# Touch and graphical displays part 5 - Menu system programming

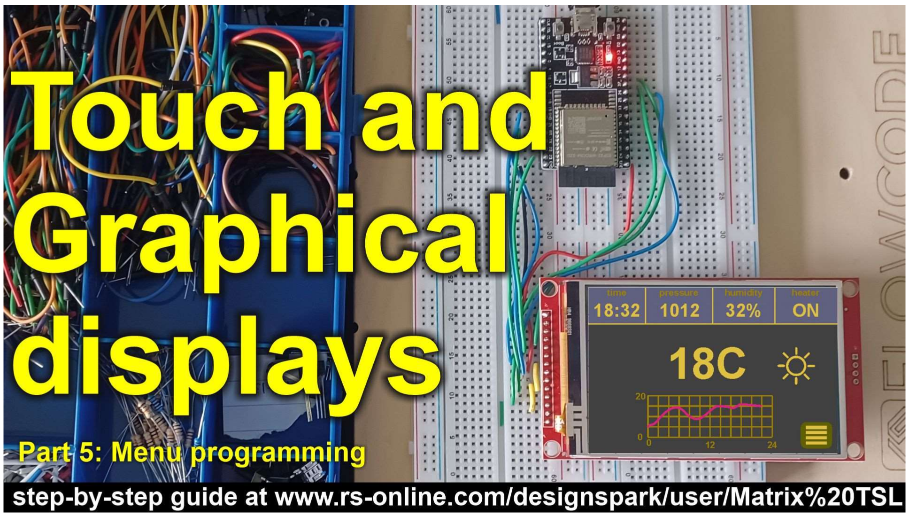

Touch and Graphical displays
Part 5: Menu programming
step-by-step guide at www.rs-online.com/designspark/user/Matrix%20TSL
18:32
1012
32%
ON
18C

Figure 1 - Youtube video link

**The project**

In this article we are going to show you how to program a full Menu system using Flowcode.

- How the components work
- Touch screen properties
- Touch screen component
- Graphical LCD display manager
- Themes
- Architecture of the program

Lets remind ourselves of what we are trying to achieve:

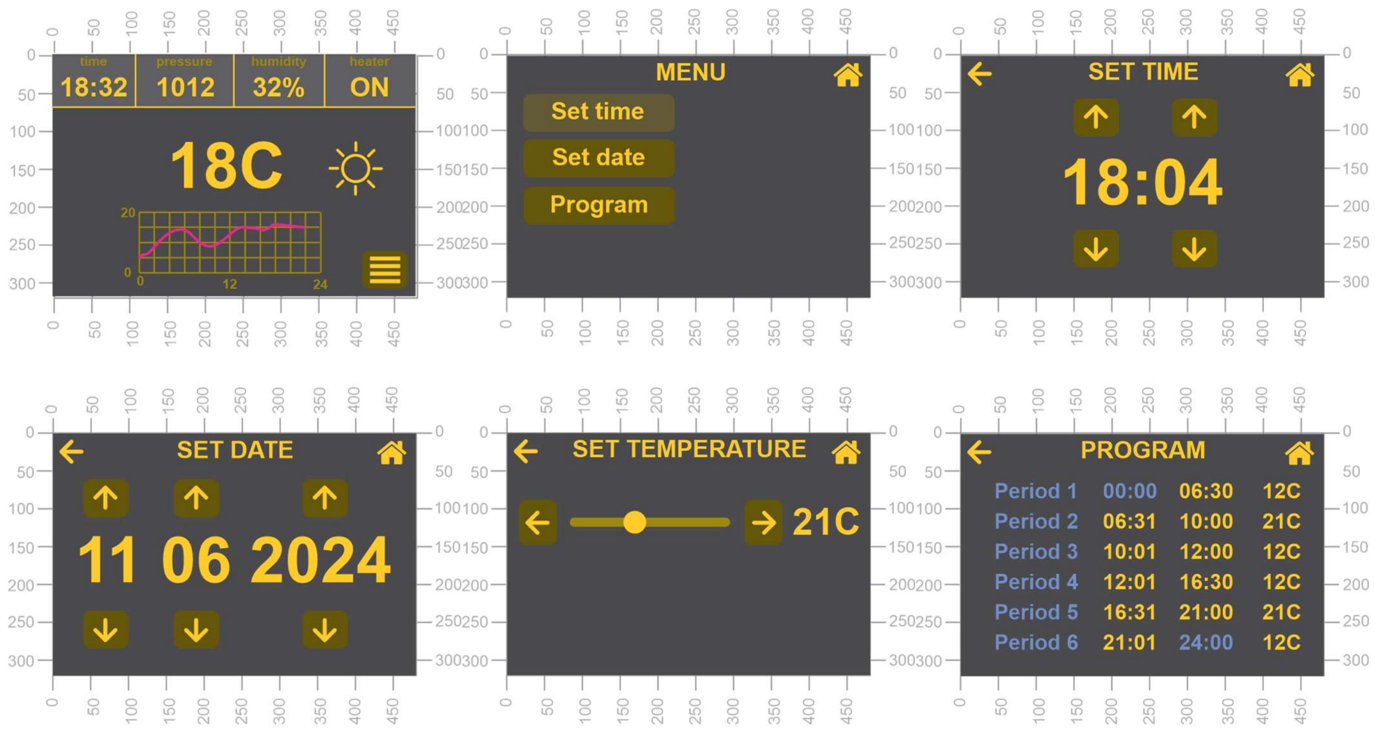
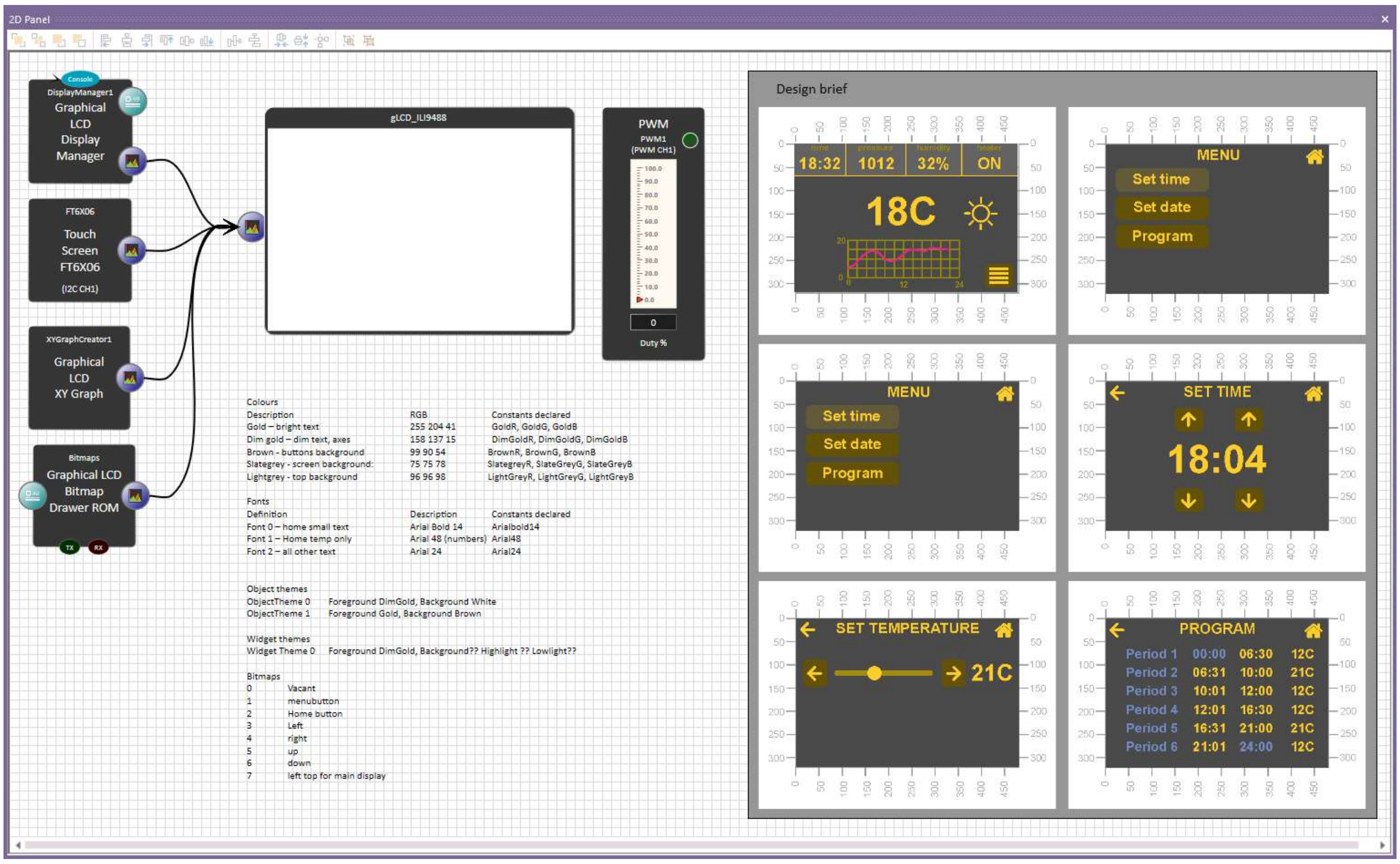


Figure 2 - The task ahead of us

Let’s start by looking at the  panel for this project:



Colours

| Description | RGB | Constants declared |
|---|---|---|
| Gold – bright text | 255 204 41 | GoldR, GoldG, GoldB |
| Dim gold – dim text, axes | 158 137 15 | DimGoldR, DimGoldG, DimGoldB |
| Brown - buttons background | 99 90 54 | BrownR, BrownG, BrownB |
| Slategrey - screen background: | 75 75 78 | SlategreyR, SlateGreyG, SlateGreyB |
| Lightgrey - top background | 96 96 98 | LightGreyR, LightGreyG, LightGreyB |

Fonts

| Definition | Description | Constants declared |
|---|---|---|
| Font 0 – home small text | Arial Bold 14 | Arialbold14 |
| Font 1 – Home temp only | Arial 48 (numbers) | Arial48 |
| Font 2 – all other text | Arial 24 | Arial24 |

Object themes

| | |
|---|---|
| ObjectTheme 0 | Foreground DimGold, Background White |
| ObjectTheme 1 | Foreground Gold, Background Brown |

Widget themes

| | |
|---|---|
| Widget Theme 0 | Foreground DimGold, Background?? Highlight ?? Lowlight?? |

Bitmaps

| | |
|---|---|
| 0 | Vacant |
| 1 | menubutton |
| 2 | Home button |
| 3 | Left |
| 4 | right |
| 5 | up |
| 6 | down |
| 7 | left top for main display |

Figure 3 -  project panel

A few things to note:

When designing a system you can use Flowcode to put the specification of the system on the panel. So here we have defined all the colours and the fonts as constants in Flowcode so we don’t need to keep thinking about what we are using where. Secondly we have put the screen graphics specification from the graphics designer as bitmaps on the panel. I can see what the goal is and the sizes of the objects are on each screen.

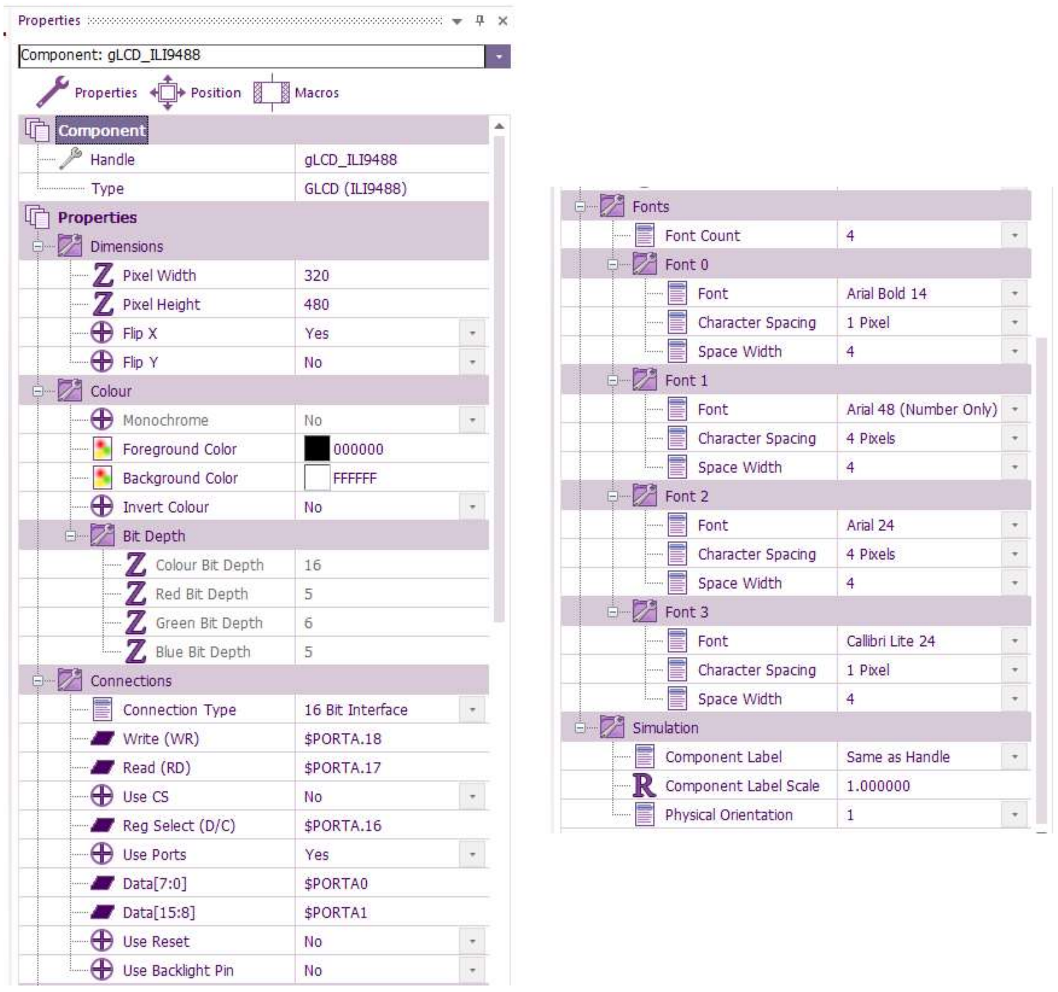
As for the components:

- The main display is an ILI9488 cap touch colour graphical LCD. The integrated touch chip is a FT6X06 and there is a component linked to the graphical display.
- The Graphical LCD Display Manager component includes routines for managing the touch widgets and menu system
- The Graphical LCD Bitmap Drawer component allows you to define bitmaps which are then compiled to the microcontroller  along with the program.
- The PWM component allows you to vary the brightness of the display.
- The target device here is an ESP32-S3_DevkitC.

Let’s look at the properties of each in turn.

**Graphical Display properties**

The display by default is portrait. We are using it in a landscape orientation. You need to set up the properties so that it is in landscape. You also need to initialise it as landscape in a hardware macro. The connections are parallel. It’s a fast display. The Font properties are set up here. There are lots of fonts to choose from. Fonts are compiled to the microcontroller and  are memory hungry so you need to declare how many you want to embed. In this case we have  4 fonts.



Properties

Component: gLCD_ILI9488

Properties Position Macros

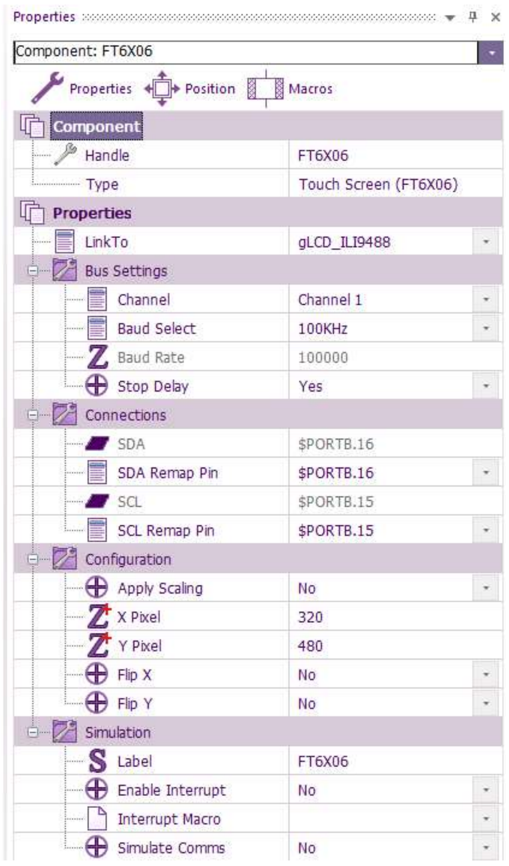
| | |
|---|---|
| **Component** | |
| Handle | gLCD_ILI9488 |
| Type | GLCD (ILI9488) |
| **Properties** | |
| Dimensions | |
| Pixel Width | 320 |
| Pixel Height | 480 |
| Flip X | Yes |
| Flip Y | No |
| Colour | |
| Monochrome | No |
| Foreground Color | ⬛ 000000 |
| Background Color | ⬜ FFFFFF |
| Invert Colour | No |
| Bit Depth | |
| Colour Bit Depth | 16 |
| Red Bit Depth | 5 |
| Green Bit Depth | 6 |
| Blue Bit Depth | 5 |
| Connections | |
| Connection Type | 16 Bit Interface |
| Write (WR) | $PORTA.18 |
| Read (RD) | $PORTA.17 |
| Use CS | No |
| Reg Select (D/C) | $PORTA.16 |
| Use Ports | Yes |
| Data[7:0] | $PORTA0 |
| Data[15:8] | $PORTA1 |
| Use Reset | No |
| Use Backlight Pin | No |

| | |
|---|---|
| Fonts | |
| Font Count | 4 |
| Font 0 | |
| Font | Arial Bold 14 |
| Character Spacing | 1 Pixel |
| Space Width | 4 |
| Font 1 | |
| Font | Arial 48 (Number Only) |
| Character Spacing | 4 Pixels |
| Space Width | 4 |
| Font 2 | |
| Font | Arial 24 |
| Character Spacing | 4 Pixels |
| Space Width | 4 |
| Font 3 | |
| Font | Callibri Lite 24 |
| Character Spacing | 1 Pixel |
| Space Width | 4 |
| Simulation | |
| Component Label | Same as Handle |
| Component Label Scale | 1.000000 |
| Physical Orientation | 1 |

Figure 4 - Display properties

**Touch screen chip properties**

The touch screen links to the Graphical Display. The properties allow you to set up the bus, connections - in this case I2C - and the XY parameters.

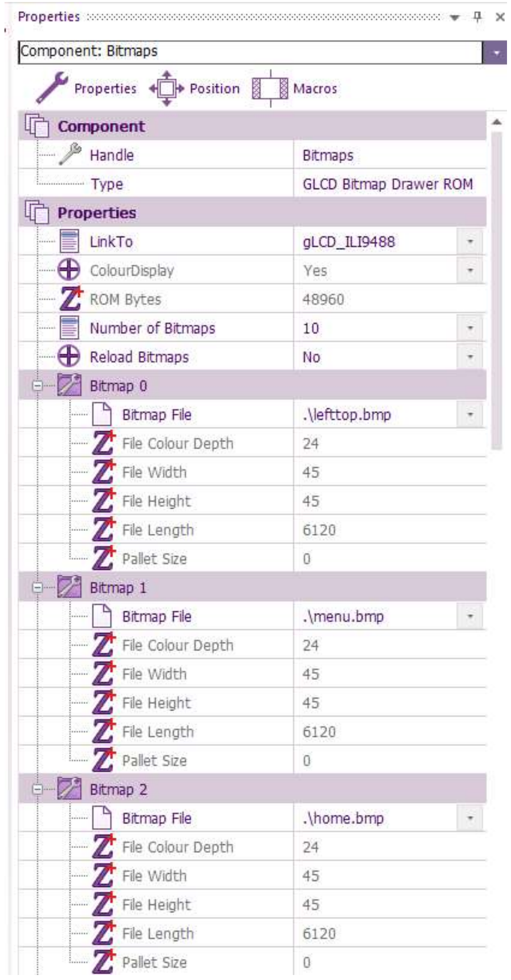


Properties
Component: FT6X06

| | |
|---|---|
| **Component** | |
| Handle | FT6X06 |
| Type | Touch Screen (FT6X06) |
| **Properties** | |
| LinkTo | gLCD_ILI9488 |
| Bus Settings | |
| Channel | Channel 1 |
| Baud Select | 100KHz |
| Baud Rate | 100000 |
| Stop Delay | Yes |
| Connections | |
| SDA | $PORTB.16 |
| SDA Remap Pin | $PORTB.16 |
| SCL | $PORTB.15 |
| SCL Remap Pin | $PORTB.15 |
| Configuration | |
| Apply Scaling | No |
| X Pixel | 320 |
| Y Pixel | 480 |
| Flip X | No |
| Flip Y | No |
| Simulation | |
| Label | FT6X06 |
| Enable Interrupt | No |
| Interrupt Macro | |
| Simulate Comms | No |

Figure 5 - touch screen component properties

**Bitmap drawer properties**



Properties
Component: Bitmaps

| | |
|---|---|
| **Component** | |
| Handle | Bitmaps |
| Type | GLCD Bitmap Drawer ROM |
| **Properties** | |
| LinkTo | gLCD_ILI9488 |
| ColourDisplay | Yes |
| ROM Bytes | 48960 |
| Number of Bitmaps | 10 |
| Reload Bitmaps | No |
| Bitmap 0 | |
| Bitmap File | .\lefttop.bmp |
| File Colour Depth | 24 |
| File Width | 45 |
| File Height | 45 |
| File Length | 6120 |
| Pallet Size | 0 |
| Bitmap 1 | |
| Bitmap File | .\menu.bmp |
| File Colour Depth | 24 |
| File Width | 45 |
| File Height | 45 |
| File Length | 6120 |
| Pallet Size | 0 |
| Bitmap 2 | |
| Bitmap File | .\home.bmp |
| File Colour Depth | 24 |
| File Width | 45 |
| File Height | 45 |
| File Length | 6120 |
| Pallet Size | 0 |

Figure 6 - Bitmap Drawer properties

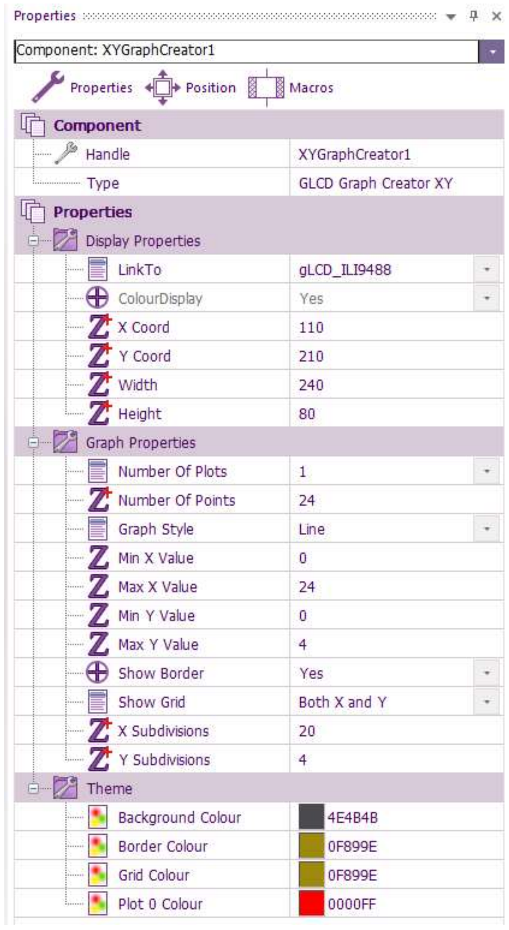
The Bitmap drawer component links to the touch screen. This is where you declare the bitmaps you will use.

When you declare a bitmap Flowcode brings the bitmap file into its memory. Currently (August 2024) if you change the bitmap file you need to Reload it using the Reload property.

Bitmaps are memory hungry.  Flowcode handles 24 bit, 8 bit and 4 bit bitmaps. The properties show you the palette used and the memory used.

You can setup to 10 bitmaps. You need to declare the number of bitmaps in the properties so that Flowcode knows how much memory to allocate during compilation.

**Graphical LCD graph properties**



Properties

Component: XYGraphCreator1

Properties | Position | Macros

| | |
|---|---|
| **Component** | |
| Handle | XYGraphCreator1 |
| Type | GLCD Graph Creator XY |
| **Properties** | |
| Display Properties | |
| LinkTo | gLCD_ILI9488 |
| ColourDisplay | Yes |
| X Coord | 110 |
| Y Coord | 210 |
| Width | 240 |
| Height | 80 |
| Graph Properties | |
| Number Of Plots | 1 |
| Number Of Points | 24 |
| Graph Style | Line |
| Min X Value | 0 |
| Max X Value | 24 |
| Min Y Value | 0 |
| Max Y Value | 4 |
| Show Border | Yes |
| Show Grid | Both X and Y |
| X Subdivisions | 20 |
| Y Subdivisions | 4 |
| Theme | |
| Background Colour | ■ 4E4B4B |
| Border Colour | ■ 0F899E |
| Grid Colour | ■ 0F899E |
| Plot 0 Colour | 🟥 0000FF |

Figure 7 - Graph properties

The XY Graph component links to the Graphical Display. The properties allow you to set the position, size and graph parameters. The graph has its own Colour theme. The colour numbers here are the Hex equivalent of the RGB parameters in the colour picker: not ideal. Our background colour is SlateGrey, the Border and grid are DimGold and the line is drawn as a red line.

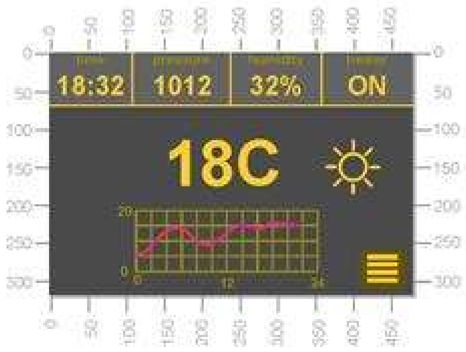


Figure 8 - The Home screen specification

**Architecture of a touch screen program**

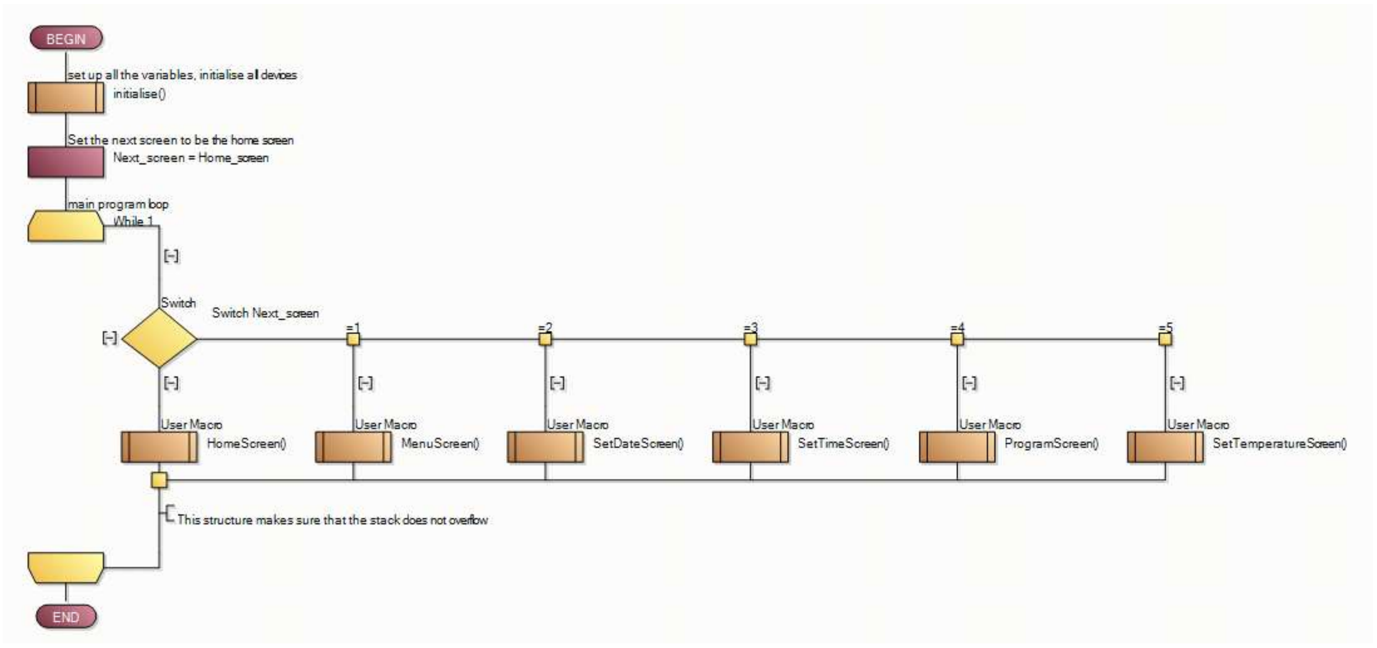


Figure 9 - Main program routine

Figure x shows the MAIN program routine. In our system we will treat each screen display as a separate program macro. That keeps things neat. We need to navigate between screens depending on which on-screen buttons are pressed. To control this navigation the  next_screen variable controls which screen is going to be displayed next

Why introduce this complexity?  Each screen sets up the screen graphics and parameters and then goes into a WHILE loop waiting for user interaction. If we navigated between screens without closing the WHILE loops then the internal stack memory would become overloaded: we always need to close the WHILE loop. So every time there is a navigation between screens we jump back into the MAIN routine which then directs where the program goes next.

The first commands initialise variables and components. Next we test to see if a variable has been adjusted and if so set next scree to Program screen. Then we test the value of Next screen in a Switch icon and invoke the relevant Macro.

This is the fundamental architecture that allows you to have many screens. Each screen has a number starting at 0.

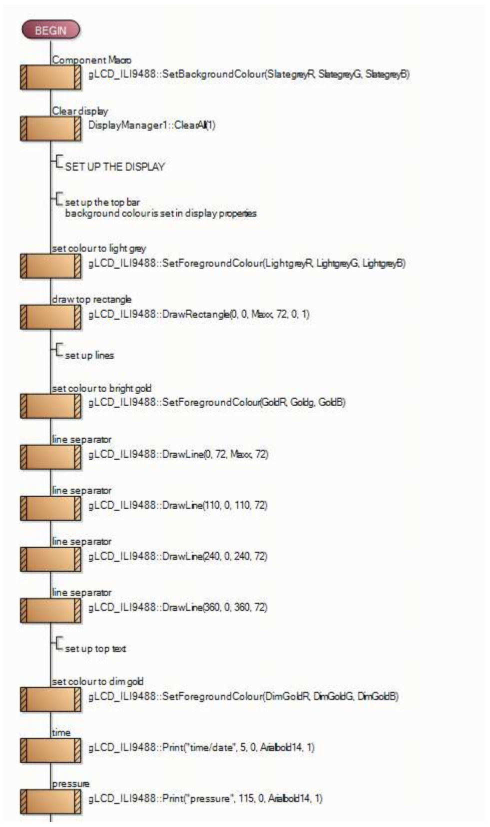
<u>Tracking variables</u>

In our system there are numbers that represent colours, screens, themes and so on. Its a lot to track mentally. To get round this we have allocated constants to represent these numbers. For screens this is as follows:

- Home_screen: 0
- Menu screen: 1
- Set_date_screen: 2
- Set_time_screen: 3
- Program_screen: 4
- Set_temperature_screen: 5

This allows us to forget about the actual numbers and just refer to these screens by Constants. The only exception to this is the SWITCH icon where numbers must be used inside the dialogue box rather than constants or variables.

This takes care of the navigation between screens. Let’s look at the Home screen details:



```
BEGIN
Component Macro
    gLCD_ILI9488::SetBackgroundColour(SlategreyR, SlategreyG, SlategreyB)
Clear display
    DisplayManager1::ClearAll(1)
SET UP THE DISPLAY
set up the top bar
background colour is set in display properties
set colour to light grey
    gLCD_ILI9488::SetForegroundColour(LightgreyR, LightgreyG, LightgreyB)
draw top rectangle
    gLCD_ILI9488::DrawRectangle(0, 0, Maxx, 72, 0, 1)
set up lines
set colour to bright gold
    gLCD_ILI9488::SetForegroundColour(GoldR, Goldg, GoldB)
line separator
    gLCD_ILI9488::DrawLine(0, 72, Maxx, 72)
line separator
    gLCD_ILI9488::DrawLine(110, 0, 110, 72)
line separator
    gLCD_ILI9488::DrawLine(240, 0, 240, 72)
line separator
    gLCD_ILI9488::DrawLine(360, 0, 360, 72)
set up top text
set colour to dim gold
    gLCD_ILI9488::SetForegroundColour(DimGoldR, DimGoldG, DimGoldB)
time
    gLCD_ILI9488::Print("time/date", 5, 0, Arialbold14, 1)
pressure
    gLCD_ILI9488::Print("pressure", 115, 0, Arialbold14, 1)
```

Figure 10 -  first part of the Home screen routine

**Home screen**

There are two parts to this: the first part sets up the screen itself. The second part is a loop that waits for user interaction.

<u>Setting up the screen</u>

These macros consist of a lot of graphics statements: set the foreground colour, draw lines, print text etc.

Keeping track of the RGB parameters that make up a colour is difficult. So what we do is set up a bunch of constants to track the colours we are using. Our colour table is as follows:

| | | |
|---|---|---|
| Gold – bright text | 255 204 41 | GoldR, GoldG, GoldB |
| Dim gold – dim text, axes | 158 137 15 | DimGoldR, DimGoldG, DimGoldB |
| Brown - buttons background | 99 90 54 | BrownR, BrownG, BrownB |
| Slategrey - screen background: | 75 75 78 | SlategreyR, SlateGreyG, SlateGreyB |
| Lightgrey - top background | 96 96 98 | LightGreyR, LightGreyG, LightGreyB |

Setting up your colours a RGB constants in this way as the advantage that if you want to tweak colours later then you can do that globally.

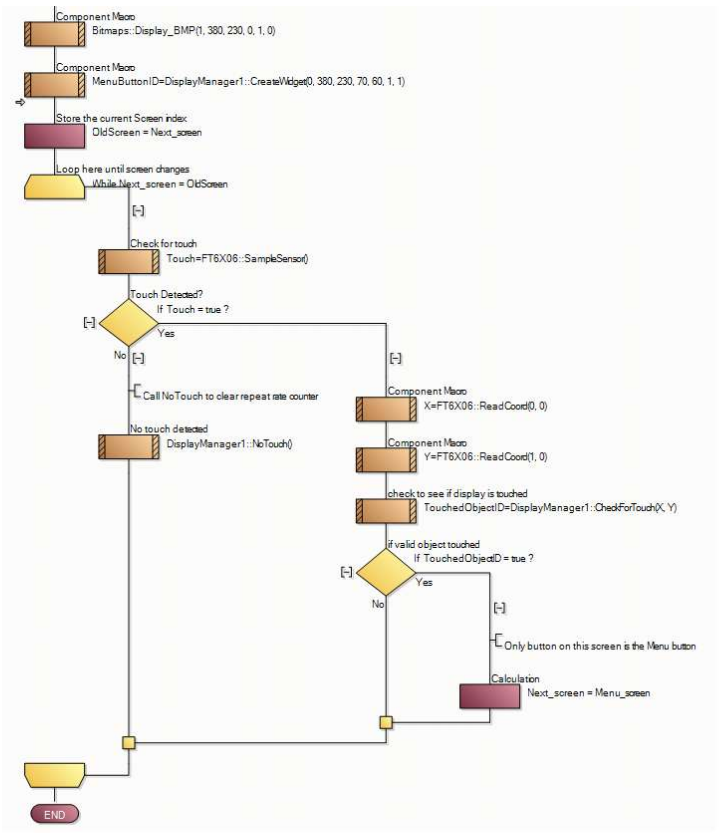

Component Macro
Bitmaps::Display_BMP(1, 380, 230, 0, 1, 0)
Component Macro
MenuButtonID=DisplayManager1::CreateWidget(0, 380, 230, 70, 60, 1, 1)
Store the current Screen index
OldScreen = Next_screen
Loop here until screen changes
While Next_screen = OldScreen
Check for touch
Touch=FT6X06::SampleSensor()
Touch Detected?
If Touch = true ?
Yes
No
Call NoTouch to clear repeat rate counter
No touch detected
DisplayManager1::NoTouch()
Component Macro
X=FT6X06::ReadCoord(0, 0)
Component Macro
Y=FT6X06::ReadCoord(1, 0)
check to see if display is touched
TouchedObjectID=DisplayManager1::CheckForTouch(X, Y)
if valid object touched
If TouchedObjectID = true ?
Yes
No
Only button on this screen is the Menu button
Calculation
Next_screen = Menu_screen
END

Figure 11 - Touch detect part of the Home screen routine

<u>Touch detect routine</u>

There is only one button on the Home screen - a menu button. The first commands displays a Menu bitmap on the display and then declares a touchable Widget on top of it - in this case the Widget is rectangular and is hidden.

The routine then goes into a loop which only ends when the Next_screen variable changes.

The touch component is sampled for a touch. If there is a touch the X, Y location of the touch is passed on to the Graphical Display component which compares it to the co-ordinates of the declared touchable areas. If the touch is on a valid component the Graphical Display returns the ID of the valid component and in the TouchedObjectID variable.

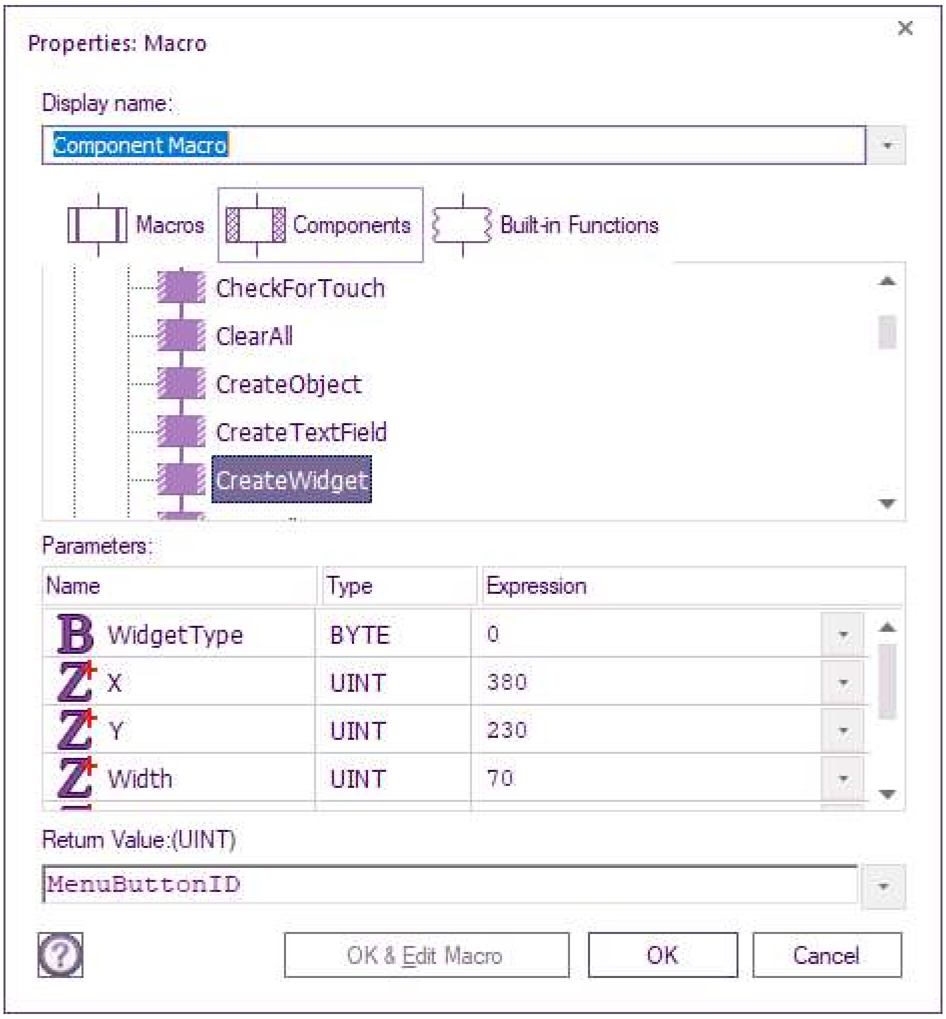
All the other screen macros are just a variation of the Home screen macro with more IF commands in the touch detection routine.

**IDs of touchable objects**

Allocating memory space is a key Flowcode task. We have seen above that Flowcode needs to allocate memory space for fonts and bitmaps and that makes sense. Because of the way Flowcode uses memory to create Objects and Widgets the maximum number of each also needs to be declared.

The Display Manager has hardware macros to allow you to draw objects and touchable widgets. As the Display Manager sets up widgets it automatically allocates a unique ID to them. The ID number starts at the maximum declared number of Objects - 50 in our case - and assigns them to Widgets incrementally as they are created.

Tracking this manually is a pain. So what we do is assign a variable to each touchable object and as the Widget is created we assign the ID value to that variable. In this case you can see that where the Menu button is assigned the CreateWidget routine assigns the ID to the variable MenuButtonID.


Properties: Macro

Display name: Component Macro

Parameters:

| Name | Type | Expression |
|---|---|---|
| WidgetType | BYTE | 0 |
| X | UINT | 380 |
| Y | UINT | 230 |
| Width | UINT | 70 |

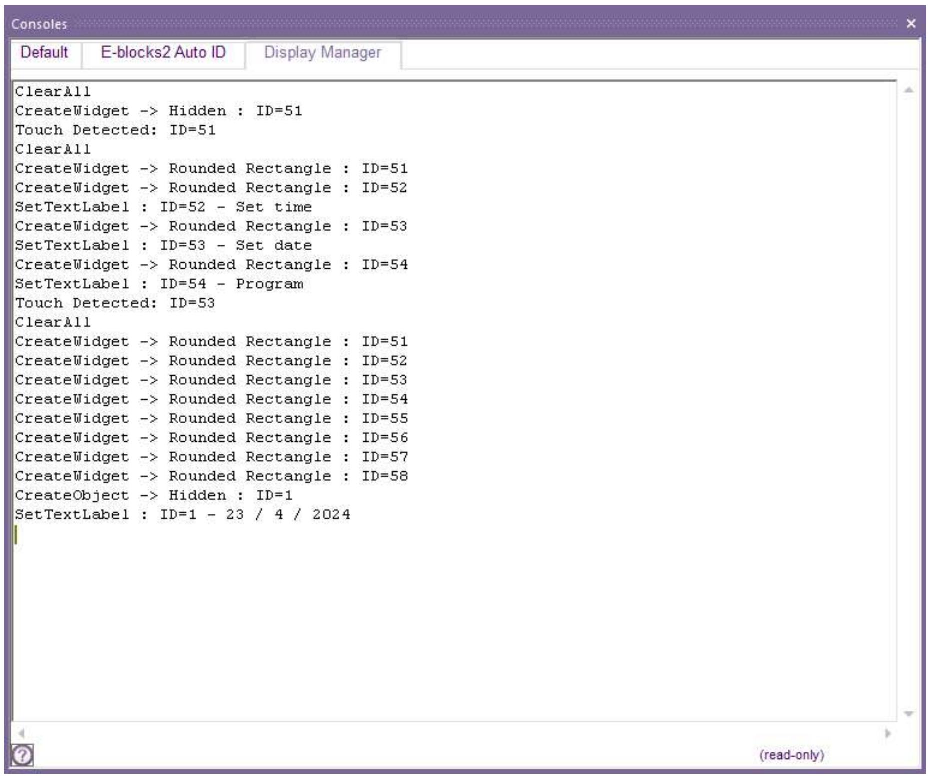
Return Value:(UINT) MenuButtonID

Figure 12 - Create Widget dialogue for the Menu button assigning ID to the MenuButtonID variable

**Debugging touch screen operation**

Making sure that your program traps the right IDs on a touch should be straightforward if you assign IDs correctly at the point of Widget creation. However The matrix simulation engine has a great debugging tool: the Console. This is a really useful tool that allows you to compare the ID number created during the screen set up with the variables you have assigned. Before simulation select VIEW….CONSOLE and click on the Display Manager tab and as the Display Manager creates a widget, or detects a touch it puts the details on here. So you can see what the Display Manager is doing at all points of the program and if you do have a navigation issue you can use this to solve it.

This shows you the basics of setting up the overall architecture and a typical screen. The rest of the program is a variation on this theme. The Program screen has 20 touchable objects and we use an array system to track the touchable IDs created by Display manager. Of course with a program like this there is a reasonable amount of code to track variables and the program function.



```
ClearAll
CreateWidget -> Hidden : ID=51
Touch Detected: ID=51
ClearAll
CreateWidget -> Rounded Rectangle : ID=51
CreateWidget -> Rounded Rectangle : ID=52
SetTextLabel : ID=52 - Set time
CreateWidget -> Rounded Rectangle : ID=53
SetTextLabel : ID=53 - Set date
CreateWidget -> Rounded Rectangle : ID=54
SetTextLabel : ID=54 - Program
Touch Detected: ID=53
ClearAll
CreateWidget -> Rounded Rectangle : ID=51
CreateWidget -> Rounded Rectangle : ID=52
CreateWidget -> Rounded Rectangle : ID=53
CreateWidget -> Rounded Rectangle : ID=54
CreateWidget -> Rounded Rectangle : ID=55
CreateWidget -> Rounded Rectangle : ID=56
CreateWidget -> Rounded Rectangle : ID=57
CreateWidget -> Rounded Rectangle : ID=58
CreateObject -> Hidden : ID=1
SetTextLabel : ID=1 - 23 / 4 / 2024
```

Figure 13 - Console dialogue during simulation

**Conclusion**

Touch screens are replacing indicators, potentiometers and switches in electronic devices. The number of products on the market is rising fast as engineers take the opportunity to incorporate touch screens into their designs and increase functionality and reduce cost.

Flowcode is compatible with thousands of touch screen graphical displays. Flowcode now includes a range of components to allow engineers to design, simulate and deploy touch screen displays.

There are three main areas of work in touch screen system design:

- Graphics
- Touchable elements, navigation and logic,
- The rest of the program

Flowcode 10.1 is equipped with software components for all of these for thousands of devices. Flowcode's simulation capabilities make it an idea choice for graphical touch screen system design.