ESP32 Development

Workshop4

Revision 1.0

Contents

1. Introduction

Workshop4 is a comprehensive software from 4D Systems providing a code and graphics editor for ESP32-S3 based modules. It can be used to design graphical interfaces for all sorts of applications using the IDE's various widgets. All application code can also be developed within the Workshop4 IDE, easily coupling it with the your design, so is a one stop shop for development with these modules.

The Workshop4 IDE utilises the Arduino IDE 2.x CLI to handle the compiling, linking and downloading of ESP32-S3 based projects, using the ESP32 Arduino Core and associated libraries, without having to interface with Arduino IDE at all.

2. Development Setup

This section describes how to setup Workshop4 and Arduino IDE for developing applications for 4D System's ESP32-S3 based display modules.

Both Workshop4 and Arduino IDE 2.x should be installed in your Windows computer.

- Workshop4 IDE
- Arduino IDE

To install Workshop4, please refer to the Installation section of the Workshop4 User Manual.

You can refer to this Arduino documentation for instructions on how to download and install Arduino IDE 2.

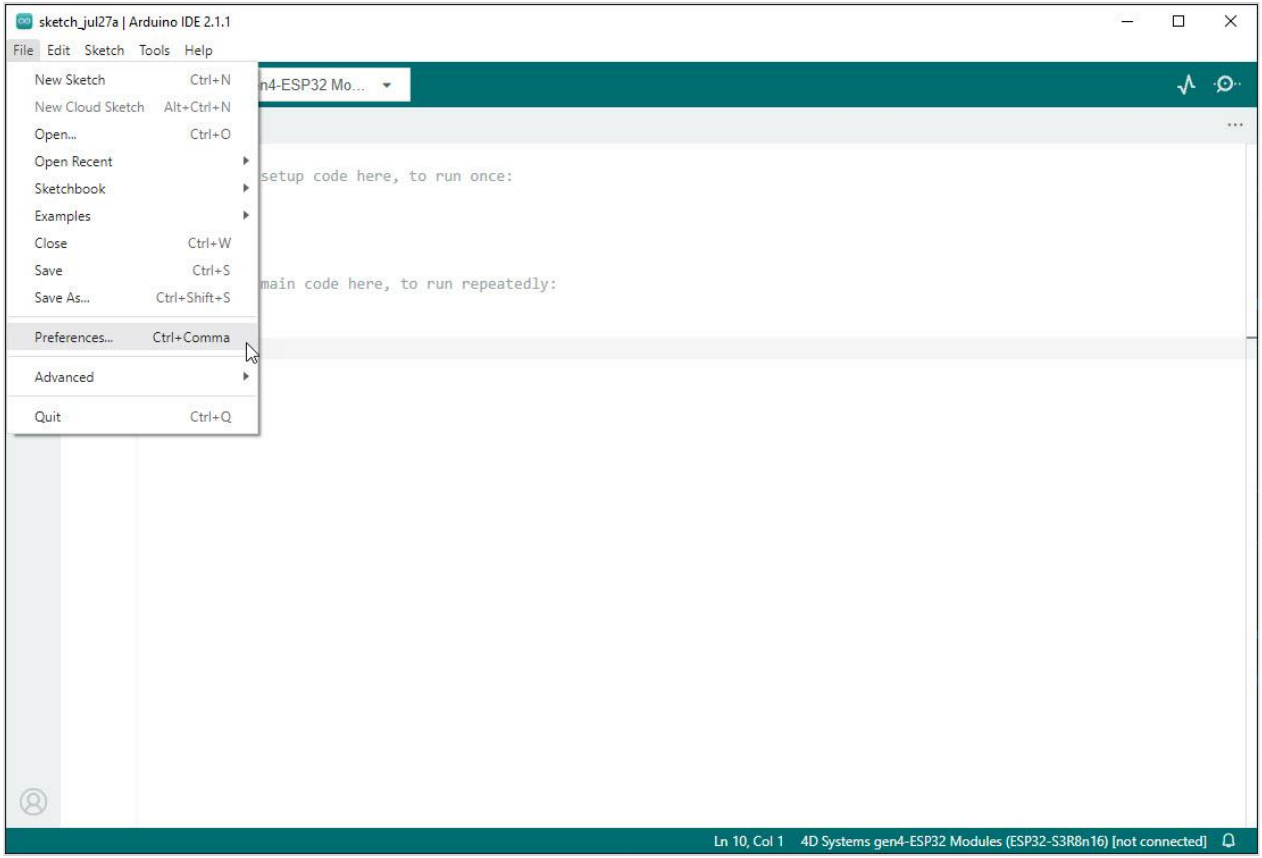
> **Note**
>
> Workshop4 is a **Windows-only** application.

2.1. ESP32 Arduino Boards

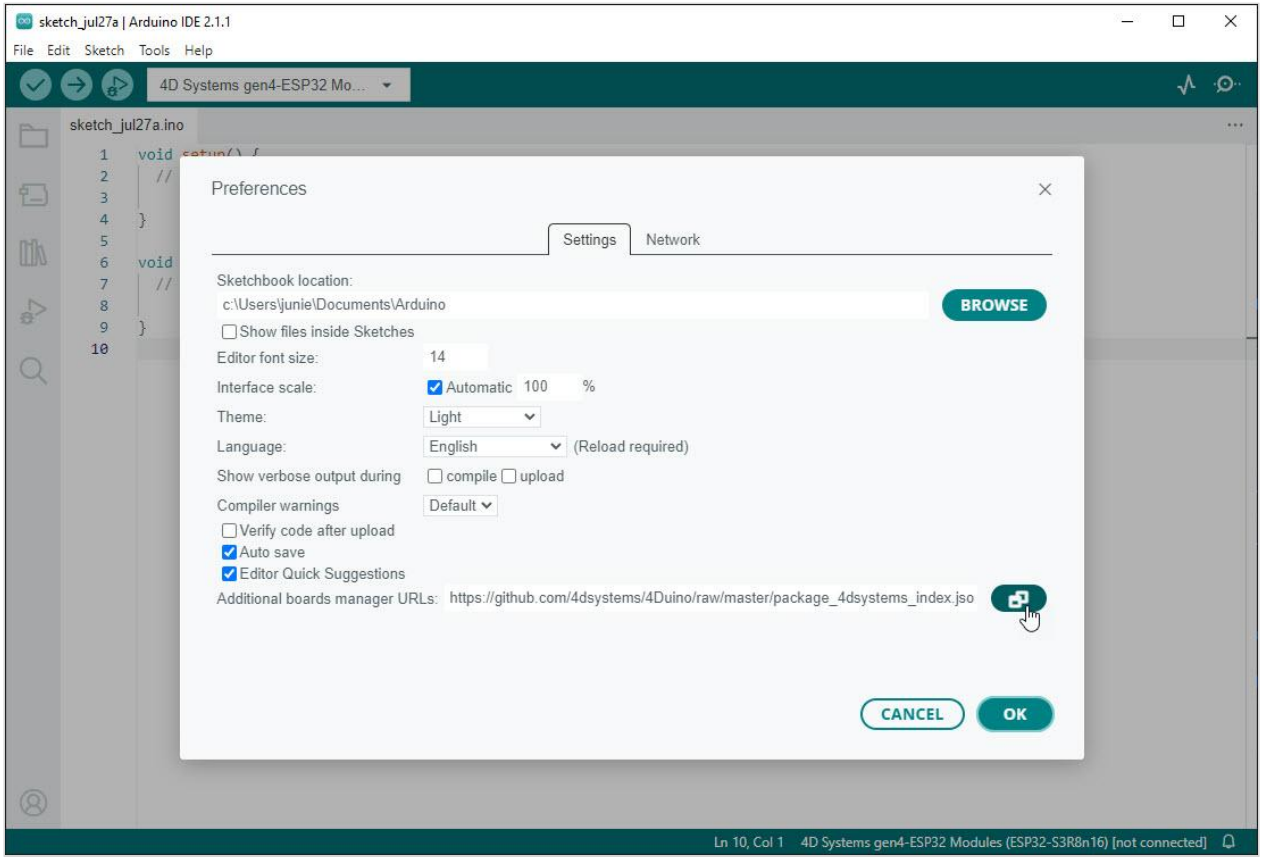
After installing both Workshop4 and Arduino IDE 2 on your system, Arduino needs to add the supported ESP32 boards which includes 4D Systems' ESP32-S3 based displays through its Boards Manager.

Follow these steps to install ESP32 Arduino compatible boards using Boards Manager.

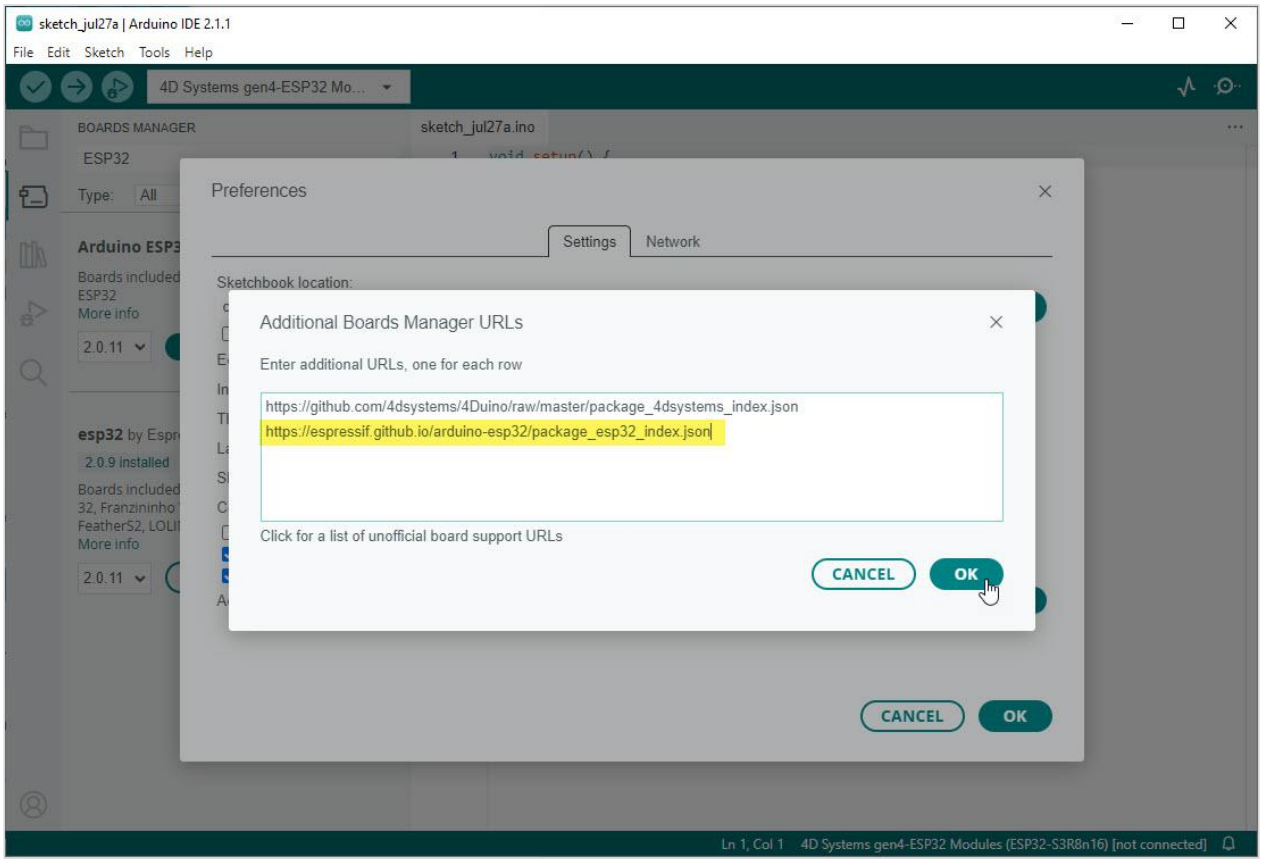
1. Start Arduino IDE and open the Preferences Window.



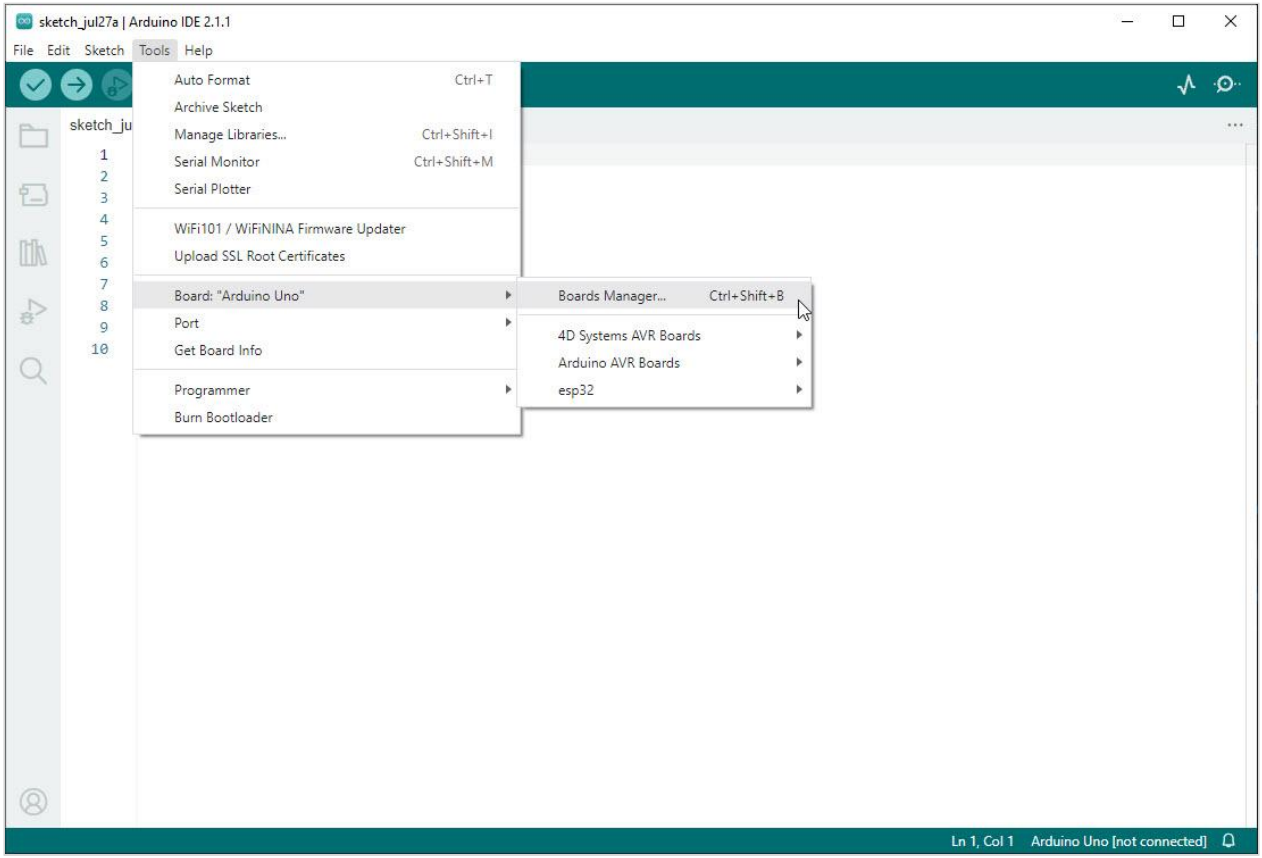
2. Find **Additional boards manager URLs** and click the button to edit the list.



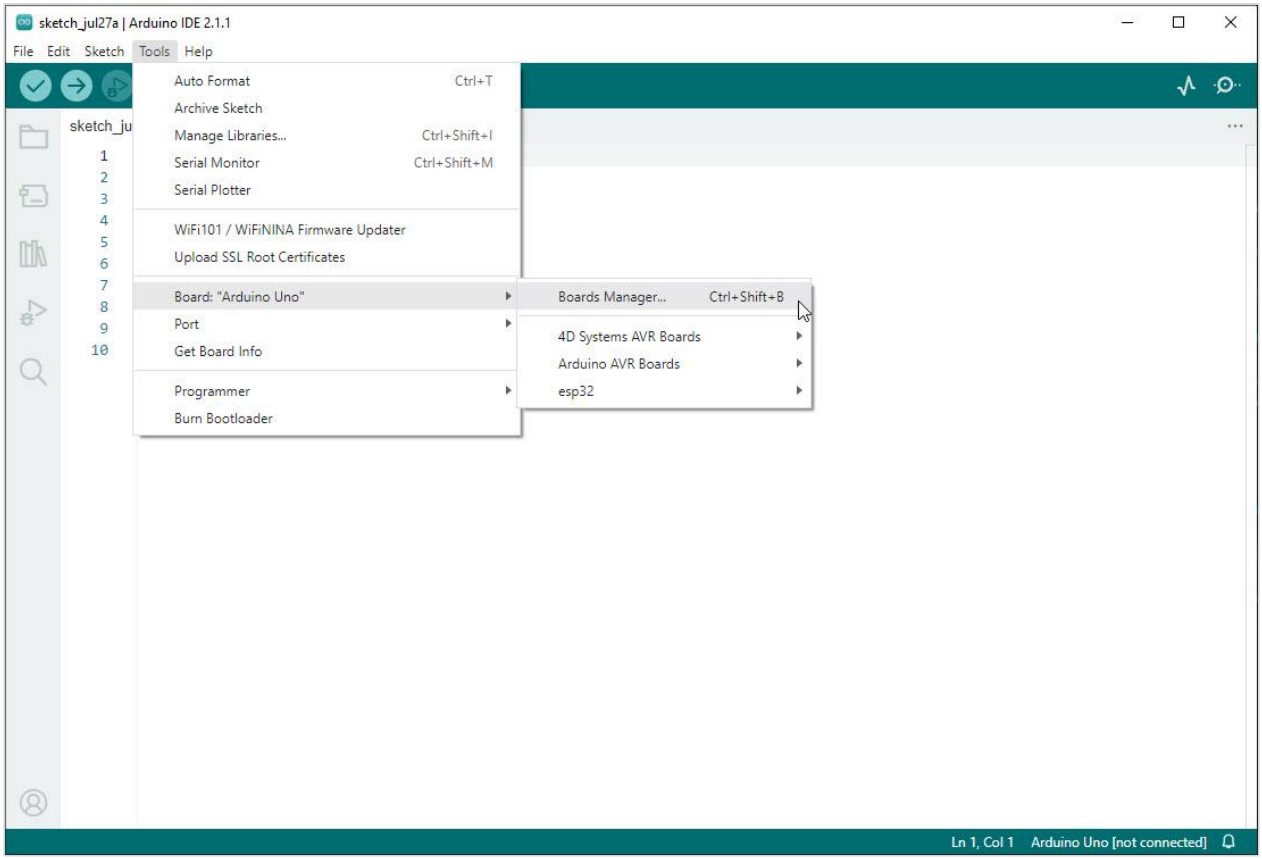
3. Add a reference to `https://espressif.github.io/arduino-esp32/package_esp32_index.json` and save.



4. Open Boards Manager from *Tools* > *Board* menu.



5. Search and install **esp32** by Espressif Systems.



You can refer to the official Espressif documentation for updated boards URL and more details.

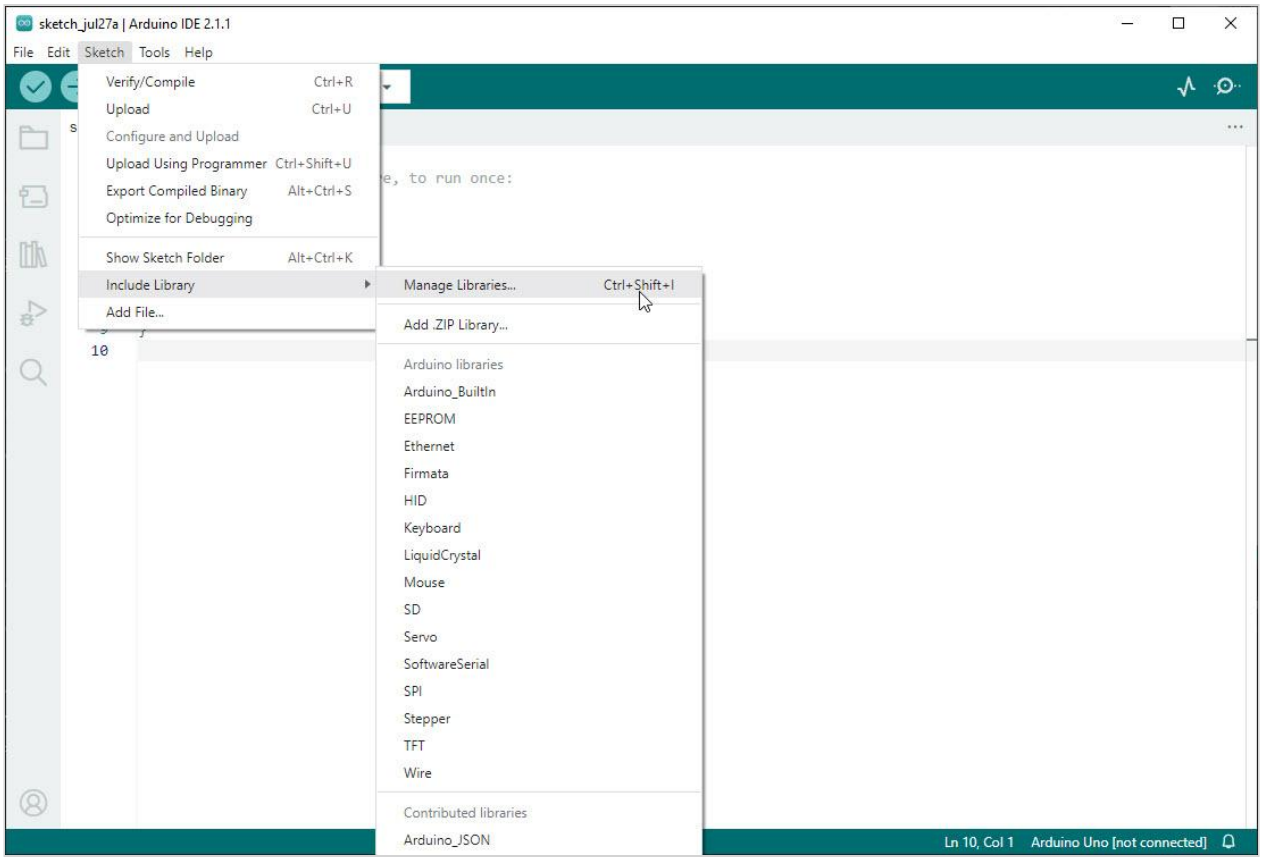
2.2. GFX4dESP32 Library

Workshop4 development for 4D Systems' ESP32-S3 based displays relies on **GFX4dESP32** library. This library needs to be installed alongside the Arduino IDE.

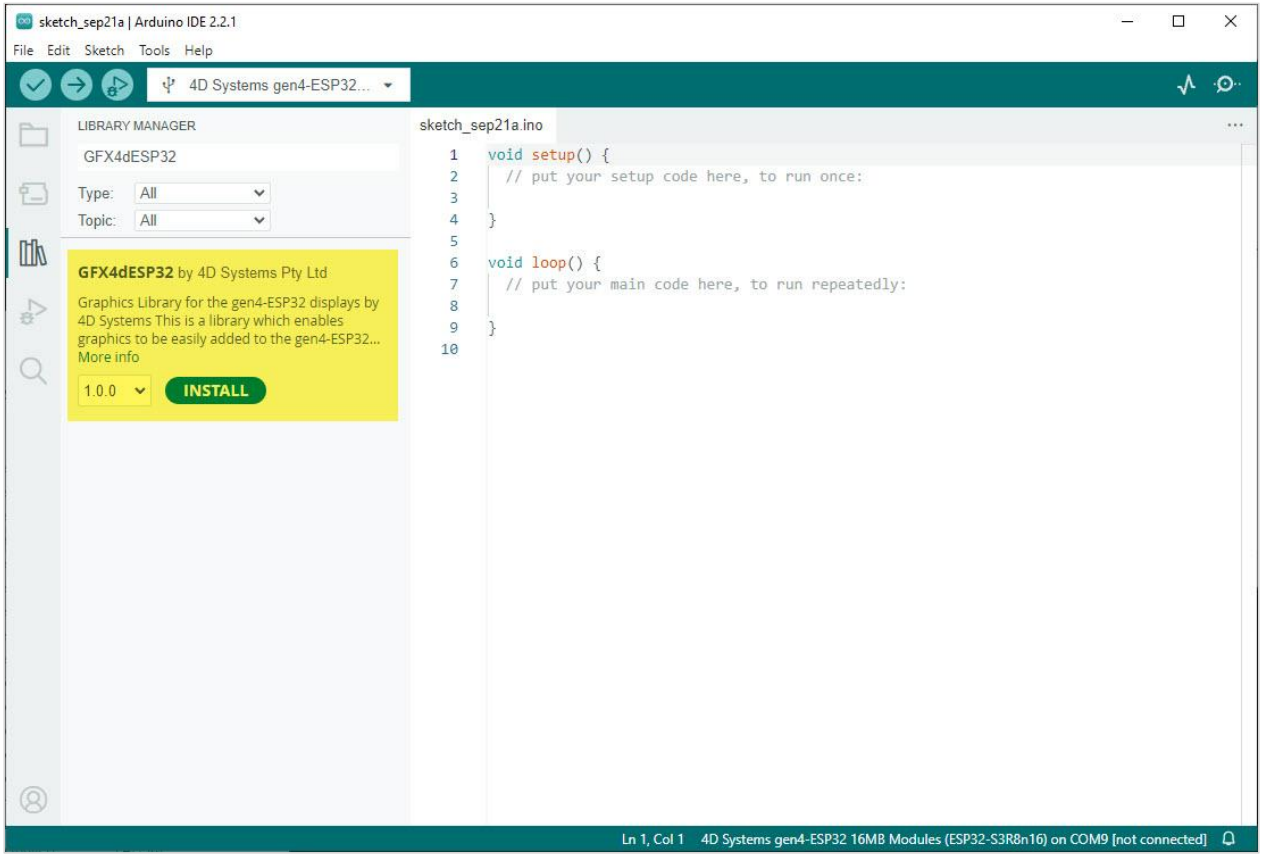
2.2.1. Install via Library Manager

Install the library using Arduino IDE's Library Manager by following the procedure below:

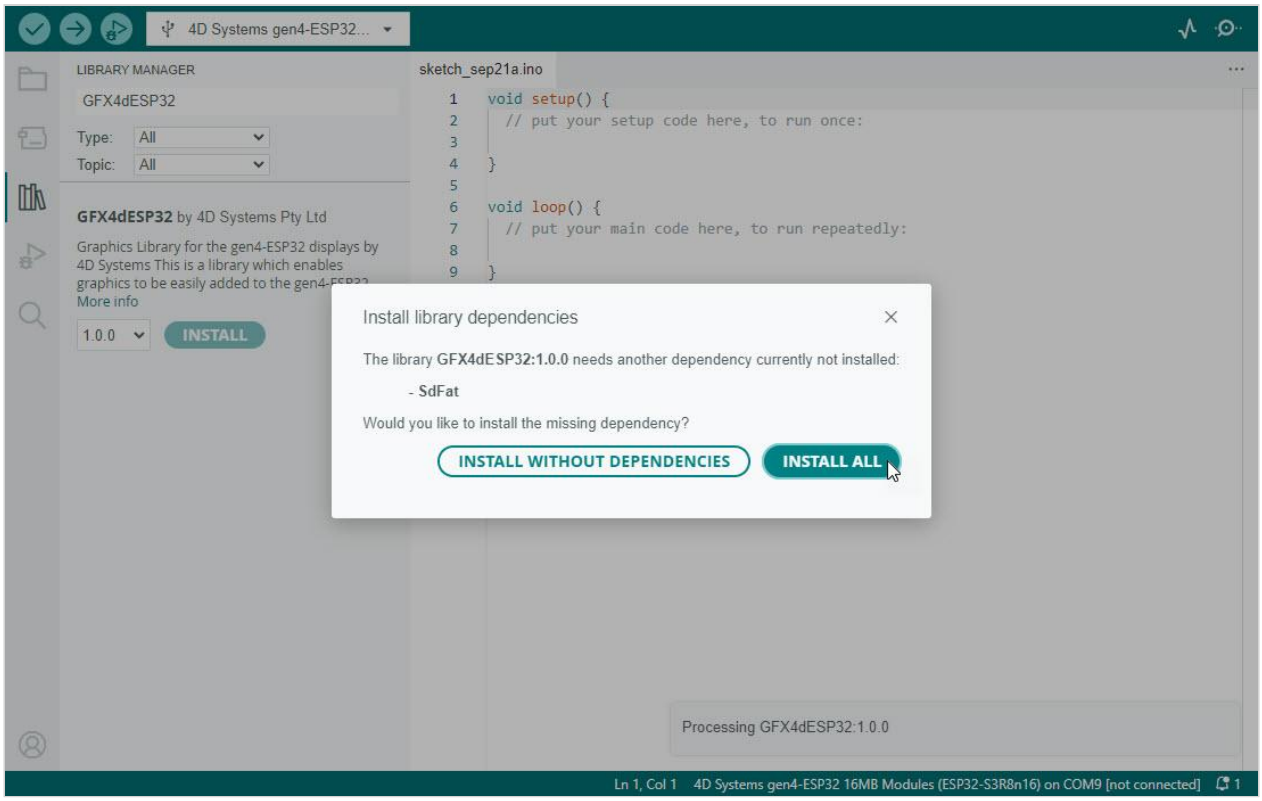
1. Open Library Manager by navigating menu: *Sketch > Include Library > Manage Libraries*



2. Search and install **GFX4dESP32** by 4D Systems Pty Ltd.

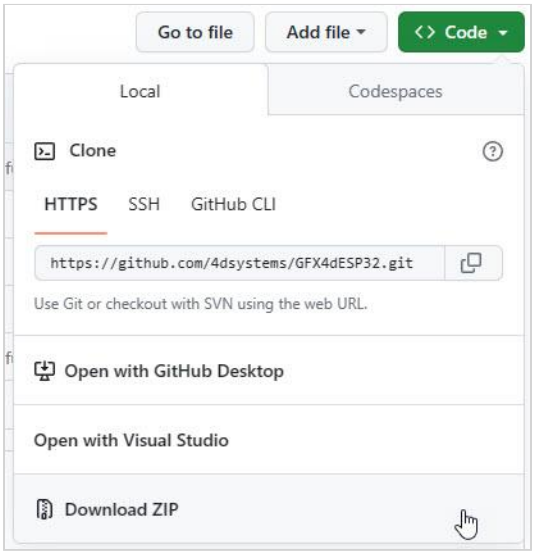


3. Arduino IDE will prompt to install additional dependencies that are not currently installed.

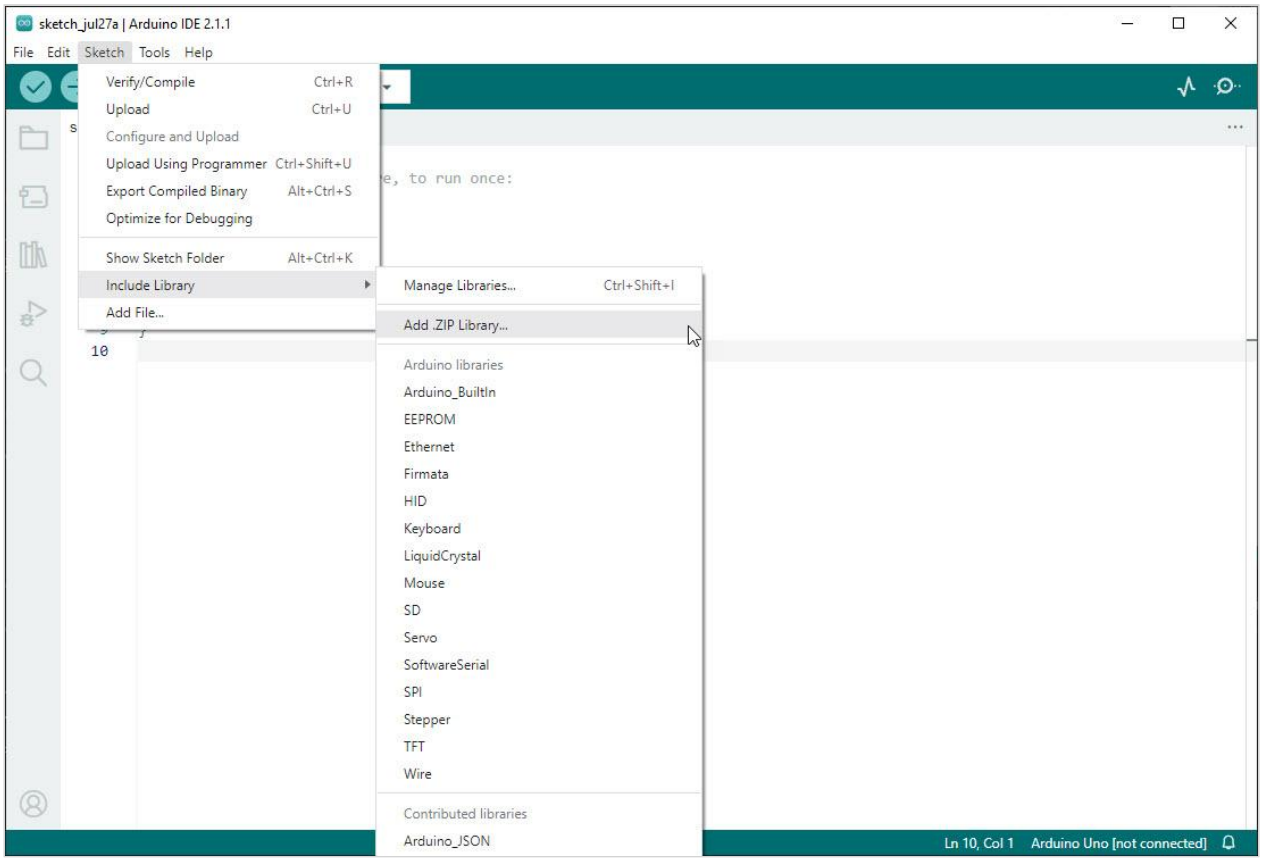


2.2.2. Install from a ZIP Library

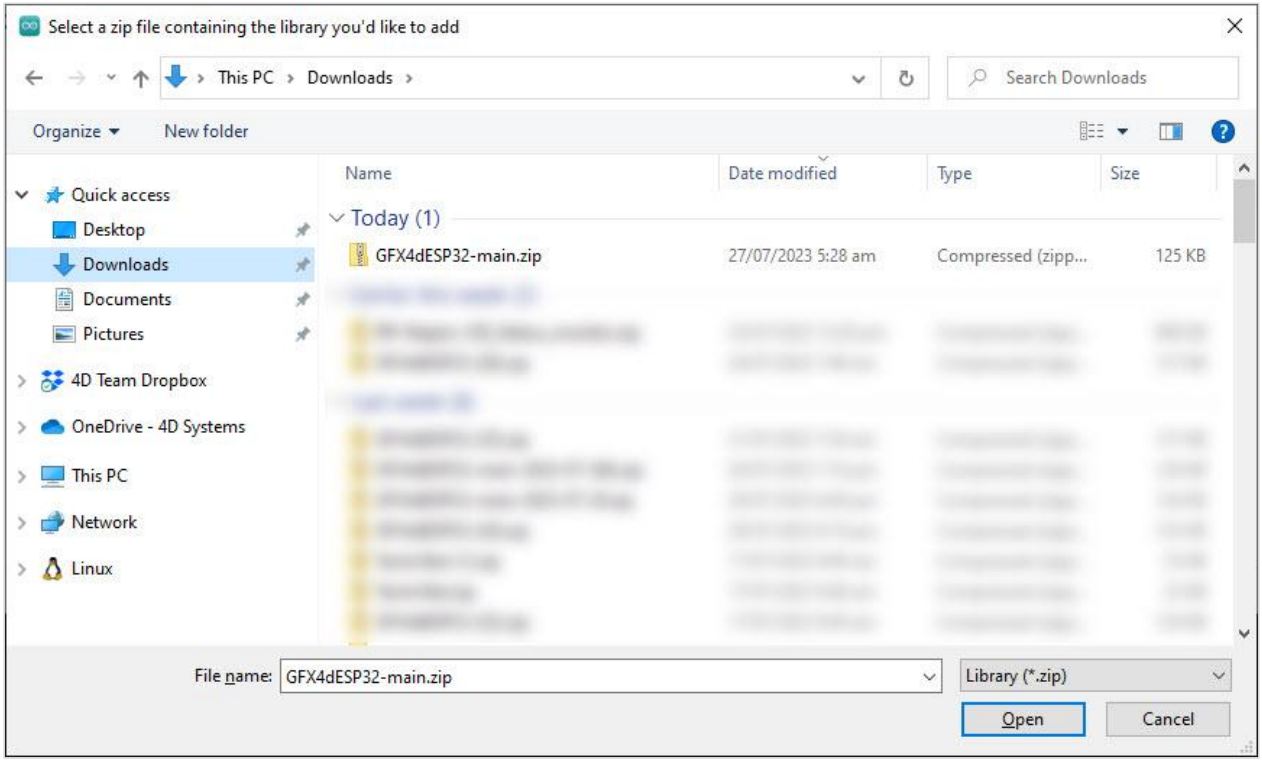
1. Download a ZIP copy of the library from the GitHub repository



2. Select the **Add ZIP Library** option from *Sketch > Include Library* menu.



3. Navigate to the downloaded zip file and click **Open**.



As of writing this document, this method doesn't prompt to automatically install dependencies. Therefore, dependencies need to be installed manually. As of the initial version of the library, the only dependency is SdFat library (GitHub Repository - v2.2.2).

3. Development Roadmap

After having completed the Development Setup, we can create and develop projects using Workshop4.

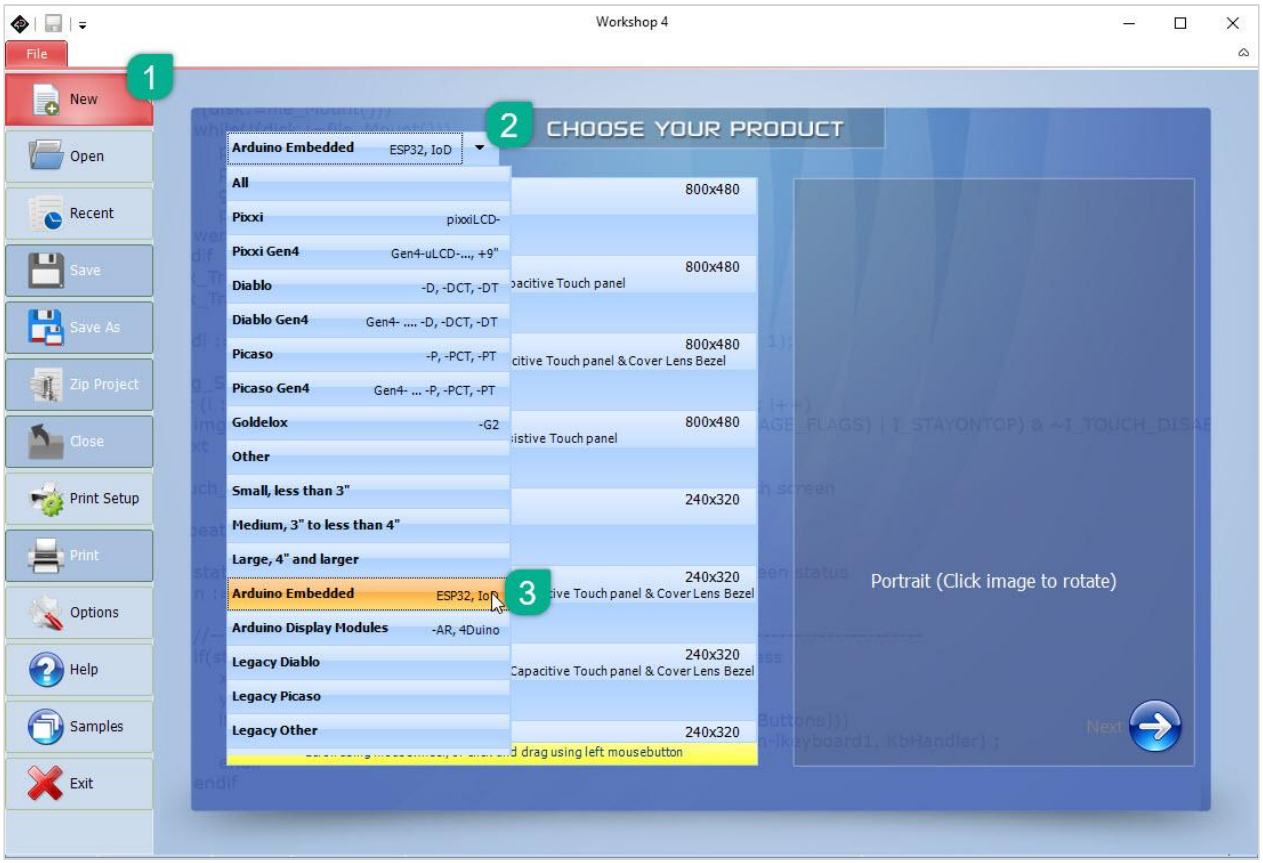
This section discusses about the overall development process including graphics design, writing code and uploading to the display module.

3.1. Creating a New Project

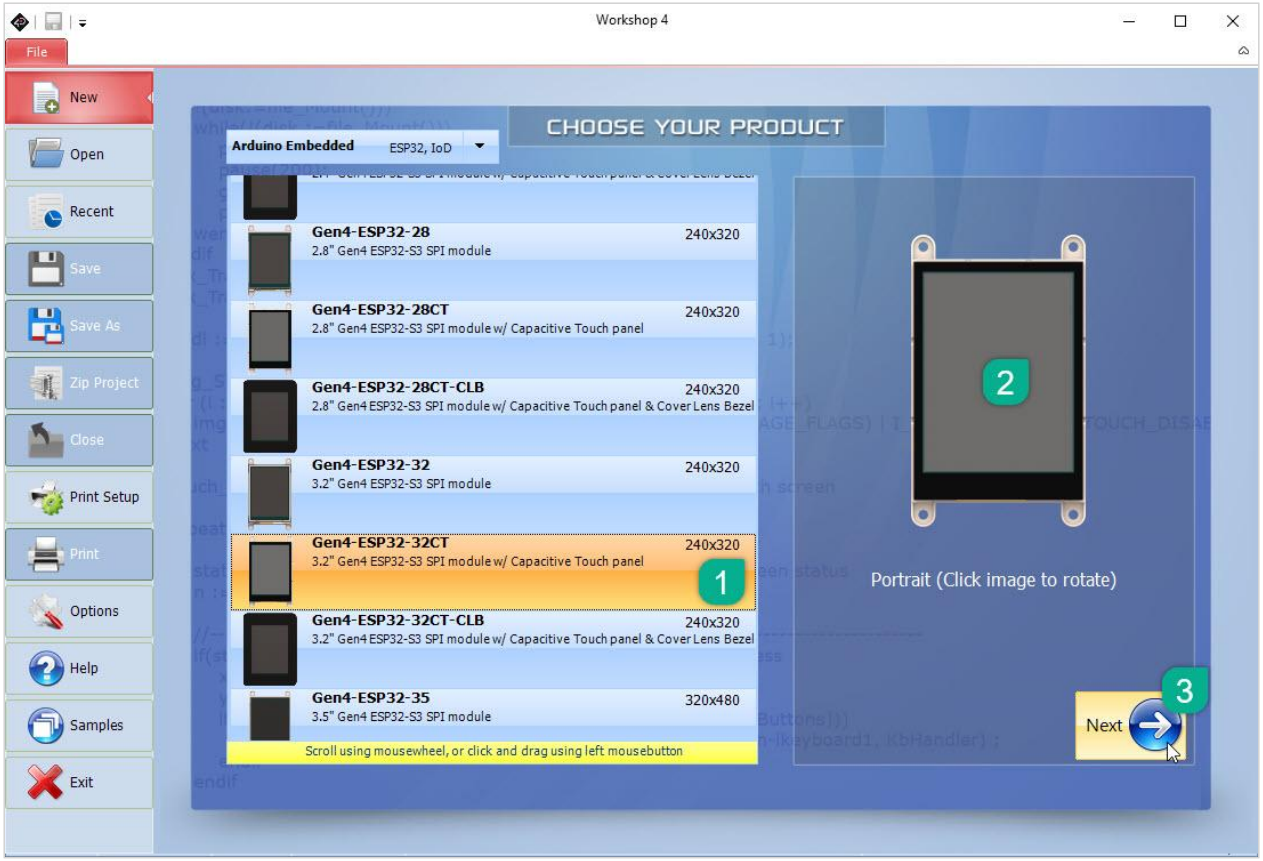
Create a new project by following the procedure below:

1. Open **Workshop4**.

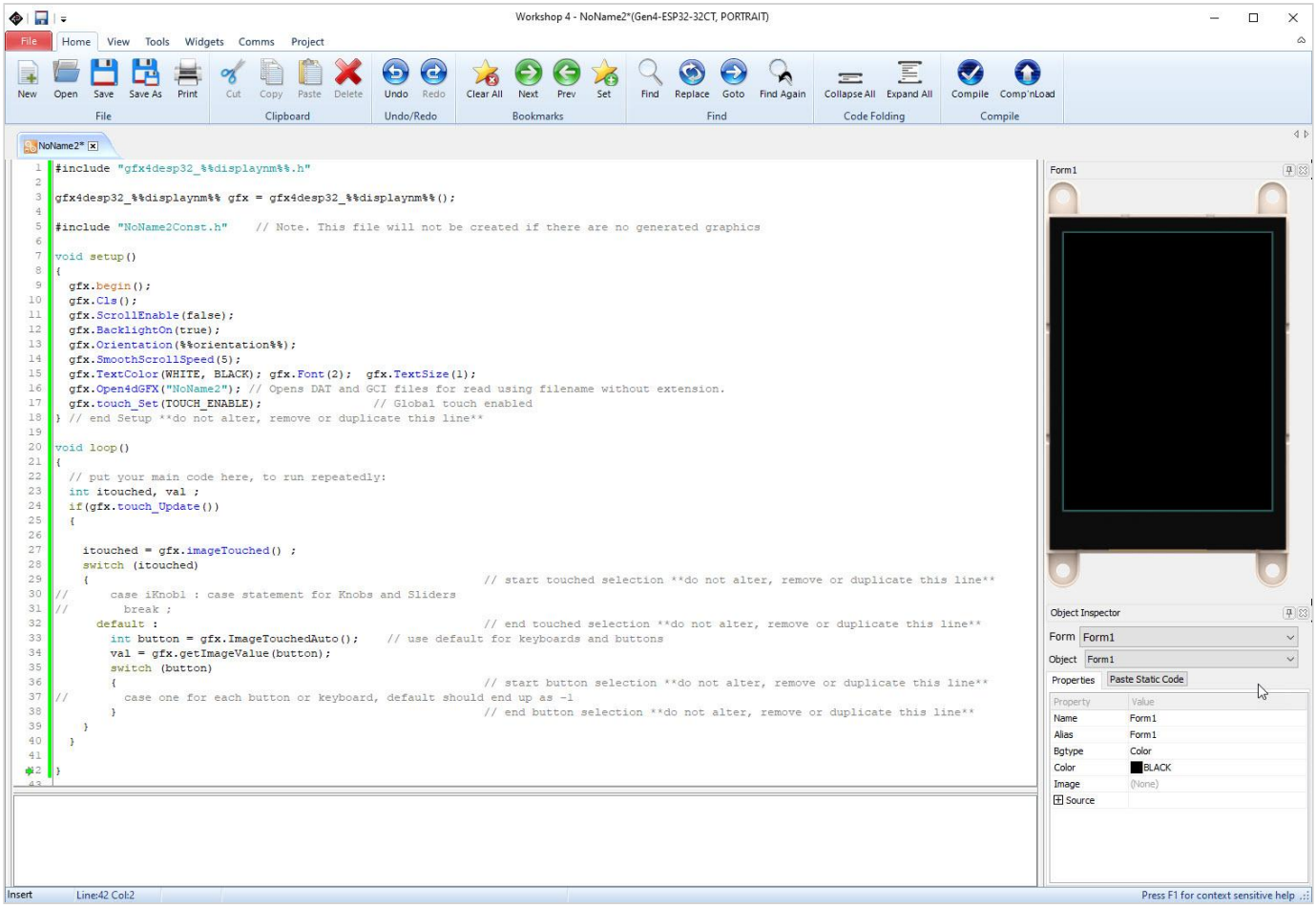
2. Select **New** and filter the ESP32 display modules by selecting **Arduino Embedded** from the dropdown menu



3. Find your display module from the list, select the orientation and confirm your selection.



A fresh project for the selected display will open in a new tab.



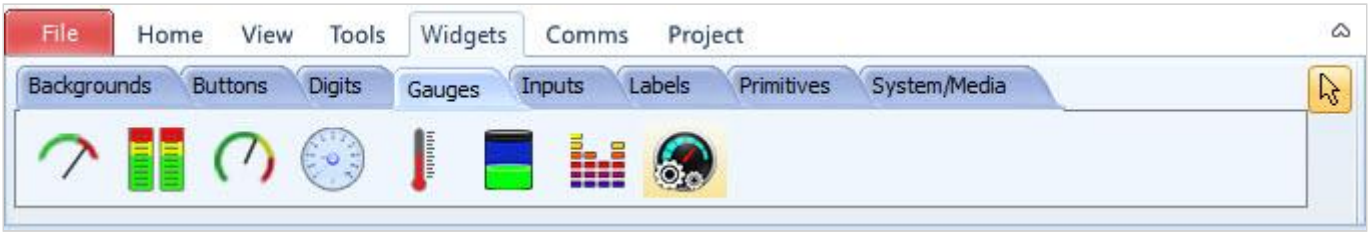
As shown, the project starts with an initial code. Please refer to the Preliminary Code section for a brief discussion.

3.2. Designing a Graphical Interface

By using Workshop4, it's easier than ever to design graphical user interfaces for ESP32 applications. It provides an easy-to-use WYSIWYG editor with support for multiple types of widgets including buttons, sliders, knobs and gauges.

You can refer to the Workshop4 Widgets Reference Manual for more information.

The widgets available for ESP32 devices mainly include GCI widgets and primitive shapes. Workshop4 provides all the available widgets for ESP32-S3 displays under the **Widgets** menu.



3.3. Writing Code

Workshop4 utilises Arduino CLI for compiling, linking and downloading of ESP32-S3 based projects. This lets user develop Arduino code from within Workshop4 IDE while working on a user interface.

3.3.1. Preliminary Code

Workshop4 provides initial code to start with. This includes setup code required for the screen and touch handling code for touch enabled display modules.

The code starts with including the appropriate header file from the GFX4dESP32 library.

```
#include "gfx4desp32_%%displaynm%%.h"

gfx4desp32_%%displaynm%% gfx = gfx4desp32_%%displaynm%%();
```

Note

`%%displaynm%%` is automatically replaced by Workshop4 with the name of the target display module, making it easier to change between different 4D Systems ESP32-S3 modules without the need to change this part of the code.

The **setup** code includes:

```
gfx.begin();
gfx.Cls();
gfx.ScrollEnable(false);
gfx.BacklightOn(true);
gfx.Orientation(%%orientation%%);
gfx.SmoothScrollSpeed(5);
gfx.TextColor(WHITE, BLACK);
gfx.Font(2);
gfx.TextSize(1);
gfx.Open4dGFX("NoName1");     // Opens DAT and GCI files for read using filename without
extension.
gfx.touch_Set(TOUCH_ENABLE);  // Global touch enabled
```

Note

- `%%orientation%%` is automatically replaced by Workshop4 depending on the target display orientation
- `NoName1` is the default format for the name of an unsaved project file. This is carried over to the GCI/DAT files that's copied to the uSD card for GCI widgets. It is also automatically replaced when the project is saved assumming it wasn't edited manually.
- `gfx.touch_Set(TOUCH_ENABLE);` is only generated for touch enabled display modules

The **loop** function includes touch handling code for touch enabled display modules:

```
int itouched, val;
if (gfx.touch_Update()) {
  itouched = gfx.imageTouched();
  // start touched selection
  switch (itouched) { // **do not alter, remove or duplicate this line**
    // case iKnob1:   // case statement for Knobs and Sliders
    //   break;
    // end touched selection
    default:          // **do not alter, remove or duplicate this line**
      int button = gfx.ImageTouchedAuto(); // use default for keyboards and buttons
      val = gfx.getImageValue(button);
      // start button selection
      switch (button) { // **do not alter, remove or duplicate this line**
      // case one for each button or keyboard, default should end up as -1
      } // end button selection **do not alter, remove or duplicate this line**
  }
}
```

Warning

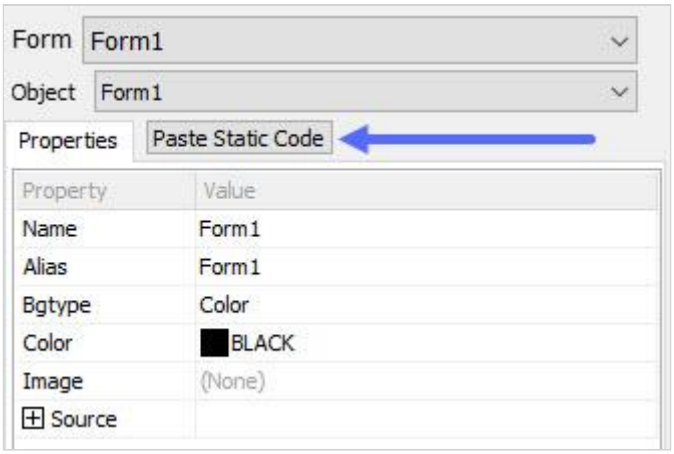
Please take note of the items are commented with **"do not alter, remove or duplicate this line"**. These lines are used by Workshop4 to locate various sections of the code, for when the Paste Code option is used.

3.3.2. Generating Widget Code

Workshop4 is primarily designed for products powered by 4D graphics processors: Pixxi44, Pixxi28, Diablo16, Picaso and Goldelox. It provides multiple environments for developers allowing different level of expertise, from no coding at all to writing code from scratch.

Workshop4's ViSi environment provides the most versatility by allowing users to write their own code while providing a graphics editor. Furthermore, it provides a simple utility that generates code for each widget or object used in the project with a click of a button.

Similar to Workshop4's ViSi environment, ESP32-S3 project provides a **Paste Code** utility that can be used to generate relevant code for the widgets.



This option generates code to update or show widgets at the current cursor position, or more appropriate location or multiple locations in the project.

The following is a list of code snippets Workshop4 generate for a target Workshop4 object.

1. **Show widget initially** - This is generated for all widgets in the **setup** function. This needs to run at least once when switching forms. The generated code must be edited to only show the current Form and widgets in it.
2. **Enabling touch** - This is generated in the **setup** function together with showing the touch input widget once. The generated code must be edited to only enable the input widgets in the current active Form.
3. **Touch handling** - This is generated in **switch-case** block in the **loop** function and can be used to handle which input widget is touched and its new value.
4. **Update widget value** - This is generated for output widgets in the current cursor position.

> **Note**
>
> Workshop4 also generates code inside a header file `<project name>Const.h` based on the widgets added to the project when the graphics is built. This contains constants that can be seen generated with the **Paste Code** option. If there's no generated graphics, this file will not be generated and should be commented out.

Here's an example single form project containing a gauge, a slider input and 2 buttons.

From this project, **Paste Code** option is used for all four widgets while the cursor is in the same position.

The generated code is as shown:

```
#include "gfx4desp32_%%displaynm%%.h"

gfx4desp32_%%displaynm%% gfx = gfx4desp32_%%displaynm%%();

#include "PasteCodeConst.h"
// Note. This file will not be created if there are no generated graphics

void setup()
{
  gfx.begin();
  gfx.Cls();
  gfx.ScrollEnable(false);
  gfx.BacklightOn(true);
  gfx.Orientation(%%orientation%%);
  gfx.SmoothScrollSpeed(5);
  gfx.TextColor(WHITE, BLACK); gfx.Font(2);  gfx.TextSize(1);
  gfx.Open4dGFX("PasteCode"); // Opens DAT and GCI files for read using
                              //   filename without extension.
  gfx.touch_Set(TOUCH_ENABLE);          // Global touch enabled
  gfx.UserImages(iCoolgauge1,0) ;       // init_Coolgauge1 show initially, if required
  gfx.imageTouchEnable(iSlider1, true); // init_Slider1 enable touch of widget (on Form1)
  gfx.UserImages(iSlider1,0) ;          // init_Slider1 show initially, if required (on Form1)
  gfx.imageTouchEnable(iWinbutton1, true, MOMENTARY);
                                        // init_Winbutton1 enable touch of widget (on Form1)
  gfx.UserImages(iWinbutton1,0) ;       // init_Winbutton1 show initially,
                                        //   if required (on Form1)
  gfx.imageTouchEnable(iWinbutton2, true, MOMENTARY);
                                        // init_Winbutton2 enable touch of widget (on Form1)
  gfx.UserImages(iWinbutton2,0) ;       // init_Winbutton2 show initially,
                                        //   if required (on Form1)
} // end Setup **do not alter, remove or duplicate this line**
```

```
void loop()
{
  // cursor position is below this comment during each use of 'Paste Code' option
  gfx.UserImages(iCoolgauge1, frame) ; // where frame is 0 to 100 (for a displayed 0 to 100)

  // cursor position is above this comment during each use of 'Paste Code' option

  int itouched, val ;
  if(gfx.touch_Update())
  {

    itouched = gfx.imageTouched() ;
    switch (itouched)
    { // start touched selection **do not alter, remove or duplicate this line**
      // case iKnob1 : case statement for Knobs and Sliders
      //  break ;
      case iSlider1 :                       // process_Slider1 process (on Form1)
        val = gfx.imageAutoSlider(iSlider1, HORIZONTAL_SLIDER, gfx.touch_GetX(), 8, 8);
        // process Slider based on val
        break ;
      default: // end touched selection **do not alter, remove or duplicate this line**
        int button = gfx.ImageTouchedAuto(); // use default for keyboards and buttons
        val = gfx.getImageValue(button);
        switch (button)
        { // start button selection **do not alter, remove or duplicate this line**
          // case one for each button or keyboard, default should end up as -1
          case iWinbutton1 :                // process_Winbutton1 process Button (on Form1)
            // process win button, for toggle val will be 1 for down and 0 for up
            break ;
          case iWinbutton2 :                // process_Winbutton2 process Button (on Form1)
            // process win button, for toggle val will be 1 for down and 0 for up
            break ;
        } // end button selection **do not alter, remove or duplicate this line**
    }
  }

}
```

Notice that despite the cursor being positioned at the same place for each widget, update code is only generated for the CoolGauge since it is the only output widget.

```
// cursor position is below this comment during each use of 'Paste Code' option
gfx.UserImages(iCoolgauge1, frame) ; // where frame is 0 to 100 (for a displayed 0 to 100)

// cursor position is above this comment during each use of 'Paste Code' option
```

Note

Notice that the generated code for CoolGauge includes the variable *frame* which pertains to the new value to update the gauge to. This variable is not automatically declared since users may prefer to use their own more meaningful variable names.

In the **setup** function, you'll find code to draw widgets initially and to enable touch for input widgets.

```
gfx.UserImages(iCoolgauge1,0) ;       // init_Coolgauge1 show initially, if required
gfx.imageTouchEnable(iSlider1, true); // init_Slider1 enable touch of widget (on Form1)
gfx.UserImages(iSlider1,0) ;          // init_Slider1 show initially, if required (on Form1)
gfx.imageTouchEnable(iWinbutton1, true, MOMENTARY);
                                      // init_Winbutton1 enable touch of widget (on Form1)
gfx.UserImages(iWinbutton1,0) ;       // init_Winbutton1 show initially,
                                      //   if required (on Form1)
gfx.imageTouchEnable(iWinbutton2, true, MOMENTARY);
                                      // init_Winbutton2 enable touch of widget (on Form1)
gfx.UserImages(iWinbutton2,0) ;       // init_Winbutton2 show initially,
                                      //   if required (on Form1)
```

This should be edited as needed as it is only generated to give a suitable starting point. Common changes that needs to be done are:

- **handling multiple forms** - not all forms are drawn at the start and therefore this needs to be edited to suit the project
- **enabling/disabling touch** - some applications may need to initially disable touch for input widgets and only enable at certain conditions
- **hiding widgets initially** - some applications may need to initially hide widgets and only show at certain conditions

Users can freely adjust their application code to suit their needs.

In the **loop** function, code is generated for each input widget inside the touch handling block from the initial code.

```
int itouched, val ;
if (gfx.touch_Update())
{
  itouched = gfx.imageTouched() ;
  switch (itouched)
  { // start touched selection **do not alter, remove or duplicate this line**
    // case iKnob1 : case statement for Knobs and Sliders
    //  break ;
    case iSlider1 :                       // process_Slider1 process (on Form1)
      val = gfx.imageAutoSlider(iSlider1, HORIZONTAL_SLIDER, gfx.touch_GetX(), 8, 8);
      // process Slider based on val
      break ;
    default: // end touched selection **do not alter, remove or duplicate this line**
      int button = gfx.ImageTouchedAuto(); // use default for keyboards and buttons
      val = gfx.getImageValue(button);
      switch (button)
      { // start button selection **do not alter, remove or duplicate this line**
        // case one for each button or keyboard, default should end up as -1
        case iWinbutton1 :                // process_Winbutton1 process Button (on Form1)
          // process win button, for toggle val will be 1 for down and 0 for up
          break ;
        case iWinbutton2 :                // process_Winbutton2 process Button (on Form1)
          // process win button, for toggle val will be 1 for down and 0 for up
          break ;
      } // end button selection **do not alter, remove or duplicate this line**
  }
}
```

The pasted code allows you to simply handle the new value of each input widget. Comments are provided to show which part of the code the touch input value can be handled.

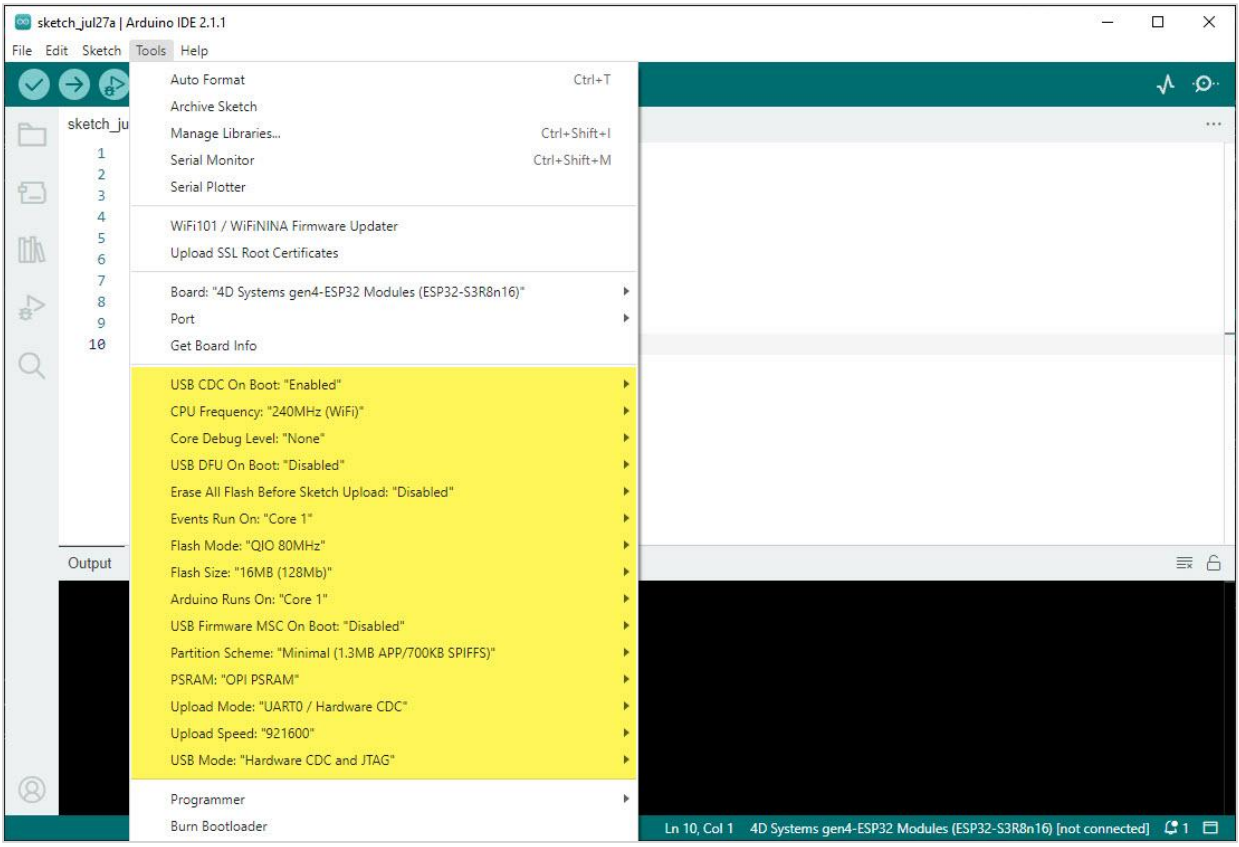
Users can freely add code for handling the new value. However, it is always advisable to refrain from using blocking code as it can affect touch handling.

> **Warning**
>
> Please take note of the items are commented with **"do not alter, remove or duplicate this line"**. These lines are used by Workshop4 to locate various sections of the code, for when the Paste Code option is used.

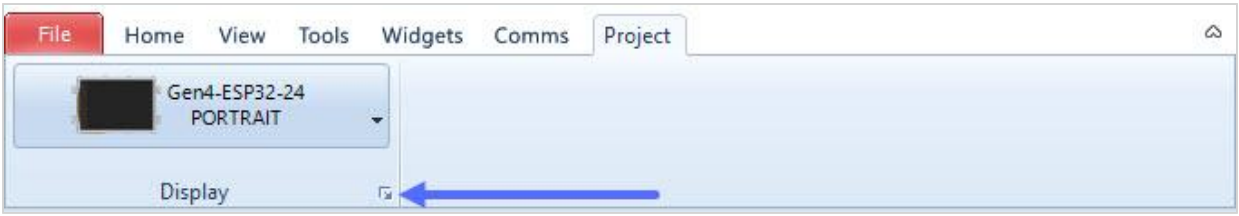
3.4. Programming the Display

3.4.1. Set Target Options

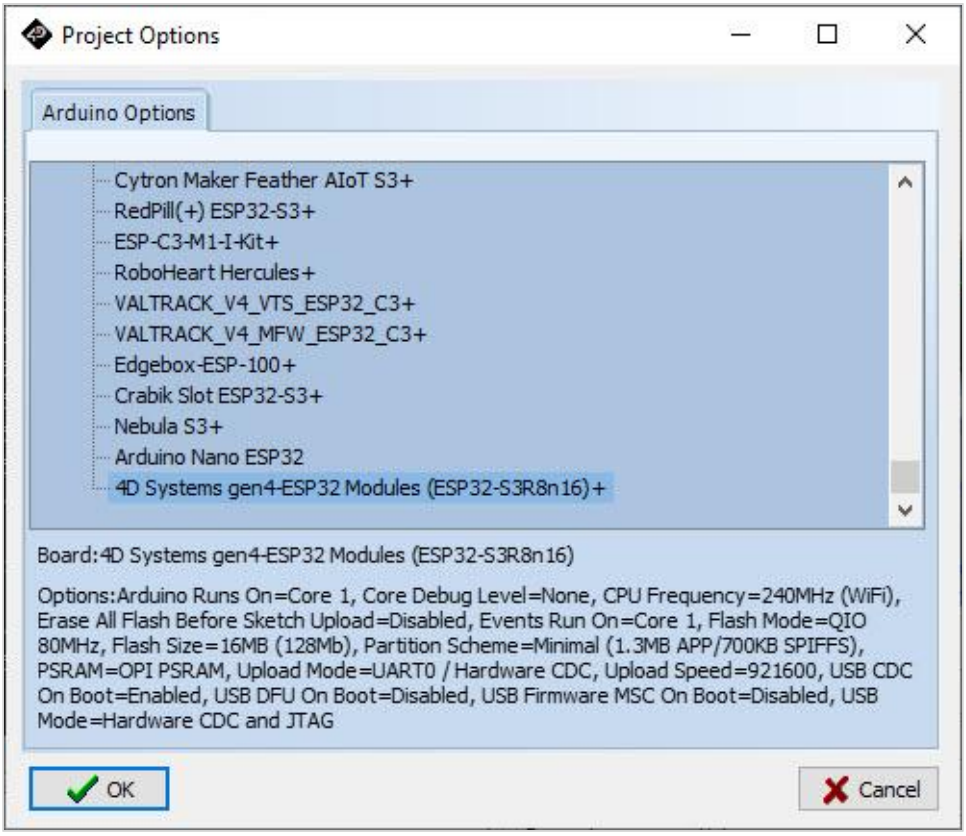
When working with ESP32-S3 devices, several options such as partition table, USB options, etc. can be set. In Arduino, these options can be set from **Tools** menu.



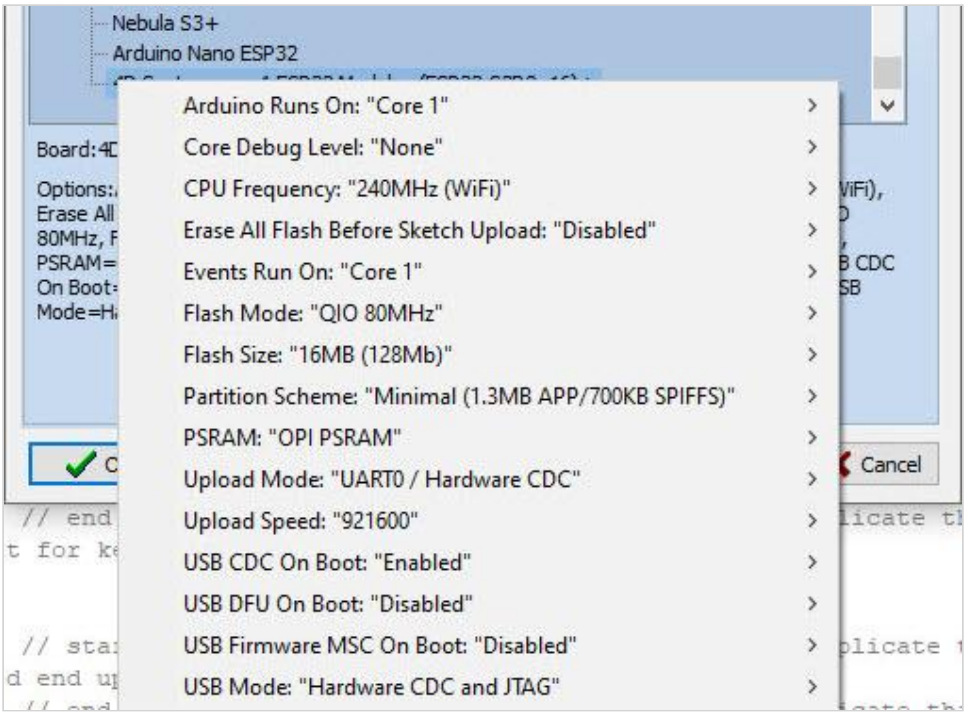
In Workshop4, the same option can be set by going to the **Project** menu and opening **Project Options** window by clicking the button as shown.



From this window, find **4D Systems gen4-ESP32 Modules (ESP32-S3R8N16)**



Right click on it to open a dropdown menu.



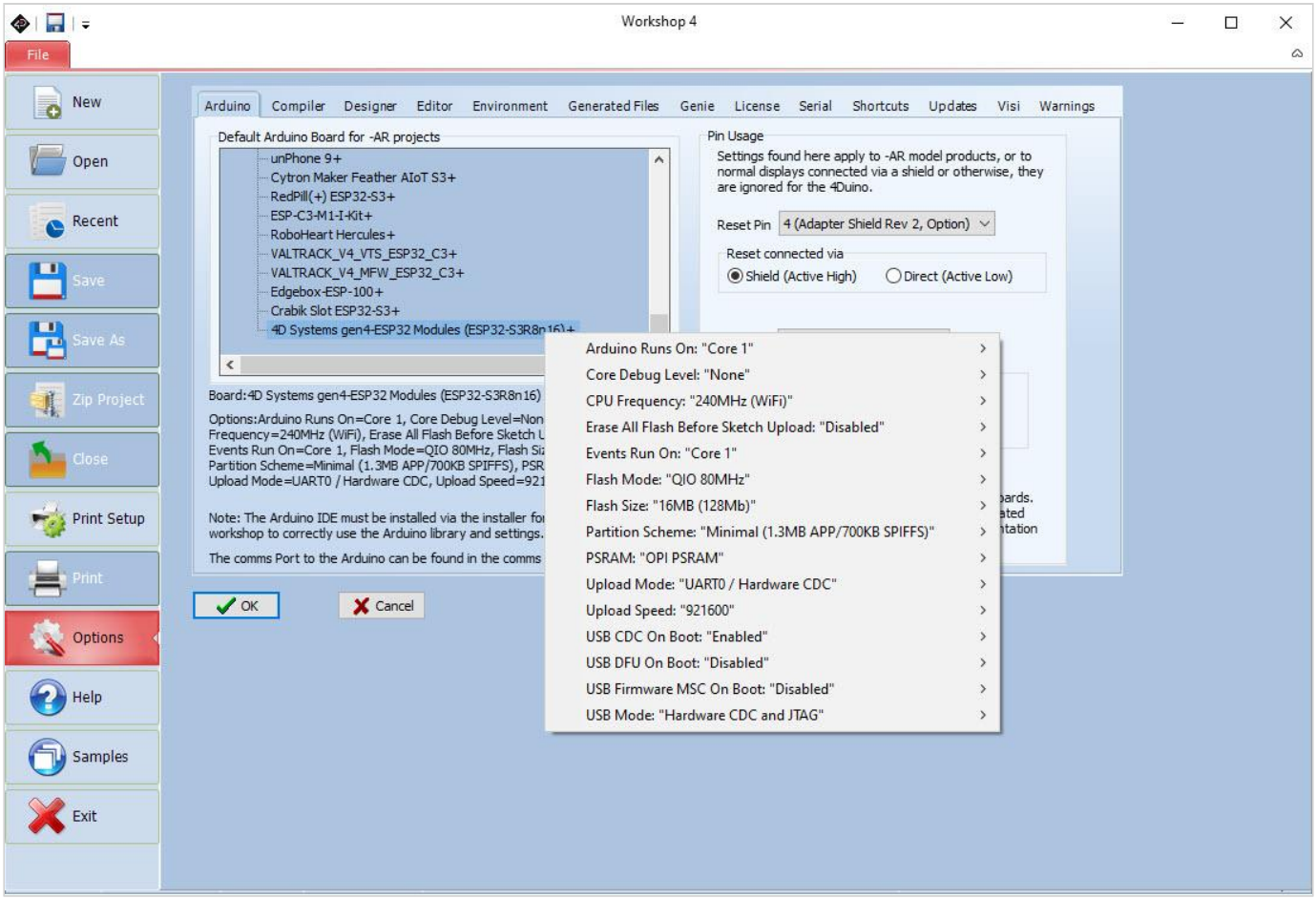
Select the options as needed by your project and press **OK**.

3.4.2. System-Wide Target Options

Unlike Arduino IDE, Workshop4 provides both system-wide (IDE-wide) and project options. Changing the project target options won't affect the system-wide settings.

The system-wide options are used when creating a new project.

To change the system-wide options in Workshop4, go to *File -> Options -> Arduino*.



3.4.3. Uploading the Project

After setting the target options, the project can be compiled and uploaded.

Connect the display module via the USB-C port or using a 4D-UPA (*revision 1.4 or higher*) via the 30-way interface.

Select the target COM port for the display module.

From the **Home** menu, click **Comp'nLoad** to compile and load the program to the display.

4. Legal Notice

4.1. Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. 4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems. 4D Systems reserves the right to modify, update or makes changes to Specifications or written material without prior notice at any time.

All trademarks belong to their respective owners and are recognised and acknowledged.

4.2. Disclaimer of Warranties & Limitations of Liabilities

4D Systems makes no warranty, either expressed or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

Images and graphics used throughout this document are for illustrative purposes only. All images and graphics used are possible to be displayed on the 4D Systems range of products, however the quality may vary.
In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.
4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail - safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems' products and devices in 'High Risk Activities' and in any other application is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.