Serial Input Basics
By: Robin2
http://forum.arduino.cc/index.php?topic=396450
Compiled by: Tiktak

Contents

The following sections are in this Tutorial

* Introduction
* Serial data is slow by Arduino standards
* Example 1 - Receiving single characters
* Why code is organized into functions
* Example 2 - Receiving several characters from the Serial Monitor

* Example 3 - A more complete system
* How many characters can be received?
* Things that are not used in the examples
* serialEvent()
* Clearing the input buffer

* Receiving numbers rather than text
* Example 4 - Receiving a single number from the Serial Monitor
* Example 5 - Receiving and parsing several pieces of data
* Binary data
* Example 6 - Program to receive binary data

Introduction

Newcomers often seem to have difficulty with the process of receiving Serial data on the Arduino - especially when they need to receive more than a single character. The fact that there are 18 different functions listed on the Serial reference page probably does not help

You could write a small book and still not cover every possible situation for data reception. Rather than write pages and pages that few would read I thought it would be more useful to present a few examples which will probably cover all of a newcomer's needs. And when you understand these examples you should be able to figure out solutions for other strange cases.

Almost all serial input data can be covered by three simple situations

A - when only a single character is required
B - when only simple manual input from the Serial Monitor is required
C - other

Serial data is slow by Arduino standards

When anything sends serial data to the Arduino it arrives into the Arduino input buffer at a speed set by the baud rate. At 9600 baud about 960 characters arrive per second which means there is a gap of just over 1 millisecond between characters. The Arduino can do a lot in 1 millisecond so the code that follows is designed not to waste time waiting when there is nothing in the input buffer even if all of the data has not yet arrived. Even at 115200 baud there is still 86 microseconds or 1376 Arduino instructions between characters.

And because data arrives relatively slowly it is easy for the Arduino to empty the serial input buffer even though all of the data has not yet arrived. Many newcomers make the mistake of assuming that something like while (Serial.available() > 0) { will pick up all the data that is sent. But it is far more likely that the WHILE will empty the buffer even though only part of the data has arrived.

Example 1 - Receiving single characters

In very many cases all that is needed is to send a single character to the Arduino. Between the upper and lower case letters and the numeric characters there are 62 options. For example you could use 'F' for forward, 'R' for reverse and 'S' for stop.

Code to receive a single character is as simple as this

```
// Example 1 - Receiving single characters

char receivedChar;
boolean newData = false;

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvOneChar();
    showNewData();
}

void recvOneChar() {
    if (Serial.available() > 0) {
        receivedChar = Serial.read();
        newData = true;
    }
}

void showNewData() {
    if (newData == true) {
        Serial.print("This just in ... ");
        Serial.println(receivedChar);
        newData = false;
    }
}
```

Why code is organized into functions

Even though this example is short and simple I have deliberately put the code to receive the character into a separate function called recvOneChar() as that makes it simple to add it into any other program. I also have the code for showing the character in the function showNewData() because you can change that to do whatever you want without upsetting the rest of the code.

If you wish to use the code from any of the examples in your own program I suggest that you just copy the complete functions from the relevant example and create the necessary global variables at the top of your own program.

Example 2 - Receiving several characters from the Serial Monitor

If you need to receive more than a single character from the Serial Monitor (perhaps you want to input people's names) you will need some method of letting the Arduino know when it has received the full message. The simplest way to do this is to set the line-ending to newline.

This is done with the box at the bottom of the Serial Monitor window. You can choose between "No line ending", "Newline", "Carriage return" and "Both NL and CR". When you select the "Newline" option a new-line character ('\n') is added at the end of everything you send.

```
// Example 2 - Receive with an end-marker

const byte numChars = 32;
char receivedChars[numChars];   // an array to store the received data

boolean newData = false;

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvWithEndMarker();
    showNewData();
}

void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        }
        else {
            receivedChars[ndx] = '\0'; // terminate the string
            ndx = 0;
            newData = true;
        }
    }
}

void showNewData() {
    if (newData == true) {
        Serial.print("This just in ... ");
        Serial.println(receivedChars);
        newData = false;
    }
}
```

Example 3 - A more complete system

The simple system in Example 2 will work well with a sympathetic human who does not try to mess it up. But if the computer or person sending the data cannot know when the Arduino is ready to receive there is a real risk that the Arduino will not know where the data starts.

If you would like to explore this, change the end marker in the previous program from '\n' to '>' so that you can include the end marker in your text for illustration purposes. (You can't manually enter a Newline character in the text you are sending from the Serial Monitor). And put the line ending back to "No line ending"

Now, with the revised code send "qwert>" and you will see that it behaves exactly the same as when you were using Newline as the end marker.

But if you try this "asdfg>zxcvb" you will only see the first part "asdfg". And then if you send "qwert>" you will see "zxcvbqwert" because the Arduino has become confused and cannot know that it should have ignored "zxcvb".

The answer to this problem is to include a start-marker as well as an end-marker.

```
// Example 3 - Receive with start- and end-markers

const byte numChars = 32;
char receivedChars[numChars];

boolean newData = false;

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvWithStartEndMarkers();
    showNewData();
}

void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }

        else if (rc == startMarker) {
            recvInProgress = true;
        }
    }
}

void showNewData() {
    if (newData == true) {
        Serial.print("This just in ... ");
        Serial.println(receivedChars);
        newData = false;
    }
}
```

To see how it works try sending "qwerty<asdfg>zxcvb" and you will see that it ignores everything except "asdfg".

In this program you will notice that there is a new variable called recvInProgress. This is necessary because a distinction needs to be made between unwanted characters that arrive before the start marker and the valid characters that arrive after the start marker.

This version of the program is very similar to the Arduino code in this Python - Arduino demo.

How it works

It is important to notice that each time the function recvWithEndMarker() or recvWithStartEndMarker() is called it reads whatever characters may have arrived in the serial input buffer and places them in the array receivedChars.

If there is nothing in the buffer recvWithEndMarker() does not waste time waiting.

In the case of recvWithStartEndMarker() all characters are discarded until the start-marker is detected.

If the end-marker has not yet arrived it will try again when loop() next repeats.

For best results it is important to ensure that loop() can repeat as quickly as possible - hundreds or even thousands of times per second.

How many characters can be received?

In the examples I have assumed that you will not need to receive more than 32 bytes. That can easily be altered by changing the value in the constant numChars.

Note that the 64 byte size of the Arduino serial input buffer does not limit the number of characters that you can receive because the code in the examples can empty the buffer faster than new data arrives.

Things that are not used in the examples

You will notice that the examples here do not use any of these Arduino functions

- Serial.parseInt()
- Serial.parseFloat()
- Serial.readBytes()
- Serial.readBytesUntil()

All of these are blocking functions that prevent the Arduino from doing something else until they are satisfied, or until the timeout expires. The examples here do exactly the same job without blocking. That allows the Arduino to do other things while it is waiting for data to arrive.

serialEvent()

I don't recommend using this function - I prefer to deal with the Serial data when it suits me. It behaves just as if you had this code as the last thing in loop().

```
if (Serial.available > 0) {
  mySerialEvent();
}
```

Clearing the input buffer

It is probably worth mentioning that the poorly named Serial.flush() function does not empty the input buffer. It is only relevant when the Arduino is sending data and its purpose is to block the Arduino until all outgoing the data has been sent.

If you need to ensure the Serial input buffer is empty you can do so like this

```
while (Serial.available() > 0) {
   Serial.read();
}
```

Receiving numbers rather than text

So far the examples have assumed you want to receive text. But perhaps you want to send a number or maybe a mix of text and numbers.

Example 4 - Receiving a single number from the Serial Monitor

The simplest case is where you want to type a number into the Serial Monitor (I am assuming you have line-ending set to newline). Let's assume you want to send the number 234. This is a variation on Example 2 and it will work with any integer value. Note that if you don't enter a valid number it will show as 0 (zero).

```
// Example 4 - Receive a number as text and convert it to an int

const byte numChars = 32;
char receivedChars[numChars];   // an array to store the received data

boolean newData = false;

int dataNumber = 0;             // new for this version

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvWithEndMarker();
    showNewNumber();
}

void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    if (Serial.available() > 0) {
        rc = Serial.read();

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        }
        else {
            receivedChars[ndx] = '\0'; // terminate the string
            ndx = 0;
            newData = true;
        }
    }
}

void showNewNumber() {
    if (newData == true) {
        dataNumber = 0;             // new for this version
        dataNumber = atoi(receivedChars);   // new for this version
        Serial.print("This just in ... ");
        Serial.println(receivedChars);
        Serial.print("Data as Number ... ");    // new for this version
        Serial.println(dataNumber);     // new for this version
        newData = false;
```

```
    }
}
```

Example 5 - Receiving and parsing several pieces of data

It is also straightforward to receive several pieces of data in a single message and parse the data to assign them to individual variables. This example assumes you send something like "<HelloWorld, 12, 24.7>". This is an extension of Example 3.

A function called parseData() has been added and the function showParsedData() takes the place of showNewData() in the earlier example.

```
// Example 5 - Receive with start- and end-markers combined with parsing

const byte numChars = 32;
char receivedChars[numChars];
char tempChars[numChars];        // temporary array for use when parsing

      // variables to hold the parsed data
char messageFromPC[numChars] = {0};
int integerFromPC = 0;
float floatFromPC = 0.0;

boolean newData = false;

//============

void setup() {
    Serial.begin(9600);
    Serial.println("This demo expects 3 pieces of data - text, an integer and a floating point
value");
    Serial.println("Enter data in this style <HelloWorld, 12, 24.7>  ");
    Serial.println();
}

//============

void loop() {
    recvWithStartEndMarkers();
    if (newData == true) {
        strcpy(tempChars, receivedChars);
            // this temporary copy is necessary to protect the original data
            //   because strtok() used in parseData() replaces the commas with \0
        parseData();
        showParsedData();
        newData = false;
    }
}

//============

void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
```

```
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }

        else if (rc == startMarker) {
            recvInProgress = true;
        }
    }
}

//============

void parseData() {      // split the data into its parts

    char * strtokIndx; // this is used by strtok() as an index

    strtokIndx = strtok(tempChars,",");      // get the first part - the string
    strcpy(messageFromPC, strtokIndx); // copy it to messageFromPC

    strtokIndx = strtok(NULL, ","); // this continues where the previous call left off
    integerFromPC = atoi(strtokIndx);     // convert this part to an integer

    strtokIndx = strtok(NULL, ",");
    floatFromPC = atof(strtokIndx);     // convert this part to a float

}

//============

void showParsedData() {
    Serial.print("Message ");
    Serial.println(messageFromPC);
    Serial.print("Integer ");
    Serial.println(integerFromPC);
    Serial.print("Float ");
    Serial.println(floatFromPC);
}
```

Binary data
======
So far we have been receiving character data - for example the number 121 is represented by the characters '1', '2' and '1'. It is also possible to send that value as binary data in a single byte - it happens to be the Ascii value for the character 'y'. Note that 121 in decimal is the same as 0x79 in HEX.

Note that if you are sending binary data it is quite likely that you will need to send as data the same values that are used for the start- and end-markers. That goes beyond the scope of this Tutorial and one way of doing it is illustrated in the demo here.

The examples that follow assume that the binary data will NEVER include the byte values used for the start- and end-markers. For simplicity I will continue to use < and > as the markers. The byte values for those characters are 0x3C and 0x3E. This will allow you to test the program from the Serial Monitor by sending, for example, <24y> which will be interpreted by the receiving program as the binary values 0x32, 0x34 and 0x79. These are the Ascii codes for 2, 4 and y.

Of course it would be more usual for binary data to be sent by another computer program - on another Arduino or on a PC.

Example 6 - Program to receive binary data
This is adapted from Example 3

```
// Example 6 - Receiving binary data

const byte numBytes = 32;
byte receivedBytes[numBytes];
byte numReceived = 0;

boolean newData = false;

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvBytesWithStartEndMarkers();
    showNewData();
}

void recvBytesWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    byte startMarker = 0x3C;
    byte endMarker = 0x3E;
    byte rb;


    while (Serial.available() > 0 && newData == false) {
        rb = Serial.read();

        if (recvInProgress == true) {
            if (rb != endMarker) {
                receivedBytes[ndx] = rb;
                ndx++;
                if (ndx >= numBytes) {
                    ndx = numBytes - 1;
                }
            }
            else {
                receivedBytes[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                numReceived = ndx;  // save the number for use when printing
                ndx = 0;
                newData = true;
            }
        }

        else if (rb == startMarker) {
            recvInProgress = true;
        }
    }
}

void showNewData() {
    if (newData == true) {
        Serial.print("This just in (HEX values)... ");
        for (byte n = 0; n < numReceived; n++) {
            Serial.print(receivedBytes[n], HEX);
            Serial.print(' ');
        }
        Serial.println();
        newData = false;
    }
}
```