What is the Arduino?

rodbird

Table of Contents

What is the Arduino?

Start simply

The Sketch

Our Liberty BASIC GUI

The message format

Current command set

Making the connection

Starting the conversation

Send Request

Get Response

The complete GUI

The command list

AddServo

GetDigital

GetAnalog

GetPWM

SetDigital

SetPWM

SetServo

SetTone

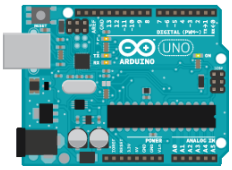
Ping

DelServo

SetSpeed

SetStepR

SetStepL



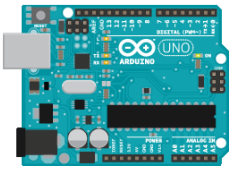
The Arduino is a much loved general purpose input and output board. It has a huge following and lots of support. This article is about how you can interface the board via the serial port and so give Liberty BASIC access to all that the board has to offer.

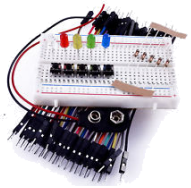
Start simply

If this is your first step into microprocessor electronics my advice would be to start simply. Purchase a Uno Kit. There are several easily obtainable kits, each provides the board, electronic components and usually a booklet to get you up and running quickly.

Some kits use soldered components, others offer plug in breadboard connection or ready made plug and socket connectors. Probably best to opt for plug in breadboard style which lets you build circuits quickly while you play and learn.

This is all the kit you need to get started:





Purchase this kind of kit and play with it for a day or so, follow the tutorials available online or use the booklet. Get to the point where you have downloaded your own sketch to make the Arduino's onboard LED flash. Now you are ready to unleash Liberty BASIC.

The Sketch

You will now know that the Arduino uses a BASIC like programming language but there are significant differences to Liberty BASIC. These programs are called Sketches and you write them on your PC and download them to the Arduino which fires up and runs autonomously. You can go and learn this language, and you probably will over time, but right now we are going to use a general purpose Sketch written by the Liberty BASIC community. This Sketch allows simple serial messages to control and receive info from the Arduino. So the Arduino is running autonomously but it is dedicated to interpreting and actioning our control requests. It has become a slave to Liberty BASIC.

Here is the Sketch, you will need to download it to your Arduino, follow the booklet instructions on how to do that.

```
#include <Stepper.h>
// initialize 100 step stepper on pins 8&9:
// alter these parameters to suit your stepper
Stepper myStepper(100, 8, 9);

#include <Servo.h>
// set up 14 servo objects in an array
Servo myservos[14] ;

void setup() {
  Serial.begin(9600);
  Serial.println("Com Open");
}
void loop() {

  // you can call your own functions here
  // lbRun() is only called by interrupt
```

```
  // when there is serial data available

}

// serialEvent checks to see if data is available on serial port
// when a message comes we just invoke our lbRun() function


void serialEvent() {
  // check if data available, if so call the lbRun() routine
  if ( Serial.available()) {
    lbRun();
  }
}


void lbRun()
{
  while (Serial.available() > 10)
  {
    // look for the next valid integer in the incoming serial stream:
    int cmd = Serial.parseInt();
    // do it again:
    int pin = Serial.parseInt();
    // do it again:
    int val = Serial.parseInt();
    // look for the "*", That's the end of our command string
    if (Serial.read() == '*')
    {
      //now select the command to run and use the pin and val argument
      if ( cmd == 0)//add servo
      {
        myservos[pin].attach(pin);
      }

      if ( cmd == 1)//getdigital
      {
        pinMode(pin, INPUT);//set to input
        digitalWrite(pin, HIGH);//turn on pullup resistor
        Serial.print(pin);
        Serial.print(",");
        Serial.print(digitalRead(pin));
        Serial.print("*");
      }

      if ( cmd == 2)//getanalog
```

```
{
  analogRead(pin);
  delay(10);
  Serial.print(pin);
  Serial.print(",");
  Serial.print(analogRead(pin));
  Serial.print("*");
}

if ( cmd == 3)//getpulse
{
  Serial.print(pin);
  Serial.print(",");
  Serial.print(pulseIn(pin, val));
  Serial.print("*");
}

if ( cmd == 4)//setdigital
{
  pinMode(pin, OUTPUT);//set to ouput
  digitalWrite(pin, val);
}

if ( cmd == 5)//setanalog
{
  analogWrite(pin, val);
}

if ( cmd == 6)//setservo
{
  myservos[pin].write(val) ;
}

if ( cmd == 7)//settone
{
  if (val == 0)
  {
    noTone(pin);
  }
  else
  {
    tone(pin, val);
  }
}

if ( cmd == 8)//ping
```

```
      {

        // establish variables for duration of the ping,
        long duration;

        // The PING))) is triggered by a HIGH pulse of 2 or more micro
seconds.
        // Give a short LOW pulse beforehand to ensure a clean HIGH pu
lse:
        pinMode(pin, OUTPUT);
        digitalWrite(pin, LOW);
        delayMicroseconds(2);
        digitalWrite(pin, HIGH);
        delayMicroseconds(5);
        digitalWrite(pin, LOW);

        // The same pin is used to read the signal from the PING))): a
 HIGH
        // pulse whose duration is the time (in microseconds) from the
 sending
        // of the ping to the reception of its echo off of an object.
        pinMode(pin, INPUT);
        duration = pulseIn(pin, HIGH);
        Serial.print(pin);
        Serial.print(",");
        Serial.print(duration);
        Serial.print("*");
      }

      if ( cmd == 9)//detach servo
      {
        myservos[pin].detach();
      }

      if ( cmd == 10)//set the stepper speed
      {
        myStepper.setSpeed(val);
      }

      if (cmd == 11)//set the stepper in motion +
      // this is a blocking call, the stepper will run at
      // the setSpeed value till steps are complete
      {
        myStepper.step(val);
        Serial.print("99,0,*");
```

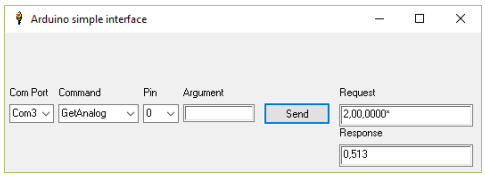
```
      }
      if (cmd == 12)//set the stepper in motion -
      {
        myStepper.step(-val);
        Serial.print("99,0,*");
      }
    }
  }
}   // end or lbRun()

// add your own functions here
```

Very briefly this Sketch loads the servo control and stepper library then starts polling the serial buffer, looking for complete messages that we have sent from Liberty BASIC. When it finds a complete message it interprets or parses that message to establish what we are asking it to do. Then on a simple select case basis it actions that request returning results if needed. We will pick out and explain what each part of the Sketch achieves as we discuss the commands that can be sent.

Our Liberty BASIC GUI

This is the GUI we will build to allow us to converse with the Arduino. This is a very simple GUI that connects with the Arduino over the USB port and allows us to format and send requests and receive responses. It will allow you to play with circuits quickly and easily.

Now we could write really simple code in the mainwin. It is perfectly possible to open the com port and send and receive messages from there. However there is likely to be a lot of user interaction so a GUI interface is abetter solution.



The message format

A Liberty BASIC message is sent in a comma delimited string with fixed length numeric values which have leading zeros:

```
"CC,PP,VVVV*"
```

Where:
CC = a two digit number specifying which command to action
PP = a two digit number specifying which pin to act on
VVVV = optionally, a four digit number passing any value to apply
followed by a terminating "*"

Responses, if issued, are received in a variable length message, comma delimited, in the following format:

```
"PP,VVVVn*"
```

Where:
PP = up to two digits specifying the pin producing the result
VVVVn = the numeric result
followed by a terminating "*"

Current command set

AddServo, You can attach servos to a pin then send rotation angle messages
GetDigital, You can read the digital state of a pin 0v=Low 5v=High
GetAnalog, You can read the analog state of a pin returning 0-1023 depending on the voltage at the pin
GetPWM, You can read the high or low timing in microseconds of pulsed pins
SetDigital, You can set the pin state to High or Low
SetPWM, You can set the duty cycle of a pulse train at a pin between 0% and 100%
SetServo, You can set the servo angle between 0o and 180o
SetTone, A pin can output a continuous tone at a specified frequency
Ping, You can measure the distance to and object via an ultrasound ping
DelServo, Servos need detached after use
SetSpeed, You can set stepper motor rotation speed in RPM
SetStepR, You can move the stepper motor right n steps
SetStepL, You can move the stepper motor left n steps

Making the connection

The first thing we need to do is open the serial link and connect to the Arduino. If you have downloaded the Sketch all you do is plug in the Arduino, Windows will open a Com port and the Arduino will start checking for messages. All we need do in Liberty BASIC is find the port number and start conversing.

Here is some code that will find the Com port and allow it to be opened. It uses a combobox, the

combobox is filled with available ports and you then pick which one you want to talk to. So you may have more than one Arduino connected but most often we will be dealing with one port and one board and the code should automatically select that port.

```
'Subs to handle the com port ==========================================
=====
    sub portClick h$
        'take com port combobox input, open choosen com port
        #main.cbport "selection? p$"
        if Port then close #port
        if p$<>"" then
            'open p$;":9600,n,8,1,ds0,cs0,rs" for random as #port
            Port=1
            call delay 500
        end if
    end sub

    sub getPorts

'test first 32 ports and load combobox list for valid serial ports
        index=1
        for p = 1 to 32
            oncomerror [trap]
            open "Com";str$(p);
":9600,n,8,1,ds0,cs0,rs" for random as #com
            port$(index)="Com";str$(p)
            index=index+1
            close #com

            [trap]
            oncomerror
        next
        #main.cbport, "reload"
        'now if there is only one port open it
        if port$(1)<>"" and port$(2)="" then
            open port$(1);":9600,n,8,1,ds0,cs0,rs" for random as #port
            Port=1
            #main.cbport, "selectindex 1"
            call delay 500
        else
            #main.cbport, "selectindex 0"
        end if
    end sub
```

Starting the conversation

I say conversation because you will be sending and receiving messages. You might change my example code and do this on a timed basis, regularly graphing and displaying the results. The system is quite fast. Obviously the serial link and message passing is a bottleneck compared with how fast the Arduino would run unfettered but you will be able to get many readings per second.

Lets look at some code that will manage the conversation. We need a send routine and a receive routine The send routine is pretty straight forwards because we initiate a single message though you might change the code to do this repetitively. The receiving is a little more complex because we may be receiving a stream of messages and we need to parse them out.

Send Request

```
'send the request if the button is clicked
 sub send h$
     #main.tbarg "!contents? v$"
     Val=val(v$)
     #main.tbresp ""
     msg$=str$(Cmd)+","
     msg$=msg$+right$("00"+str$(Pin),2)+","
     msg$=msg$+right$("0000"+str$(Val),4)+"*"
     #main.tbrequ msg$
     #port msg$
 end sub
```

Get Response

```
 sub getresponse
     if Port then
         if lof(#port)>0 then
             Buffer$=Buffer$+input$(#port, lof(#port))
             endofdata=instr(Buffer$,"*",1)
             [loop]
             if endofdata>0 then
                 'we have a valid end
                 dat$=left$(Buffer$,endofdata-1)
                 Buffer$=right$(Buffer$,len(Buffer$)-endofdata)
                 pin=val(word$(dat$,1,","))
                 dat=val(word$(dat$,2,","))
                 #main.tbresp pin;",";dat
```

```
                    endofdata=instr(Buffer$,"*",1)
                    if endofdata>0 then [loop]
                end if
            end if
        end if
    end sub
```

The complete GUI

```
 nomainwin
    'We need an array to store the Com port names in
    dim port$(256)
    'since we plan to use Subs we need a few globals
    'I tend to identify global variables with a capital letter
    global Port,Cmd,Pin,Val,Buffer$
    Port=0
    dim cmd$(14)
    cmd$(1)="AddServo"
'servos need attached to a pin before sending turn commands
    cmd$(2)="GetDigital"
'this gets the state of any digital pin, just ground it for low
    cmd$(3)="GetAnalog"
'this gets the analog value of the voltage on any analog pin 0v-5v
    cmd$(4)="GetPWM"
'this gets the pulse width in microseconds on any digital pin
    cmd$(5)="SetDigital"    'this sets any digital pin High or Low
    cmd$(6)="SetPWM"
'this sets the pulse width modulation 0%-100% (analog out)
    cmd$(7)="SetServo"
'sets preattached servo in degerees 0o-180o
    cmd$(8)="SetTone"
'sets a tone on any digital pin send 0 to silence
    cmd$(9)="Ping"
'pings an ultrasonic transducer to measure distance in air
    cmd$(10)="DelServo"     'detaches a servo from its named pin
    cmd$(11)="SetSpeed"
'sets the speed in RPM that a stepper will move
    cmd$(12)="SetStepR"
'sets the stepper moving right n steps will respond 0,99 when done
    cmd$(13)="SetStepL"
'sets the stepper moving left n steps will respond 0,99 when done

    dim pin$(20)
    for n= 0 to 13
```

```
        pin$(n+1)=str$(n)
    next

    WindowWidth = 550
    WindowHeight = 195
    UpperLeftX=int((DisplayWidth-WindowWidth)/2)
    UpperLeftY=int((DisplayHeight-WindowHeight)/2)
    combobox #main.cbport, port$(, portClick,    5,  80,  50, 100
    statictext #main.stport, "Com Port",   5,  60, 60,  20
    combobox #main.cbcmd, cmd$(, cmdClick,   60,  80,  90, 100
    statictext #main.stcmd, "Command",  60,  60,  60,  20
    combobox #main.cbpin, pin$(, pinClick,  155,  80,  40, 100
    statictext #main.stpin, "Pin", 155,  60,  20,  20
    textbox #main.tbarg, 200,  80,  80,  20
    statictext #main.starg, "Argument", 200,  60,  90,  20
    button #main.send,"Send",send, UL, 290,  78,  75,  25
    textbox #main.tbrequ, 375,  80, 150,  25
    statictext #main.strequ, "Request", 375,  60,  60,  20
    textbox #main.tbresp, 375,  125, 150,  25
    statictext #main.stresp, "Response", 375,  105,  60,  20
    open "Arduino simple interface" for window as #main
    #main "trapclose quit"

    'find out what com ports are available and load the combobox
    call getPorts


'we need an endless loop to clear out the serial buffer Arduino only s
tores 64 bytes
    while 1
        scan
        call getresponse
    wend




'Subs to handle comboboxes=============================================
=====


'what command did the user click,if it needs no argument set val to 0
    sub cmdClick h$
        #main.cbcmd "selectionindex? i"
```

```
'since our combobox array is numbered 1-13 deduct 1 to get 0-12
        Cmd=i-1
        if (Cmd>=0 and Cmd<=3) or (Cmd>=8 and Cmd<=9) then #
main.tbarg "0"
        if (Cmd>=10 and Cmd<=12) then #main.cbpin
"selectindex 1" : Pin=0
    end sub

    'what pin was selected
    sub pinClick h$
        #main.cbpin "selectionindex? i"
        Pin=i-1
    end sub


    'send the request if the button is clicked
    'we need to make it a fixed length string
    'because thats how the Arduino knows it
    'has a complete message
    sub send h$
        #main.tbarg "!contents? v$"
        Val=val(v$)
        #main.tbresp ""
        msg$=right$("00"+str$(Cmd),2)+","
        msg$=msg$+right$("00"+str$(Pin),2)+","
        msg$=msg$+right$("0000"+str$(Val),4)+"*"
        #main.tbrequ msg$
        #port msg$
    end sub


'suck the input buffer dry, keep the remnants of the message if the wh
ole
    'message has not been received, thats why Buffer$ is global
    sub getresponse
        if Port then
            if lof(#port)>0 then
                Buffer$=Buffer$+input$(#port, lof(#port))
                endofdata=instr(Buffer$,"*",1)
                [loop]
                if endofdata>0 then
                    'we have a valid end
                    dat$=left$(Buffer$,endofdata-1)
                    Buffer$=right$(Buffer$,len(Buffer$)-endofdata)
                    pin=val(word$(dat$,1,","))
                    dat=val(word$(dat$,2,","))
```

```
                    #main.tbresp pin;",";dat
                    endofdata=instr(Buffer$,"*",1)
                    if endofdata>0 then [loop]
                end if
            end if
        end if
    end sub


'Subs to handle the com port ==========================================
=====
    sub portClick h$
        'take com port combobox input, open choosen com port
        #main.cbport "selection? p$"
        if Port then close #port
        if p$<>"" then
            'open p$;":9600,n,8,1,ds0,cs0,rs" for random as #port
            Port=1
            call delay 500
        end if
    end sub

    sub getPorts

'test first 32 ports and load combobox list for valid serial ports
        index=1
        for p = 1 to 32
            oncomerror [trap]
            open "Com";str$(p);
":9600,n,8,1,ds0,cs0,rs" for random as #com
            port$(index)="Com";str$(p)
            index=index+1
            close #com

            [trap]
            oncomerror
        next
        #main.cbport, "reload"
        'now if there is only one port open it
        if port$(1)<>"" and port$(2)="" then
            open port$(1);":9600,n,8,1,ds0,cs0,rs" for random as #port
            Port=1
            #main.cbport, "selectindex 1"
            call delay 500
        else
            #main.cbport, "selectindex 0"
```

```
        end if
    end sub

    sub delay m
        CallDLL #kernel32, "Sleep", m As ulong, Sleep As void
    end sub

    sub quit h$
        close #main
        if Port then close #port
        end
    end sub
```

The command list

AddServo

Command = 0 , Pin Range = 0-13, Argument = Null, Response = Null
Servos must first be assigned a pin, use of the library disables PWM on pins 9 and 10

```
if ( cmd == 0)//add servo
{
myservos[pin].attach(pin);
}
```

GetDigital

Command = 1, Pin Range = 0-13, Argument = Null, Response = pin,0 or 1

The pin is set to input and the pullup resistor set high,ground the pin via a switch/sensor to pull it low.
If the pin is low 0 will be returned if high 1 will be returned.

```
if ( cmd == 1)//getdigital
{
pinMode(pin, INPUT);//set to input
digitalWrite(pin, HIGH);//turn on pullup resistor
Serial.print(pin);
Serial.print(",");
Serial.print(digitalRead(pin));
```

```
Serial.print("*");
}
```

GetAnalog

Command = 2, Pin Range 0-5 (analog), Argument = Null, Response = pin,0-1023

The analog pins 0-5 return 0-1023 measuring 0-5v on the pin, connect
pots, LDRs, Thermistors or any resistance based sensor. The pin is read twice, the delay settles the reading and improves accuracy.

```
if ( cmd == 2)//getanalog
{
analogRead(pin);
delay(10);
Serial.print(pin);
Serial.print(",");
Serial.print(analogRead(pin));
Serial.print("*");
}
```

GetPWM

Command = 3, Pin Range 0-13, Argument = 1 or 0, Response = pin,microseconds

Sending 0 as the Argument will measure the low pulse 1 will measure the high pulse,
connect gyros and accelerometers that provide PWM output.

```
if ( cmd == 3)//getpulse
{
Serial.print(pin);
Serial.print(",");
Serial.print(pulseIn(pin, val));
Serial.print("*");
}
```

SetDigital

Command = 4, Pin Range = 0-13, Argument = 0 or 1, Response = Null

This will set the pin HIGH or LOW use with LEDs or driver electronics to switch relays or pulse motors.

```
if ( cmd == 4)//setdigital//
{
pinMode(pin, OUTPUT);//set to ouput
digitalWrite(pin, val);
}
```

SetPWM

Command = 5, Pin Range = 3,5,6,9,10,11 , Argument = 0-255, Response = Null

This sets the PWM ratio on any of the legal pins from 0% HIGH to 100% HIGH connect LEDs to fade or brighten or with driver electronics control motor speed.

```
if ( cmd == 5)//setPWM
{
analogWrite(pin, val);
}
```

SetServo

Command = 6 , Pin Range = 0-13, Argument = 0o to 180o, Response = Null

This sets the servo angle in degrees. But many electronic motor control (ESCs) and gyro gadgets use this form of PWM output. Forward and reverse motor speed control and gyro stabilised servo control are all possible.

```
if ( cmd == 6)//setservo
{
myservos[pin].write(val) ;
}
```

SetTone

Command = 7 , Pin Range = 0-13, Argument = Hz 31 - 4978, Response = Null

Only one pin can output a tone at any time, sending 0 silences the tone.
A tone between 31Hz and 4978Hz can be specified it plays till silenced.

```
if ( cmd == 7)//settone
{
if (val == 0)
{
```

```
noTone(pin);
}
else
{
tone(pin, val);
}
}
```

Ping

Command = 8, Pin Range = 0-13, Argument = Null, Response = pin,microseconds

This pings an ultrasonic transducer, divide the result by 29 then 2 to get cm distance.
29 is the number of cm sound will travel in a microsecond, we divide by 2 because
the sound is bounced out and back.

```
if ( cmd == 8)//ping
{

// establish variables for duration of the ping,
long duration;

// The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
// Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
pinMode(pin, OUTPUT);
digitalWrite(pin, LOW);
delayMicroseconds(2);
digitalWrite(pin, HIGH);
delayMicroseconds(5);
digitalWrite(pin, LOW);

// The same pin is used to read the signal from the PING))): a HIGH
// pulse whose duration is the time (in microseconds) from the sending
// of the ping to the reception of its echo off of an object.
pinMode(pin, INPUT);
duration = pulseIn(pin, HIGH);
Serial.print(pin);
Serial.print(",");
Serial.print(duration);
Serial.print("*");
}
```

DelServo

Command = 9 , Pin Range = 0-13, Argument = Null, Response = Null

Servos must be detached for the pin to be reused, when all are detached pin 9 and 10
are reenabled for PWM

```
if ( cmd == 9)//detach servo
{
myservos[pin].detach();
}
```

SetSpeed

Command = 10 , Pin Range = Null, Argument = RPM, Response = Null

This uses the stepper library and sets the rotational speed of the stepper motor. The Sketch contains starting
parameters for the stepper 100,8,9 that is 100 steps is one complete revolution. This has to be the correct step value for the RPM value to work correctly

```
      if ( cmd == 10)//set the stepper speed
      {
        myStepper.setSpeed(val);
      }
```

SetStepR

Command = 11 , Pin Range = Null, Argument = N Steps, Response = 0,99 when complete

This sets the stepper motor in motion and it will complete N steps to the right and then respond. The stepper control is blocking, the Sketch will handle nothing else till it is complete. So use high seed and short movements to keep things flexible.

```
 if (cmd == 11)//set the stepper in motion +
      // this is a blocking call, the stepper will run at
      // the setSpeed value till steps are complete
      {
        myStepper.step(val);
        Serial.print("99,0,*");

      }
```

SetStepL

Command = 12 , Pin Range = Null, Argument = N Steps, Response = 0,99 when complete

As for SetStepR only the motor moves in the opposite direction.

What is the Arduino? | Start simply | The Sketch | Our Liberty BASIC GUI | The message format | Current command set | Making the connection | Starting the conversation | Send Request | Get Response | The complete GUI | The command list