Placeholder for Future Automation: Arduino Sketch Documentation

Introduction:

The Arduino GPIO Control Device is a C-based framework developed by Krishna Ganta, a student of Northshore School District (Student ID: 2032673). It is a script to run on the Arduino family of microcontrollers, that allows manual control of the GPIO pins via a USB Serial connection. Note that to function, there must be a wired Serial interface between the controller and the Arduino. If such an interface is not feasible, or the Arduino is not available, consider using the Raspberry Pi GPIO control device, which can function using a wireless interface.

Purpose:

The purpose of this Arduino sketch is to create a plug-and-play device that acts as a placeholder for future automation. This device allows manual control of specified GPIO (General Purpose Input/Output) pins on the Arduino board. It serves as a convenient interface for manually controlling devices or circuits until they are fully automated in the future.

Prerequisites:

Before using this sketch, ensure you have the following:

- An Arduino board (e.g., Arduino Uno or similar).
- LEDs, relays, or other devices connected to the GPIO pins for manual control.
- An understanding of the GPIO pins you want to control and their corresponding numbers.
- The Arduino IDE or compatible software for uploading the sketch to your Arduino board.

Code Overview:

The sketch provides a simple yet flexible framework for controlling GPIO pins using serial communication. Having the advantage of using USB Serial makes it useful in region where no Wireless Access Point can be set up, or if the corresponding wireless hardware is not available. It listens for commands via the serial interface and responds accordingly to turn pins on or off.

GPIO Pin Configuration:

- The `gpio_pins` array at the beginning of the sketch should be modified to include the actual GPIO pins you want to control. Replace the placeholder values (2, 3, 4) with the pins you are using.

Setup Function:

- In the `setup()` function:
    - The sketch initializes the GPIO pins specified in the `gpio_pins` array as outputs.
    - It sets the initial state of these pins to LOW (OFF).
    - Serial communication is initialized at a baud rate of 9600, making it compatible with common serial terminals.

Loop Function:

- The `loop()` function continuously monitors the serial port for incoming messages.
- When a message is received, it is processed to determine whether to turn a pin on or off.
- Messages should have the format: "`ON <PIN_NUMBER>`" or "`OFF <PIN_NUMBER>`", where `<PIN_NUMBER>` is the GPIO pin number to control.

Message Processing:

- The sketch checks whether the incoming message starts with "ON " or "OFF ".
- If the message starts with "ON ", it extracts the pin number and validates it.
- If the pin is valid, it sets the corresponding GPIO pin to HIGH (ON) and sends "OK" back via serial communication.
- If the message starts with "OFF ", it extracts the pin number and validates it.
- If the pin is valid, it sets the corresponding GPIO pin to LOW (OFF) and sends "OK" back via serial communication.
- If the message format is not recognized or the pin number is invalid, it sends "Invalid Command" as a response.

Validating Pins:

- The `isValidPin()` function checks whether a provided pin number is valid by comparing it with the pins listed in the `gpio_pins` array.

Usage:

1. Connect the Arduino to the GPIO pins and devices you want to control manually.
2. Upload this sketch to your Arduino using the Arduino IDE.
3. Open the Serial Monitor or a compatible serial terminal.
4. Send commands in the format "`ON <PIN_NUMBER>`" or "`OFF <PIN_NUMBER>`" to control the GPIO pins.

5. The sketch will respond with "OK" for successful commands or "Invalid Command" for invalid ones.

Extending for Future Automation:

- Once you are ready to automate the control of your devices or circuits, you can replace the manual input with automated triggers, such as sensors or timers.
- Modify the sketch to include additional logic and conditions for automated control based on your specific requirements.

Notes:

- Ensure that the Arduino board is powered appropriately and safely to avoid damage.
- Double-check your connections and GPIO pin numbers to ensure correct functionality.
- Customize the sketch further to suit your specific needs and integration with future automation systems.

```
C Arduino Debugger.ino
// Define the GPIO pins you want to control
const int gpio_pins[] = {2, 3, 4}; // Replace with the actual GPIO pins you want to use

void setup() {
  // Initialize GPIO pins as outputs
  for (int i = 0; i < sizeof(gpio_pins) / sizeof(gpio_pins[0]); i++) {
    pinMode(gpio_pins[i], OUTPUT);
    digitalWrite(gpio_pins[i], LOW);
  }

  // Initialize Serial communication
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    String message = Serial.readStringUntil('\n');
    message.trim();

    if (message.startsWith("ON ")) {
      int pin_number = message.substring(3).toInt();
      if (isValidPin(pin_number)) {
        digitalWrite(pin_number, HIGH);
        Serial.println("OK");
      } else {
        Serial.println("Invalid Command");
      }
    } else if (message.startsWith("OFF ")) {
      int pin_number = message.substring(4).toInt();
      if (isValidPin(pin_number)) {
        digitalWrite(pin_number, LOW);
        Serial.println("OK");
      } else {
        Serial.println("Invalid Command");
      }
    } else {
      Serial.println("Invalid Command");
    }
  }
}

bool isValidPin(int pin) {
  for (int i = 0; i < sizeof(gpio_pins) / sizeof(gpio_pins[0]); i++) {
    if (gpio_pins[i] == pin) {
      return true;
    }
  }
  return false;
}
```