

Forum:[Hack Forums](#)

Thread: [Arduino to C++ Serial Communication](#)

Kaszz - April 7, 2017, 8:14 am

Im currently working on serial communicatain from my arduino to a C++ program that is running on my PC.

Currently i just want to send a binary value to the PC when i press a button on my board, however ive found some inconsistencies when i go for anything larger than 2 digits.

My arduino code - which is basically just the original debounce example with an added serial.print():

```
Code int inPin = 2;      // the number of the input pin
```

```
int outPin = 13;        // the number of the output pin
```

```
int state = HIGH;      // the current state of the output pin
```

```
int reading;           // the current reading from the input pin
```

```
int previous = LOW;    // the previous reading from the input pin
```

```
long time = 0;         // the last time the output pin was toggled
```

```
long debounce = 200;  // the debounce time, increase if the output flickers
```

```
void setup()
```

```
{
```

```
  pinMode(inPin, INPUT);
```

```
  pinMode(outPin, OUTPUT);
```

```
  Serial.begin(9600);
```

```
}
```

```
void loop()
```

```
{
```

```
  reading = digitalRead(inPin);
```

```
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
```

```
    if (state == HIGH){
```

```
      state = LOW;
```

```
    }
```

```
    else {
```

```
      state = HIGH;
```

```
    }
```

```
    Serial.print("1234"); // For showcasing of the example ive not only used 0 and 1.
```

```
                        // This way it is easier to actually see what is going on.
```

```
    time = millis();
```

```
  }
```

```
  digitalWrite(outPin, state);
```

```
  previous = reading;
```

```
}
```

This example just sends out the value 1234 whenever i press my button (i know this isnt binary, but it is easier to see where it goes from when we have more different symbols).

Now my C++ code is based off of this serial communications class:

<http://playground.arduino.cc/Interfacing/CPPWindows>

And my main loop looks like this:

```
Code int main() {
    Serial* SP = new Serial("\\\\.\\COM3");
    if (SP->IsConnected())
    {
        cout << "We're connected" << endl;
        cout << "_____ " << endl;
    }

    char incomingData[256] = "";
    int readResult = 0;

    while (SP->IsConnected())
    {
        readResult = SP->ReadData(incomingData);
        incomingData[readResult] = 0;

        if (strlen(incomingData) > 0)
        {
            string str(incomingData);
            cout << endl << "String: " << str << endl;

            for (int i = 0; i < strlen(incomingData); i++)
            {
                cout << "Char on place " << i << ": " << incomingData[i] << endl;
            }
            cout << "_____ " << endl;
        }
    }
}
```

If we now take a look at the desired console output with an example where i only send the number 1 from the arduino when i press a key it will look something like this:

As we see it registers it completely fine, however, if i use the example from the actual arduino code posted the console looks something more like this:

So as you see it somehow doesnt always put the entire char array into the string before it prints and moves on. I dont know if this is because the char array isnt filled up yet, and i dont know how to account for this.

I hope someone have an idea as to how i can receive the entire string of data every time.

Dostoevsky - April 7, 2017, 12:38 pm

I'm not 100% sure about Arduino programming but, in my opinion, it kinda looks like an issue with the actual transfer of the serial data rather than a real logic issue in your code (e.g. it isn't actually receiving the data correctly, have you tried spamming the button? Does it give the same result? Looks awfully like some read synchronisation fault). I'm kinda drunk right now but there's a few things you should address in your code anyway:

1) `std::endl` isn't just a new line. It will actually flush the stream. This is not always desired/required so you should replace the instances of `std::endl` with `'\n'`.

2) The comparison within your loop relies on the result of a call. This means it's gonna have to call it every iteration. This is not what you want since the string length should be the same. You should get the length before the loop and store it in a local variable and compare against that within your for loop.

3) You never delete your Serial. I know people will tell you that "it doesn't matter because the program ends there so it's superfluous" but, for good code, you shouldn't really ever rely on the fact that the termination will clean your stuff up. Just a small, petty, note.

As for a general test case, I recommend just simplifying your code so that it doesn't print the data it receives but only prints the length of the data. Then, spam the button and check if it gives varying lengths/ If that's the case, there's probably some issue with the actual sending/receiving of the buffer. Look into the other serial methods available to you on the Arduino:

<https://www.arduino.cc/en/Serial/Flush> etc. The code looks like it should generally be blocking until it receives something and should wait until it receives all of it. As I said, I've never worked with the Arduino so I dunno.

Kaszz - April 7, 2017, 3:25 pm

(Apr 7, 2017 12:38 PM)Dostoevsky Wrote: [\[CITE\]](#) I'm not 100% sure about Arduino programming but, in my opinion, it kinda looks like an issue with the actual transfer of the serial data rather than a real logic issue in your code (e.g. it isn't actually receiving the data correctly, have you tried spamming the button? Does it give the same result? Looks awfully like some read synchronisation fault). I'm kinda drunk right now but there's a few things you should address in your code anyway:

1) `std::endl` isn't just a new line. It will actually flush the stream. This is not always desired/required so you should replace the instances of `std::endl` with `'\n'`.

2) The comparison within your loop relies on the result of a call. This means it's gonna have to call it every iteration. This is not what you want since the string length should be the same. You should get the length before the loop and store it in a local variable and compare against that within your for loop.

3) You never delete your Serial. I know people will tell you that "it doesn't matter because the program ends there so it's superfluous" but, for good code, you shouldn't really ever rely on the fact that the termination will clean your stuff up. Just a small, petty, note.

As for a general test case, I recommend just simplifying your code so that it doesn't print the data it receives but only prints the length of the data. Then, spam the button and check if it gives varying lengths/ If that's the case, there's probably some issue with the actual sending/receiving of the buffer. Look into the other serial methods available to you on the Arduino:

<https://www.arduino.cc/en/Serial/Flush> etc. The code looks like it should generally be blocking until it receives something and should wait until it receives all of it. As I said, I've never worked with the Arduino so I dunno.

I was recommended this:

<http://forum.arduino.cc/index.php?topic=396450.0>

And it helped me out. I relied on the serial transfer to be way faster than it actually was.

As for your point about `endl` and `/n`, can you possibly elaborate in a more general matter? Seems like something that is good to know.

And about deleting my object i am aware that it is recommended, however fiddling around with the code for a good while apparantly i had deleted it again :>

Thanks for your answer anyways.

Dostoevsky - April 7, 2017, 3:46 pm

(Apr 7, 2017 03:25 PM)Overpass Wrote: [\[CITE\]](#) I was recommended this:
<http://forum.arduino.cc/index.php?topic=396450.0>

And it helped me out. I relied on the serial transfer to be way faster than it actually was.

As for your point about endl and /n, can you possibly elaborate in a more general matter? Seems like something that is good to know.

And about deleting my object i am aware that it is recommended, however fiddling around with the code for a good while apparantly i had deleted it again :>

Thanks for your answer anyways.

std::endl is basically the equivalent of '\n' followed by a std::flush.

This behaviour is not always desirable or required. Like, if you're looping std::cout, may as well just use '\n' (I'd bring up performance reasons but it's kinda pointless).

<http://stackoverflow.com/questions/213907/c-stdendl-vs-n>