250 \text{ ns} \Rightarrow 1.4 \text{ Wait States}$$



Writing a Driver

Parallel Master Port – PMP

- Configure PMP for LCD System Interface
 - PMP Mode Register (PMMODE)
 - PMP Control Register (PMCON)
- Example from `ResetDevice()` in `LGDP4531.c` driver...

CSF1:CFS0

Chip Select Function Bits

0 – PMSC2:1 used as Addr15:14

1 – PMCS2 = chip select

PMCS1 = Addr14

2 -- PMCS2:1 = chip selects

```
// PMP setup
PMMODE = 0;
PMAEN = 0;
PMCON = 0;
PMMODEbits.MODE = 2;
PMMODEbits.WAITB = 0;
PMMODEbits.WAITM = 1;
PMMODEbits.WAITE = 0;
PMMODEbits.CSF = 0;
PMCONbits.PTRDEN = 1;
PMCONbits.PTWREN = 1;
PMCONbits.PMPEN = 1;
```



Writing a Driver

Parallel Master Port – PMP

- Configure PMP for LCD System Interface
 - PMP Mode Register (PMMODE)
 - PMP Control Register (PMCON)
- Example from `ResetDevice()` in `LGDP4531.c` driver...

PTRDEN: R/W Strobe Enable

0 = Disabled

1 = Enabled

```
// PMP setup
PMMODE = 0;
PMAEN = 0;
PMCON = 0;
PMMODEbits.MODE = 2;
PMMODEbits.WAITB = 0;
PMMODEbits.WAITM = 1;
PMMODEbits.WAITE = 0;
PMCONbits.CSF = 0;
PMCONbits.PTRDEN = 1;
PMCONbits.PTWREN = 1;
PMCONbits.PMPEN = 1;
```



Writing a Driver

Parallel Master Port – PMP

- Configure PMP for LCD System Interface
 - PMP Mode Register (PMMODE)
 - PMP Control Register (PMCON)
- Example from `ResetDevice()` in `LGDP4531.c` driver...

PTWREN:
Write Enable Strobe Enable
0 = Disabled
1 = Enabled

```
// PMP setup
PMMODE = 0;
PMAEN = 0;
PMCON = 0;
PMMODEbits.MODE = 2;
PMMODEbits.WAITB = 0;
PMMODEbits.WAITM = 1;
PMMODEbits.WAITE = 0;
PMCONbits.CSF = 0;
PMCONbits.PTRDEN = 1;
PMCONbits.PTWREN = 1;
PMCONbits.PMPEN = 1;
```



Writing a Driver

Parallel Master Port – PMP

- Configure PMP for LCD System Interface
 - PMP Mode Register (PMMODE)
 - PMP Control Register (PMCON)
- Example from `ResetDevice()` in `LGDP4531.c` driver...

PMPEN: PMP Enable

0 = Disabled

1 = Enabled

```
// PMP setup
PMMODE = 0;
PMAEN = 0;
PMCON = 0;
PMMODEbits.MODE = 2;
PMMODEbits.WAITB = 0;
PMMODEbits.WAITM = 1;
PMMODEbits.WAITE = 0;
PMCONbits.CSF = 0;
PMCONbits.PTRDEN = 1;
PMCONbits.PTWREN = 1;
PMPENbits.PMPEN = 1;
```



MICROCHIP

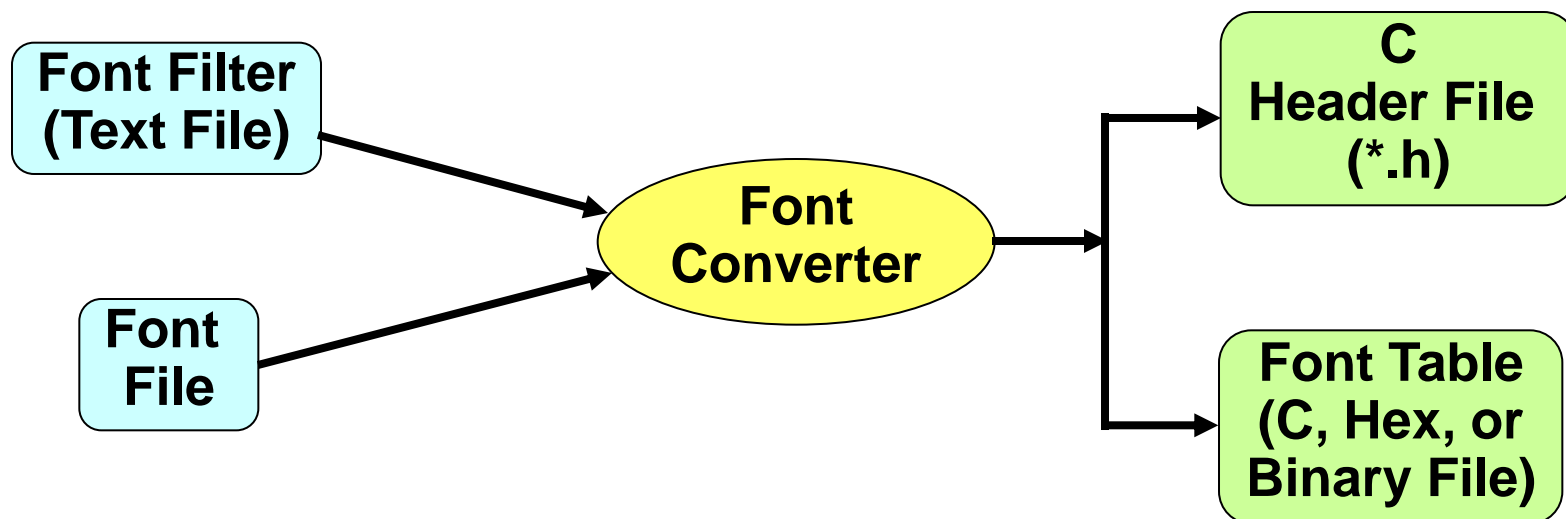
Regional Training Centers

APPENDIX B

**Font Conversion, Filtered Font, and
External Memory Examples**

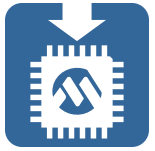
Font Filters

- Used to reduce memory for font image
 - Glyphs are remapped to new character ID
 - Requires filter text file (font filter file)
 - Must include generated reference files in project



Lab Exercise 3

Application Code



GFXLab3.c – Main application code

```
// calibrate Touch Screen if needed
TouchLoadCalibration();

// create menu style scheme
menuScheme = GOLCreateScheme();

while(1) {
    if(GOLDraw()){ // Draw GOL objects
        TouchGetMsg(&msg); // Get message from touch screen
        GOLMsg(&msg); // Process message

#ifdef USE_KEYBOARD
        if ((tick - sideBtnTick) > 100) {
            sideBtnTick = tick;

            SideButtonsMsg(&msg); // Get side buttons message
            GOLMsg(&msg); // Process message
        }
#endif
    }
}
```

WAIT FOR DRAW TO COMPLETE

Touch Screen Messages

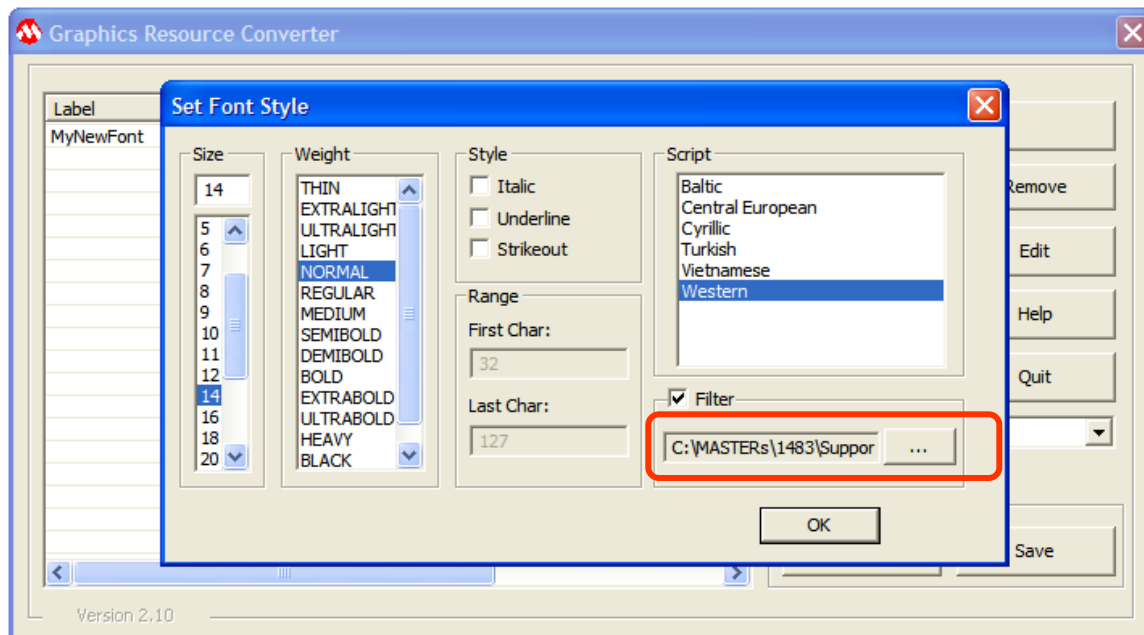
Side Button Messages

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
- Recreate image to edit strings
 - Can't reference characters directly
- Refer to App Note 1182
(Fonts in the Microchip Graphics Library)

Using Fonts Example

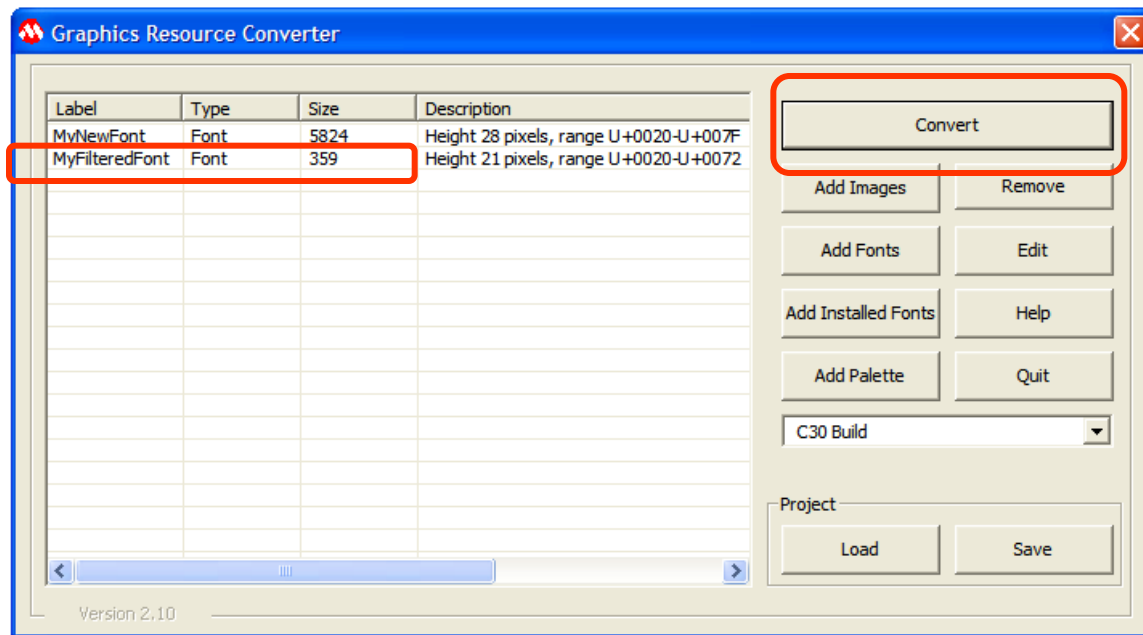
- Convert a True Type font ...
- Add a font filter



- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
(Raster -- *.fnt or
True Type Font -- *.ttf)
- 4) Set the Font Style
- 5) Select filter option and
navigate to filter text
file

Using Fonts Example

- Convert a True Type font ...
- Add a font filter



- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
(Raster -- *.fnt or
True Type Font -- *.ttf)
- 4) Set the Font Style
- 5) Select filter option and
navigate to filter text
file (if desired)
- 6) Select Convert



Font Filter Sample File

- Font filter file: `filter.txt` ...
 - Created and saved in Microsoft Wordpad

```
HelloWorldStr1:    Hello World!           // first string
HelloworldStr2:    How are you?           //second string
```

- Tool generates: `filtered_font.h` and `filtered_font.c`
 - Must be included in application file

```
#include "Graphics\Graphics.h"
```

```
// Automatically generated reference arrays for the filter.txt file.
```

```
XCHAR HelloWorldStr1[] = {0x0020, 0x0020, 0x0020, 0x0020, 0x0022,
0x0029, 0x0025, 0x002A, 0x002B, 0x002D, 0x0020, 0x0023, 0x0024, 0x002D,
0x0027, 0x0028, 0x002C, 0x0029, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020,
0x0020, 0x0000};           // first string
```

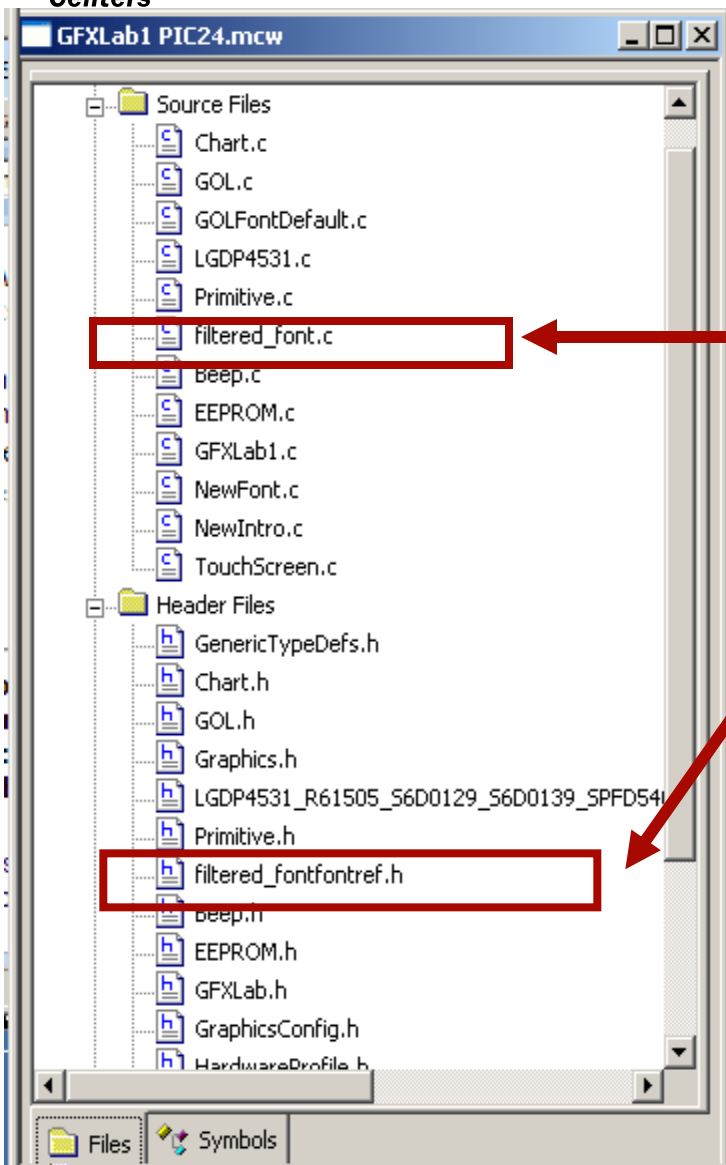
```
XCHAR HelloWorldStr2[] = {0x0020, 0x0020, 0x0020, 0x0021, 0x0024,
0x0026, 0x0026, 0x0028, 0x0020, 0x0022, 0x0029, 0x0025, 0x002A, 0x002B,
0x002D, 0x0000};           // second string
```



MICROCHIP

Regional Training
Centers

Using Filtered Fonts Internal Memory



**Add generated
files to project**



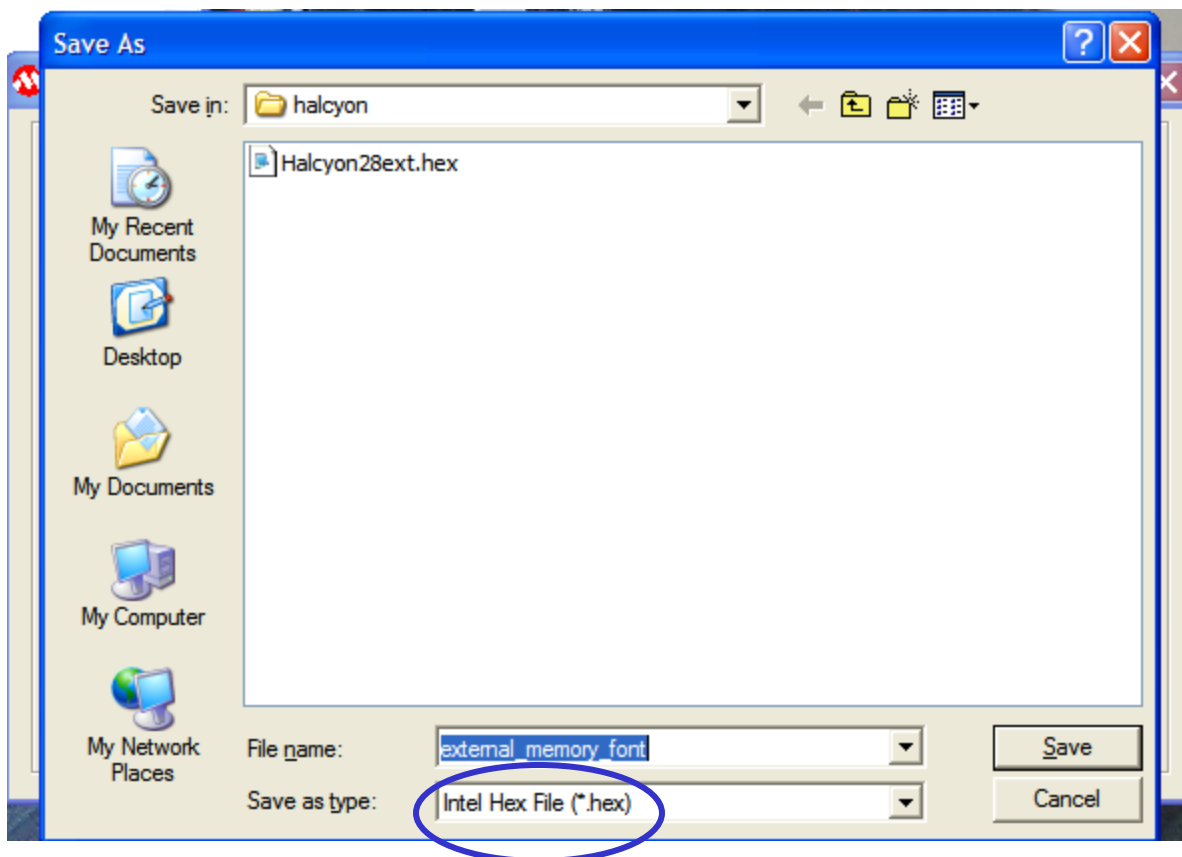
**Only filtered fonts generate
reference header files. Must
#include in application file.**

Using Fonts

Example – External Memory

- Convert a True Type font ...
- Store in external memory

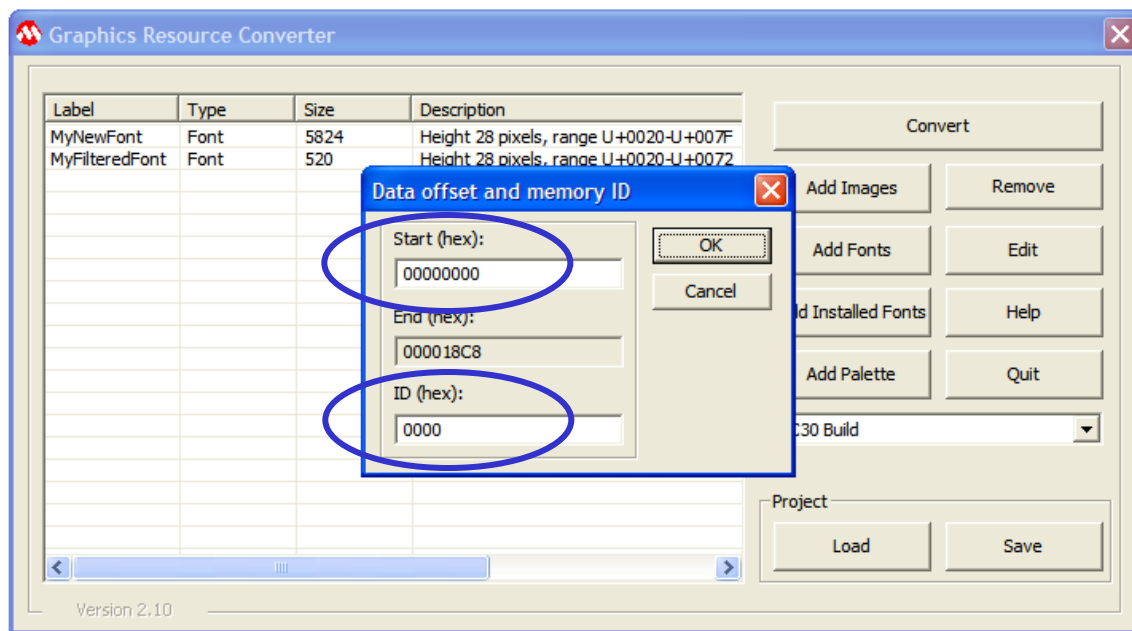
- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
(Raster -- *.fnt or
True Type Font -- *.ttf)
- 4) Set the Font Style
- 5) Select filter option and
navigate to filter text
file (if desired)
- 6) Select Convert
- 7) For external memory,
choose *.hex type



Using Fonts

Example – External Memory

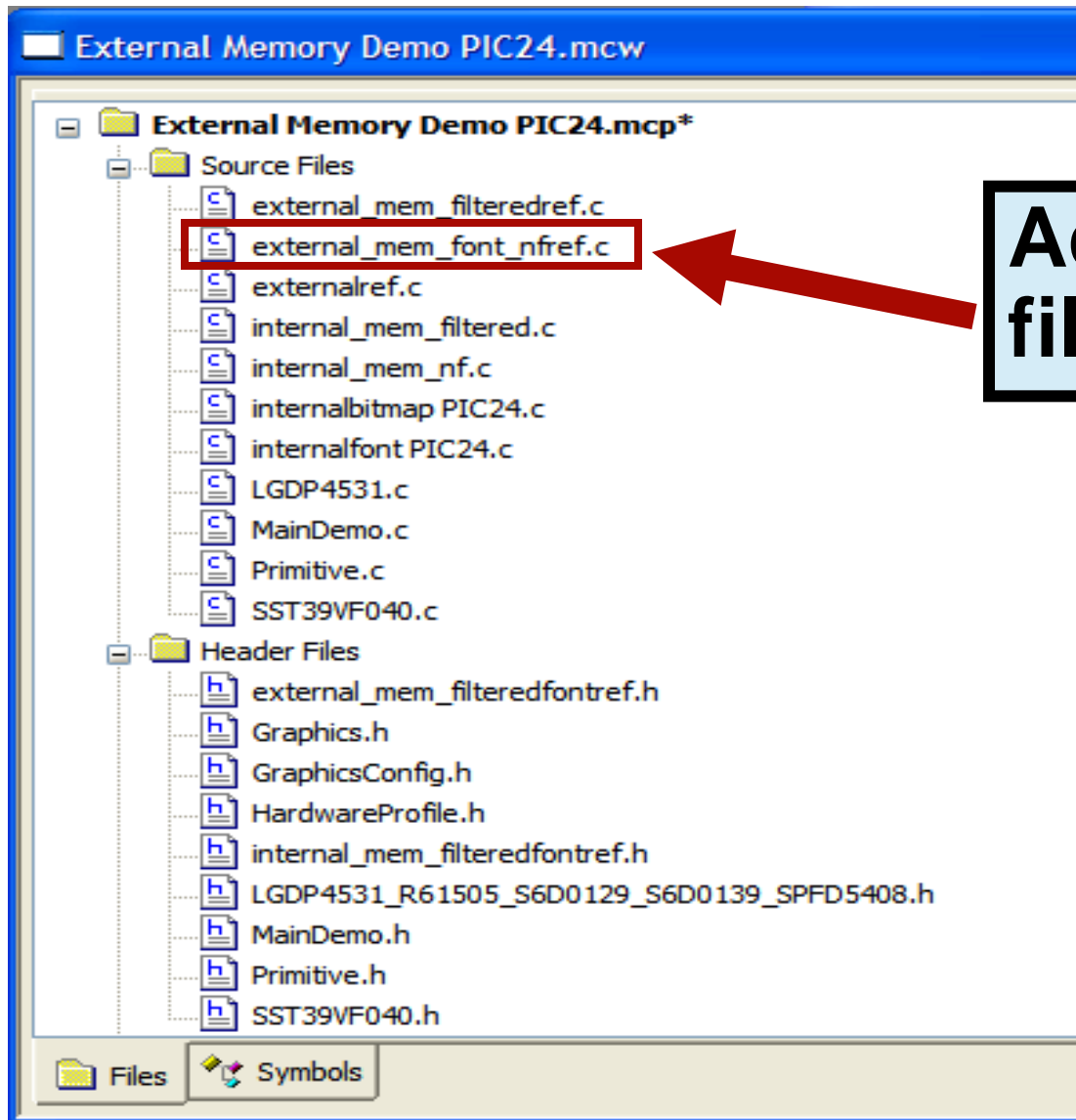
- Convert a True Type font ...
- Store in external memory



- 1) Select Add Fonts
- 2) Navigate to font folder
- 3) Select Font File
(Raster -- *.fnt or
True Type Font -- *.ttf)
- 4) Set the Font Style
- 5) Select filter option and
navigate to filter text
file (if desired)
- 6) Select Convert
- 7) Select file location
- 8) Set memory address
range and ID number



Using Fonts Internal Memory



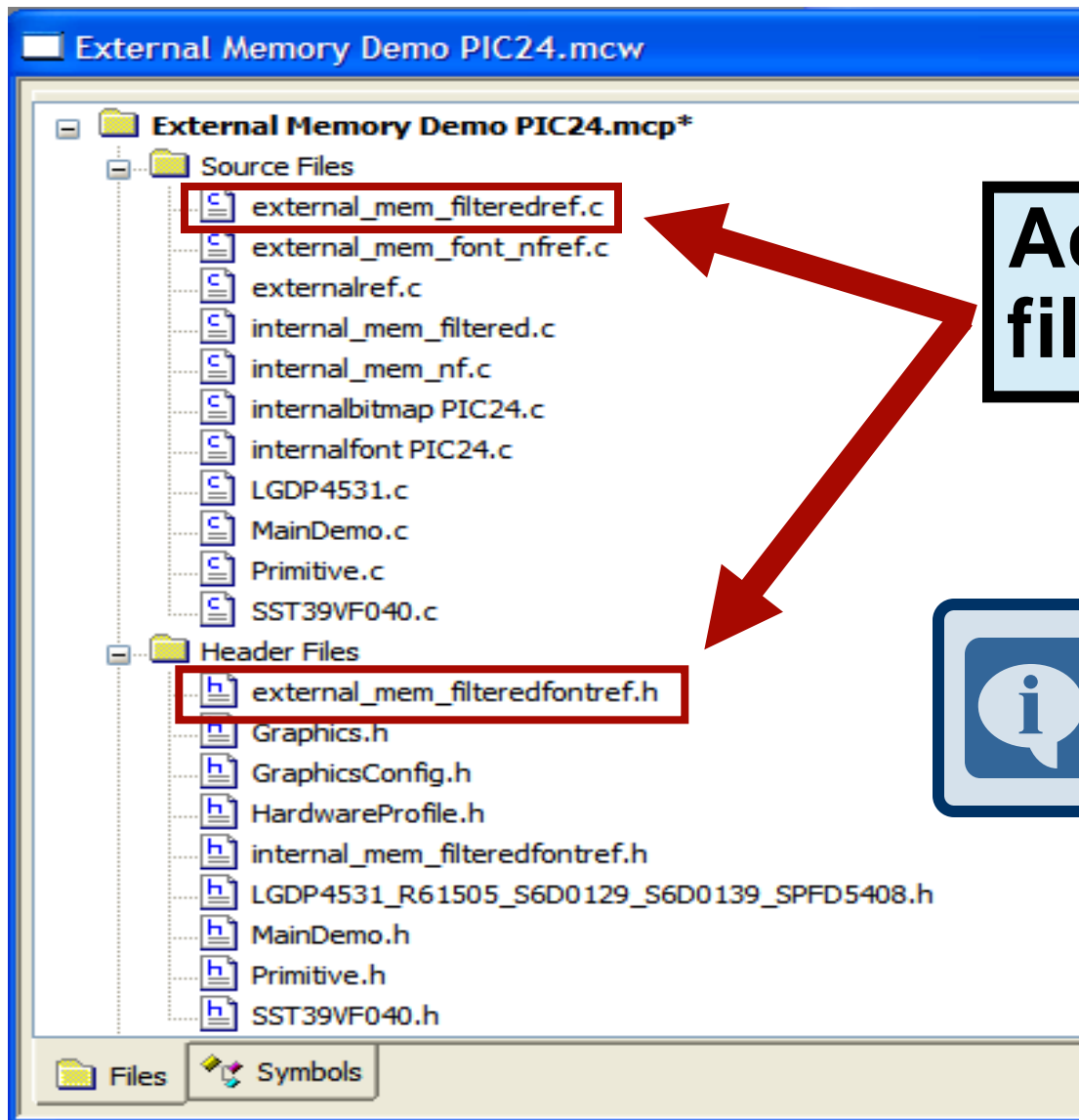
**Add generated
files to project**



MICROCHIP

Regional Training
Centers

Using Filtered Fonts External Memory



**Add generated
files to project**



Only filtered fonts generate
reference header files. Must
#include in application file.



MICROCHIP

Regional Training Centers

APPENDIX C

DA210

Graphics Processing Units

- **Rectangle Copy Graphics Processing Unit (RCCGPU)**
- **Character Graphics Processing Unit (CHRGPU)**
- **Inflate Processing Unit (IPU)**
- **Command FIFO**



RCCGPU

Example of usage:

- **Line()**
- **Bar()**
- **Rectangle()** - composed of multiple lines
- **PutPixel()** - basic rendering function to create complex shapes.
 - **Diagonal Lines**
 - **Circle()**
- **Animation of rectangular shapes**
 - **Example: Animate a bitmap to make it appear it moves from left to right?**

Character Graphics Processing Unit (CHRGPU)

- **CHRGPU is used to render characters of a specified font table stored in data memory.**
- **Font tables are assumed to be initialized in data memory**
- **The font table format is the same format defined in the Graphics Library**
- **Transparency mode is supported**
- **No dynamic character rotation**
 - **if rotation is required it should performed in software**
 - **CHRGPU with RCCGPU can be used to implement dynamic character rotation**

Inflate Processing Unit (IPU)

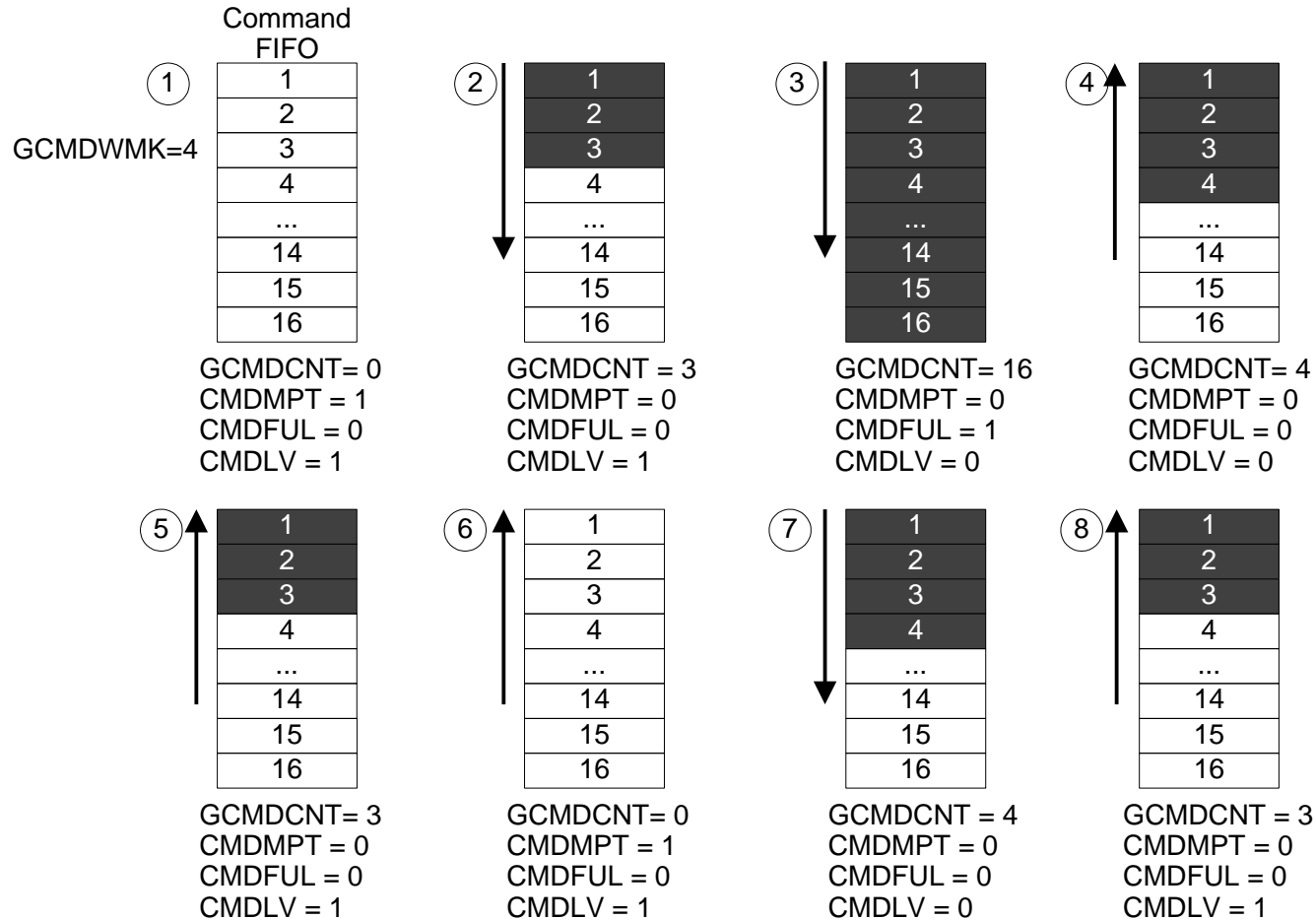
- **IPU is used to decompress data located in one memory location to another.**
- **Uses DEFLATE algorithm**
 - **Fixed Huffman codes only (no dynamic codes)**
 - **PNG like compression (normally PNG uses dynamic Huffman code)**
- **IPU is useful in applications with limited memory**
 - **Example: when using multiple fonts, compress the font tables in memory. Use IPU to decompress the currently used font table in memory. When changing the font to use, decompress the new font into the same area used by the previous font table.**
 - **Images can also be compressed and use IPU to place them in the display buffer**



Command FIFO

- **Application uses the Command FIFO to execute commands to the GPUs.**
- **16 deep 32-bit word FIFO**
- **The Command FIFO is shared by all the GPUs**
- **Only one GPU will be executing at any time**
- **All queued commands will be waiting in the FIFO until a GPU is available to process the command**
- **Command Watermark mechanism is available to optimize GPU-command FIFO operation**

Command FIFO Water Mark





Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KeeLoq, KeeLoq logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC32 logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2008, Microchip Technology Incorporated, All Rights Reserved.