MICROCHIP

Regional Training Centers

HIF 2131A

Designing with Microchip's Graphic Library

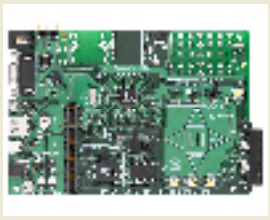
Lab Manual

Hardware

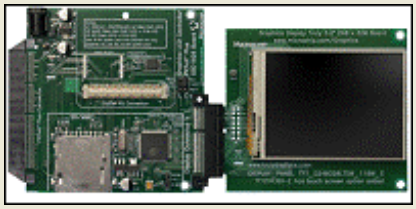
Special Instructions for Hardware Setup

Procedure to prepare firmware for correct hardware platform.

PIC24FJ256DA210 Development Board



Graphics PICTail Version 3



Each lab directory will contain a file named **HardwareProfile.h**. The appropriate hardware profile should be included in the code. This is set in this file. The following instructions outline the steps to set the correct hardware profile for your chosen hardware.

1 To run code using PIC24FJ256DA210 Development Board with the 3.2 inch Truly Display: In the directory of the current hands on lab open the **HardwareProfile.h** file and look for the following lines. Uncomment the #include line that will load the hardware profile for Truly display.
NOTE: Only one uncommented line will be used in each section.

HardwareProfile.h

```
#if defined (__PIC24F__) || defined(__dsPIC33F__) || defined(__PIC24H__)

#if defined (__PIC24FJ256DA210__)
/*********************************************************************
* Hardware Configuration for
* PIC24FJ256DA210 Development Board (DM240312)
*********************************************************************/
/* ------------------------------------------------------------ */
/* To use Graphics Display Truly 3.2" 320x240 Board (AC164127-4)     */
/* with TFT-G240320LTSW-118W-E display panel                        */
/* ------------------------------------------------------------ */
1 #include "Alternative Configura-
tions\HardwareProfile_PIC24FJ256DA210_DEV_BOARD_16PMP_MCHP_DA210_TFT_G240320LTSW_118W_E.h"

/* ------------------------------------------------------------ */
/* To use Graphics Display Powertip 4.3" 480x272 Board (AC164127-6) */
/* with PH480272T-005-I11Q display panel                            */
/* ------------------------------------------------------------ */
2 //#include "Alternative Configura-
tions\HardwareProfile_PIC24FJ256DA210_DEV_BOARD_16PMP_MCHP_DA210_PH480272T_005_I11Q.h"

#else
...
```

2 To run code using PIC24FJ256DA210 Development Board with the 4.3 inch Powertip Display, Uncomment the #include line that will load the hardware profile for Powertip display

3 To run code using Explorer 16 with a PIC24FJ128GA010 with the 3.2 inch Truly Display: In the directory of the current hands on lab open the **HardwareProfile.h** file and look for the following lines. Uncomment the #include line that will load the hardware profile for Truly display.

HardwareProfile.h

```
#if defined (__PIC24F__) || defined(__dsPIC33F__) || defined(__PIC24H__)

#if defined (__PIC24FJ256DA210__)
...
/*********************************************************************
* Hardware Configuration for
* PIC24FJ256DA210 Development Board (DM240312)
*********************************************************************/
/* ------------------------------------------------------------------ */
/* To use Graphics Display Truly 3.2" 320x240 Board (AC164127-4)    */
/* with TFT-G240320LTSW-118W-E display panel                         */
/* ------------------------------------------------------------------ */
#include "Alternative Configura-
tions\HardwareProfile_PIC24FJ256DA210_DEV_BOARD_16PMP_MCHP_DA210_TFT_G240320LTSW_118W_E.h"

/* ------------------------------------------------------------------ */
/* To use Graphics Display Powertip 4.3" 480x272 Board (AC164127-6) */
/* with PH480272T-005-I11Q display panel                             */
/* ------------------------------------------------------------------ */
//#include "Alternative Configura-
tions\HardwareProfile_PIC24FJ256DA210_DEV_BOARD_16PMP_MCHP_DA210_PH480272T_005_I11Q.h"

#else
/*********************************************************************
* Hardware Configuration for
* Explorer 16 (DM240001)
*********************************************************************/
/* ------------------------------------------------------------------ */
/* To use Graphics Display Truly 3.2" 320x240 Board (AC164127-4)    */
/* with TFT-G240320LTSW-118W-E display panel                         */
/* ------------------------------------------------------------------ */
#include "Alternative Configura-
tions\HardwareProfile_GFX_PICTAIL_V3_8PMP_SSD1926_TFT_G240320LTSW_118W_E.h"

/* ------------------------------------------------------------------ */
/* To use Graphics Display Powertip 4.3" 480x272 Board (AC164127-6) */
/* with PH480272T-005-I11Q display panel                             */
/* ------------------------------------------------------------------ */
//#include "Alternative Configura-
tions\HardwareProfile_GFX_PICTAIL_V3_8PMP_SSD1926_PH480272T_005_I11Q.h"
```

4 To run code using Explorer 16 with a PIC24FJ128GA010 with the 4.3 inch Powertip Display, Uncomment the #include line that will load the hardware profile for Powertip display When using the 4.3” display, the application code in these labs will be skewed to the left side of the screen.

5 To run code using Explorer 16 with a PIC32MX360F512L with 8-bit PMP interface (Graphics LCD Controller PICtail™ Plus SSD1926 Board jumper JP2 set to **parallel—8 Bit**) and the 3.2 inch Truly Display: In the directory of the current hands on lab open the **HardwareProfile.h** file and look for the following lines. Uncomment the #include line that will load the hardware profile for Truly display.

HardwareProfile.h

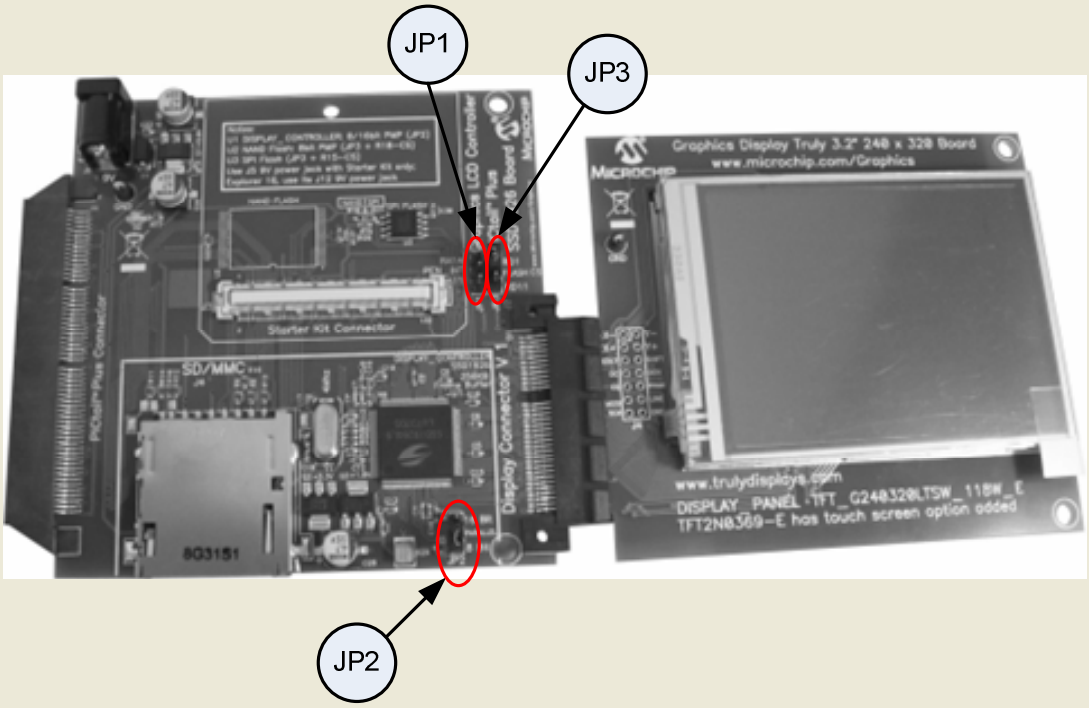
```
#if defined (__PIC24F__) || defined(__dsPIC33F__) || defined(__PIC24H__)
...
#else
#elif defined (__PIC32MX__)

/*********************************************************************
* Hardware Configuration for
* Explorer 16 (DM240001)
*********************************************************************/
/* ------------------------------------------------------------------ */
/* To use Graphics Display Truly 3.2" 320x240 Board (AC164127-4)      */
/* with TFT-G240320LTSW-118W-E display panel                          */
/* ------------------------------------------------------------------ */
#include "Alternative Configura-
tions\HardwareProfile_GFX_PICTAIL_V3_8PMP_SSD1926_TFT_G240320LTSW_118W_E.h"
//#include "Alternative Configura-
tions\HardwareProfile_GFX_PICTAIL_V3_16PMP_SSD1926_TFT_G240320LTSW_118W_E.h"

/* ------------------------------------------------------------------ */
/* To use Graphics Display Powertip 4.3" 480x272 Board (AC164127-6)   */
/* with PH480272T-005-I11Q display panel                              */
/* ------------------------------------------------------------------ */
//#include "Alternative Configura-
tions\HardwareProfile_GFX_PICTAIL_V3_8PMP_SSD1926_PH480272T_005_I11Q.h"
//#include "Alternative Configura-
tions\HardwareProfile_GFX_PICTAIL_V3_16PMP_SSD1926_PH480272T_005_I11Q.h"
```

6 To run code using Explorer 16 with a PIC32MX360F512L with 8-bit PMP interface (Graphics LCD Controller PICtail™ Plus SSD1926 Board jumper JP2 set to **parallel—8 Bit**) and the 4.3 inch Powertip Display, Uncomment the #include line that will load the hardware profile for Powertip display

7 To run code using Explorer 16 with a PIC32MX360F512L with 16-bit PMP interface (Graphics LCD Controller PICtail™ Plus SSD1926 Board jumper JP2 set to **parallel—16 Bit**) and the 4.3 inch Powertip Display, Uncomment the #include line that will load the hardware profile for Powertip display

8 To run code using Explorer 16 with a PIC32MX360F512L with 16-bit PMP interface (Graphics LCD Controller PICtail™ Plus SSD1926 Board jumper JP2 set to **parallel—16 Bit**) and the 4.3 inch Powertip Display, Uncomment the #include line that will load the hardware profile for Powertip display

Procedure to set-up Graphics LCD Controller PICtail™ Plus SSD1926 Board.



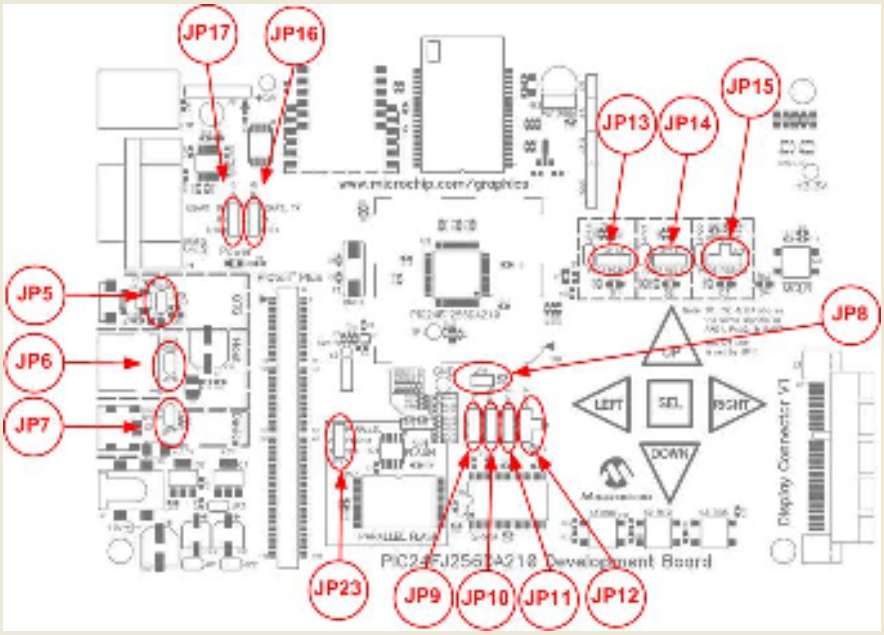
When using the Graphics LCD Controller PICtail™ Plus SSD1926 Board, jumpers on the board must be properly set.

Check that all the jumpers are set to the following positions:

- JP1 – don't care, leave it open
- JP2 – set to PARALLEL – 8-Bit
- JP3 – set to RD1 – FLASH CS

Procedure to set-up PIC24FJ256DA210 Development Board.

PIC24FJ256DA210 Development Board Jumpers



When using the PIC24FJ256DA210 Development Board jumpers on the board must be properly set.

Check that all the jumpers are set to the following positions (text in bold are not default settings of the board):

- JP8 – closed
- JP9 – set to 1 – 2
- JP10 – set to 1 – 2
- JP11 – set to 1 – 2
- JP12 – set to 2 – 4
- JP13 – set to RG8 – S1
- JP14 – set to **RE9 – PAD2**
- JP15 – set to RB5 – POT(RB5)
- JP17 – set to USART_RX – RX
- JP18 – set to USART_TX – TX
- JP23 – set to PMCS2 – SPI
- JP5, JP6 and JP7 – don't care

HardwareProfile.h

```
    /*********************************************************************
     * Hardware Configuration for
     * Starter Kit
     * MultiMedia Development Board
     * Display TFT-G240320LTSW-118W-E
     ********************************************************************/
    //#include "Alternative Configura-
tions\HardwareProfile_MULTI_MEDIA_BOARD_DM00123_16PMP_PIC32_STK_SSD1926_TFT_G240320LTSW_118W_E.h"
    //#include "Alternative Configura-
tions\HardwareProfile_MULTI_MEDIA_BOARD_DM00123_16PMP_PIC32_USB_STK_SSD1926_TFT_G240320LTSW_118W_
E.h"
    #include "Alternative Configura-
tions\HardwareProfile_MULTI_MEDIA_BOARD_DM00123_16PMP_PIC32_ENET_STK_SSD1926_TFT_G240320LTSW_118W
_E.h"

    //#include "Alternative Configura-
tions\HardwareProfile_MULTI_MEDIA_BOARD_DM00123_8PMP_PIC32_STK_SSD1926_TFT_G240320LTSW_118W_E.h"
    //#include "Alternative Configura-
tions\HardwareProfile_MULTI_MEDIA_BOARD_DM00123_8PMP_PIC32_ENET_STK_SSD1926_TFT_G240320LTSW_118W_
E.h"
    //#include "Alternative Configura-
tions\HardwareProfile_MULTI_MEDIA_BOARD_DM00123_8PMP_PIC32_USB_STK_SSD1926_TFT_G240320
LTSW_118W_E.h"
```

9

9 To run code using Multimedia Expansion Board with PIC32 Ethernet Starter Kit with a PIC32MX795F512L with 16-bit PMP interface.

Procedure to set-up PIC32 Ethernet Starter Kit and ♂Multimedia Expansion Board

The PIC32 Ethernet Starter Kit contains everything needed to begin development for Ethernet and USB Host/Device/OTG applications with the PIC32 Microcontroller family.

The Multimedia Expansion Board is an integrated yet flexible solution for development of high impact User Interfaces. The board comes with a 3.2” Color TFT touch-screen display and interfaces to any PIC32 Starter kit* .

PIC32 Starter Kit connector, which can be found on back side of Multimedia Expansion Board, can connect with PIC32 Ethernet starter kits. It can connect with PC via standard A to mini B USB cable for debugging and programming..The starter kit can be powered by USB.

Lab Exercise 1

Creating a Splash Screen

Purpose

In this first lab, you will create a Splash Screen for our GUI. The **Graphics Resource Converter** tool will be used to generate font and bitmap image files for use in this project. You will learn how to integrate the generated font and image files into your project. Additionally, you will learn to use some of the basic drawing shapes and attributes that come with the library. We will be working in the Lab1.c file included in the Lab 1 project. For your convenience, a full solution is provided in **class folder**.

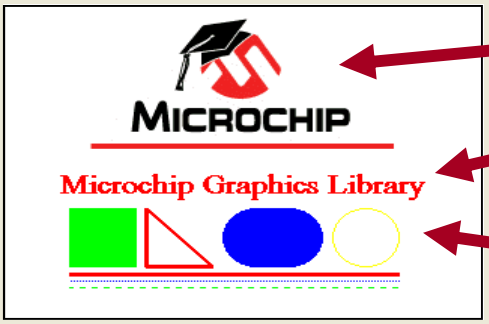
Requirements

Development Environment:	MPLAB® v8.60 or newer
Software:	Microchip Graphics Library v2.10 Graphics Resource Converter
C Compiler:	MPLAB® C Compiler for PIC24 and dsPIC OR MPLAB® C Compiler for PIC32
Hardware Tools:	Explorer 16 with PIC24FJ128GA010 or PIC32MX360F512L PIM and Graphics LCD Controller PICtail™ Plus SSD1926 Board or PIC24FJ256DA210 Development Board MPLAB® Real ICE or MPLAB® ICD3 Debugger
Lab files on class PC:	C:\...\Lab1…

Objectives

1) Using the Graphics Resource Converter, convert bitmap and font into structures the Graphics Library can use.
2) Write a program that will use the converted bitmaps and fonts along with a variety of primitive shapes to create and display the splash screen shown below.

Touchscreen and erase animation functions have been provided so that at the completion of this lab you will be able to touch the screen to display a new string.



Logo bitmap file (mchpLogo.bmp)

Text string using DoulosSILR true type font

Drawings using primitive layer functions

Procedure

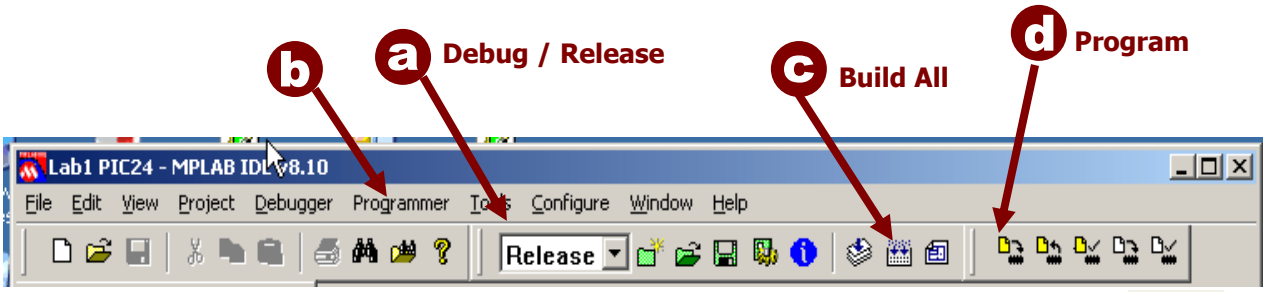
1 If you still have a previous project open, you must first close it by selecting from the menu:
File ▸ Close Workspace

Then, open Lab 1 by selecting from the menu:
File ▸ Open Workspace...and opening the workspace file appropriate for you development board setup located at:

C:\...\Lab1

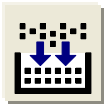
Choose:
Lab1 Ex 16 PIC24 Truly 3.2.mcw for the Explorer 16 with PIC24FJ128GA010 PIM or ...
Lab1 DA210 Truly 3.2.mcw for the PIC24FJ256DA210 Development Board or...
Lab1 Ex 16 PIC32 Truly 3.2.mcw for the Explorer 16 with PIC32MX360F512L PIM

2 To verify your setup, compile the project and program the microcontroller.

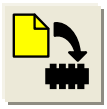


a Select **Release** mode.

c Click on the **Build All** button.

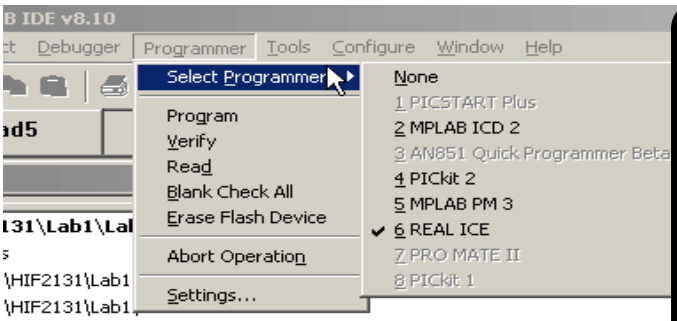


d If no errors are reported, click on the **Program** button.



b Select the programmer by selecting from the menu:

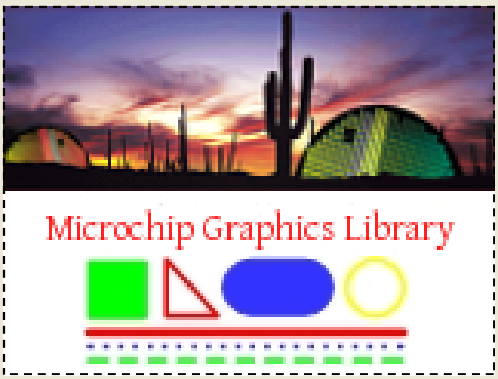
Programmer ▸ Select Programmer ▸ MPLAB® ICD 3 or **REAL**



Some of the graphics boards may require touchscreen calibration. Please complete the calibration procedure following instructions on the LCD panel to insure correct operation throughout the labs. If necessary, you may manually enter the calibration mode by holding switch S3 (for Explorer 16) or S1 (for PIC24FJ256DA210 Dev Board) as you press and release MCLR.

If your set up is correct, you will see this screen.

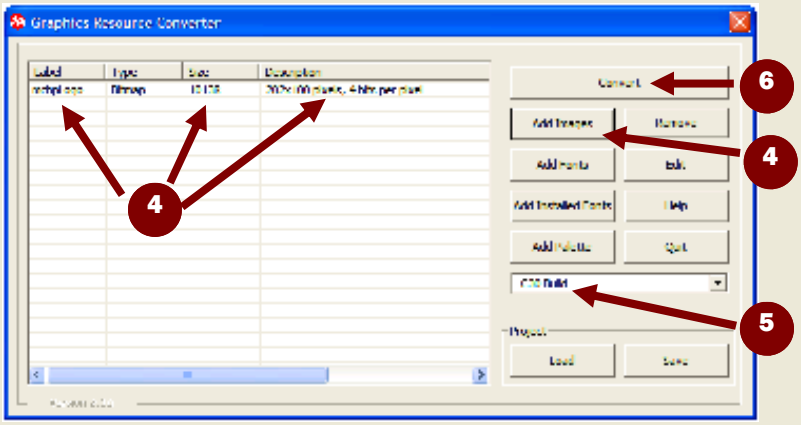
What you see on the screen is a 16 bpp bitmap stored in a 160 x 120 pixel format. To display the image over the full screen width, we specified the IMAGE_X2 stretch factor in the PutImage API.



3 To begin, we will replace the Silicon Desert portion of the image shown above with a Microchip logo. To do this, you must first transform the logo bitmap file to a usable format using the Bitmap and Font conversion tool. Launch the tool using:

C:\...\Graphics Resource Converter v2.10 ▸ Graphics Resource Converter.exe

Bitmap File Path:
C:\...\Bitmaps
mchpLogo.bmp



4 Load the bitmap file mchpLogo.bmp by clicking **Add Images** in the tool.
Verify that image size is 202x100 pixels and color depth is 4 bpp. Also verify the label reads mchpLogo.

5 For PIC24, set the build to C30.
For PIC32, set the build to C32.

If you select the wrong build option for your project, you will get compiler errors later.

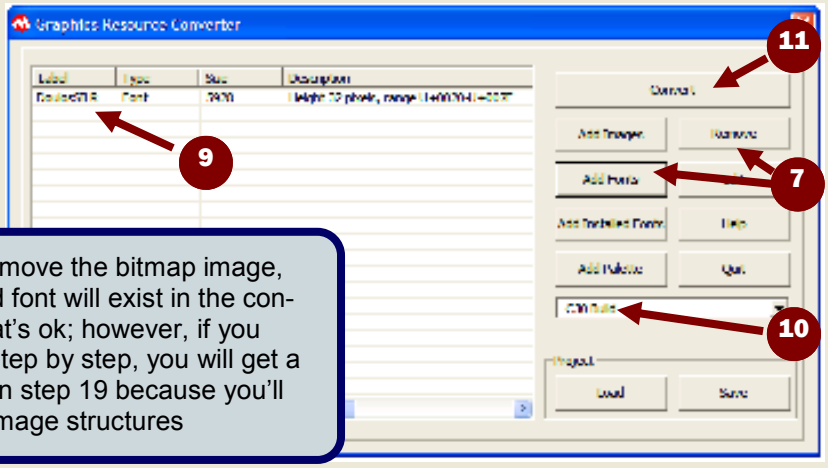
6 Click **Convert**.

Save file as type "Array in Internal Flash.c"
Save converted File to : C:\...\Lab1\NewIntro.c

DO NOT CLOSE THE CONVERTER. WE WILL USE IT AGAIN IN THE NEXT STEP.

7 Highlight the mchpLogo and click remove. Now, we will convert the font we wish to use for this project. Click **Add Fonts** in the converter and navigate to the **DoulosSILR.ttf** file.

Font Source Path:
C:\...Fonts
DoulosSILR.ttf



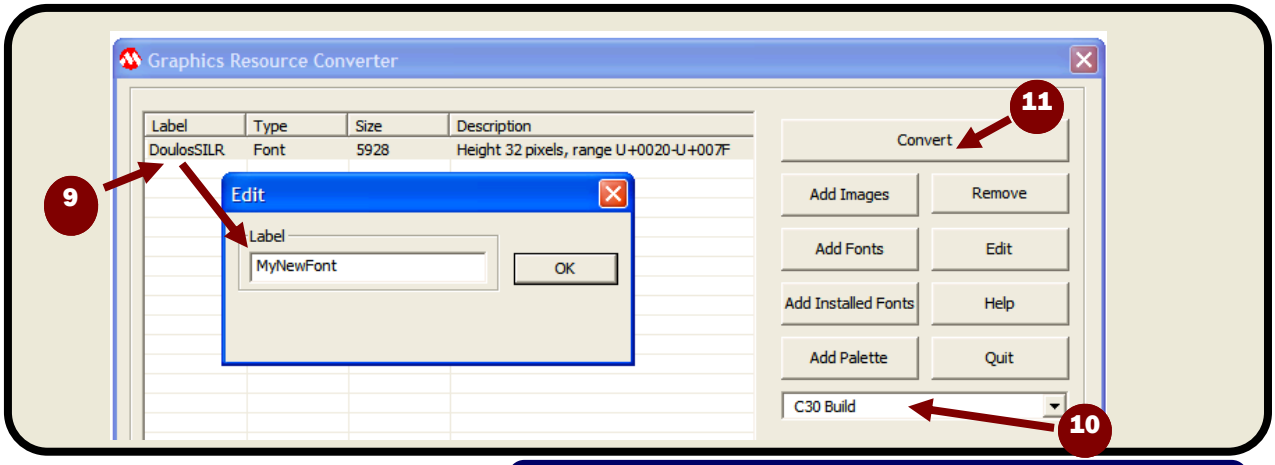
If you do not remove the bitmap image, both image and font will exist in the converted file. That's ok; however, if you follow this lab step by step, you will get a compiler error in step 19 because you'll end up with 2 image structures

8 In the Set Font Style window, select:

1. **Size set to 12**
2. **Weight set to NORMAL**
3. **Range set to (32-127)**
4. **Script set to Western**
5. **Click OK.**

You may limit the size of your font structure by limiting the character range. For instance, if you will only use lower case letters, change the range to 97-122.

9 Rename the **DoulosSILR** font to **MyNewFont** by double clicking the label field.



10 For PIC24, set the build to C30.
For PIC32, set the build to C32.

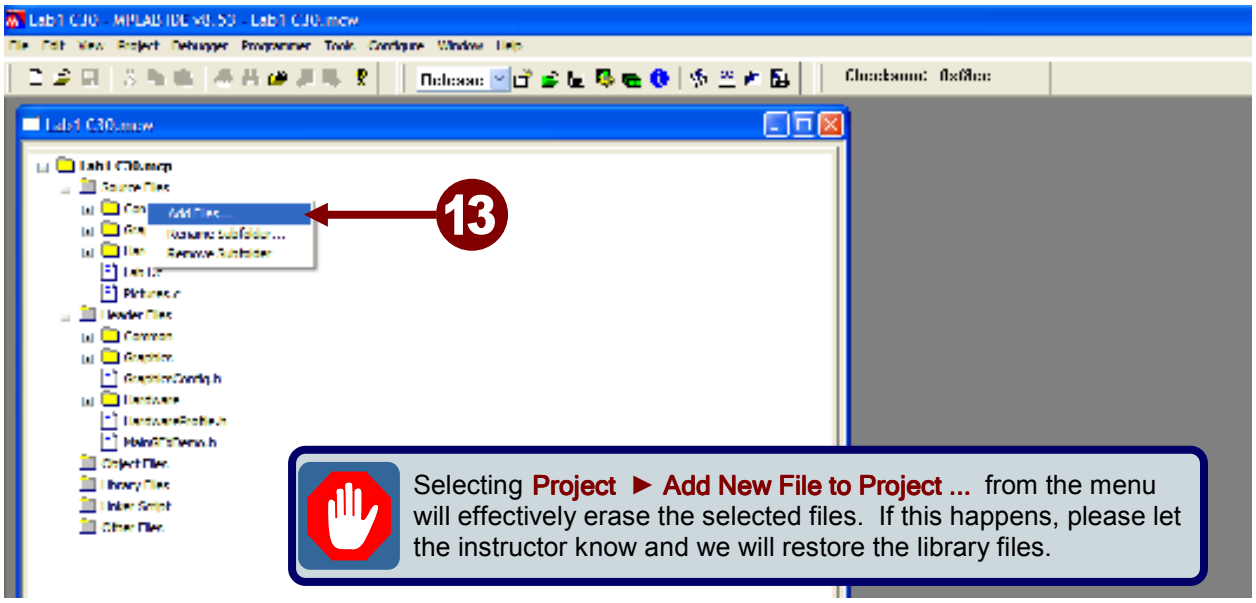
If you select the wrong build option for your project, you will get compiler errors later.

11 Click **Convert**.

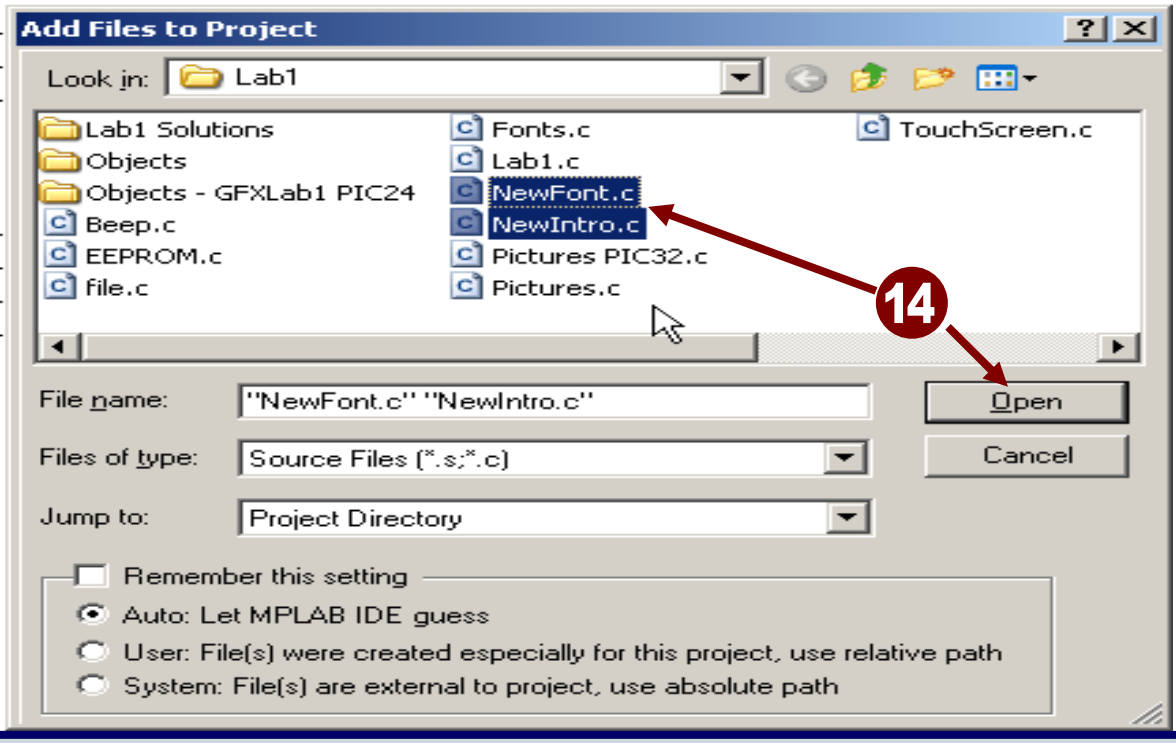
Save file as type "Array in Internal Flash.c"
Save converted File to : C:\...\Lab1\NewFont.c

12 Click **Quit** to exit the Graphics Resource Converter tool.

13 Both our image and font source files have been successfully converted. The next step is to add the converted files into the Lab 1 project. In the MPLAB® project window, add the files to the project by right clicking source files, then select Add Files from the drop down menu. Alternatively, you may select **Project ▸ Add Files to Project ...** from the menu bar.



14 From the Lab 1 directory, select **NewFont.c, NewFont.h, NewIntro.c, and NewIntro.h** then click open.



Since we no longer need the Pictures.c (or Pictures32.c) file, please remove it from the project. To do so, highlight the file then right click to select "Remove File". Alternatively, once the file is highlighted, you may hit the delete key. If using PIC24, failure to remove this file will result in a linker error indicating you need the large code

We are now ready to begin writing our program. To keep it simple, you will simply modify the Lab 1 code as instructed in the following steps. We begin by declaring the font and bitmap structures.

15 **On line 90, change the image name to mchpLogo**

```
extern const BITMAP_FLASH intro;  //### <--- Change image name here ###    15
```

16 **On line 95, add a declaration statement to declare MyNewFont as an extern FONT_FLASH type.**

```
// ### Your Code Here ###          //### <--- Declare MyNewFont here ###    16

extern const FONT_FLASH GOLFontDefault;          // Default GOL font.
```

On line 276, locate the StartScreen() function. In this function we will write the necessary code to display our new splash screen. We will compile and program several times in this process.

17 **Starting on line 289, insert code to set the screen background color to white.** Note that the value WHITE is a pre-defined color in the LCD driver header file.

Step 17 Reference Information

SetColor Macro — Sets the current drawing color

```
#define  SetColor(color)
```

ClearDevice Function — clears the screen with the current drawing color

```
void ClearDevice(void);
```

Reference Information

The colors named below have been defined in the driver header file:

BLACK	BRIGHTBLUE	BRIGHTGREEN	BRIGHTCYAN
BRIGHTMAGENTA	BRIGHTRED	BRIGHTYELLOW	BLUE
GREEN	CYAN	RED	MAGENTA
BROWN	LIGHTGRAY	DARKGRAY	LIGHTBLUE
LIGHTGREEN	LIGHTCYAN	LIGTHRED	LIGHTMAGENTA
YELLOW	WHITE	GRAY0	GRAY1
GRAY2	GRAY3	GRAY4	GRAY5

using the `RGB565Convert(R,G,B)` macro. You'll need to provide values for Red, Green, and Blue intensity levels as an 8-bit integer value with 255 being maximum intensity. For example:

```
#define ORANGE RGB565Convert(255, 127, 0)
```

will create a color with the name ORANGE.

18 **Starting on line 291, change the code to replace the intro image with the mchpLogo image.**

a. Comment out call to PutImage(…) API.
b. Insert code to calculate the x coordinate for the new image such that the logo will be centered horizontally on the screen. HINT: (Maximum Screen Width - Image Width) / 2 will give the horizontal center point. Store the result in x_image variable.
c. Insert code to display the image on the screen at position (x, y) = (x_image, 10) using a NORMAL stretch factor. Where x is the left position and y is the top value. HINT: Use PutImage(…) API with image name mchpLogo.

Step 18 Reference Information

GetMaxX() Macro — Returns the maximum horizontal screen value

```
#define GetMaxX()
```

GetImageWidth Function — Returns the image width

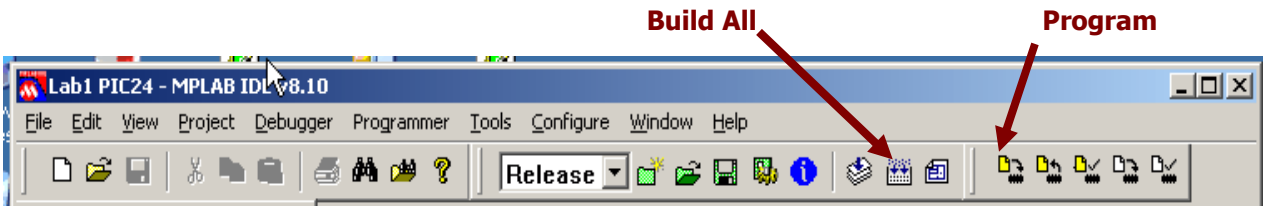
```
SHORT GetImageWidth( void* bitmap );
```

PutImage Function — Displays the image starting from the left, top coordinates

```
void PutImage( SHORT left, SHORT top, void* bitmap, BYTE stretch);
```

Valid values for stretch are IMAGE_X2 and IMAGE_NORMAL.

19 Compile the project and program the device. Verify that your image is shown in the expected location.



If all went well, you will see the following on your screen.

20 **Starting on line 298, insert code required to display a pre-defined string stored in text[].** The string should be BRIGHTRED, use MyNewFont, and positioned in the horizontal center at y=120.

a. Set the current color to BRIGHTRED (a predefined color value).
b. Set the font image to MyNewFont.
c. Insert code to calculate the x coordinate for the string such that the string will be centered horizontally on the screen. HINT: (Max X—String Width) / 2 will give the horizontal center point. Store the result in x_text variable.
d. Insert code to display text[] on the screen using coordinates x_text, 120. HINT: OutTextXY(…)

Step 21 Reference Information

SetColor Macro — Sets the current drawing color

```
#define  SetColor(color)
```

SetFont Function — Sets the current font used in OutTextXY(), OutText(), and OutChar() functions.

```
void SetFont( void* font);
```

GetMaxX() Macro — Returns the maximum horizontal screen value

```
#define GetMaxX()  (SCREEN_HOR_SIZE–1)
```

GetTextWidth Function — Returns the width of the specified string using the specified font.

```
SHORT GetTextWidth( XCHAR* textstring, void* font);
```

OutTextXY Function — Displays a null terminated string of characters starting at given x,y position.

```
WORD OutTextXY( SHORT x, SHORT y, XCHAR* textstring);
```

This function uses the current active font.

For non-blocking configurations, **OutTextXY()** may return control to the program if the display is busy. When this happens, a 0 is returned and **OutTextXY(..)** must be called again to continue outputting the string. For blocking configurations, this function always returns a '1'.

21 Compile the project and program the device.
Verify that your image and strings are displayed in the expected locations.

If all went well, you will see this on your screen.

21

Your final task is to draw a couple of shapes on the bottom of the screen using primitive APIs. The other shapes are already pre-coded for your reference.

22 **Starting on Line 305 in Lab1.c, insert code to display a BRIGHTGREEN filled box at location: left = 44, top = 155, right = 88, bottom = 199.** NOTE: BRIGHTGREEN is a defined color.

23 **Starting on Line 308 in Lab1.c, insert code to display a BRIGHTYELLOW unfilled circle at location: x = 241, y = 177 with a radius = 22.** NOTE: BRIGHTYELLOW is a defined color.

Step 22-23 Reference Information

SetColor Macro — Sets the current drawing color

```
#define  SetColor(color)  _color = color //color coded in 5:6:5 RGB Format
```

Bar Function — Draws a bar given left, top, right, bottom corners using the current color.

```
void Bar( SHORT left, SHORT top, SHORT right, SHORT bottom);
```

Circle Macro — Draws a circle with the given center and radius using the current color.

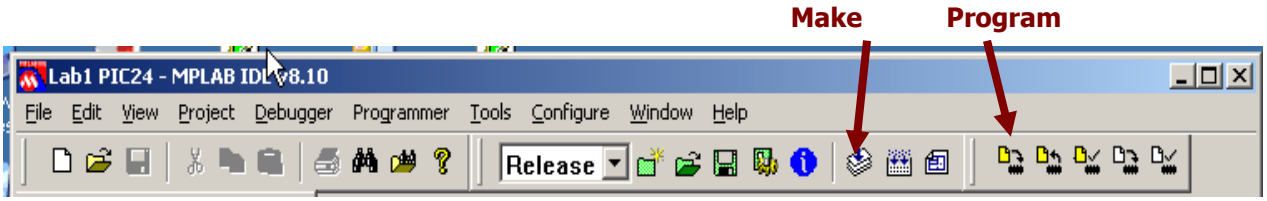
```
#define Circle( x, y, radius )  Bevel( x, y, x, y, radius)
```

NOTE: x, y, and radius are of SHORT type.

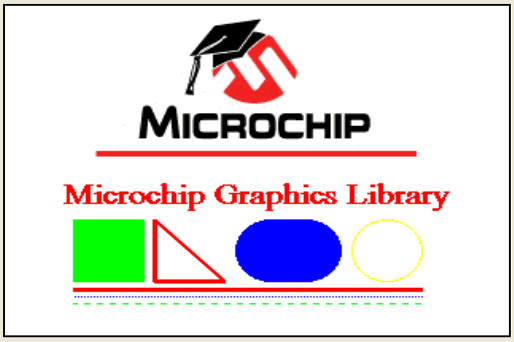
The Bar() and the Circle() APIs will return a 1 when the primitive is successfully drawn on the screen. If the USE_NONBLOCKING_CONFIG option is enabled, the primitive functions may return even if it has not drawn the primitive completely. To force the rendering recursively call the functions until they return a 1. Example: `while(!Bar(x1,y1,x2,y2));`

24 **On line 278 in Lab1.c, display the other shapes by turning on the compiler switch.**

25 Compile the project and program the device.
Verify that your splash screen matches the expected results shown on page 1-1.



Touch the screen.
Did the surprise phrase change? If it did, you have now completed Lab 1. **Great Job!**



25

Results

You have just learned how to integrate images and fonts to your application with the Graphics Library. Also you have learned how to implement primitive rendering functions and control the global level drawing attributes such as color, line type, line size and fonts used. If you finish early, you can try out the various other primitive functions found in the Graphics Library Help file.

Code Analysis

The simple application that you have modified implements a touch screen module and uses primitive rendering functions to draw a simple splash screen. Each primitive function is affected by the global drawing attributes such as line type, line size and color. For texts, the font and current color settings affect the way the text appears on the screen.

Conclusions

Having learned how to use the Primitive Layer of the Graphics Library you are now ready to move upward to the next layer and learn the Graphic Object Layer that implements the Widgets. This gets you closer to fully integrating your application with a graphical solution. The next lab will give you a introduction to the Graphics Object Layer and use the Widgets that come with the library. Since the widgets in the Graphics Object layer call upon the primitive layer functions, you are now armed and ready to create your own widgets. An app note on this topic will be out soon.

Lab Exercise 2

Creating Widgets

Purpose

Now that the splash screen is complete, we will create a simple menu for our application. In this lab you will draw a few button and static text widgets. Our focus in this lab is exploring widget creation and drawing APIs. As such, touchscreen and side button inputs have been disabled. The next few labs will explore integration of user and system inputs. For your convenience, a full solution is provided in **C:\...\Lab2\Lab2 Solution**.

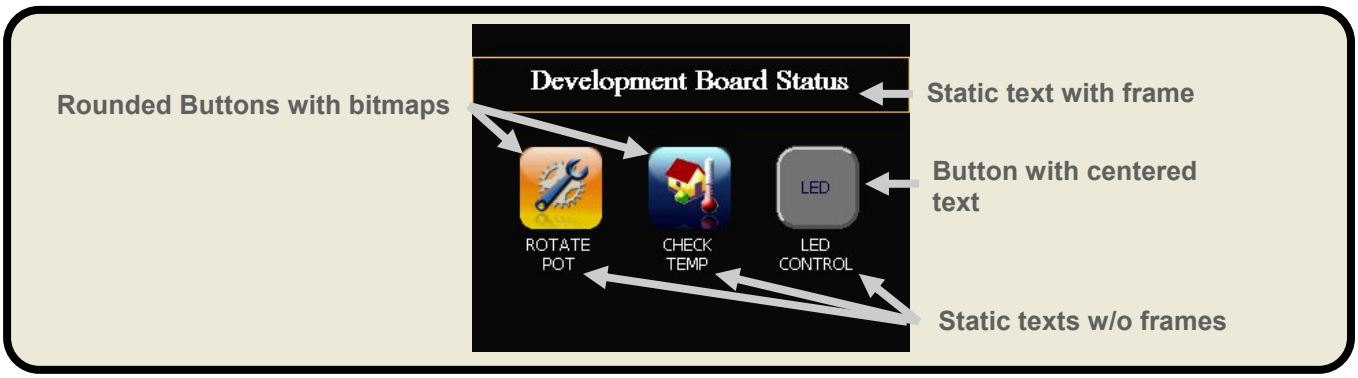
Requirements

Development Environment:	MPLAB® v8.50 or newer
Software:	Microchip Graphics Library v2.10 Graphics Resource Converter
C Compiler:	MPLAB® C Compiler for PIC24 and dsPIC OR MPLAB® C Compiler for PIC32
Hardware Tools:	Explorer 16 with PIC24FJ128GA010 or PIC32MX360F512L PIM and Graphics LCD Controller PICtail™ Plus SSD1926 Board or PIC24FJ256DA210 Development Board MPLAB® Real ICE or MPLAB® ICD3 Debugger
Lab files on class PC:	C:\...\Lab2...

Objectives

- Use GOLCreateScheme() API to establish 2 color schemes.
- Use the appropriate ObjCreate(…) APIs to create:
  - 3 Rounded buttons. One with text and Two with an image (no text).
  - 3 static texts to label the buttons. One static text with frame to label the screen.
- Use GOLDraw() API to render the widgets to the display.

Expected Results



Procedure

1 If you still have a previous project open, you must first close it by selecting from the menu:
File ▸ Close Workspace

Then, open Lab 2 by selecting from the menu:
File ▸ Open Workspace... and opening the workspace file appropriate for you development board setup located at:

C:\...\Lab 2

Choose:
Lab1 Ex 16 PIC24 Truly 3.2.mcw for the Explorer 16 with PIC24FJ128GA010 PIM or ...
Lab1 DA210 Truly 3.2.mcw for the PIC24FJ256DA210 Development Board or...
Lab1 Ex 16 PIC32 Truly 3.2.mcw for the Explorer 16 with PIC32MX360F512L PIM

The following colors have been defined in the driver layer of the Graphics Library:

BLACK	BRIGHTBLUE	BRIGHTGREEN	BRIGHTCYAN
BRIGHTMAGENTA	BRIGHTRED	BRIGHTYELLOW	BLUE
GREEN	CYAN	RED	MAGENTA
BROWN	LIGHTGRAY	DARKGRAY	LIGHTBLUE
LIGHTGREEN	LIGHTCYAN	LIGHTRED	LIGHTMAGENTA
YELLOW	WHITE	SADDLEBROWN	SIENNA

If you'd like to add your own colors, you may do so using the `RGB565Convert( )` macro. You'll need to supply values for Red, Green, and Blue in that order. For example:
`#define ORANGE RGB565Convert(255, 127, 0)`
will create the color ORANGE.

2 To use widgets in our application, we must add their source and header files to our project. In the project window, **add the following source files** to the project: **GOL.c, Button.c, and StaticText.c**

Library Files Source Directory:
C:\...\Microchip Graphics Library v2.10\Graphics

3 **Add the following header files** to the project: **GOL.h, Button.h, and StaticText.h**

Library Files Includes Directory:
C:\...\Microchip Graphics Library v2.10\Include\Graphics

WARNING! Selecting **Project ► Add New File to Project ...** from the menu will effectively erase the selected files. If this happens, please let the instructor know and we will restore the library files.

4 Additionally, we must enable the widgets we wish to use in the **GraphicsConfig.h** header file. To do so, open the file and uncomment the Button and Static Text compiler switches.

GraphicsConfig.h

```
86  /*****************************************************************
87  * Overview: To save program memory, unused Widgets or Objects can be
88  *                     removed at compile time.
89  *
90  *****************************************************************/
91  #define USE_GOL                 // Enable Graphics Object Layer.
92  #define USE_BUTTON              // Enable Button Object.
93  //#define USE_WINDOW            // Enable Window Object.
94  //#define USE_CHECKBOX          // Enable Checkbox Object.
95  //#define USE_RADIOBUTTON       // Enable Radio Button Object.
96  //#define USE_EDITBOX           // Enable Edit Box Object.
97  //#define USE_LISTBOX           // Enable List Box Object.
98  //#define USE_SLIDER            // Enable Slider or Scroll Bar Object.
99  //#define USE_PROGRESSBAR       // Enable Progress Bar Object.
100 #define USE_STATICTEXT          // Enable Static Text Object.
101 //#define USE_PICTURE           // Enable Picture Object.
102 //#define USE_GROUPBOX          // Enable Group Box Object.
103 //#define USE_ROUNDDIAL         // Enable Dial Object.
104 //#define USE_METER             // Enable Meter Object.
105 //#define USE_TEXTENTRY         // Enable TextEntry Object.
106 //#define USE_GRID              // Enable Grid Object.
107 //#define USE_CHART             // Enable Chart Object
108 //#define USE_CUSTOM           // Enable Custom Control Object
```

Now that we have properly set up the widgets for use in our application, we need to establish the Style Schemes they will use. The 3 buttons will initially use the library's default style scheme defined in **GOL.h**. The 3 static texts used to label the buttons/icons will use a different style scheme and the static text used for the title will be the same as the other 3 static text except for the font used.

5 **In the `main()` function located in Lab2.c, on line 157, add code to initialize the display, initialize the library, and create the default style scheme.** HINT: **`GOLInit()`** API will accomplish all of these tasks; therefore, only one line of code is needed.

Style Scheme—is a structure used by the library to define the colors and font used to draw widgets. Creating widgets with NULL assigned to the style scheme will assign the default style scheme to the objects.

```
typedef struct {
    WORD  EmbossDkColor;      // Emboss dark color used for 3d effect.
    WORD  EmbossLtColor;      // Emboss light color used for 3d effect.
    WORD  TextColor0;         // Character color 0 used for objects that supports text.
    WORD  TextColor1;         // Character color 1 used for objects that supports text.
    WORD  TextColorDisabled;  // Character color used when object is in a disabled state.
    WORD  Color0;             // Color 0 usually assigned to an Object state.
    WORD  Color1;             // Color 1 usually assigned to an Object state.
    WORD  ColorDisabled;      // Color used when an Object is in a disabled state.
    WORD  CommonBkColor;      // Background color used to hide Objects.
    void  *pFont;             // Font selected for the scheme.
} GOL_SCHEME;
```

The Static Text widgets will use a modified style scheme, we will need to add code to define that scheme.

6 **In the CreateScheme() function located in Screens.c, on line 77, add two lines of codes to allocate memory to the alternate scheme pointers statusbox and labelbox.** Note that the two pointers has already been defined.

Step 5 and 6 Reference Information

GOLInit() function — initializes both the display and the library then creates a default style scheme with default settings referenced by the global scheme pointer. This function MUST be called before graphics object layer (GOL) functions may be used.

```
void GOLInit(void);
```

GOLCreateScheme() function — creates a new style scheme structure with parameters initialized to default values as defined in GOL.h. Returns a GOL_SCHEME pointer to the new structure.

```
GOL_SCHEME *GOLCreateScheme(void);
```

7 For simplicity, the choice of colors for the style scheme used for the Static Text has been predefined. This is shown starting in line 82 of Screens.c file.
On line 82 in Screens.c, modify the style scheme by turning on the compiler switch.

Step 8 Reference Information

`BtnCreate()` returns the address of the newly created Widget. Make sure you assign the pointer (that is initialized to NULL) to the address of the newly created Buttons. The pointers are declared starting in line number 131.

```
// pointers to widgets on this screen
BUTTON *pOBJ_BUTTON_POT = NULL;
BUTTON *pOBJ_BUTTON_TEMP = NULL;
BUTTON *pOBJ_BUTTON_LED = NULL;
```

With the Style Schemes defined, we are now ready to start creating our widgets. We'll do this in 2 steps. First, we'll create the buttons and verify their look on the display. Then, we'll add the static texts to label the buttons and add the title of the main menu.

8 Locate the CreateMenuScreen() function on line 127 in Screens.c. **Starting on line 148, add code to create three rounded buttons using the parameters given below** (HINT: make 3 calls to the **`BtnCreate()`** function).

For your convenience, the values given below are predefined macros that can be found in Screens.h. Note that Buttons 1 and 2 uses bitmaps while Button 3 uses a text. For Button 3, the string used to label the button is already pre-defined as a static string Btn_Label.

Refer to the Graphics Library help file for the BtnCreate() API prototype.

BUTTON 1 parameters

- ID = OBJ_BUTTON_ROTATE_POT
- left = POT_STARTX
- top = POT_STARTY
- right = POT_STARTX+BTN_WIDTH
- bottom= POT_STARTY+BTN_HEIGHT
- radius = BTN_RADIUS
- state = BTN_DRAW
- *pBitmap = (void*) &Rotate_Icon
- *pText = NULL
- *pScheme = NULL

BUTTON 2 parameters

- ID = OBJ_BUTTON_CHECK_TEMP
- left = TEMP_STARTX
- top = TEMP_STARTY
- right = TEMP_STARTX+BTN_WIDTH
- bottom= TEMP_STARTY+BTN_HEIGHT
- radius = BTN_RADIUS
- state = BTN_DRAW
- *pBitmap = (void*) &Temp_Icon
- *pText = NULL
- *pScheme = NULL

BUTTON 3 parameters

- ID = OBJ_BUTTON_CONTROL_LED
- left = LED_STARTX
- top = LED_STARTY
- right = LED_STARTX+BTN_WIDTH
- bottom= LED_STARTY+BTN_HEIGHT
- radius = BTN_RADIUS
- state = BTN_DRAW
- *pBitmap = NULL
- *pText = (XCHAR*) Btn_Label
- *pScheme = NULL

To render the buttons in the screen we need to call an API that takes care of the rendering of the widgets in the screen.

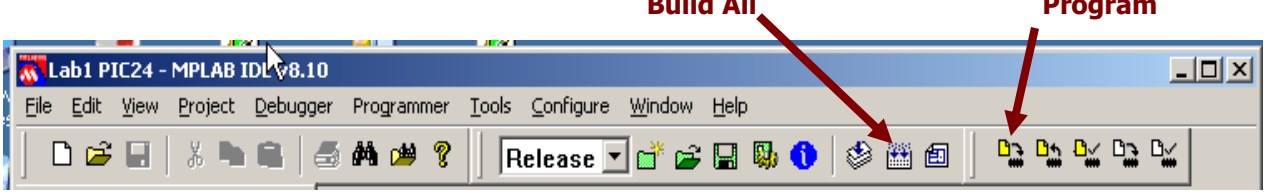
9 Locate the main() in Lab2.c file. **Add a line of code within the application while loop (line 185) to enable the rendering of the widgets to the screen.**

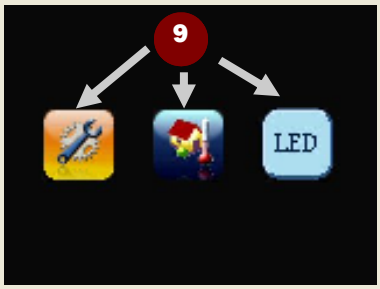
Step 9 Reference Information

GOLDraw() function — loops through the active linked list and redraws widgets that need to be redrawn. Partial or full redrawing is determined by the state field in the widget structure. Returns TRUE when drawing of all widgets is complete.

```
WORD GOLDraw(void);
```

10 **Compile the project and program the device. Verify that your buttons are in the expected locations and use the correct style scheme.**





If you forgot to enable the widgets in the GraphicsConfig.h file, the compiler will give errors related to BTN_DRAW.

If you forgot to add the widget source and header files, you will get linker errors.

Notice that the LED button is using the default style scheme. Normally, users will not redefine the default style scheme. Instead create an alternate style scheme just like the two static text style schemes that was allocated memory in step 6. For simplicity the new colors and font used for the LED button has been predefined by the pointer “flatbuttons”.

11 **On line 111 of Screens.c, enable the “flatbuttons” style scheme for the LED button by turning on the compiler switch.**

12 **On line 198 of Screens.c , modify the LED button style scheme to use “flatbuttons” style scheme using the GOLSetScheme() API.**

Step 12 Reference Information

GOLSetScheme() macro — sets the style scheme to be used for the specified wiget.

```
#define GOLSetScheme(pObj, pScheme)
```

Since GOLSetSCheme() call may change the font used (font sizes can be different), it is necessary to call the BtnSetTxt() function to force the recalculation of the width and height of the string used. This is already added in the code. Uncomment the line to enable the code.

Now that the buttons are properly defined and positioned, let’s add the 3 static texts to label the buttons. The style scheme of our static texts are already predefined and enabled in Step 7.

Step 13 Reference Information

`StCreate()` returns the address of the newly created Widget. Make sure you assign the pointer (that is initialized to NULL) to the address of the newly created Static Texts. The pointers are declared starting in line number 135.

```
STATICTEXT *pOBJ_STATICTEXT_TITLE = NULL;
STATICTEXT *pOBJ_STATICTEXT_POT = NULL;
STATICTEXT *pOBJ_STATICTEXT_TEMP = NULL;
STATICTEXT *pOBJ_STATICTEXT_LED = NULL;
```

13 To create the static texts, we will make 3 calls to the StCreate() function. **Starting on line 205 in Screens.c file, add code to create three static text instances using the parameters given below** For your convenience, the values given below are predefined macros that can be found in GFXLab.h. Refer to Graphics Library Help file for StCreate() prototype.

STATIC TEXT 1 parameters

- ID = OBJ_STATICTEXT_POT
- left = POT_STARTX
- top = POT_STARTY+BTN_HEIGHT+5,
- right = POT_STARTX+BTN_WIDTH
- bottom = POT_STARTY + (BTN_HEIGHT*2)+ 5
- state = ST_CENTER_ALIGN | ST_DRAW
- *pText = "ROTATE\nPOT"
- *pScheme = labelbox

STATIC TEXT 2 parameters

- ID = OBJ_STATICTEXT_TEMP
- left = TEMP_STARTX
- top = TEMP_STARTY+BTN_HEIGHT+5,
- right = TEMP_STARTX+BTN_WIDTH
- bottom = TEMP_STARTY + (BTN_HEIGHT*2)+ 5
- state = ST_CENTER_ALIGN | ST_DRAW
- *pText = "CHECK\nTEMP"
- *pScheme = labelbox

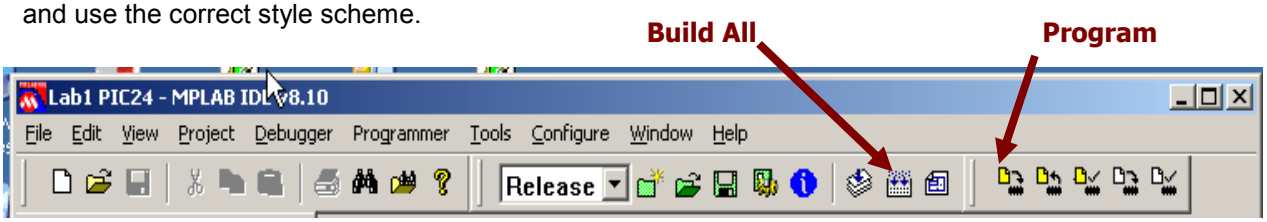
STATIC TEXT 3 parameters

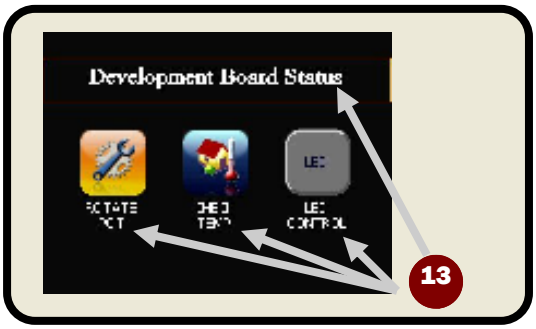
- ID = OBJ_STATICTEXT_LED
- left = LED_STARTX
- top = LED_STARTY+BTN_HEIGHT+5,
- right = LED_STARTX+BTN_WIDTH
- bottom = LED_STARTY + (BTN_HEIGHT*2)+ 5
- state = ST_CENTER_ALIGN | ST_DRAW
- *pText = "LED\nCONTROL"
- *pScheme = labelbox

Lastly, enable the code to include the creation of the Title Static Text. Note that this static text is created with the frame enabled (ST_FRAME state bit is set).

14 **On line 237 in Screens.c, enable the title static text turning on the compiler switch.**

15 Compile the project and program the device. Verify that your static texts are in the expected locations and use the correct style scheme.





If you forgot to enable the widgets in the GraphicsConfig.h file, the compiler will give errors related to ST_DRAW .

If you forgot to add the widget source and header files, you will get linker errors.

In step 13 you have set the static text alignment to be center aligned (ST_CENTER_ALIGN). This is an example of a state setting of a widget. Each type of widget will have its unique state settings. For the title static text, another state setting is also enabled. This is the frame of the static text (ST_FRAME).

There are two ways to set the state settings of the widget. The first is to set it at creation. In steps 8 and 11, you have used two create functions (StCreate() and BtnCreate()). The second, is to use the state setting API SetState(). Please refer to the Graphics Library Help file for details on this API and the respective widget documentation for details on state bits.

The action and appearance of a widget are defined by it's state field. We can modify the state after creation using the SetState() macro. State bits are bitwise ORd so that changing one bit will not affect the others. The macros for the state bits are defined in the widget header file. You may find more information in the Graphics Library help file.

Bonus Procedure

Play time! In step 12 you have modified the style scheme of the LED button widget using the API GOLSetScheme().

For this bonus step, use the same API GOLSetScheme() function to assign the "flatbuttons" style scheme to OBJ_STATICTEXT_LED. But instead of placing the function call in Screens.c place it in the main function located in Lab2.c file. Place the code in line 180.

HINT: Use the API GOLGetScheme() to get the address of the style scheme and use it to set the style scheme of the other widget.

Bonus Step Reference Information

GOLFindObject() function — finds a widget in the active linked list using the given object ID and returns a pointer to the widget.

```
OBJ_HEADER* GOLFindObject(WORD ID);
```

GOLSetScheme() macro — sets the style scheme to be used for the specified wiget.

```
#define GOLSetScheme(pObj, pScheme)
                    ((OBJ_HEADER *)pObj)->pGolScheme = pScheme
```

GOLGetScheme() macro — finds a widget in the active linked list using the given object ID and returns a pointer to the widget.

```
#define GOLGetScheme(pObj)
                    ((OBJ_HEADER *)pObj)->pGolScheme
```

Results

You have just learned how to use the Widgets in the Graphics Library. You have learned how multiple objects and with one single function call manage the rendering of the objects in the screen. You have also learned how to enable available features of the Widgets using its states. You have also learned how to modify the style schemes assigned to objects to change the appearance of the objects when drawn.

Code Analysis

The code that you have modified created separate objects and with GOLDraw() leave the management of rendering of objects to the library functions. Using the state bits you can modify how an object is rendered. Also with the style scheme you can easily modify the color values or font to change the style scheme assigned to the object.

Conclusions

Having learned how to use the Objects in the Graphics Library you are now ready to move upward and learn how to modify the behavior of the objects depending on the user inputs such as touch screen.

THIS PAGE INTENTIONALLY LEFT BLANK

Lab Exercise 3

Interfacing the User

Purpose

The purpose of this lab is to learn to implement a simple user interface by implementing widgets to control the LEDs in the Explorer 16 board or the PIC24FJ256DA210 Development Board. In this lab, you will learn how to provide system and widget control in response to user inputs like the touch screen interface.

Requirements

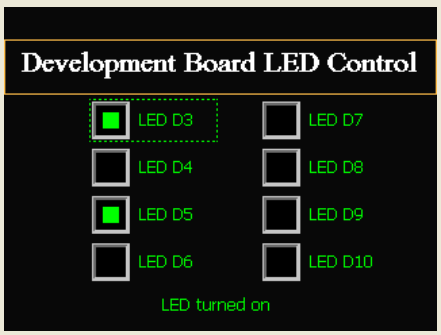
Development Environment:	MPLAB® v8.50 or newer
Software:	Microchip Graphics Library v2.10 Graphics Resource Converter
C Compiler:	MPLAB® C Compiler for PIC24 and dsPIC OR MPLAB® C Compiler for PIC32
Hardware Tools:	Explorer 16 with PIC24FJ128GA010 or PIC32MX360F512L PIM and Graphics LCD Controller PICtail™ Plus SSD1926 Board or PIC24FJ256DA210 Development Board MPLAB® Real ICE or MPLAB® ICD3 Debugger
Lab files on class PC:	C:\...\Lab3...

Objectives

1) Provide customized control for both the system and widgets via the GOLMsgCallback() function.
2) Modify behavior of one widget based on a state of another widget.
3) Modify behavior of a system resource (such as LEDs) based on a state of a widget.

Expected Results

Each Checkbox will control an LED



For Explorer 16 Board all LEDs (D3-D10) are enabled

For PIC24FJ256DA210 Development Board LEDs (D2 & D4) are enabled

Procedure

If you still have a previous project open, you must first close it by selecting from the menu:
File ▸ Close Workspace

Then, open the Lab 3 project by selecting from the menu:
File ▸ Open Workspace... and opening the workspace file appropriate for you development board setup located at:

C:\...\Lab 3

Choose:
Lab1 Ex 16 PIC24 Truly 3.2.mcw for the Explorer 16 with PIC24FJ128GA010 PIM or ...
Lab1 DA210 Truly 3.2.mcw for the PIC24FJ256DA210 Development Board or...
Lab1 Ex 16 PIC32 Truly 3.2.mcw for the Explorer 16 with PIC32MX360F512L PIM

In this lab exercise we will use the check box and static text widgets. The widgets are already declared and created in the function named CreateLEDScreen() in the file Screens.c. In order for the user to interact with the check box widgets the resistive touch screen overlaid on the display is used.

The touch screen driver (TouchScreen.c & TouchScreen.h) implements the sampling of the user touch. Sampling is done by the function TouchProcessTouch(). This function sets up the 2 analog signals and 2 digital signals on the touch screen and uses the A/D module to obtain the actual samples. The way the lab code is set up, is that the Timer interrupt calls the TouchProcessTouch() to update the touch positions variable in the touch screen driver.

The application can check if there is a valid user touch by calling the `TouchGetMsg()` function. This function populates the message structure GOL_MSG. That message in turn gets processed in the library by calling the `GOLMsg()` function. The library evaluates the message and determines which widget is affected by the message. This is done by parsing through the linked list of widgets.

The message structure is used to pass event information in a prescribed format to the library's message manager (i.e. GOLMsg() function). The format of the structure for a touch screen inputs is:

```
typedef struct {
    BYTE type,    // input type TYPE_TOUCHSCREEN for touch screens
    BYTE uiEvent, // EVENT_PRESS, EVENT_RELEASE,
                  // EVENT_STILLPRESS, EVENT_MOVE or EVENT_INVALID
    SHORT param1, // The x position of the event
    SHORT param2  // The y position of the event
} GOL_MSG;
```

Another great reference on the use of messaging is explained in Application Note AN1227, Using a Keyboard with the Microchip Graphics Library. In this case keyboard or side button style input is used for interfaces.

The affected widget then passes a translated message on the GOLMsgCallback() function. This function is implemented in the application and provides the opportunity for the application to interpret the message (GOL_MSG) and the translated message. Based on that interpretation, the application can then modify some system states, update system variables and widget states.

The **GOLMsgCallback()** function is a user defined function called by the library's message manager (**GOLMsg()**) after the passed message has been translated. Within this function, the programmer may add or modify the widgets default behavior depending on the received message.

```
WORD GOLMsgCallback (WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg);
```

Input:
objMsg — this is the translated message
***pObj** — pointer to the widget affected by the event
pMsg — pointer to the message structure

Return:
1 — to have the message manager complete the widget's default actions
0 — to skip the default actions

In this case we modify the LED's of the system to reflect the state of the check boxes. This is done on the first portion of lab3. In the second portion we modify other widgets based on the state of another widget.

1. **Open the Lab3.c** file. **On line 183, locate the main while() loop**. Inside the `if GOLDraw() { ... }` routine **add the function call to retrieve the touch message from the touch screen module**. **Add the code on line 187**.

2. On **line 190 add the code** to invoke the library's message manager.

What is the advantage of waiting for GOLDraw() to complete before calling the message manager?

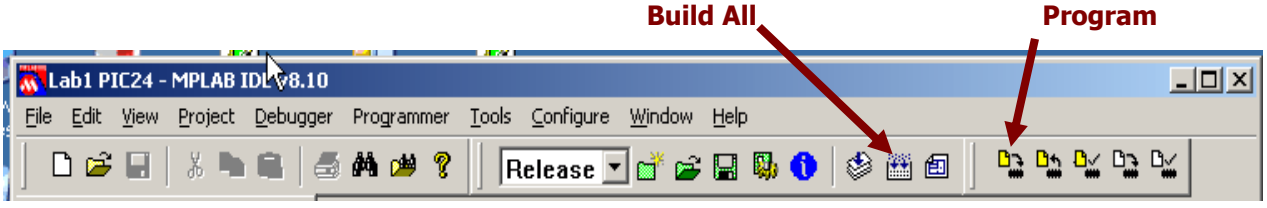
Steps 1 and 2: Reference Information

Any user interface can provide a function that will populate a message structure. Some common examples are side buttons, keyboard, touch pads and touch screen. Application in turn calls this function with the pointer to the message. The message is used to pass the user event information to the library's message manager (i.e. GOLMsg() function).

void TouchGetMsg(GOL_MSG *pMsg) — function that retrieves the touch message from the touch screen module. A pointer to the message structure must be passed to this function.

void GOLMsg(GOL_MSG *pMsg);
Where *pMsg is a pointer to the message structure

3. **Compile the project and program the device. Verify that the check boxes can be controlled by the touch screen.**



Information regarding a widget's default behavior is located in the Graphics Help file under the ObjMsgDefault function description. By default, the appropriate draw bits will be set for the widget receiving the message. NOTE: Not all widgets accept messages by default. For instance, Static Text does not accept messages; however, we can control the widget by adding code to the `GOLMsgCallback()` function.

Using the check box states, we will control the illumination of the LEDs on the board. Also notice that there is a text at the bottom of the screen that says "No Event". This is a static text widget and we want to use this widget to reflect the events that occur on the check boxes.

The static text string will be modified using the following conditions:

- Whenever a text box is checked, the string displayed on the static text should be "LED turned on".
- Whenever a text box is unchecked, the string displayed on the static text should be "LED turned off".

There are many ways to implement this. One way is to use the translated message to determine if the check box will be checked or unchecked.

In the `GOLMsgCallback()` function, a switch statement is used to set the variable **`whichLED`** to indicate which check box received the msg. The variable **`whichLED`** is set accordingly whenever there is an event on the check boxes. Recall, pObj is a pointer to the widget affected by the touch event. It isprovided everytime GOLMsgCallback() is called when there is a valid event on a widget.

4. **Locate** the function **GOLMsgCallback()** on Lab3.c. **On lines 267** and **277**, objMsg is used to determine if the translated message is **CB_MSG_CHECKED** and **CB_MSG_UNCHECKED**.

5. **On line 270 add** the code to turn on the LED by calling SetLED() and using the variable whichLED.

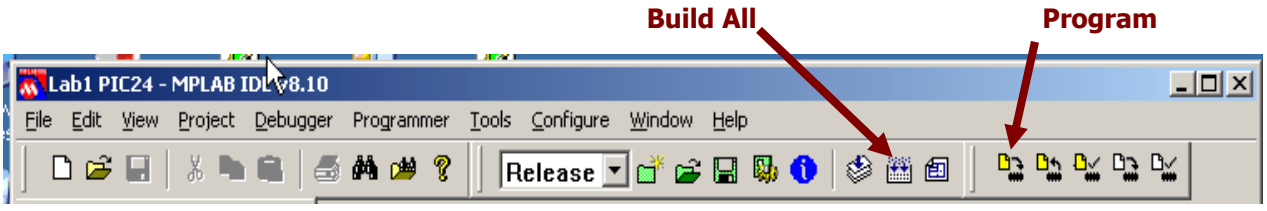
6. **On line 283 add** the code to turn off the LED by calling SetLED() and using variable whichLED.

Steps 5 and 6 Reference Information

SetLED() is a predefined function in the Lab3.c that sets the LED on the Explorer 16 or PIC24FJ256DA210 Development Board. status (LED_OFF, LED_ON) turns on or off the LED specified by whichOne.

```
void SetLED(BYTE status, BYTE whichOne)
```

7. **Compile the project and program the device. Verify that the LEDs can be controlled by the state of the check boxes.**
Note:
   - **For Explorer 16 all 8 LEDs can be controlled.**
   - **For PIC24FJ256DA210 only LEDs D2 and D4 are controlled.**



On the Explorer 16, the eight LEDs (D3-D10) are connected to PORTA of the processor. To use the LEDs, jumper JP2 must be installed.

On the PIC24FJ256DA210 Development Board 2 LEDs (D2 and D4) are connected to PORTE9 and PORTA7 respectively. To use the 2 LEDs, jumper JP14 must be set to RE9-S2 and jumper JP11 must be set to 1-2.

All ports used for LED are initialized for you in the `InitLED()` function located in Lab3.c

Now implement the code to modify the static text.

8 **On line 274 add** the code to change the string displayed on the static text to "LED turned on". On the next line add the code that will redraw the static text.

9 **On line 288 add** the code to modify the string displayed on the static text to "LED turned off". On the next line add the code that will redraw the static text.

Step 8 and 9 Reference Information

StSetText() function — Sets the string displayed on the referenced static text.

```
void StSetText( STATICTEXT* pSt, XCHAR* text);
```

SetState() macro — sets the specified state bits of a given object. The object must be re drawn to display the changes. It is possible to set several state bits using this macro.

```
#define SetState(pObj, stateBits)
```

HINT: The **pObj** parameter passed in the `GOLMsgCallback()` function is a pointer to the widget affected by a user input event. Recall, Static Text objects do NOT accept messages; therefore, you will need to provide a pointer when you call any Static Text APIs.

RECALL: `GOLFindObject(objID)` will return a pointer to the object associated with the **objID.**

EXAMPLE: `StSetText((STATICTEXT*)GOLFindObject(OBJ_ST_ID), "text")` will change the text string in the `OBJ_ST_ID` structure to "text".

10 **Compile the project and program the device**. Verify that the static text string changes as you toggle the state of the check boxes using the touch screen.

Results

We have created the LED control using widgets and messaging. In a real world scenario, those LEDs can be any control signals to any external devices such as motors and lights. We have also learned how to control widgets based on a state of another widget.

Code Analysis

In this lab you have learned how to modify the object behavior using the GOLMsgCallback() function. Using the available API for each objects you have linked up multiple objects to function as one. By determining the state change in one object, due to user actions, you have controlled how the other objects will behave. Using the same state changes you have learned how to insert controls on external functions such as LEDs that can represent real world modules like motors.

Conclusions

Now that you have learned how to use the GOLMsgCallback() to control object behavior you are ready to take a look at the GOLDrawCallback() function that can also control object behavior.

THIS PAGE INTENTIONALLY LEFT BLANK

Lab Exercise 4

Advanced control using `GOLDrawCallback()`

Purpose

In Lab 3, you have learned how to implement a user interface through the message callback. In this lab you will be introduced to an advanced feature of control. This is control through the drawing callback.

Requirements

Development Environment:	MPLAB® v8.50 or newer
Software:	Microchip Graphics Library v2.10 Graphics Resource Converter
C Compiler:	MPLAB® C Compiler for PIC24 and dsPIC OR MPLAB® C Compiler for PIC32
Hardware Tools:	Explorer 16 with PIC24FJ128GA010 or PIC32MX360F512L PIM and Graphics LCD Controller PICtail™ Plus SSD1926 Board or PIC24FJ256DA210 Development Board MPLAB® Real ICE or MPLAB® ICD3 Debugger
Lab files on class PC:	C:\...\Lab4...

Objectives

1) Implement auto control based on timer events in the GOLDrawCallback() function.
2) Monitor the user action on a Potentiometer using a widget.
3) Use a Meter widget to monitor the Potentiometer value.

Expected Results

The Meter widget will display the Potentiometer value continuously.

Procedure

If you still have a previous project open, you must first close it by selecting from the menu:
File ▸ Close Workspace

Then, open the Lab 4 project by selecting from the menu:
File ▸ Open Workspace... and opening the workspace file appropriate for you development board setup located at:

C:\...\Lab 4

Choose:
Lab1 Ex 16 PIC24 Truly 3.2.mcw for the Explorer 16 with PIC24FJ128GA010 PIM or ...
Lab1 DA210 Truly 3.2.mcw for the PIC24FJ256DA210 Development Board or...
Lab1 Ex 16 PIC32 Truly 3.2.mcw for the Explorer 16 with PIC32MX360F512L PIM

Lab 4 Reference Information

TICK Timer: A tick timing routine has been in implemented in the application code. This routine uses an interrupt from Timer 4. The timer is set up to interrupt approximately every 100 us and is initialized using the `TickInit()` function (also provided in the application code). The following variables are affected by the tick count.

`tick:` Global variable that holds the tick count.
`prevTick:` Global variable that holds the old tick count.

The sampling of the potentiometer, as well as the touch screen, is based on this tick counter. The effective value reflected on the **Meter** widget is determined by the `SAMPLETIME` macro that has been defined in the lab application code (see Lab4.c). The TICK is application provided, it is NOT part of the Graphics Library.

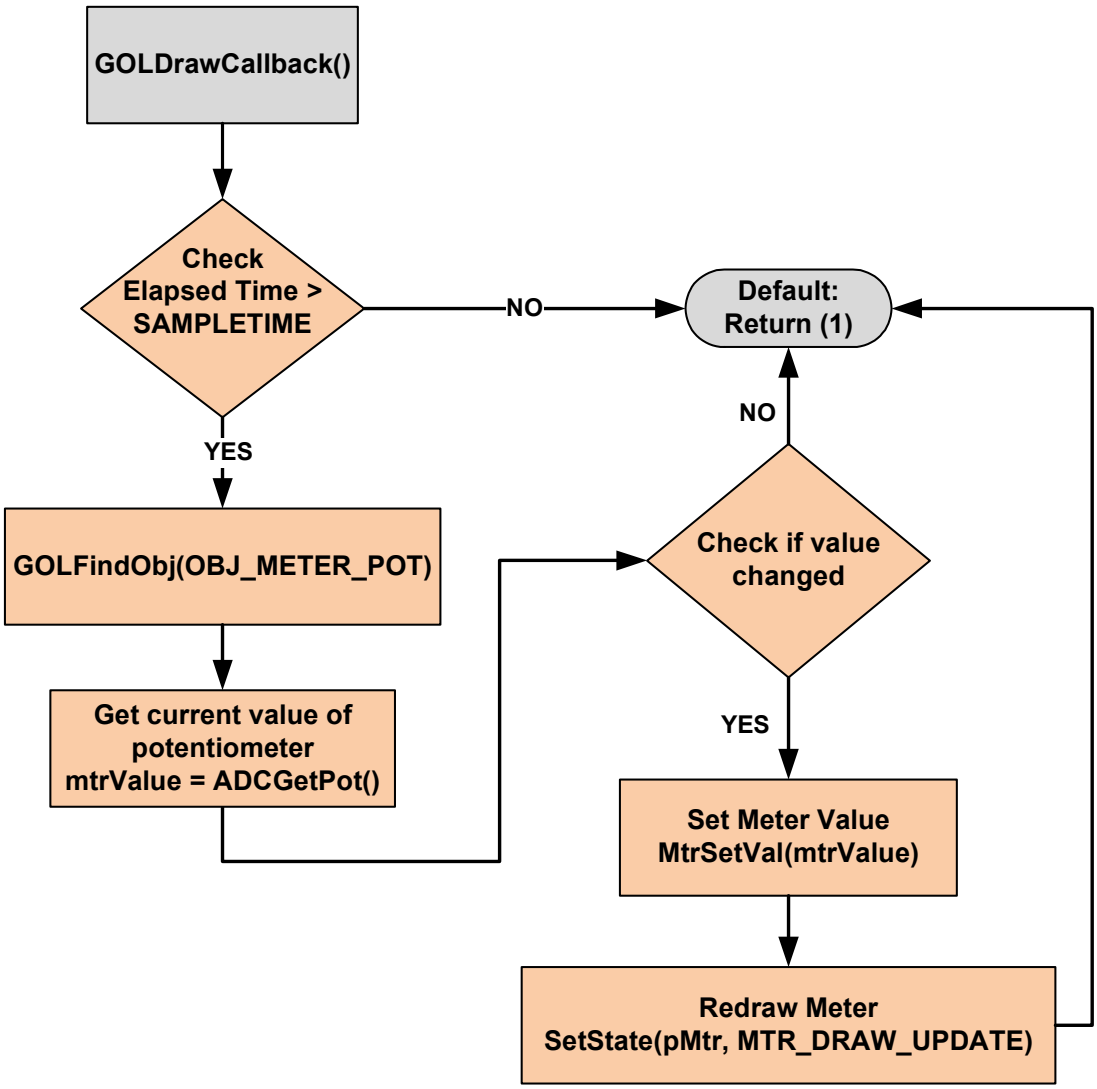
Widget Information: The **Meter** Widget is created in the `CreatePotentiometerScreen()` function located in the Screens.c file. **Meter** Widget is updated in the `GOLDrawCallback()` function.

WIDGET	DESCRIPTION	OBJECT ID	DEFINED POINTER
Meter	Display Potentiometer Value	OBJ_METER_POT	pMtr

In the previous lab, you added code to the `GOLMsgCallback()` function to individually control LEDs on the development board.

In steps 1-4 of this lab, you will add code in the `GOLDrawCallback()` that will modify the Meter to reflect the value of the potentiometer as the potentiometer is turned (clockwise or counter clock wise). Details of the sequence of the operation of the meter are shown on the flow chart shown on the next page.

The reason we implement this functionality in the `GOLDrawCallback()` is that we know for certain that the linked list has been fully parsed. In other words, we know none of the widgets are in the middle of being rendered. How do we know this? Because `GOLDraw()` calls the `GOLDrawCallback()` function once it has finished parsing the linked list.



HINT: `GOLDrawCallback()` does not pass any parameters. The application must either establish pointers to widgets that need to be modified.

RECALL: `GOLFindObject(objID)` will return a pointer to the object associated with the `objID`.

EXAMPLE: To establish a pointer to a button with objID BTN_128:
BUTTON *pBtn
pBtn= `GOLFindObject(BTN_128);` .

1 **On line 240, add code to initialize the Meter handler (pMtr).**

2 Obtain the current value of the Potentiometer using the function `ADCGetPot()`. **Save this value to the predefined variable, mtrValue.**

Step 2 Reference Information

`ADCGetPot()` will return the value for the potentiometer. No parameters are required.

EXAMPLE: To save the potentiometer value to the potValue variable:

```
WORD potValue;
potValue = ADCGetPot();
```

3 Use the Graphics Library Help file to locate the APIs for the METER widget. Specifically locate `MtrSetVal` and `MtrGetVal` APIs. **Starting on line 250, add the code to modify the value of the Meter and modify the state to partially redraw the Meter.**

Step 3 Reference Information

USE THE GRAPHICS HELP FILE TO FIND WIDGET STATEBITS AND WIDGET SPECIFIC APIs.

Start ▸ Programs ▸ Microchip ▸ Graphics Library v2.10 ▸ Graphics Library Help

Expand GOL Objects then expand the Widget you are looking for. Statebits are in the [Widget] States section. [Widget] specific APIs are listed in the [Widget] section.

4 **Compile the project and program the device to verify that steps 1-3 were successfully completed.**

Bonus Procedure

Notice that the value of the Meter is always modified everytime the GOLDrawCallback() is called. What if the potentiometer is not modified? Do we need to redraw the Meter?

a **On line 247, add code to skip the setting of the value and redrawing of the Meter when the value of the Potentiometer is not changing. (Add check to see if mtrValue = MtrGetVal(pMtr))**

Hint: Use MtrGetVal(METER *pMtr) to obtain the Meter value.

b Compile the project and program the device to verify operation.

Results

This lab has introduced you to another way of modifying Object behavior based on your application inputs. This option can be an alternative to add to the application for a better user experience.

Code Analysis

Similar to the GOLMsgCallback() the GOLDrawCallback() can also be used to modify object behavior. Unlike the message callback that depends on the messages, the draw callback can be used and changes to the Objects can be implemented using system level variables.

Another additional use of the draw callback is that it provides a safe place for users to implement customized drawings where modification of global drawing settings such as LineTypes, LineThickness, Colors and Fonts is safe and drawing of Widgets is not affected by these changes.

Object behavior modification in the drawing callback can be derived by any system event such as timer interrupt, external event, or global variables.

Conclusions

Having the knowledge on how to modify object behavior based on messages and on system level variables you are now ready to take a look at advanced management of Objects and screens. Lab5 will show you how to manage multiple screens.

Lab Exercise 5

Putting it all together...

Purpose

The purpose of this lab is to examine the dynamic switching of screens in detail when moving between screens in the application. In lab 1 you created a splash screen. In labs 2, 3 and 4 you created an Main Menu screen and two more screens for your application. To save time, we have replicated the Potentiometer screen to mimic a temperature monitoring. In this final lab, you will only use one linked list for each screen. As you move between screens, the list previous list is deleted and the new list for the next screen is dynamically created.

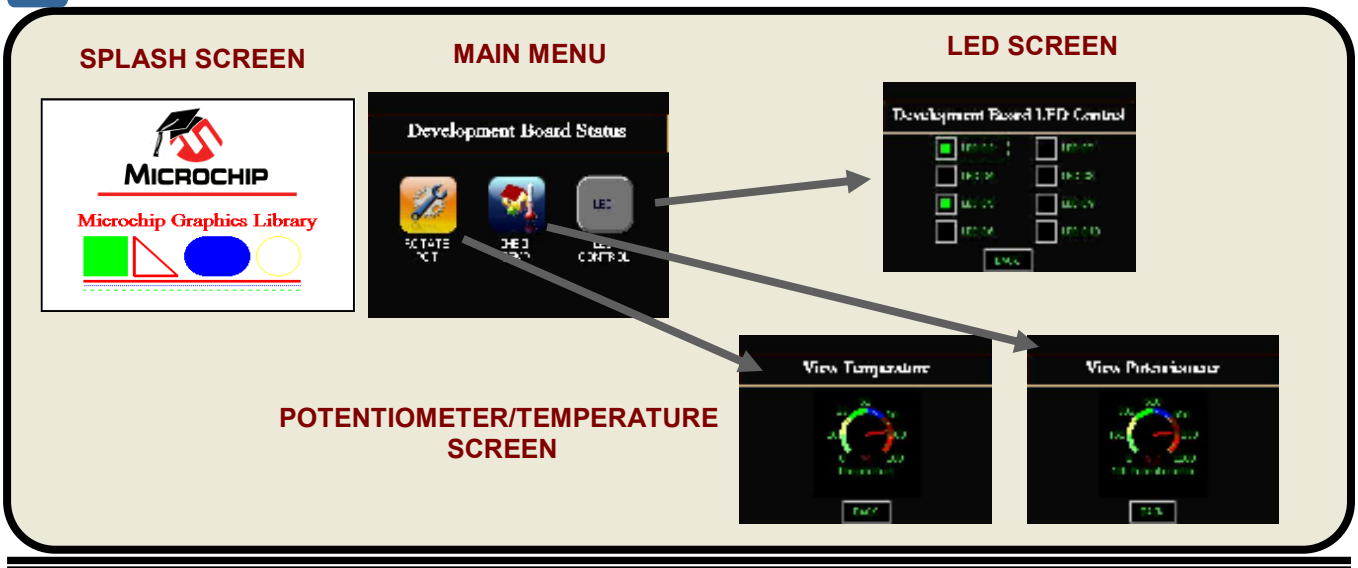
Requirements

Development Environment:	MPLAB® v8.50 or newer
Software:	Microchip Graphics Library v2.10 Graphics Resource Converter
C Compiler:	MPLAB® C Compiler for PIC24 and dsPIC OR MPLAB® C Compiler for PIC32
Hardware Tools:	Explorer 16 with PIC24FJ128GA010 or PIC32MX360F512L PIM and Graphics LCD Controller PICtail™ Plus SSD1926 Board or PIC24FJ256DA210 Development Board MPLAB® Real ICE or MPLAB® ICD3 Debugger
Lab files on class PC:	C:\...\Lab5...

Objectives

1) Switch from one screen to another.
2) Dynamically create one linked list for each screen.

Expected Results



Procedure

If you still have a previous project open, you must first close it by selecting from the menu:
File ▸ Close Workspace

Then, open the Lab 5 project by selecting from the menu:
File ▸ Open Workspace... and opening the workspace file appropriate for your development board setup located at:

C:\...\Lab 5

Choose:
Lab1 Ex 16 PIC24 Truly 3.2.mcw for the Explorer 16 with PIC24FJ128GA010 PIM or ...
Lab1 DA210 Truly 3.2.mcw for the PIC24FJ256DA210 Development Board or...
Lab1 Ex 16 PIC32 Truly 3.2.mcw for the Explorer 16 with PIC32MX360F512L PIM

The following chart shows the information you may need for each of the screens. Because we have multiple screens, we implement a state machine to move us to the appropriate callback function. The state machine variable is screenState. State movement is implemented in the appropriate GOLMsgCallback() because the next state is dependent on which Widget the user has touched. This implementation of the callback functions helps with code maintainability.

Lab 5 Reference Information

MAIN MENU SCREEN:

Screen creation function:	**CreateMenuScreen()**
List pointer name:	**default**
GOLMsgCallback():	**MainMenuMsgCallback()**
GOLDrawCallback():	**NONE**

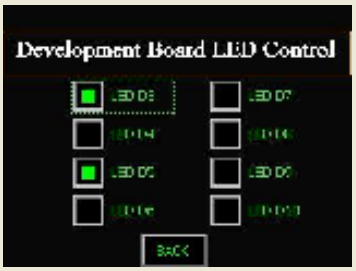
VIEW POTENTIOMETER SCREEN:

Screen creation function:	**CreatePotentiometerScreen()**
List pointer name:	**default**
GOLMsgCallback():	**PotentiometerMsgCallback()**
GOLDrawCallback():	**UpdatePotentiometer()**

TEMPERATURE SCREEN:

Screen creation function:	**CreateTemperatureScreen()**
List pointer name:	**default**
GOLMsgCallback():	**TemperatureMsgCallback()**
GOLDrawCallback():	**UpdateTemperature()**

Lab 5 Reference Information (continued...)

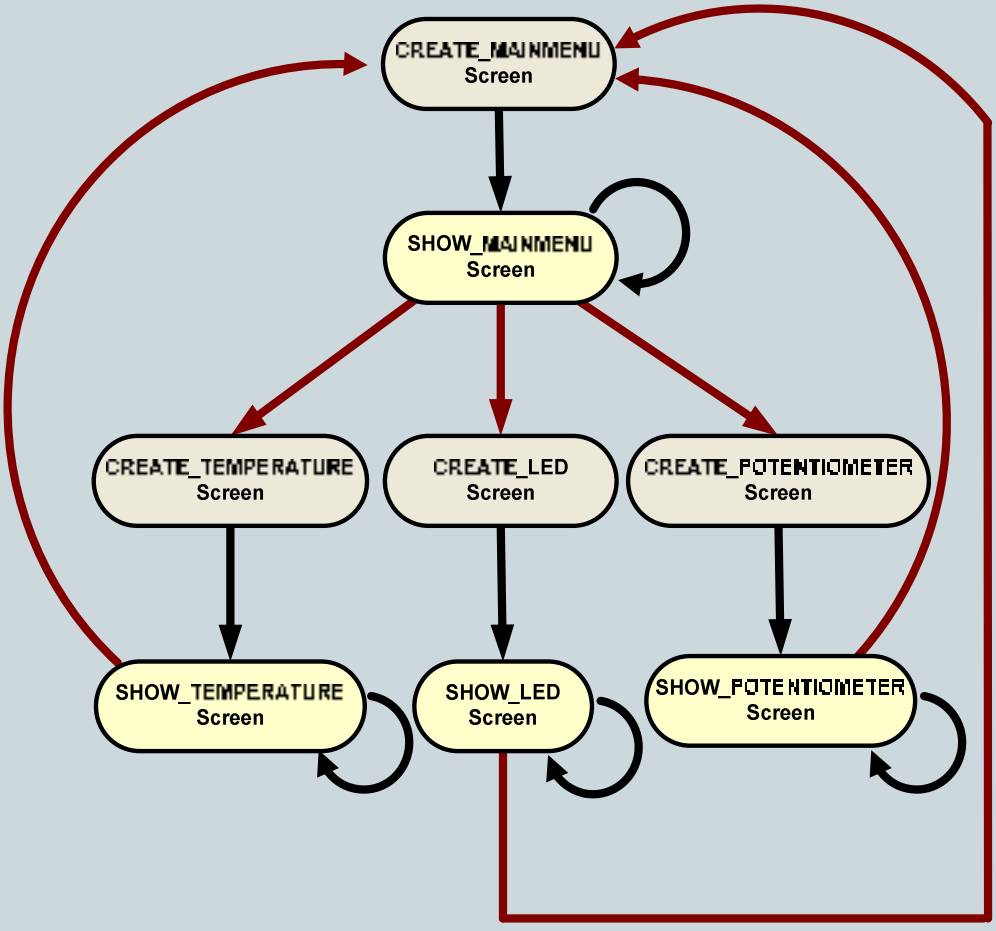


LED SCREEN:

Screen creation function:	CreateLEDScreen()
List pointer name:	default
GOLMsgCallback():	LEDMsgCallback()
GOLDrawCallback():	NONE

Lab 5 Reference Information

Simplified State Diagram



The code to create the objects is provided for you. The only things to add are the APIs that will free the current active list of widgets every time the application moves to another screen and the switching of states.

In steps 1-3, you will add the state transitions that will allow movement from one screen to another. In steps 4-6 you will add the API required to free up the current active list and prepare for the creation of new widgets for the next screen.

NOTE: In the lecture portion you have learned two possible ways to implement screen switching. This lab will show you in detail how the dynamically created screens are implemented using states.

Step 1-3: Reference Information

The screenStates variable defines the current state of the application based on the enumerated `SCREEN_STATES`.

```
typedef enum {
    CREATE_MAINMENU = 0,
    SHOW_MAINMENU,
    CREATE_POTENTIOMETER,
    SHOW_POTENTIOMETER,
    CREATE_TEMPERATURE,
    SHOW_TEMPERATURE,
    CREATE_LED,
    SHOW_LED,
} SCREEN_STATES;
```

1. In Lab5.c, look for the function named `MainMsgMenuCallback()`. **In line number 328, add the code to switch from current state to CREATE_POTENTIOMETER state when the rotate potentiometer icon (Button) with OBJ_BUTTON_ROTATE_POT ID is pressed**.

2. **In line number 350, add the code to switch from current state to CREATE_TEMPERATURE state when the check temperature icon (Button) with OBJ_BUTTON_CHECK_TEMP ID is pressed**.

3. **In line number 368, add the code to switch from current state to CREATE_LED state when the LED control icon (Button) with OBJ_BUTTON_CONTROL_LED ID is pressed**.

4. **Compile the project and program the device.** The application should now switch from one screen to another.

Try switching from one screen to another until you break the program. The run time error appears when one of the dynamically created Widgets failed to get memory. This is caused by the fact that the active list keeps on growing as you switch from one screen to another because the create functions for each screen creates additional widgets.

Now that the states can switch, add the code that will clear the current active list to prepare it for the new widgets that will be dynamically created by the `ObjCreate()` functions.

5 **In the file Screens.c go to line 146 in `CreateMenuScreen()` and add the code to clear the current active list.**

6 **In the file Screens.c go to line 259 in `CreateLEDScreen()` and add the code to clear the current active list.**

7 **In the file Screens.c go to line 465 in `CreatePotentiometerScreen()` and add the code to clear the current active list.**

8 **In the file Screens.c go to line 535 in `CreateTemperatureScreen()` and add the code to clear the current active list.**

Step 4-8: Reference Information

GOLFree() — This function frees all the memory used by objects in the active list and initializes the active list pointer to NULL to start a new empty list. This function must be called only inside the `GOLDrawCallback()` function when using `GOLDraw()` and `GOLMsg()` functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.

Other relevant APIs that may be used to switch screens:

GOLNewList() macro — starts a new linked list by setting the global active list pointer to NULL. The keyboard focus pointer is also reset. If the previous list is no longer needed, memory should by freed using GOLFree().

```
#define GOLNewList()
```

GOLSetList() macro — sets the global active list pointer to the specified list.

```
#define GOLSetList(objsList)
```

GOLGetList() macro — returns the pointer to the current active list.

```
#define GOLGetList()
```

9 **Compile the project and program the device.** The application should now switch from one screen to another with the active list initialized properly for each screen.

_pGolObjects—is the global pointer that refers to the current active list of objects. This active list of objects is the list that will be used to render objects into the screen. Both **GOLDraw()** and **GOLMsg()** functions parse this list to determine which widgets are affected by the user messages and at the same time which widgets need to be redrawn or removed form the screen.

_pGolObjects is retrieved using the **GOLGetList()** function. It is initialized to NULL using the **GOLNewList()**. **GOLFree()** on the other hand removes all the widgets in the list and frees the memory used by those removed objects. To assign **_pGolObjects** to a saved list use **GOLSetList()**.

Results

This lab has shown you how to manage multiple screen applications. Using the API's available in the Library, it is easy to implement multiple pages with multiple lists or create and destroy lists on the fly. In the lecture portion of the class, you learned other methods for switching between screens. As home-work, try to modify Lab 5 to switch between multiple linked lists instead.

Code Analysis

The Library has a special rule in modifying the current active list. It is very important that the current active list should only be modified whenever GOLDraw() is not actively rendering objects in the screen. The only safe place to do without checking if GOLDraw() is busy with the list is in GOLDrawCallback(). If it is not possible to do this, user must make sure that the last call to GOLDraw() function has returned a 1. It is also important that GOLMsg() is not active when modifying the current active list.
Applications with multiple screens can be implemented using multiple lists or creating and destroying lists on the fly. The advantage of using multiple lists is that it is faster to switch and render the screens but at the cost of RAM space. If speed is not an important requirement in the application, use of on the fly creation and destruction of lists when switching screens may be a good alternative. All Objects included in the library have been optimized so the delay caused by the deletion of objects in memory and calling of Create() functions for each Object can be unnoticeable.

Conclusions

Now that you have the knowledge of using the Microchip Graphics Library, from the Primitive Layer to the Graphics Object Layer, you are now able to integrate cool Graphics to your applications easily. You have seen how easy it is to create multiple screens and control the Objects using standard user interfaces such as touch screen.

THIS PAGE INTENTIONALLY LEFT BLANK

Lab Exercise 6

Graphics on MDB

Purpose

To familiarize you with the Microchip Graphics library and learn how to create a graphics application and run on Multimedia development board

Requirements

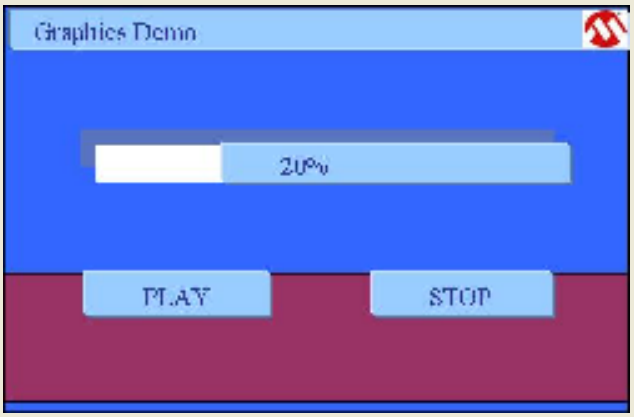
Development Environment:	MPLAB® v8.50 or newer
Software:	Microchip Graphics Library v2.10
	Graphics Resource Converter
C Compiler:	MPLAB® C Compiler for PIC32
Hardware Tools:	PIC32 Ethernet Starter Kit and
	Multimedia Expansion Board
	MPLAB® Real ICE or MPLAB® ICD3 Debugger
Lab files on class PC:	C:\...\Lab6...

Objectives

1) Switch from one screen to another.
2) Dynamically create one linked list for each screen.

Expected Results

MAIN MENU



Procedure

If you still have a previous project open, you must first close it by selecting from the menu:
File ▸ Close Workspace

Then, open the Lab 6 project by selecting from the menu:
File ▸ Open Workspace... and opening the workspace file appropriate for your development board setup located at:

C:\...\Lab 6

Choose:

Lab6.mcw for PIC32 Ethernet Starter Kit and Multimedia Expansion Board

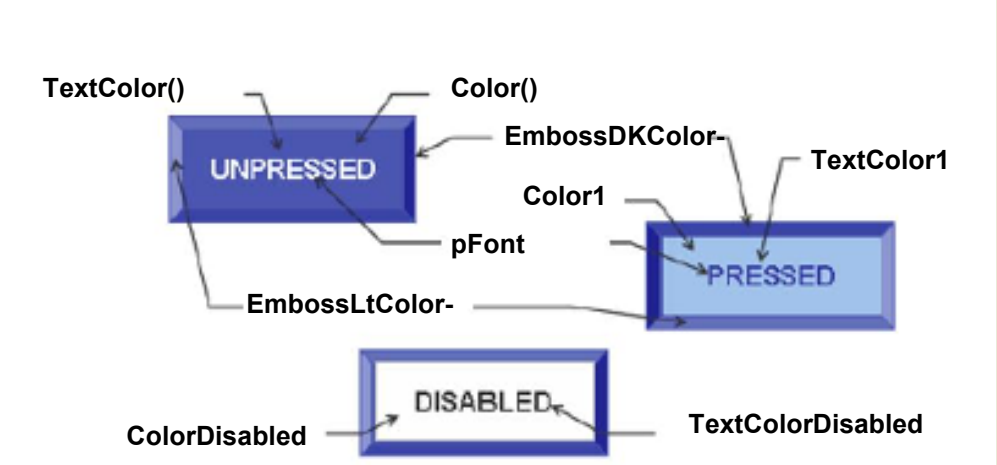
Lab 6 Reference Information

Overview:
Create two buttons and one progress bar on the LCD display.
Integrate the Touch Screen with the buttons and control the progress bar on button click.

Graphics Object Layer (GOL) objects:

1. Button:

The button object is rendered using the assigned style scheme. The following figure illustrates the color assignments for the button.



Lab 6 Reference Information (continued...)

BtnCreate Function

```
BUTTON  *BtnCreate(
        WORD        ID,             // Unique user defined ID for the object instance.
        SHORT       left,           // Left most position of the object.
        SHORT       top,            // Top most position of the object.
        SHORT       right,          // Right most position of the object.
        SHORT       bottom,         // Bottom most position of the object.
        SHORT       radius,         // Radius of the rounded edge.
        WORD        state,          // Sets the initial state of the object.
        void        *pBitmap,       // Pointer to the bitmap used on the face of the button.
        XCHAR       *pText,         // Pointer to the text of the button.
        GOL_SCHEME  *pScheme        // Pointer to the style scheme used.
);
```

This function creates a BUTTON object with the parameter given. It automatically attached the new object into a global linked list of the objects and returns the address of the object.

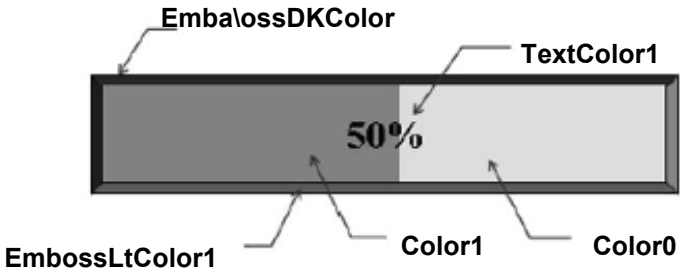
Example:

```
GOL_SCHEME *pScheme;
BUTTON *buttons;

pScheme = GOLCreateScheme():

Buttons = BtnCreate(1,20,64,50,118,0, BTN_DRAW, NULL,”ON”, pScheme);

if (buttons[0] == NULL)          // check if button 0 is created
   Return 0;
```

Lab 6 Reference Information (continued...)

2. Progress Bar

The Progress Bar Object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



CommonBKColor - used to hide/remove the progress bar from the screen

PbCreate Function

```
PROGRESSBAR *PbCreate(
    WORD        ID,             // Unique user defined ID for the object instance.
    SHORT       left,           // Left most position of the object.
    SHORT       top,            // top most position of the object.
    SHORT       right,          // right most position of the object.
    SHORT       bottom,         // bottom most position of the object.
    WORD        state,          // Sets the initial position of the progress.
    WORD        pos,            // Defines the initial position of the progress.
    WORD        range,      // This specifies the maximum value of the progress bar at 100% position.
    GOL_SCHEME  *pScheme        //Pointer to the style scheme used for the object.
);
```

This function creates a PROGRESSBAR object with the parameters given. It automatically attaches the ne object into a global linked list of objects and returns the address of the object.

Example:

```
PROGRESSBAR *pPBar;
Void CreateProgressBar(){
pPBar = PbCreate(ID_PROGRESSBAR1,       // ID
        50,90,270,140,                  // dimension
        PB_DRAW,                        // Draw the object
        25,                             // position
        50,                             // set the range
        NULL,);                         // use default GOL scheme
}
```

Lab 6 Reference Information (continued...)

GOL Call back Functions

3. GOLMsgCallback Function

WORD GOLMsgCallBack(
WORD objMsg, // Message for the object or the action ID response from the object
OBJ_HEADER *pObj, // Pointer to the object that processed the message.
GOL_MSG *pMsg // Pointer to the GOL message from user
);

The user MUST implement this function. GOLMsg() calls this function when a valid message for an object in the active list is received, User action for the message should be implemented here. If this function returns non-zero, the message for the object will be processed by default. If zero is returned, GOL will not perform any action.

4. GOLDrawCallback Function

WORD GOLDrawCallback();

GOLDrawCallback() function MUST BE implemented by the user. This is called inside the GOLDraw() function when the drawing of the objects in the active list is completed. User drawing must be done here. Drawing color, line type, clipping region, graphic cursor position and current font will not be changed by GOL if this function returns a zero. To pass drawing control to GOL this function must return a non-zero value. If GOL messaging is not using the active link list, it is safe to modify the list here.

Example:-

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg)
{
        if(objMsg == BTN_MSG_PRESSED) {
                if(GetObjID(pObj) == ID_BUTTON_PLAY){
                        // Do some action
                }
                else if(GetObjID(pObj) == ID_BUTTON_STOP){
                        // Do some action
                }
         }
        return 1;

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

THIS PAGE INTENTIONALLY LEFT BLANK