`Serial.print` in a Nutshell

Gerald Recktenwald
`gerry@pdx.edu`

October 27, 2020

Introduction

The Serial Monitor is part of the Arduino IDE that can receive text messages from a running Arduino sketch. The communication between the Arduino and the Serial Monitor uses the USB cable. Therefore, the Serial Monitor only works when the Arduino board is tethered to the host computer that is running the Arduino IDE.

Figure 1 is an annotated screenshot of a Serial Monitor window running on a Macintosh computer. The Serial Monitor on a Windows computer will have a slight different appearance, but will have the same user interface controls.
See `https://www.arduino.cc/reference/en/language/functions/communication/serial/print/`.

Printing to the Serial Monitor

Starting the Serial Monitor

You must first call `Serial.begin`, usually in the `setup()` function of your Arduino sketch.

1. `Serial.begin(9600)` and `Serial.begin(115200)` are common. Those commands set the baud rate to 9600 and 115200, respectively. The baud rate is the number of bits per second[1].

2. The valid baud rates for the communication between the Arduino and your PC are shown in Figure 1. A 9600 baud rate is usually sufficient. Recent example sketches from vendors like

[1]See, e.g., `https://en.wikipedia.org/wiki/Baud`.

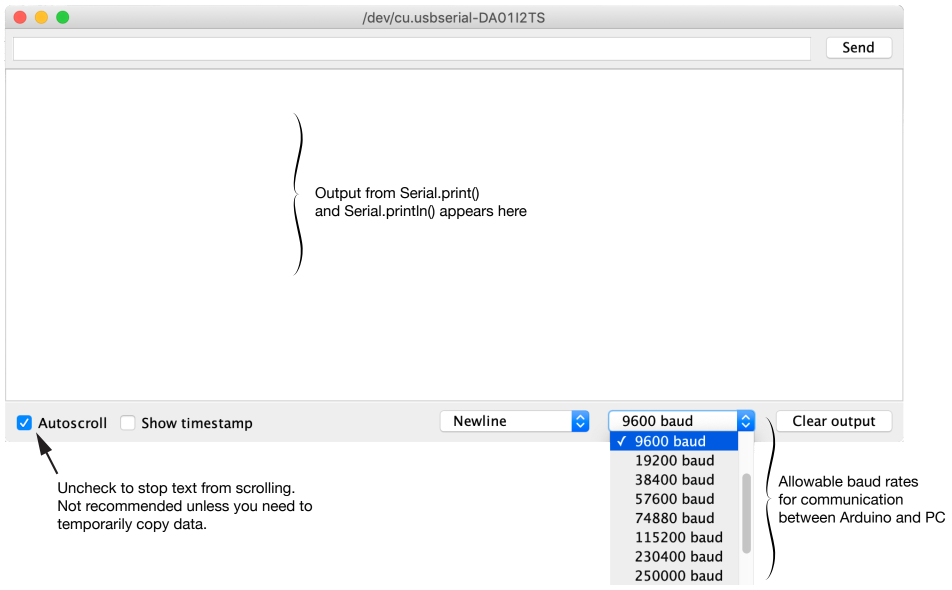
/dev/cu.usbserial-DA01I2TS
Send
Output from Serial.print() and Serial.println() appears here
Autoscroll
Show timestamp
Newline
9600 baud
Clear output
✓ 9600 baud
19200 baud
38400 baud
57600 baud
74880 baud
115200 baud
230400 baud
250000 baud
Uncheck to stop text from scrolling. Not recommended unless you need to temporarily copy data.
Allowable baud rates for communication between Arduino and PC

Figure 1: Image of the Serial Monitor showing options for freezing the display and adjusting the baud rate.

Adafruit use the higher 115200 baud rate.

3. Baud rate is also set in the Serial Monitor. The value specified in `Serial.begin` must be consistent with the value in the Serial Monitor so that the Arduino and your PC can talk to each other.

Waiting for the Serial Monitor to Start

1. On Arduino UNO boards, the connection between the board and the Serial Monitor program will be reset in these circumstances.
    - When a new sketch is successfully uploaded to the board.
    - When the reset button on the board is pressed.

2. For boards with native USB support, like the Arduino Leonardo, Arduino Due and Feather nRF52840, the connection between the board and the Serial Monitor is not automatically reset. When starting a sketch it is usually necessary wait for the `Serial` object in the sketch to establish a connection with the USB port on the host computer. One way to wait for the serial connection is to include these lines in the `setup` function of your sketch.

    ```
    Serial.begin(nnnn);
    while (!Serial) yield();
    ```

    where `nnnn` is the baud rate. The `while (!Serial)` test will be true (`Serial` is false and `!Serial` is true) until the serial port is ready[2]. The call to `yield();` tells the multitasking scheduler that the current Arduino sketch will let other threads run to completion.

3. **Warning:** If your Arduino is not connected to your host computer via USB, e.g., if you are running on battery power, the `while (!Serial) yield();` code will hang your sketch. If a serial port connection cannot be established, the sketch will stay in the `while(...)` loop indefinitely. Therefore, be sure to remove `while (!Serial) yield();` code when compiling your sketch for untethered operation.

Use of Printing Commands in Arduino Sketches

1. `Serial.print(x)` prints the value of `x` (converted to a string) to the Serial monitor.

2. `Serial.println(x)` prints the value of `x` (converted to a string) to the Serial monitor and advances the cursor to the next line.

3. Only one value (one `x`) can be printed at a time.

4. The `x` in `Serial.print(x)` or `Serial.println(x)` can be a string or a single number. The number can be either an integer or a floating point value.
    - If `x` is an integer, `Serial.print(x)` prints the string version of the value stored in x
    - If `x` is a floating point value, `Serial.print(x)` prints the string representing the four most significant digits of `x`.

[2]See e.g., `https://www.arduino.cc/reference/en/language/functions/communication/serial/ifserial/` and `https://arduino.stackexchange.com/questions/4556/what-does-the-line-while-serial-do-in-an-arduino-program`.

- When numerical values are printed, two arguments can be used in the `Serial.print()` and `Serial.println()` commands. The second argument determines the format or precision of numerical value. See `https://www.arduino.cc/reference/en/language/functions/communication/serial/print/`, and the output from Listing 1.
- Controlling horizontal position with spaces or tabs is helpful when one or more numerical values are printed on the same line in the Serial Monitor. Listing 2 and Listing 4 show examples of controlling the horizontal space between subsequent print commands.

5. Horizontal cursor position

   - The cursor location is the starting position of the next `Serial.print` or `Serial.println` statement. The cursor is analogous to the insertion point for text input in a word processor.
   - The cursor is left at the end of the string printed by `Serial.print(x)`, or it is left at the start of the next line after the string printed by `Serial.println(x)`.
   - The horizontal position of the start of a `Serial.print(x)` or `Serial.println(x)` can be controlled by adding the special characters `\t` (tab) or `\n` (newline) as discussed below. The horizontal position can also be controlled with one or more `Serial.print("...")` statements that print additional characters without moving to the next line.

6. The argument of `Serial.print(x)` or `Serial.println(x)` can be a string enclosed in double quotes, for example,

   ```
   Serial.print("Hello, world")
   ```

   - The string can include any number of tab characters (`\t`) or new line characters (`\n`).
   - A tab adds horizontal space. In the Serial Monitor, tabs are expanded to 8 spaces *or* advances to an integral number of 8 spaces from the left margin. See the output from Listing 2.
   - A tab can be used to indent a first line or add space between two subsequent `Serial.print()` commands.
   - A new line advances the cursor to the next row in the Serial Monitor.
   - Examples:

     `Serial.print("\t x = ")` adds a tab (8 blank spaces) *and* a space before "`x =` " is printed

     `Serial.print("\tx = ")` adds a tab before "`x =` " is printed. The lack of space between `\t` and `x` is OK.

     `Serial.print("My message ...\n\n")` prints `"My message ..."` followed by one blank line. The first "`\n`" advances the cursor to the line after "End of setup" and the second "`\n`" adds a blank line.

     `Serial.println("My message ...\n")` is equivalent to
     `Serial.print("My message ...\n\n")`.

Examples

Use of Format Specifiers

```
//  File: demoSerialPrint.ino
//
//  Demonstrate features of Serial.print and Serial.println

void setup() {

  int n = 12345;
  float x = 12.3456789;

  Serial.begin(9600);

  Serial.println("Print n:");
  Serial.println(n);
  Serial.println(n,BIN);
  Serial.println(n,HEX);

  Serial.println("\nPrint x:");
  Serial.println(x);
  Serial.println(x,0);
  Serial.println(x,2);
  Serial.println(x,4);
}

void loop() {}  // intentionally blank
```

Listing 1: Code to demonstrate features of the `Serial.print` command.

Output from the `demoSerialPrint` sketch in Listing 1:

```
Print n:
12345
11000000111001
3039

Print x:
12.35
12
12.35
12.3457
```

Horizontal Spaces and Tabs

```
//  File: demoSerialPrintSpaces.ino
//
//  Demonstrate ways of adding horizontal spaces in sequences of Serial.print
//  and Serial.println commands

void setup() {

  int n = 12345;
  float x = 12.3456789, y, z;

  Serial.begin(9600);

  y = 2*x;
  z = sqrt(x);

  Serial.println("\nPrint x, y and z with different horizontal spacing:");
  Serial.print("\t");
  Serial.print(x);
  Serial.print("    ");
  Serial.print(y);
  Serial.print("\t");
  Serial.println(z);
  Serial.println("0123456789012345678901234567890123456789");  // indicate position

  Serial.println("\nPrint n with different tab spacings:");
  Serial.print("\t");
  Serial.print(n);
  Serial.print("\t\t");
  Serial.print(n);
  Serial.print("        ");   // 8 spaces
  Serial.println(n);
  Serial.println("0123456789012345678901234567890123456789");  // indicate position
}

void loop() {}  // intentionally blank
```

Listing 2: Code to demonstrate how to print columns of numbers with `Serial.print` and `Serial.println`.

Output from the `demoSerialPrintSpaces` sketch in Listing 2:

```
Print x, y and z with different horizontal spacing:
        12.35    24.69  3.51
0123456789012345678901234567890123456789

Print n with different tab spacings:
        12345           12345        12345
0123456789012345678901234567890123456789
```

Labeling Parameters of a Mathematical Function

```
//  File: demoSerialPrintParameters.ino
//
//  Use Serial.print and Serial.println to print labeled
//  parameters, e.g. constants used in a math formula.

void setup() {

  int i;
  float c1 = 5, c2 = -3.0, t = 0.732194, y;

  Serial.begin(9600);

  Serial.println("\nParameters of an exponential decay function");

  Serial.print("\n\tc1 = ");   Serial.println(c1);
  Serial.print("\tc2 = ");     Serial.println(c2);

  Serial.print("\nEvaluate y = c1*exp(c2*t) at t = ");
  Serial.println(t,5);

  y = c1*exp(c2*t);
  Serial.print("\n\ty = ");    Serial.println(y,5);

}

void loop() {}  // intentionally blank
```

Listing 3: Code to demonstrate how to print labelled parameters of a mathematical function with `Serial.print` and `Serial.println`.

Output from the `demoSerialPrintParameters` in Listing 3:

```
Parameters of an exponential decay function

        c1 = 5.00
        c2 = -3.00

Evaluate y = c1*exp(c2*t) at t = 0.73219

        y = 0.55591
```

Printing Columns

```
//  File: demoSerialPrint.ino
//
//  Demonstrate how to print a table of numbers with
//  Serial.print and Serial.println

#define PI 3.1415926535897932    //  Excess digits included

void setup() {

  int i,n = 5;
  float dtheta, theta, s,c,t;

  Serial.begin(9600);
  Serial.println("\ntheta   sin(theta)  cos(theta)  tan(theta)");

  dtheta = 0.5*PI/float(n);

  theta = 0.0;
  for ( i=1; i<=n; i++) {
    s = sin(theta);
    c = cos(theta);
    t = tan(theta);
    Serial.print(theta,4);
    Serial.print("    ");
    Serial.print(s,4);
    Serial.print("      ");
    Serial.print(c,4);
    Serial.print("      ");
    Serial.println(t,4);

    theta += dtheta;
  }
}

void loop() {}  // intentionally blank
```

Listing 4: Code to demonstrate how to print columns of numbers with `Serial.print` and `Serial.println`.

Output from the `demoSerialPrintTable` sketch in Listing 4:

```
theta   sin(theta)  cos(theta)  tan(theta)
0.0000    0.0000      1.0000      0.0000
0.3142    0.3090      0.9511      0.3249
0.6283    0.5878      0.8090      0.7265
0.9425    0.8090      0.5878      1.3764
1.2566    0.9511      0.3090      3.0777
```