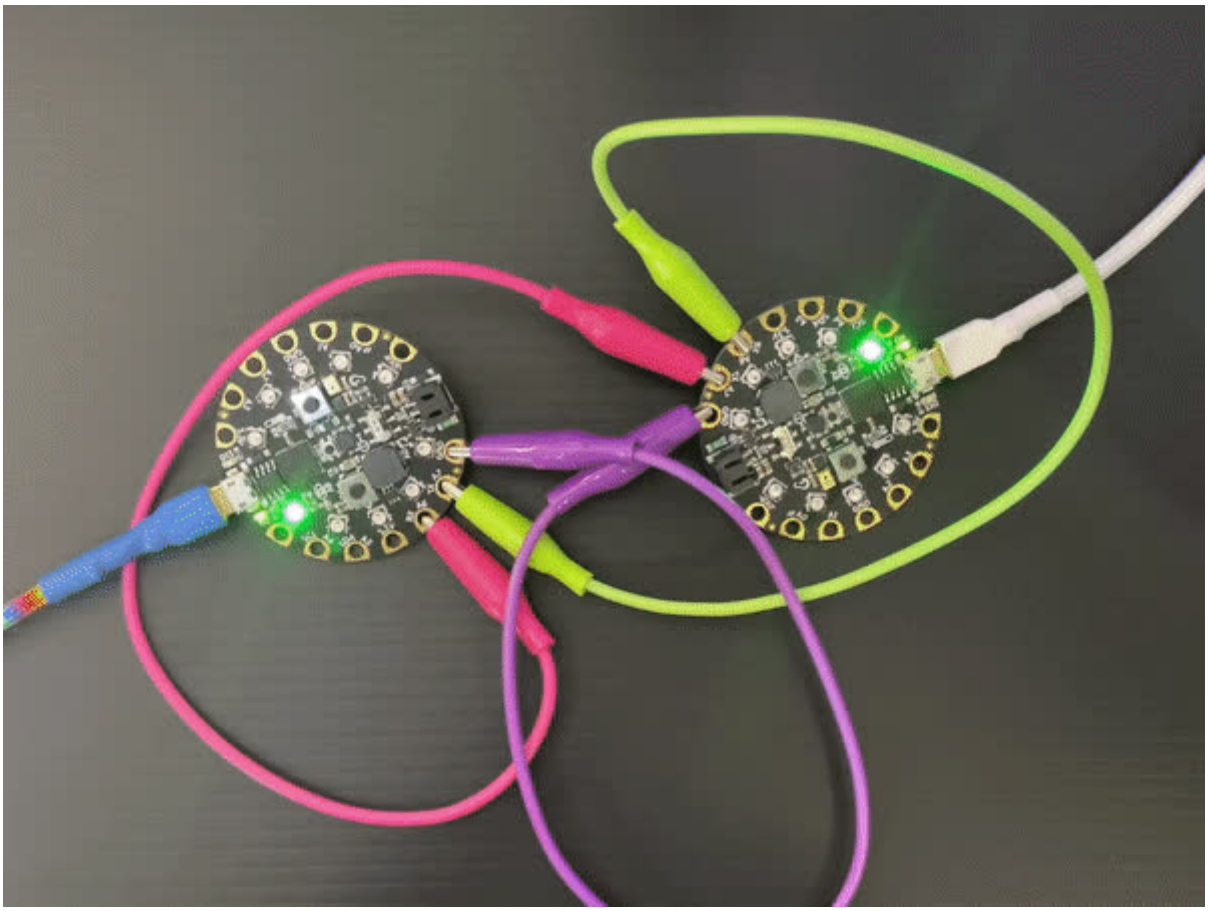
UART Communication Between Two CircuitPython Boards

Created by Eva Herrada

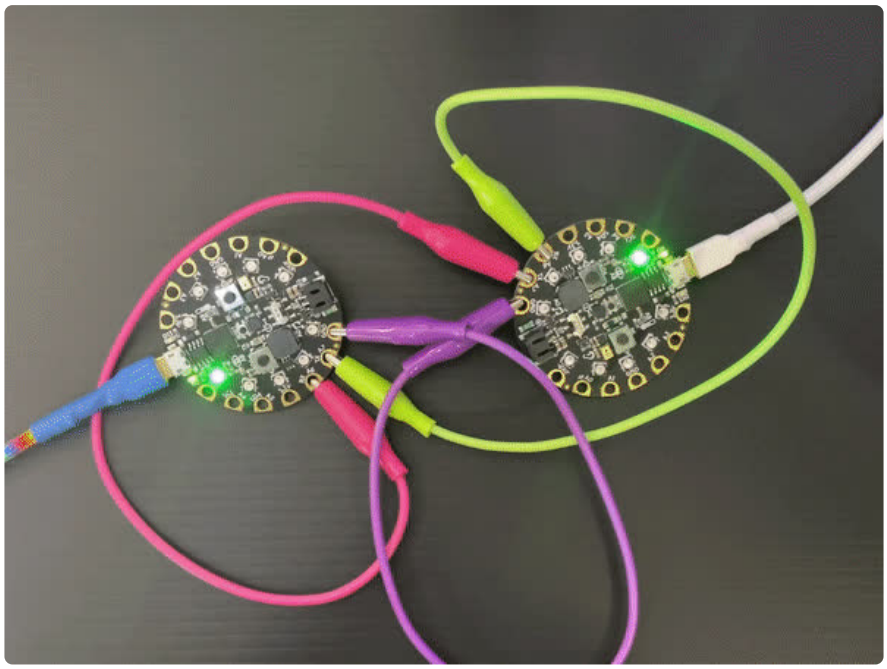


https://learn.adafruit.com/uart-communication-between-two-circuitpython-boards

Last updated on 2024-06-03 03:27:38 PM EDT

Table of Contents

Overview



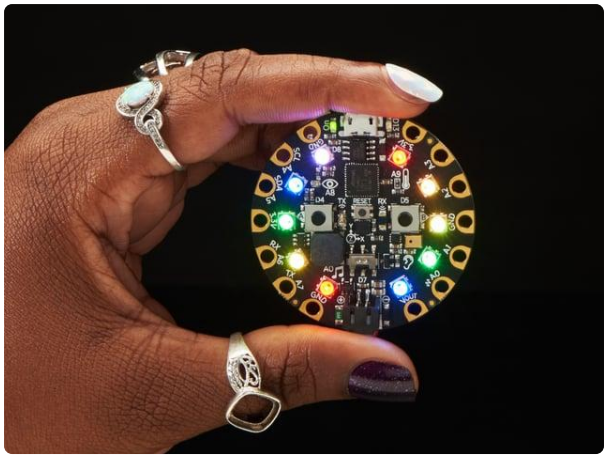
In this project, you'll learn how to communicate between two different CircuitPython boards using UART.

UART, universal asynchronous receiver-transmitter, is a serial communication protocol that works asynchronously. Unlike I2C, it isn't structured and does not require pull-ups. UART is also, like the name implies, universal in that just about every microcontroller and microcomputer has a UART port. This can make it really easy to set up as cross-platform.

This guide will go through a really simple example where the boards send packets back and forth between each other that tell the other board to make the built-in NeoPixel brighter. The example in this guide is very simple, but it should give you a really good start for using UART to communicate between boards in your project.

Parts

The examples apply to nearly the full range of CircuitPython boards that have UART available. For this guide, two Circuit Playground Express boards were used.



Circuit Playground Express

Circuit Playground Express is the next step towards a perfect introduction to electronics and programming. We've taken the original Circuit Playground Classic and...

https://www.adafruit.com/product/3333

Colorful Clips - Tropical Alligator Test Clip Leads - 10 Pieces

Designed by Making & Hacking extraordinaire Lee Cyborg, these colorful clips are a nice switch-up from the standard colors. Connect this to that...

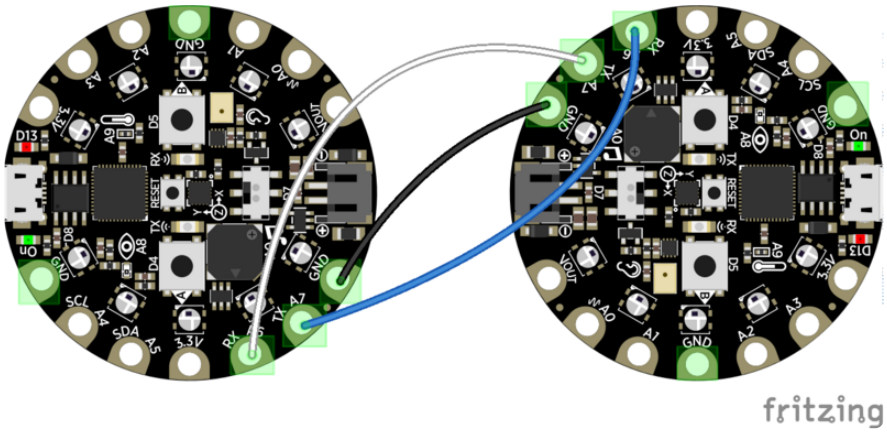
https://www.adafruit.com/product/5152

USB cable - USB A to Micro-B

This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...

https://www.adafruit.com/product/592

Wiring

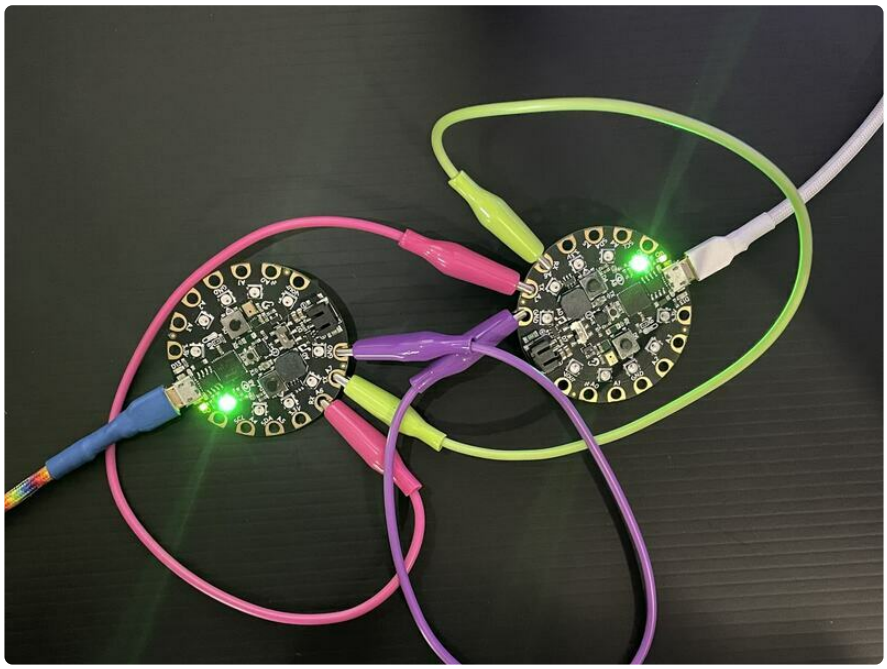


Wiring for this project is really simple. Just connect the **RX** of one board to the **TX** of the other, and the **TX** of one board to the **RX** of the other. Then connect the **GND** pin from one board to the **GND** pin on the other.

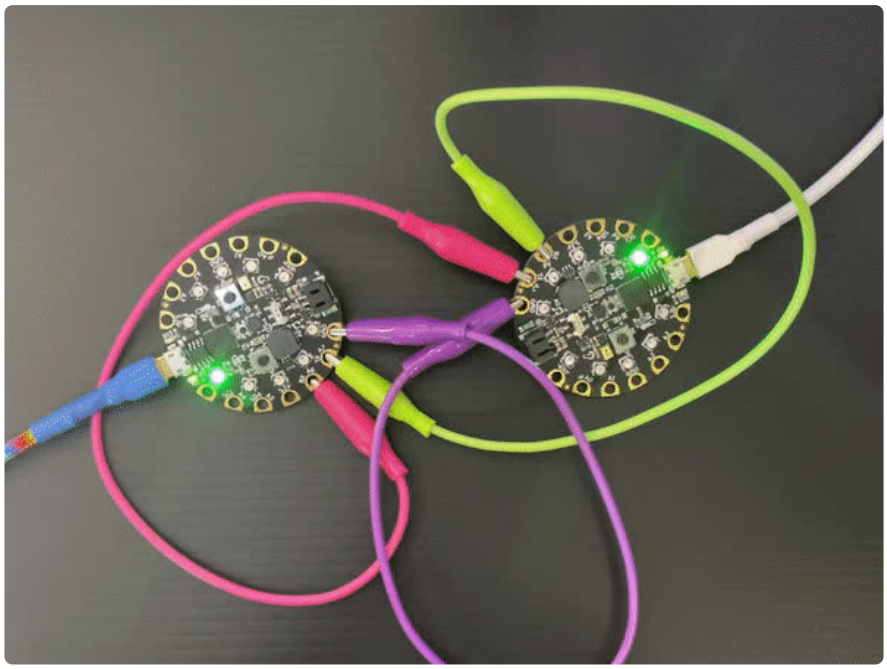
Make sure that it's wired **RX-TX** and **TX-RX** and not **RX-RX** and **TX-TX**.

Mixing up RX and TX is pretty common, since they are 'relative' designators. If your setup isn't working, try swapping them!

When you've wired them up, they should look something like this:



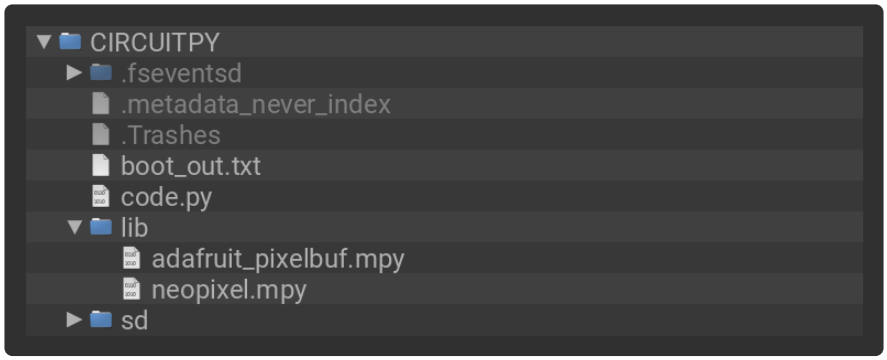
Code



Installing the Project Code

Download a zip of the project by clicking **Download Project Bundle** below.

After unzipping the file, the contents to both of the **CIRCUITPY** drives (the second one appears as **CIRCUITPY1** on some operating systems) which appear when the Circuit Playgrounds are connected to your computer with USB cables. After you've copied everything over, it should look something like this:




```
# SPDX-FileCopyrightText: Copyright (c) 2021 Eva Herrada for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Dan Halbert for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
Code for communicating between two CircuitPlayground Express boards using UART.
Sends value from the onboard light sensor to the other board and the other board
sets its
NeoPixels accordingly.
```

```
"""

import time
import board
import busio
import digitalio
import neopixel
import analogio

pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=0.1, auto_write=False)

light_sensor = analogio.AnalogIn(board.LIGHT)

btn_A = digitalio.DigitalInOut(board.BUTTON_A)
btn_A.switch_to_input(pull=digitalio.Pull.DOWN)

btn_B = digitalio.DigitalInOut(board.BUTTON_B)
btn_B.switch_to_input(pull=digitalio.Pull.DOWN)

# Use a timeout of zero so we don't delay while waiting for a message.
uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=0)

# Messages are of the form:
# "<TYPE,value,value,value,...>"
# We send and receive two types of messages:
#
# Message contains a light sensor value (float):
# <L,light>
#
# Message contains statuses of two buttons. Increment NeoPixel brightness by 0.1 if
the second
# button is pressed, and reduce brightness by 0.1 if the first button is pressed.
# <B,btn_A,btn_B>

UPDATE_INTERVAL = 3.0
last_time_sent = 0

# Wait for the beginning of a message.
message_started = False

while True:
    # Send light sensor value only every UPDATE_INTERVAL seconds.
    now = time.monotonic()
    if now - last_time_sent >= UPDATE_INTERVAL:
        light = light_sensor.value
        uart.write(bytes(f"<L,{light}>", "ascii"))
        print("sending light value", light)
        last_time_sent = now

    if any((btn_A.value, btn_B.value)):
        # Send values of built-in buttons if any are pressed
        uart.write(bytes(f"<B,{btn_A.value},{btn_B.value}>", "ascii"))
        print(f"Sent ({btn_A.value}, {btn_B.value})")

        # Don't do anything else until both buttons are released
        time.sleep(0.1)
        while any((btn_A.value, btn_B.value)):
            pass

    byte_read = uart.read(1)  # Read one byte over UART lines
    if not byte_read:
        # Nothing read.
        continue

    if byte_read == b"<":
        # Start of message. Start accumulating bytes, but don't record the "<".
        message = []
        message_started = True
        continue
```

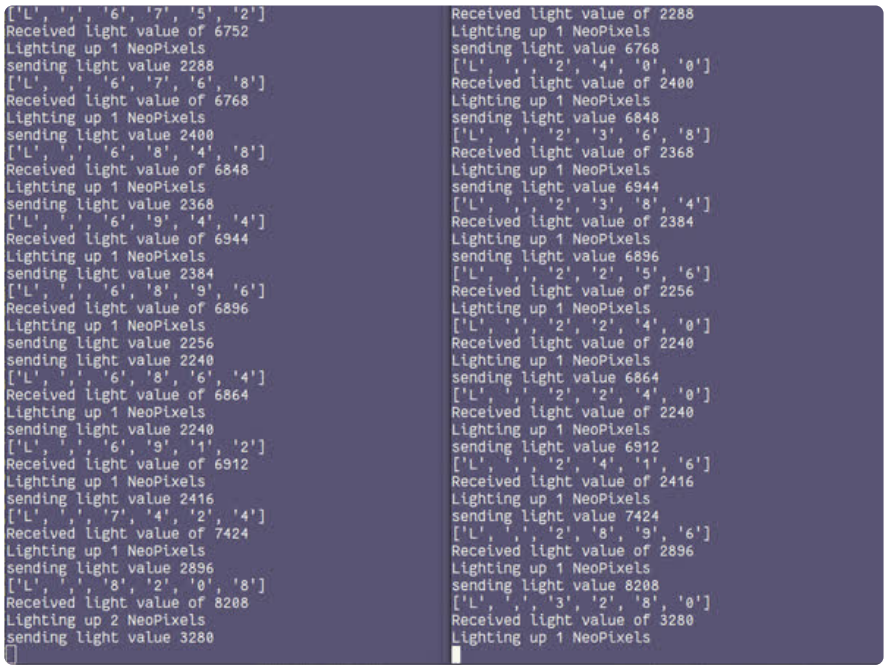
```
    if message_started:
        if byte_read == b">":
            # End of message. Don't record the ">".
            # Now we have a complete message. Convert it to a string, and split it
up.
            print(message)
            message_parts = "".join(message).split(",")
            message_type = message_parts[0]
            message_started = False

            if message_parts[0] == "L":
                # Received a message telling us a light sensor value
                peak = int(((int(message_parts[1]) - 2000) / 62000) * 10)
                for i in range(0, 10):
                    if i <= peak:
                        pixels[i] = (0, 255, 0)
                    else:
                        pixels[i] = (0, 0, 0)
                pixels.show()
                print(f"Received light value of {message_parts[1]}")
                print(f"Lighting up {peak + 1} NeoPixels")

            elif message_parts[0] == "B":
                # Received a message asking us to change our brightness.
                if message_parts[1] == "True":
                    pixels.brightness = max(0.0, pixels.brightness - 0.1)
                    print(f"Brightness set to: {pixels.brightness}")
                if message_parts[2] == "True":
                    pixels.brightness = min(1.0, pixels.brightness + 0.1)
                    print(f"Brightness set to: {pixels.brightness}")

        else:
            # Accumulate message byte.
            message.append(chr(byte_read[0]))
```

If you open up the REPL for both boards in two separate windows, you can see messages being passed back and forth like this:



Code Run-Through

First, the code imports the required libraries.

```
import time
import board
import busio
import digitalio
import neopixel
import analogio
```

Next, it sets up the NeoPixel and UART, as well as the buttons for controlling the brightness.

```
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=0.1, auto_write=False)

light_sensor = analogio.AnalogIn(board.LIGHT)

btn_A = digitalio.DigitalInOut(board.BUTTON_A)
btn_A.switch_to_input(pull=digitalio.Pull.DOWN)

btn_B = digitalio.DigitalInOut(board.BUTTON_B)
btn_B.switch_to_input(pull=digitalio.Pull.DOWN)

# Use a timeout of zero so we don't delay while waiting for a message.
uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=0)
```

Before the code starts the main loop, it needs to define a few variables it will be using. The first one is a variable for how often to send a new message over the UART bus. The second variable will store the last time that a message was sent, and the final one is used to

```
UPDATE_INTERVAL = 3.0
last_time_sent = 0

# Wait for the beginning of a message.
message_started = False
```

Now, the code enters the main loop. It first checks to see how long since the last packet was sent, and if it has been long enough, it writes the packet containing the value from the light sensor to the other board over UART.

```
while True:
    # Send light sensor value only every UPDATE_INTERVAL seconds.
    now = time.monotonic()
    if now - last_time_sent &gt;= UPDATE_INTERVAL:
        light = light_sensor.value
        uart.write(bytes(f"&lt;L,{light}&gt;", "ascii"))
        print("sending light value", light)
        last_time_sent = now
```

If any of the built-in buttons are pressed, the code will send a different packet to the other device telling it the status of both of the buttons. This packet will be used to set the brightness of the NeoPixels.

```
     if any((btn_A.value, btn_B.value)):
        # Send values of built-in buttons if any are pressed
        uart.write(bytes(f"&lt;B,{btn_A.value},{btn_B.value}&gt;", "ascii"))
        print(f"Sent ({btn_A.value}, {btn_B.value})")

        # Don't do anything else until both buttons are released
        time.sleep(0.1)
        while any((btn_A.value, btn_B.value)):
            pass
```

The code now reaches the part where it tries to read a packet over the UART bus. It first tries to read one byte. If it doesn't read one, it will run the main loop over again.

Assuming it has received a byte, the code then goes on to check if the byte received is a "<", which is the defined starting byte. If it has received this byte, it then clears the `message` variable, and sets the variable to indicate that it is in the process of reading a packet.

Now that the packet is being read, it checks to see if it has reached the final byte. Up until then, it will go through and keep adding to the `message` variable. When it has reached the end, it prints the message, and sets a few variables to make reading the message a bit easier.

```
    byte_read = uart.read(1)  # Read one byte over UART lines
    if not byte_read:
        # Nothing read.
        continue

    if byte_read == b"&lt;":
        # Start of message. Start accumulating bytes, but don't record the "&lt;".
        message = []
        message_started = True
        continue

    if message_started:
        if byte_read == b"&gt;":
            # End of message. Don't record the "&gt;".
            # Now we have a complete message. Convert it to a string, and split it
up.
            print(message)
            message_parts = "".join(message).split(",")
            message_type = message_parts[0]
            message_started = False
```

Now that the packet is easily readable, the code checks to see what type of packet it is. If it's a light sensor value, it converts the light sensor value to an integer out of 10. It then turns on the NeoPixels based on that value.

If the packet contains instructions for brightness, the code then raises or lowers the brightness of the NeoPixels accordingly.

At the end of the file, and moving out one level, the code adds the byte it has just read to the `message` variable.

```
            if message_parts[0] == "L":
                # Received a message telling us a light sensor value
                peak = int(((int(message_parts[1]) - 2000) / 62000) * 10)
                for i in range(0, 10):
                    if i &lt;= peak:
                        pixels[i] = (0, 255, 0)
                    else:
                        pixels[i] = (0, 0, 0)
                pixels.show()
                print(f"Received light value of {message_parts[1]}")
                print(f"Lighting up {peak + 1} NeoPixels")

            elif message_parts[0] == "B":
                # Received a message asking us to change our brightness.
                if message_parts[1] == "True":
                    pixels.brightness = max(0.0, pixels.brightness - 0.1)
                    print(f"Brightness set to: {pixels.brightness}")
                if message_parts[2] == "True":
                    pixels.brightness = min(1.0, pixels.brightness + 0.1)
                    print(f"Brightness set to: {pixels.brightness}")

        else:
            # Accumulate message byte.
            message.append(chr(byte_read[0]))
```