4

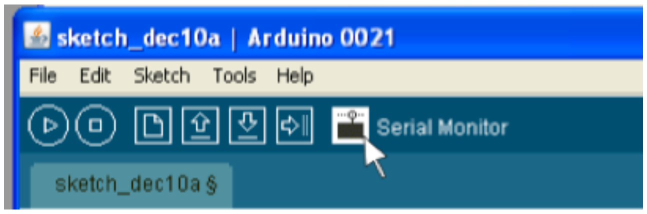
Serial

Name:
Date:

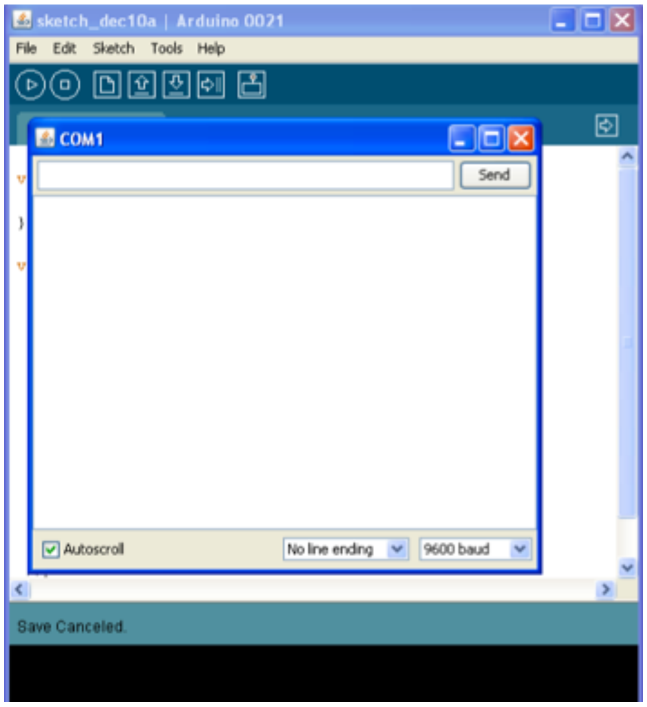
Serial is used to communicate between your computer and the RedBoard as well as between RedBoard boards and other devices. Serial uses a serial port also known as UART, which stands for universal asynchronous receiver/transmitter to transmit and receive information. In this case the computer outputs Serial Communication via USB while the RedBoard receives and transmits Serial using, you guessed it, the RX and TX pins. You use serial communication every time you upload code to your Arduino board. You will also use it to debug code and troubleshoot circuits. Basic serial communication is outlined in the following pages along with a simple activity to help you understand the concepts.

Serial Monitor:
This is where you monitor your serial communication and set baud rate.

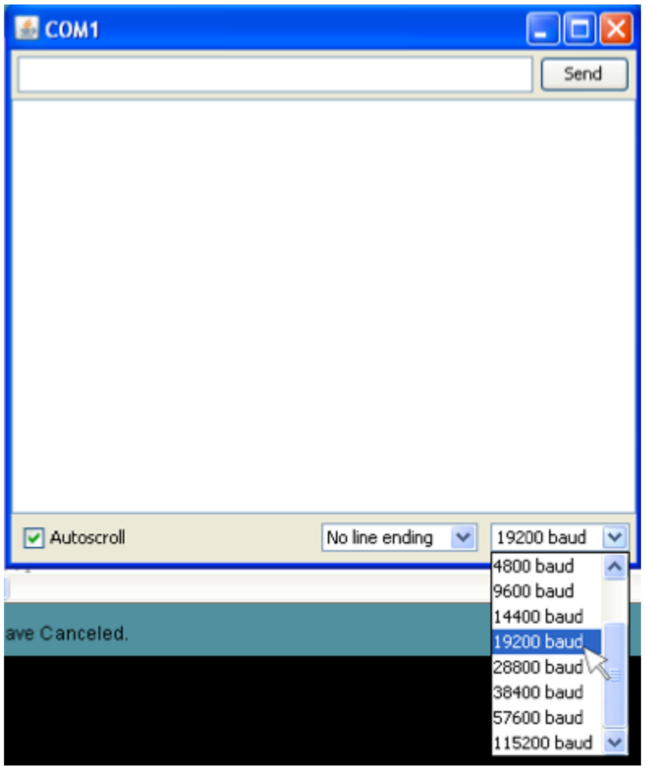
Activating the Serial Monitor:



What the activated Serial Monitor looks



Setting the Serial Monitor baud rate:



There are many different baud rates, (9600 is the standard for Arduino) the higher the baud rate the faster the machines are communicating.

In the examples above there is no Serial communication taking place yet. When you are running code that uses Serial any messages or information you tell Serial to display will show up in the window that opens when you activate the monitor.

Things to remember about Serial from this page:
1. Serial is used to communicate, debug and troubleshoot.
2. Serial baud rate is the rate at which the machines communicate.

Name:
Date:

Serial setup:

The first thing you need to know to use Serial with your Arduino code is Serial setup. To setup Serial you simply type the following line inside your setup( ) function:
Serial.begin (9600);

This line establishes that you are using the Digital Pins # 0 and # 1 for Serial communication. This means that you will not be able to use these pins as Input or Output because you are dedicating them to Serial communication. The number 9600 is the baud rate, this is the rate at which the computer and the Arduino communicate. You can change the baud rate depending on your needs but you need to make sure that the baud rate in your Serial setup and the baud rate on your Serial Monitor are the same. If your baud rates do not match up the Serial Monitor will display what appears to be gibberish, but is actually the correct communication incorrectly translated.

Using Serial for code debugging and circuit troubleshooting:

Once Serial is configured using the basic communication for debugging and troubleshooting is pretty easy. Anywhere in your sketch you wish the Arduino board to send a message type the line Serial.println("communication here");. This command will print whatever you type inside the quotation marks to the Serial Monitor followed by a return so that the next communication will print to the next line. If you wish to print something without the return use Serial.print("communication here");. To display the value of a variable using println simple remove the quotation marks and type the variable name inside the parenthesis. For example, type Serial.println(i); to display the value of the variable named i. This is useful in many different ways, if, for example, you wish to print some text followed by a variable or you want to display multiple variables before starting a new line in the Serial Monitor.

These lines are useful if you are trying to figure out what exactly your Arduino code is doing. Place a println command anywhere in the code, if the text in the println command shows up in your Serial Monitor you will know exactly when the Arduino reached that portion of code, if the text does not show up in the Serial Monitor you know that portion of code never executed and you need to rewrite.

To use Serial to troubleshoot a circuit use the println command just after reading an input or changing an output. This way you can print the value of a pin signal. For example, type Serial.print("Analog pin 0 reads:"); and Serial.println(analogRead(A0)); to display the signal on Analog Input Pin # 0. Replace the second portion with Serial.println(digitalRead(10)); to display the signal on Digital Pin # 10.

Things to remember about Serial from this page:

1. If Serial is displaying gibberish check the baud rates.
2. Use Serial.print("communication here"); to display text.
3. Use Serial.println("communication here"); to display text and start a new line.
4. Use Serial.print(variableName); to display the value stored in variableName.
5. Use Serial.print(digitalRead(10)); to display the state of Digital Pin # 10.

Name:
Date:

Using Serial for communication:
This is definitely beyond the scope of the S.I.K. but here are some basics for using Serial for device to device communication (other than your computer), not just debugging or troubleshooting. (The following paragraphs assume that you have Serial Communication hardware properly connected and powered on two different devices.)

First set up Serial as outlined on the previous page.

Use Serial.println("Outgoing communication here"); to send information out on the transmit line.

When receiving communication the Serial commands get a little more complicated. First you need to tell the RedBoard to listen for incoming communication. To do this you use the command Serial.available();, this command tells the computer how many bytes have been sent to the receive pin and are available for reading. The Serial receive buffer (computer speak for a temporary information storage space) can hold up to 128 bytes of information.

Once the RedBoard knows that there is information available in the Serial receive buffer you can assign that information to a variable and then use the value of that variable to execute code. For example to assign the information in the Serial receive buffer to the variable incomingByte type the line; incomingByte = Serial.read(); Serial.read() will only read the first available byte in the Serial receive buffer, so either use one byte communications or study up on parsing and string variable types. Below is an example of code that might be used to receive Serial communication at a baud rate of 9600.

```
//declare the variable incomingByte and assign it the value 0.
int incomingByte = 0;

void setup ( ) {
//establish serial communication at a baud rate of 9600
        Serial.begin(9600);
}
void loop ( ) {
//if there is information in the Serial receive buffer
        if (Serial.available() > 0){
//assign the first byte in buffer to incomingByte
                incomingByte = Serial.read();
        }
        if (incomingByte == 'A'){     //if incomingByte is A
                //execute code inside these brackets if
incomingByte is A
        }
        if (incomingByte == 'B'){     //if incomingByte is B
                //execute code inside these brackets if
incomingByte is B
        }
}
```

Additional things to note about Serial:
You cannot transmit and receive at the same time using Serial, you must do one or the other. You cannot hook more than two devices up to the same Serial line. In order to communicate between more than two devices you will need to use an Arduino library such as NewSoftSerial.

Things to remember about Serial from this page:
1. Serial communication requires knowing some code, but you can just look it up!
2. You cannot transmit and receive at the same time or hook up more than two devices.

Name:
Date:

// Activity

Practicing simple Serial for debugging code:
If you would like to practice using Serial for code debugging open the code file Serial01, copy the text and paste it into an Arduino sketch. This sketch is mainly empty and waiting for you to add the Serial commands.

1. First type in the command line that begins Serial at a baud rate of your choosing below where the comment reads “place serial setup here”.

2. Next open the Serial Monitor and make sure it matches the baud rate you chose. This is an example of setting up Serial communication.

3. Next type a single Serial command that will display the text “Loop starts here” below the comment “place serial statement 1 here”. Make sure the command you use starts a new line after this text is displayed.

4. Then type a single Serial command that will display the text “Variable i is equal to “ below the comment “place serial statement 2 here”. Make sure you use the command that does not start a new line after this text is displayed.

5. Now add a command below this that will display the variable i. If you are having trouble with this portion don’t forget that only text needs quotation marks around it for display in the Serial Monitor. This is an example of how you can use Serial communication to label your communication when you are trying to debug a troublesome variable.

6. Below the comment “place serial statement 3 here” add a command that will display the text “this text displays when i is equal to 8” and then start a new line. This is an example of how to display a variable value for debugging.

7. Below the comment “place serial statement 4 here” add a command that will display the text “this text displays when i is equal to 9” and then start a new line. This is an example of using Serial communication see if a portion of code ever actually executes.

8. Below the comment “place serial statement 5 here” add a command that will display the text “Loop ends here” and then start a new line.

Name:
Date:

// Activity

Using Serial for communication:
This is definitely beyond the scope of the S.I.K. but here are some basics for using Serial for device to device communication (other than your computer), not just debugging or troubleshooting. (The following paragraphs assume that you have Serial Communication hardware properly connected and powered on two different devices.)

First set up Serial as outlined on the previous page.

Use Serial.println("Outgoing communication here"); to send information out on the transmit line.

When receiving communication the Serial commands get a little more complicated. First you need to tell the Arduino to listen for incoming communication. To do this you use the command Serial.available();, this command tells the computer how many bytes have been sent to the receive pin and are available for reading. The Serial receive buffer (computer speak for a temporary information storage space) can hold up to 128 bytes of information.

Once the RedBoard knows that there is information available in the Serial receive buffer you can assign that information to a variable and then use the value of that variable to execute code. For example to assign the information in the Serial receive buffer to the variable incomingByte type the line; incomingByte = Serial.read(); Serial.read() will only read the first available byte in the Serial receive buffer, so either use one byte communications or study up on parsing and string variable types. Below is an example of code that might be used to receive Serial communication at a baud rate of 9600.