Chapter 5
User Interface

Dr. Zhang Jianwen

elezhan@nus.edu.sg

Office: E4-05-21

User interface (UI) in IoT

- Web-based UI
  - Hosting a web server at the sensor node device.
  - Control and monitor the device remotely from a web browser.
  - UI can include buttons, sliders, text inputs, and display data like sensor readings.
  - Design UI and handle user interaction with HTML/CSS and C/C++.

Web-based UI example

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>

// Replace with your network credentials
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

// HTML content
const char* htmlContent = R"rawliteral(
<html>
<head>
    <title>ESP32 Web UI</title>
</head>
<body>
    <h1>ESP32 Control Panel</h1>
    <button onclick="sendCommand('LED_ON')">Turn ON LED</button>
    <button onclick="sendCommand('LED_OFF')">Turn OFF LED</button>
    <script>
        function sendCommand(command) {
            fetch("/command?cmd=" + command);
        }
    </script>
</body>
</html>
)rawliteral";

// LED pin
const int ledPin = 2;
```

```
void setup() {
  // Start the Serial Monitor
  Serial.begin(115200);

  // Set LED pin as an output
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW); // Start with LED off

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
```

```
 // Serve the HTML content at the root URL
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/html", htmlContent);
  });

// Handle LED control via GET request
  server.on("/command", HTTP_GET, [](AsyncWebServerRequest
*request){
    String command;
    if (request->hasParam("cmd")) {
      command = request->getParam("cmd")->value();

      if (command == "LED_ON") {
        digitalWrite(ledPin, HIGH); // Turn LED ON
        Serial.println("LED turned ON");
        request->send(200, "text/plain", "LED is ON");
      } else if (command == "LED_OFF") {
        digitalWrite(ledPin, LOW); // Turn LED OFF
        Serial.println("LED turned OFF");
        request->send(200, "text/plain", "LED is OFF");
      } else {
        request->send(400, "text/plain", "Invalid Command");
      }
    }
  });

  // Start the server
  server.begin();
}

void loop() {
  // Nothing to do here, everything is handled asynchronously
}
```

User interface (UI) in IoT

- OLED/Display-based UI
  - Real-tine on-device interaction by OLED or LCD display
  - Create graphic UI with libraries and tools:
    - Adafruit_GFX
    - TFT_eSPI
    - U8g2
    - Light and Versatile Graphics Library（LVGL）
    - SquareLine

Libraries

- Adafruit_GFX
  - Lightweight graphics library designed for basic 2D graphics on small displays.
  - Supports drawing primitives such as lines, circles, rectangles, and text.
  - Works with many display types but is often used in combination with Adafruit display libraries.
  - Simplicity, ease of use, well-documented, and widely used in basic projects.
  - Not suitable for advanced UI elements or high-performance graphics. For large or complex UIs, it can be insufficient.

Libraries

- TFT_eSPI
  - For high-performance TFT displays optimized for ESP32 and STM32 microcontrollers. Support LCD display ST7735, ST7789, ILI9341, and others.
  - High-speed drawing, touchscreen support, advanced font handling, image rendering.
  - Good support for text and font handling, including custom fonts.
  - High speed and suitable for large, high-resolution displays. It’s also great for more sophisticated graphics and interfaces.
  - Requires more memory and is more complex to configure.

Libraries

- TFT_eSPI basic APIs

```
tft.fillScreen(TFT_BLACK);  // Clear the screen

tft.setCursor(x, y);  // Set cursor for text

tft.setTextColor(TFT_WHITE, TFT_BLACK);
// Set text color and background

tft.print("Hello ESP32");  // Print text to the display

tft.drawRect(x, y, width, height, color);
// Draw a rectangle

tft.drawCircle(x, y, radius, color);  // Draw a circle
```

TFT_eSPI example

```
#include <TFT_eSPI.h>  // Graphics and font library for ST7735 and ILI9341

TFT_eSPI tft = TFT_eSPI(); // Invoke custom library

void setup() {
    tft.init();
    tft.setRotation(1);  // Set orientation of the screen

    tft.fillScreen(TFT_BLACK);  // Clear screen

    tft.setTextSize(2);  // Set text size
    tft.setTextColor(TFT_WHITE, TFT_BLACK);  // Set text and background colors
    tft.setCursor(10, 10);
    tft.print("Hello, ESP32!");
}

void loop() {
    // You can update the display dynamically here
}
```

Libraries

- U8g2
  - Designed for monochrome displays like OLED or e-ink screens. It supports a wide variety of small, low-power displays like SSD1306, SH1106.
  - Focuses on low-resource devices with minimal display memory requirements.
  - Font rendering is highly optimized, with support for multiple sizes and styles. Rotated text and graphics are supported.
  - Good support for text and font handling, including custom fonts.
  - Memory-efficient. Able to handle multiple fonts and display types.

Libraries

- LVGL
  - Powerful and full-featured graphics library designed to create advanced graphical user interfaces, especially on low-power devices.
  - Advanced GUI elements such as buttons, sliders, text boxes, charts, etc.
  - Support for multiple themes, anti-aliasing, image rendering, and fonts.
  - Touch support with gesture recognition.
  - Suitable for complex UIs with animations, transitions, and multi-page layouts.

Libraries

- LVGL
  - Works with different color-depth configurations and display resolutions.
  - The most advanced of the libraries optimized for both speed and memory consumption.
  - Build modern and highly interactive graphical user interfaces (GUIs) on devices with limited resources.
  - Steeper learning curve and higher memory/CPU requirements. More suitable for projects where advanced interfaces are necessary, but not ideal for simpler display needs.

LVGL

- Key Features of LVGL
  - Lightweight: Optimized for microcontrollers with limited RAM and Flash.
  - Flexibility: It supports various display drivers and resolutions.
  - Rich Widgets: Buttons, sliders, labels, charts, and more.
  - Animations: Smooth animations for better user experience.
  - Theming: Easily customizable themes and styles.
  - Input Device Support: Handles touch, encoder, or button inputs.

LVGL

- Use Cases for LVGL
  - Smart Home Controller panel.
  - Wearables.
  - Industrial Automation.
  - Dashboards and Data Visualizations.
  - Media Players interface.
  - Custom Touchscreen Devices.

LVGL

- Basic LVGL elements
  - Widgets: LVGL provides basic building blocks called widgets, like labels, buttons, sliders, switches, and more.
  - Styles: Styles define the appearance (colors, fonts, borders) of widgets.
  - Screens: These are the containers where the widgets are placed. An LVGL application can have multiple screens.
  - Events: User interaction triggers events, such as button clicks or slider changes, which can be handled by callback functions.

LVGL

- Setting Up LVGL with ESP32-S3 in Arduino IDE
  - LVGL Library: Install the LVGL library from the Arduino Library Manager.
  - Display Driver: Install the appropriate display driver for your screen (e.g., TFT_eSPI for SPI-based TFT displays).
  - Configure the Display and Input: Set up the display and touch screen parameters in sketch.

Example: ILI9341 display on ESP32-S3

- Install Libraries.
  - Install the TFT_eSPI library in Arduino IDE (Display driver)

    Sketch > Include Library > Manage Libraries and search for TFT_eSPI
  - Install the LVGL library either from Arduino IDE library manager or from the Github LVGL repo.
  - Configure TFT_eSPI for ILI9341.
    - Editt *User_Setup.h* in TFT_eSPI library folder (*Documents/Arduino/libraries/TFT_eSPI*) to match the hardware connections for ILI9341 as shown in next slide.

Modify the User_Setup.h file to match your ILI9341 configuration

```
#define ILI9341_DRIVER      // Define the driver for ILI9341

#define TFT_CS   15  // Chip select control pin
#define TFT_RST  4   // Reset pin (could be connected to ESP32 RESET)
#define TFT_DC   2   // Data/Command control pin
#define TFT_SCLK 18  // SPI clock pin
#define TFT_MOSI 23  // SPI MOSI pin (Master Out Slave In)
#define TFT_MISO 19  // SPI MISO pin (Master In Slave Out)
```

Minimal setup to initialize LVGL with an ESP32

```
#include <TFT_eSPI.h>    // Include the TFT_eSPI library
#include <lvgl.h>        // Include the LVGL library

TFT_eSPI tft = TFT_eSPI(); // Create an instance of the TFT_eSPI class

#define LVGL_TICK_PERIOD 5

void lv_tick_handler(void) {
  lv_tick_inc(LVGL_TICK_PERIOD);
}

void setup() {
  Serial.begin(115200);

  // Initialize TFT
  tft.begin();
  tft.setRotation(1);  // Adjust based on your display orientation

  // Initialize LVGL
  lv_init();

  // Create a buffer for LVGL
  static lv_color_t buf1[240 * 10];  // Buffer size: width * height (rows)
  static lv_disp_draw_buf_t draw_buf;
  lv_disp_draw_buf_init(&draw_buf, buf1, NULL, 240 * 10);

  // Initialize the display driver in LVGL
  static lv_disp_drv_t disp_drv;
  lv_disp_drv_init(&disp_drv);

  // Set the flush function, which sends the buffer to the display
  disp_drv.flush_cb = my_disp_flush;
  disp_drv.draw_buf = &draw_buf;
  disp_drv.hor_res = 240;  // Horizontal resolution of your display
  disp_drv.ver_res = 320;  // Vertical resolution of your display
  lv_disp_drv_register(&disp_drv);
```

Minimal setup to initialize LVGL with an ESP32

```
// Initialize a timer to call `lv_tick_handler`
  const esp_timer_create_args_t periodic_timer_args = {
      .callback = &lv_tick_handler,
      .name = "periodic_gui"
  };
  esp_timer_handle_t periodic_timer;
  esp_timer_create(&periodic_timer_args, &periodic_timer);
  esp_timer_start_periodic(periodic_timer, LVGL_TICK_PERIOD * 1000);  // Call every 5 ms

}

void loop() {
  lv_timer_handler();  // Handle LVGL tasks
  delay(5);
}

// Flush the buffer to the display
void my_disp_flush(lv_disp_drv_t *disp, const lv_area_t *area, lv_color_t *color_p) {
  uint16_t c;
  tft.startWrite();
  tft.setAddrWindow(area->x1, area->y1, area->x2 - area->x1 + 1, area->y2 - area->y1 + 1);

  for (int y = area->y1; y <= area->y2; y++) {
    for (int x = area->x1; x <= area->x2; x++) {
      c = color_p->full;
      tft.writeColor(c, 1);
      color_p++;
    }
  }
  tft.endWrite();
  lv_disp_flush_ready(disp);
}
```

Example: ILI9341 display on ESP32-S3

- Customizing and Adding UI Elements using LVGL API.
  - Add a label

```
void create_label() {
    lv_obj_t *label = lv_label_create(lv_scr_act()); // Create a label on the active screen
    lv_label_set_text(label, "Hello, LVGL!");        // Set the text of the label
    lv_obj_align(label, LV_ALIGN_CENTER, 0, 0);      // Align the label to the center of the screen
}
```

Example: ILI9341 display on ESP32-S3

- Add a button with an event callback

```
// Event handler function for the button
void btn_event_handler(lv_event_t *e) {
    lv_obj_t *btn = lv_event_get_target(e); // Get the button object
    LV_UNUSED(btn);

    lv_obj_t *label = lv_label_create(lv_scr_act());
    lv_label_set_text(label, "Button Clicked!");
    lv_obj_align(label, LV_ALIGN_CENTER, 0, 40); // Show message below the button
}

void create_button() {
    lv_obj_t *btn = lv_btn_create(lv_scr_act()); // Create a button on the active screen
    lv_obj_set_size(btn, 100, 50);               // Set the button size
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -40);  // Align the button to the center of the screen

    lv_obj_t *label = lv_label_create(btn);      // Create a label on the button
    lv_label_set_text(label, "Press Me");        // Set the button text

    lv_obj_add_event_cb(btn, btn_event_handler, LV_EVENT_CLICKED, NULL); // Add click event
}
```

Example: ILI9341 display on ESP32-S3

- Add a slider

```
void slider_event_handler(lv_event_t *e) {
    lv_obj_t *slider = lv_event_get_target(e); // Get the slider object
    int value = lv_slider_get_value(slider);   // Get the current value of the slider

    // Display the value on a label
    static lv_obj_t *label = NULL;
    if (!label) {
        label = lv_label_create(lv_scr_act());
        lv_obj_align(label, LV_ALIGN_CENTER, 0, 20);
    }
    char buf[32];
    snprintf(buf, sizeof(buf), "Slider Value: %d", value);
    lv_label_set_text(label, buf);
}

void create_slider() {
    lv_obj_t *slider = lv_slider_create(lv_scr_act()); // Create a slider on the active screen
    lv_obj_set_width(slider, 200);                     // Set the slider width
    lv_obj_align(slider, LV_ALIGN_CENTER, 0, 0);       // Align the slider to the center of the screen
    lv_obj_add_event_cb(slider, slider_event_handler, LV_EVENT_VALUE_CHANGED, NULL); // Add event callback
}
```

LVGL core APIs

- Creating Basic Objects
  - Objects are the building blocks in LVGL, like labels, buttons, and images. Each object has a type and properties like position, size, style, and more.
  - lv_obj_create(): Create a new object on a specified parent (or the default parent if none is provided).

Create a button

```
lv_obj_t * btn = lv_btn_create(lv_scr_act());  // Create a button on the active screen
lv_obj_set_size(btn, 100, 50);                 // Set button size
lv_obj_align(btn, LV_ALIGN_CENTER, 0, 0);      // Center the button on the screen
```

LVGL core APIs

- Labels
  - Labels are used to display text on the screen. You can create, style, and manipulate text using the label APIs.
  - lv_label_create(): Create a new label.

Create and Set Text in a Label

```
lv_obj_t * label = lv_label_create(lv_scr_act());  // Create a label on the active screen
lv_label_set_text(label, "Hello, LVGL!");          // Set label text
lv_obj_align(label, LV_ALIGN_CENTER, 0, -50);      // Align label 50 px above the center
```

LVGL core APIs

- Button Matrices
  - Button matrices are used for creating keypads or menus. Each button has an index and a text label.
  - lv_btnmatrix_create(): Create a new button matrix.

Create a Button Matrix with Multiple Buttons

```
// Button labels
static const char * btn_map[] = { "1", "2", "3", "\n", "4", "5", "6", "\n", "7", "8", "9", "" };

lv_obj_t * btnm = lv_btnmatrix_create(lv_scr_act());   // Create button matrix
lv_btnmatrix_set_map(btnm, btn_map);                   // Assign map to button matrix
lv_obj_align(btnm, LV_ALIGN_CENTER, 0, 50);            // Align at the center (50px down)
```

LVGL core APIs

- Image Objects
  - Images are an important part of GUIs. LVGL supports different image formats (e.g., raw C arrays, binary files, and compressed formats).
  - lv_img_create(): Create a new image object.

Display an Image from a File

```
lv_obj_t * img = lv_img_create(lv_scr_act());          // Create an image object

// Set image source (e.g., `my_image` defined in C)
lv_img_set_src(img, &my_image);

lv_obj_align(img, LV_ALIGN_CENTER, 0, 0);              // Align at the center
```

LVGL core APIs

- Creating and Handling Events
  - LVGL supports events such as button presses, value changes, and other interactions. These can be handled using callback functions.
  - lv_obj_add_event_cb(): Add an event callback to an object.

Handling Button Click Events

```
void button_event_handler(lv_event_t * e) {
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * btn = lv_event_get_target(e);
    if(code == LV_EVENT_CLICKED) {
        LV_LOG_USER("Button clicked");
    }
}

lv_obj_t * btn = lv_btn_create(lv_scr_act());  // Create a button
lv_obj_add_event_cb(btn, button_event_handler, LV_EVENT_CLICKED, NULL);  // Add click event
lv_obj_align(btn, LV_ALIGN_CENTER, 0, 0);      // Align button at the center
```

Creating a Slider with an Event Callback

```
// Event callback for the slider
void slider_event_cb(lv_event_t *e) {
    lv_obj_t *slider = lv_event_get_target(e);            // Get the slider object
    int value = lv_slider_get_value(slider);              // Get slider value
    lv_label_set_text_fmt(label, "Value: %d", value);     // Update label with slider value
}

// Create a slider
lv_obj_t *slider = lv_slider_create(lv_scr_act());        // Create a slider
lv_obj_set_width(slider, 200);                            // Set slider width
lv_obj_align(slider, LV_ALIGN_CENTER, 0, 0);              // Align to center

// Create a label to display the slider value
lv_obj_t *label = lv_label_create(lv_scr_act());
lv_label_set_text(label, "Value: 0");
lv_obj_align(label, LV_ALIGN_CENTER, 0, 40);

// Add event callback for the slider
lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED, NULL);
```

Creating an Animation

```
// Animation callback function
void anim_x_cb(void * obj, int32_t v) {
    lv_obj_set_x((lv_obj_t *)obj, v);  // Set new x position during animation
}

// Create an object
lv_obj_t * obj = lv_obj_create(lv_scr_act());
lv_obj_set_size(obj, 50, 50);
lv_obj_set_pos(obj, 10, 10);

// Create an animation
lv_anim_t a;
lv_anim_init(&a);
lv_anim_set_var(&a, obj);              // Set object to be animated
lv_anim_set_values(&a, 10, 200);       // Set animation start and end values
lv_anim_set_time(&a, 1000);            // Set animation duration (1 second)
lv_anim_set_exec_cb(&a, anim_x_cb);    // Set callback to execute during animation
lv_anim_start(&a);                     // Start the animation
```

Handling Touchscreen Input

```
void touch_read(lv_indev_drv_t *drv, lv_indev_data_t *data) {
    int16_t touchX, touchY;
    bool touched = touch.readTouch(touchX, touchY);   // Read touch coordinates from your touchscreen controller

    if (!touched) {
        data->state = LV_INDEV_STATE_REL;
    } else {
        data->state = LV_INDEV_STATE_PR;
        data->point.x = touchX;
        data->point.y = touchY;
    }
}

void setup_touch() {
    static lv_indev_drv_t indev_drv;
    lv_indev_drv_init(&indev_drv);             // Initialize the input driver
    indev_drv.type = LV_INDEV_TYPE_POINTER;    // Register as a pointer device (touchscreen)
    indev_drv.read_cb = touch_read;            // Set the callback for touch input
    lv_indev_drv_register(&indev_drv);         // Register the input device driver
}
```

LVGL core APIs

- Creating Charts and Graphs
  - Charts are useful for data visualization and monitoring real-time data.
  - lv_chart_create(): Create a new chart object.

Creating a Chart and Adding Data

```
lv_obj_t * chart = lv_chart_create(lv_scr_act());   // Create a chart object
lv_obj_set_size(chart, 200, 150);                   // Set chart size
lv_obj_center(chart);                               // Center the chart on the screen

lv_chart_set_type(chart, LV_CHART_TYPE_LINE);       // Set the chart type to line

// Create a new data series
lv_chart_series_t * ser = lv_chart_add_series(chart, lv_color_hex(0xFF0000), LV_CHART_AXIS_PRIMARY_Y);

// Add points to the chart
lv_chart_set_point_id(chart, ser, 10, 0);           // Set the first point to 10
lv_chart_set_point_id(chart, ser, 20, 1);           // Set the second point to 20
lv_chart_set_point_id(chart, ser, 30, 2);           // Set the third point to 30
```

LVGL core APIs

- Slider Objects
  - Sliders allow users to set a value by dragging a thumb indicator.
  - lv_slider_create(): Create a new slider object.

Create a Slider and Display Value

```
lv_obj_t * slider = lv_slider_create(lv_scr_act());   // Create a slider
lv_obj_set_width(slider, 200);                        // Set width
lv_obj_center(slider);                                // Center the slider

// Create a label to show the slider value
lv_obj_t * label = lv_label_create(lv_scr_act());
lv_label_set_text(label, "Value: 0");
lv_obj_align(label, LV_ALIGN_CENTER, 0, 30);          // Position below slider

void slider_event_cb(lv_event_t * e) {
    lv_obj_t * slider = lv_event_get_target(e);
    int value = lv_slider_get_value(slider);
    lv_label_set_text_fmt(label, "Value: %d", value);  // Update label with slider value
}

lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED, NULL);  // Attach event callback
```

LVGL core APIs

- Text Areas
  - Text areas allow for multi-line text input and editing.
  - lv_textarea_create(): Create a new text area.

Create a Text Area with Placeholder Text

```
lv_obj_t * textarea = lv_textarea_create(lv_scr_act());   // Create a text area
lv_textarea_set_text(textarea, "");                       // Initialize with empty text
lv_textarea_set_placeholder_text(textarea, "Enter text..."); // Set placeholder text
lv_obj_set_size(textarea, 200, 100);                      // Set size of text area
lv_obj_center(textarea);                                  // Center the text area
```

LVGL core APIs

- Custom Styling
  - Styles control colors, borders, shadows, and more.
  - lv_style_set_bg_color(): Set the background color for a style.

Create a Button with a Custom Style

```
lv_style_t style;
lv_style_init(&style);
lv_style_set_bg_color(&style, lv_color_hex(0x00FF00));   // Set background color to green

lv_obj_t * btn = lv_btn_create(lv_scr_act());            // Create a button
lv_obj_add_style(btn, &style, LV_PART_MAIN);             // Apply the style
lv_obj_center(btn);                                      // Center the button
```

LVGL core APIs

- Page Navigation and Tabs
  - Create multiple pages and navigate between them.
  - lv_tabview_create(): Create a tab view with multiple pages.

Create a Tab View

```
lv_obj_t * tabview = lv_tabview_create(lv_scr_act(), LV_DIR_TOP, 50);
lv_obj_t * tab1 = lv_tabview_add_tab(tabview, "Tab 1");
lv_obj_t * tab2 = lv_tabview_add_tab(tabview, "Tab 2");

lv_obj_t * label1 = lv_label_create(tab1);
lv_label_set_text(label1, "This is the content of Tab 1");

lv_obj_t * label2 = lv_label_create(tab2);
lv_label_set_text(label2, "This is the content of Tab 2");
```

LVGL core APIs

- Display Management APIs
  - lv_disp_drv_register(): Registers a display driver with LVGL.
  - lv_disp_flush_ready(): Indicates that the display flushing is completed.
  - lv_disp_t *lv_disp_get_default(): Gets the default display for your system.

LVGL core APIs

- Object Management APIs
  - lv_obj_create(): Creates a base LVGL object. All widgets (buttons, labels, etc.) inherit from this base object.
  - lv_obj_align(): Aligns an object relative to other objects or the screen.
  - lv_obj_set_size(): Sets the size of an object (width and height).
  - lv_obj_set_pos(): Sets the position of an object (x, y coordinates).

LVGL core APIs

- Widgets APIs LVGL provides APIs for creating various UI components.
  - Buttons: lv_btn_create(), lv_btn_set_label()
  - Labels: lv_label_create(), lv_label_set_text()
  - Sliders: lv_slider_create(), lv_slider_set_value()
  - Check Boxes: lv_checkbox_create(), lv_checkbox_set_text()

LVGL core APIs

- Event Handling APIs
  - lv_obj_add_event_cb(): Adds a callback function to handle events (like button press or release).
  - lv_event_get_code(): Retrieves the event code when an event is triggered.
  - lv_event_get_target(): Gets the target object of the event.
- Animation APIs
  - lv_anim_create(): Creates animations for any object properties (like position, size, etc.).
  - lv_anim_set_time(): Sets the duration of the animation.
  - lv_anim_start(): Starts the animation.

LVGL core APIs

- Styles and Themes APIs
  - lv_style_init(): Initializes a style object.
  - lv_style_set_bg_color(): Sets the background color for objects.
  - lv_theme_apply(): Applies a theme across the UI objects.
- Input Devices APIs
  - lv_indev_drv_init(): Initializes the input device driver (touchscreen, keyboard, etc.).
  - lv_indev_drv_register(): Registers the input device driver with LVGL.
  - lv_indev_data_t: Structure to handle input device data (touch coordinates, state, etc.).

LVGL core APIs

- Drawing APIs
  - lv_draw_line(): Draws lines.
  - lv_draw_rect(): Draws rectangles.
  - lv_draw_arc(): Draws arcs.
- Image Handling APIs
  - lv_img_create(): Creates an image object.
  - lv_img_set_src(): Sets the source (image file) of the image object.
- Memory Management APIs
  - lv_mem_alloc(): Allocates dynamic memory.
  - lv_mem_free(): Frees dynamically allocated memory.

LVGL callback

- Callback functions
  - Callback functions are an essential part of handling events for GUI elements.
  - LVGL defines these functions using specific signatures and attaches them to objects to respond to events like button clicks, value changes, etc.
- Define a callback function in LVGL

```
void my_event_callback(lv_event_t * e) {
    // Event handling code
}
```

my_event_callback: The name of your custom callback function.
lv_event_t * e: The event object that contains information about the event, including the source object and the type of event.

LVGL callback

- Registering a Callback Function with an Object
  - Link a callback function with an object and its related event.

```
lv_obj_add_event_cb(obj, my_event_callback, LV_EVENT_TYPE, user_data);
```

obj: The LVGL object (e.g., a button or slider) that the callback function will be linked to.
my_event_callback: The function that will be called when the specified event occurs.
LV_EVENT_TYPE: The specific event type you want to handle (e.g., LV_EVENT_CLICKED). Use LV_EVENT_ALL if you want to handle all events.
user_data: A pointer that can be used to pass custom data to the callback function (often NULL if not needed).

Button Click Event

```c
#include "lvgl.h"

// Callback function for the button event
void button_event_cb(lv_event_t * e) {
    lv_event_code_t event_code = lv_event_get_code(e);  // Get the event code
    lv_obj_t * btn = lv_event_get_target(e);            // Get the object that triggered the event

    if (event_code == LV_EVENT_CLICKED) {
        printf("Button clicked!\n");
        lv_obj_set_style_bg_color(btn, lv_color_hex(0x00FF00), 0);  // Change button color to green on click
    }
}

void create_button_with_callback() {
    // Create a button on the current screen
    lv_obj_t * button = lv_btn_create(lv_scr_act());
    lv_obj_set_size(button, 100, 50);   // Set size of the button
    lv_obj_center(button);              // Center the button on the screen

    // Add the callback function for the button click event
    lv_obj_add_event_cb(button, button_event_cb, LV_EVENT_CLICKED, NULL);
}
```

Using User Data in Callback

```c
#include "lvgl.h"

// Structure to hold custom data
typedef struct {
    const char * name;
    int id;
} custom_data_t;

// Callback function that uses user data
void custom_event_cb(lv_event_t * e) {
    custom_data_t * data = (custom_data_t *)lv_event_get_user_data(e); // Retrieve the user data
    lv_event_code_t event_code = lv_event_get_code(e);

    if (event_code == LV_EVENT_CLICKED) {
        printf("Button '%s' with ID %d clicked!\n", data->name, data->id);
    }
}

void create_button_with_custom_data() {
    // Create a button
    lv_obj_t * button = lv_btn_create(lv_scr_act());
    lv_obj_set_size(button, 100, 50);
    lv_obj_center(button);

    // Create custom data
    custom_data_t data = {
        .name = "MyButton",
        .id = 42
    };

    // Register the callback with user data
    lv_obj_add_event_cb(button, custom_event_cb, LV_EVENT_CLICKED, &data);
}
```

Screen management in LVGL

- The application may have multiple screens showing different contents
  - Multiple screens transitioning in sequence.
  - Screen stack.
- The active screen is the screen currently displayed on the device.
  - Call *lv_scr_act()* to obtain a pointer to the currently active screen.

```
lv_obj_t * current_screen = lv_scr_act();
```

  - Use *lv_scr_load()* or *lv_scr_load_anim()* to switch the active screen.

Screen management in LVGL

- Screen objects
  - A screen in LVGL is an object (lv_obj) that serves as the base container for other objects (buttons, labels, sliders, etc.).
  - When LVGL is initialized, a default screen is created, and this screen is the active screen.
- Creating multiple screens.
  - Create multiple screens using *lv_obj_create()* without a parent, which makes them standalone screens. (Not displayed until setting as active.)

```
lv_obj_t * screen1 = lv_obj_create(NULL);  // Create a new screen
lv_obj_t * screen2 = lv_obj_create(NULL);  // Create another screen
```

Screen management in LVGL

- Setting the Active Screen
  - Set a screen as the active screen using *lv_scr_load()*.

    lv_scr_load(screen1); // Switch to `screen1`
- Screen Transition Animations
  - LVGL provides the lv_scr_load_anim() function to transition between screens with animations like fades, slides, or zoom effects.

    lv_scr_load_anim(screen2, LV_SCR_LOAD_ANIM_MOVE_LEFT, 500, 0, true);

Screen management in LVGL

- Parent-Child Relationship
  - While screens themselves are usually created as top-level objects (NULL parent), objects within those screens can have a parent-child hierarchy.
  - Buttons, labels, and other widgets can be added to a specific screen.

    lv_obj_t * button = lv_btn_create(screen1); // Add a button to `screen1`

Creating and switching between two screens

```
lv_obj_t * screen1;
lv_obj_t * screen2;

void setup() {
  // Create two screens
  screen1 = lv_obj_create(NULL);
  lv_obj_t * label1 = lv_label_create(screen1);
  lv_label_set_text(label1, "Screen 1");
  lv_obj_align(label1, LV_ALIGN_CENTER, 0, 0);

  screen2 = lv_obj_create(NULL);
  lv_obj_t * label2 = lv_label_create(screen2);
  lv_label_set_text(label2, "Screen 2");
  lv_obj_align(label2, LV_ALIGN_CENTER, 0, 0);

  // Set screen1 as the active screen initially
  lv_scr_load(screen1);
}

void loop() {
  lv_task_handler();  // Keep LVGL running
  delay(5);

  // Example of switching to screen2 after some condition (e.g., button press)
  if (/* some condition */) {
    lv_scr_load(screen2);  // Switch to screen2
  }
}
```

Graphical user interface design tool

- SquareLine Studio 
  - Built on top of LVGL and simplify the process of building complex graphical UIs for embedded devices. Suitable for fast UI development.
  - A WYSIWYG GUI designer. Provide a visual editor with dragging and dropping widgets, buttons, and other UI elements. Writing code for layout and design manually is not needed.
  - SquareLine generates C code for multiple platforms that is compatible with LVGL. – A graphical front-end for working with LVGL.
  - Support multi-screen applications. Link interactions and navigation between different screens and interface.

Design UI using SquareLine

- Install SquareLine Studio
  - Download SquareLine Studio: Visit the SquareLine Studio website and download the installer for your operating system.
  - Install the Software: Follow the on-screen instructions to install SquareLine Studio.
- Setup and Create a New Project
  - Launch SquareLine Studio and create a new project
    - Select the screen resolution that matches your target display.
    - Select the framework as LVGL and choose the version of LVGL you are targeting

Design UI using SquareLine

- Designing the GUI
  - Adding Screens
    - Click on the Screens tab and add a new screen
  - Adding Widgets
    - Ddrag and drop widgets onto the screen from the the widget library. Button, Label, Slider, Images.
  - Configuring Widgets
    - Click on a widget to configure its properties.
    - Adjust size, position, color, and other attributes in the property editor panel.

Design UI using SquareLine

- Adding Events
  - Define actions that happen when a user interacts with a widget (e.g., clicking a button).
  - Select a widget and go to the Events tab to add events like onClick or onValueChange.
  - These events can trigger actions, like changing screens or sending commands to the ESP32.

- Exporting the Design
  - After designing your GUI, export the code to integrate it with ESP32 project
    - Click on File > Export. Choose the LVGL format to generate C files containing the GUI code.
    - This will generate files such as **ui.c** and **ui.h**, which contain the code needed to initialize and display the GUI on your hardware.

Design UI using SquareLine

- Integrating with ESP32 in the Arduino IDE
  - Include LVGL Library: Ensure the LVGL library is installed with Arduino IDE. Install it from the Library Manager by searching for "LVGL".
  - Include Generated Files: Copy the exported ui.c and ui.h files into current Arduino project directory.
  - Setup Display Driver: Configure display driver according to specific hardware. For example, if using an SPI-based display, include the correct driver files and configure the pins accordingly.
  - Initialize LVGL: In setup() function, initialize LVGL and your display.

Design UI using SquareLine

- Adding More Functionality
  - Expand application by adding more events, interacting with sensors, sending data over Wi-Fi, etc.
  - Use the LVGL API functions in conjunction with custom logic to create dynamic, interactive interfaces.

Integrate the SquareLine exported files in Arduino code

```
#include <lvgl.h>
#include <TFT_eSPI.h>  // Use appropriate library for your display
#include "ui.h"        // Include the generated UI file from SquareLine

TFT_eSPI tft = TFT_eSPI();  // Initialize TFT display
lv_disp_draw_buf_t draw_buf;
lv_color_t buf[LV_HOR_RES_MAX * 10];

// Touchscreen initialization (for example, XPT2046 driver)
void touch_read(lv_indev_drv_t *indev_driver, lv_indev_data_t *data) {
    // Add your touch reading code here
    // Set data->point.x and data->point.y to the detected touch coordinates
    // Set data->state = LV_INDEV_STATE_PR if touched, LV_INDEV_STATE_REL otherwise
}

void setup() {
    Serial.begin(115200);

    // Initialize TFT display
    tft.begin();
    tft.setRotation(1);  // Set orientation

    // Initialize LVGL
    lv_init();
    lv_disp_draw_buf_init(&draw_buf, buf, NULL, LV_HOR_RES_MAX * 10);

    // Initialize display buffer
    static lv_disp_drv_t disp_drv;
    lv_disp_drv_init(&disp_drv);          // Initialize display driver
    disp_drv.hor_res = 320;               // Set the horizontal resolution (e.g., 320px)
    disp_drv.ver_res = 240;               // Set the vertical resolution (e.g., 240px)
    disp_drv.flush_cb = my_disp_flush;    // Callback to flush the buffer to the screen
    disp_drv.draw_buf = &draw_buf;
    lv_disp_drv_register(&disp_drv);      // Register the display driver
```

Integrate the SquareLine exported files in Arduino code

```
    // Initialize touch input
    static lv_indev_drv_t indev_drv;
    lv_indev_drv_init(&indev_drv);        // Initialize input driver
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touch_read;       // Touchscreen reading function
    lv_indev_drv_register(&indev_drv);    // Register the input driver

    // Initialize your GUI from SquareLine
    ui_init();  // This function is generated by SquareLine Studio
}

void loop() {
    lv_timer_handler();  // Let LVGL process its tasks
    delay(5);
}

// Flush callback to transfer the content of the buffer to the display
void my_disp_flush(lv_disp_drv_t *disp, const lv_area_t *area, lv_color_t *color_p) {
    // Implement the flushing to your specific display
    tft.startWrite();
    tft.setAddrWindow(area->x1, area->y1, area->x2 - area->x1 + 1, area->y2 - area->y1 + 1);
    tft.pushColors((uint16_t *)&color_p->full, (area->x2 - area->x1 + 1) * (area->y2 - area->y1 + 1),
true);
    tft.endWrite();
    lv_disp_flush_ready(disp);  // Tell LVGL that flushing is done
}
```

Callbacks in SquareLine

- Overview of callbacks in SquareLine
  - Design the GUI visually, adding elements like buttons, sliders, labels, etc.
  - Assign event callbacks directly to these elements through the interface.
  - The generated code includes placeholders for these callback functions, which can be modified to add custom logic.
- Define callbacks in SquareLine
  - When an event is created in SquareLine, it generates a function with a specific signature that matches the LVGL event system.

Callback function template generated by SquareLine

```
void ui_event_my_button(lv_event_t * e) {
    lv_event_code_t event_code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);

    if (event_code == LV_EVENT_CLICKED) {
        // Your custom logic here
    }
}
```

ui_event_my_button is the function generated for handling events from a button named my_button.

lv_event_t * e provides event data, like in standard LVGL callbacks.

lv_event_code_t event_code is used to identify the type of event, such as LV_EVENT_CLICKED.

Callbacks in SquareLine

- Set up callbacks in SquareLine
  - Design the UI: Drag and drop a widget onto the canvas.
  - Select the Widget: Click on the widget to open its properties.
  - Add an Event: In the properties panel, look for the "Events" section and add a new event.
  - Specify the Function Name: Assign a name to the event function (e.g., ui_event_my_button).
  - Generate Code: Click on "Generate" or "Export" to produce the LVGL-compatible C code, including the callback function template.
  - SquareLine generates the callback function and links it with the UI element.

Example of the code generated for a button click event

```
// This part is auto-generated by Squareline Studio
void ui_event_my_button(lv_event_t * e) {
    lv_event_code_t event_code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);

    if (event_code == LV_EVENT_CLICKED) {
        // Custom code: Execute when the button is clicked
        printf("Button clicked!\n");
        lv_label_set_text(ui_label, "Button was clicked!");
    }
}
```

ui_event_my_button is the callback function generated for a button click.

lv_event_code_t event_code is used to determine the type of event (here, LV_EVENT_CLICKED).

Inside the *if* block, custom actions are defined, such as printing a message or changing a label's text.

Callbacks in SquareLine

- Link generated code with application
  - Include the Generated Files: Include the ui.h file in main project file.
  - Initialize the UI: Call the initialization function generated by SquareLine Studio, such as ui_init(), to set up the UI and link the callbacks.

Example code integrate with SquareLine code

```
#include "ui.h"

void setup() {
    lv_init();
    ui_init(); // Initialize the UI elements from Squareline Studio
}

void loop() {
    lv_timer_handler(); // Call this periodically to handle LVGL tasks
    delay(5);           // Adjust as needed to avoid overloading the CPU
}
```

Screen management in SquareLine

- Visual Screen Management
  - Each screen is treated as a separate object, similar to how LVGL treats them in the code.
  - The screens are displayed in a project view, and one can name them and define their content (e.g., buttons, labels, sliders) in a visual manner.
- Screen Navigation Logic
  - The sequence and navigation between screens are controlled by events that one can define in SquareLine.
  - Navigation action are set up through the Squareline Studio interface by adding events to UI elements.

Screen management in SquareLine

- Generated LVGL code
  - SquareLine generates the underlying LVGL code that corresponds to the screen sequence and transitions you have defined visually.
  - It handles the creation of screens, widgets, and the functions that perform the navigation, making it easier for developers.
- Using lv_scr_load() or lv_scr_load_anim()
  - Behind the scenes, SquareLine generates code that uses LVGL functions like lv_scr_load() or lv_scr_load_anim() to make the switch.
  - If it is set up as animations in SquareLine (like sliding transitions), it uses lv_scr_load_anim() with the appropriate parameters for that effect.

Screen Navigation in SquareLine

- Drag a button onto Screen1 and define an event like On Click -> Go to Screen2.
- SquareLine generates code similar to below.

```
void btn_event_handler(lv_event_t * e) {
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * btn = lv_event_get_target(e);

    if(code == LV_EVENT_CLICKED) {
        lv_scr_load(screen2);  // Load screen2 when the button is clicked
    }
}
```

- The generated event handler btn_event_handler() is automatically connected to the button, and lv_scr_load() is used to switch to screen2 when the button is clicked.