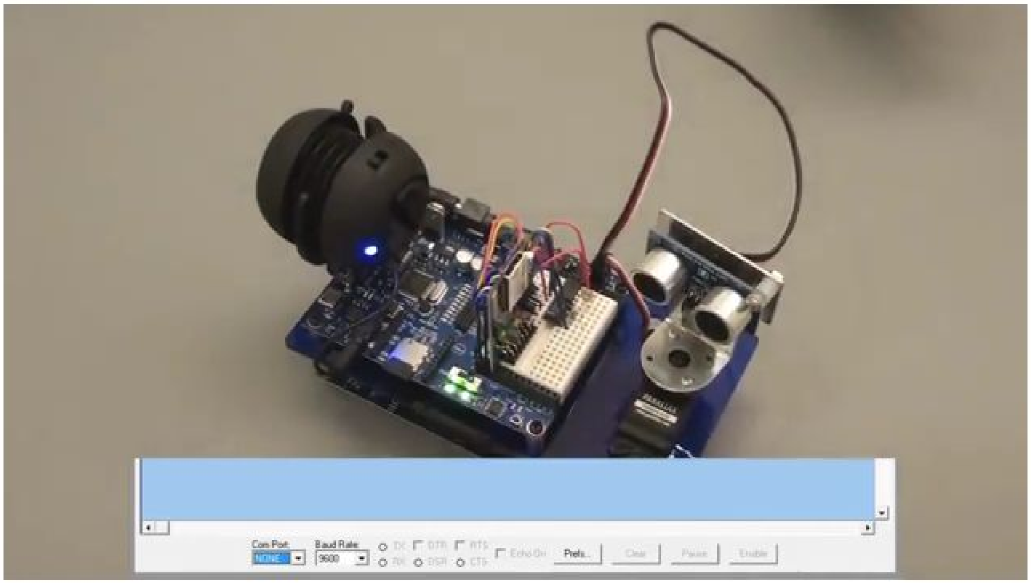
Web Site: www.parallax.com
Learn: learn.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267
EDU Hotline: (916) 625-6801

RN-42 Bluetooth to PC Demo

LEVEL: Intermediate
SKILLS REQUIRED: Spin Programming
HOURS TO COMPLETE: 2



View this project's YouTube video on the ParallaxInc YouTube channel.

The RN-42 Bluetooth to PC Demo provides an example of communication between the RN-42 Bluetooth Module and another device capable of supporting Bluetooth SPP, such as a PC or SmartPhone.

The demo provides a two-way communication channel for a remote device which has two functions. One is to send system telemetry from on-board sensors. The other is to receive and act upon known commands from a remote serial terminal or command interface, including a PC or SmartPhone application.

While this demo application is small, it represents a scalable real-world application where you might be receiving weather or sensor telemetry from a farm or industrial control system and need to be able to send commands to such a system, perhaps to open or close a valve in response to data received.

What's Required:

- (1) Propeller Board of Education (#32900, discontinued)

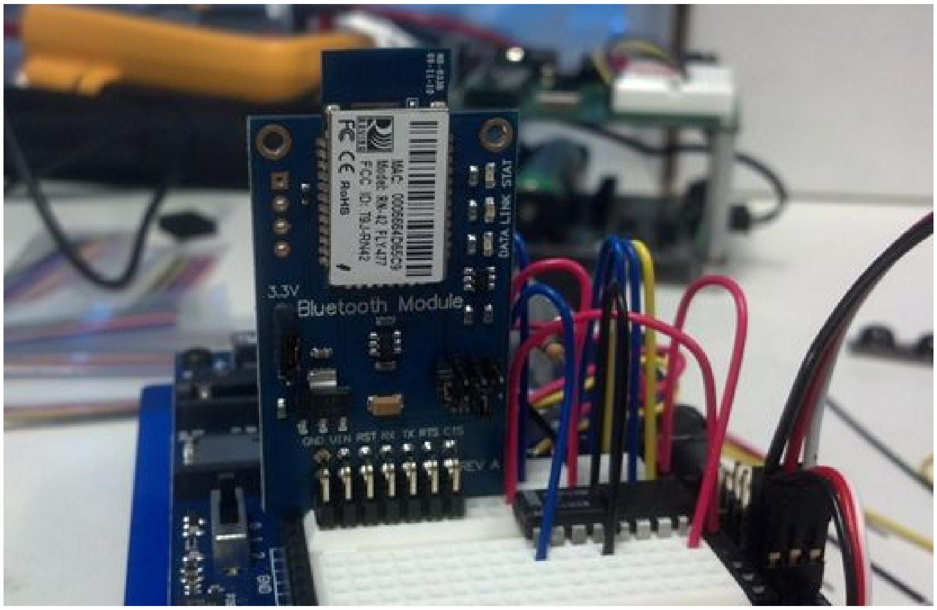
- (1) RN-42 Bluetooth Module (#30086)
- (1) ADC0831 8-bit A/D Converter (#ADC0831)
- (1) DS1620 Digital Thermometer (#604-00002, discontinued)
- (1) PING))) Ultrasonic Sensor + Mounting Bracket (#910-28015A)
- (1) Photoresistor (#350-00009, discontinued)
- (1) 10K ohm, ¼ W Resistor (#150-01030)
- (1) 4.7K ohm, ¼ W Resistor (#150-04720)
- (1) Li-ion Power Pack Charger (#28986, optional)
- (2) High-Capacity Li-ion Cell (#28987, optional)
- (1) Veho 360 Speaker (#900-00018)
- (1) Veho Speaker Stand for Propeller BOE (#725-32900, discontinued)
- (1) 2 GB microSD Card (#32319)
- Connecting wires
- USB Adapter for connecting microSD card to PC and load WAV files
- Mounting plate or chassis for PING))) Mounting Bracket and PCB

Note: Many of these parts are discontinued for sale or manufacture by Parallax, but may be found through other retailers, or have suitable replacements available through Parallax or other retailers.

Assemble the Bluetooth Unit

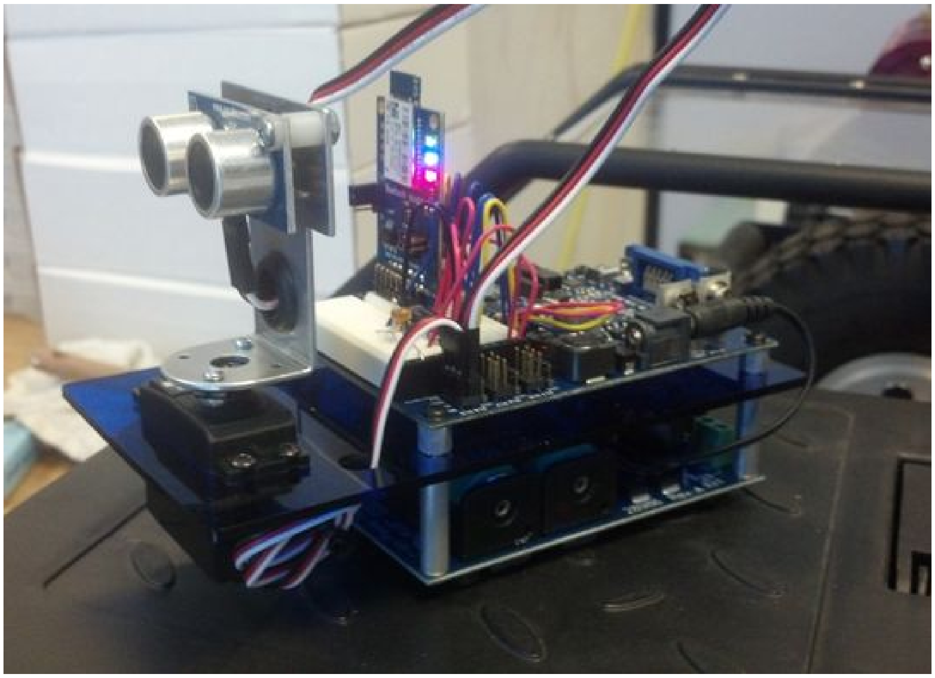
- ✓ Open the project schematic.
- ✓ As you add each component to the Propeller BOE you should run the associated test code for that device. For example, once the DS1620 is connected run the Test_DS1620.spin code.
- ✓ Once the ADC and photoresistor are connected run the Test_ADC0831.spin code.

The Parallax Serial Terminal will display the results @ 9600 bps connected to the COM Port used to program the Propeller BOE. There are test programs for all subsystems connected to the Propeller BOE.

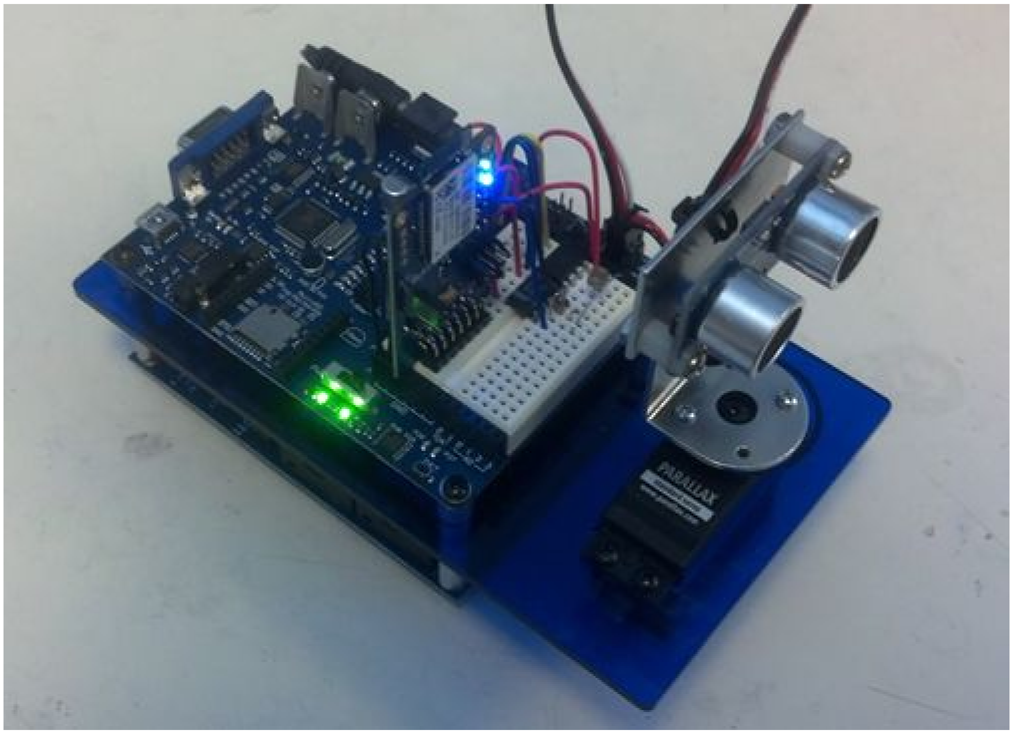


- ✓ Plug the RN-42 Bluetooth Module, ADC0831, and DS1620 as shown in the photo above. Use the provided schematic for reference.
- ✓ It’s a good idea to wire these devices before connecting the PING))) Mounting Bracket, which, depending on how you mount everything might make it difficult to wire up.
- ✓ Be sure to set the voltage jumper on the RN-42 Bluetooth Module to 3.3V and not 5V.
- ✓ Only configuration jumper 4 (default baud rate) is shorted forcing the baud rate to 9600 bps. The connections are outlined in the demo code in the constants section.

In our case, a custom base was laser cut for the project. The Propeller BOE and the PING))) Mounting Bracket are both attached to this base. They could also have been mounted to a BOE-Bot Chassis or home-made frame/chassis. The Li-ion Power Pack is optional. You can power the demo from a wall-adapter if you prefer.

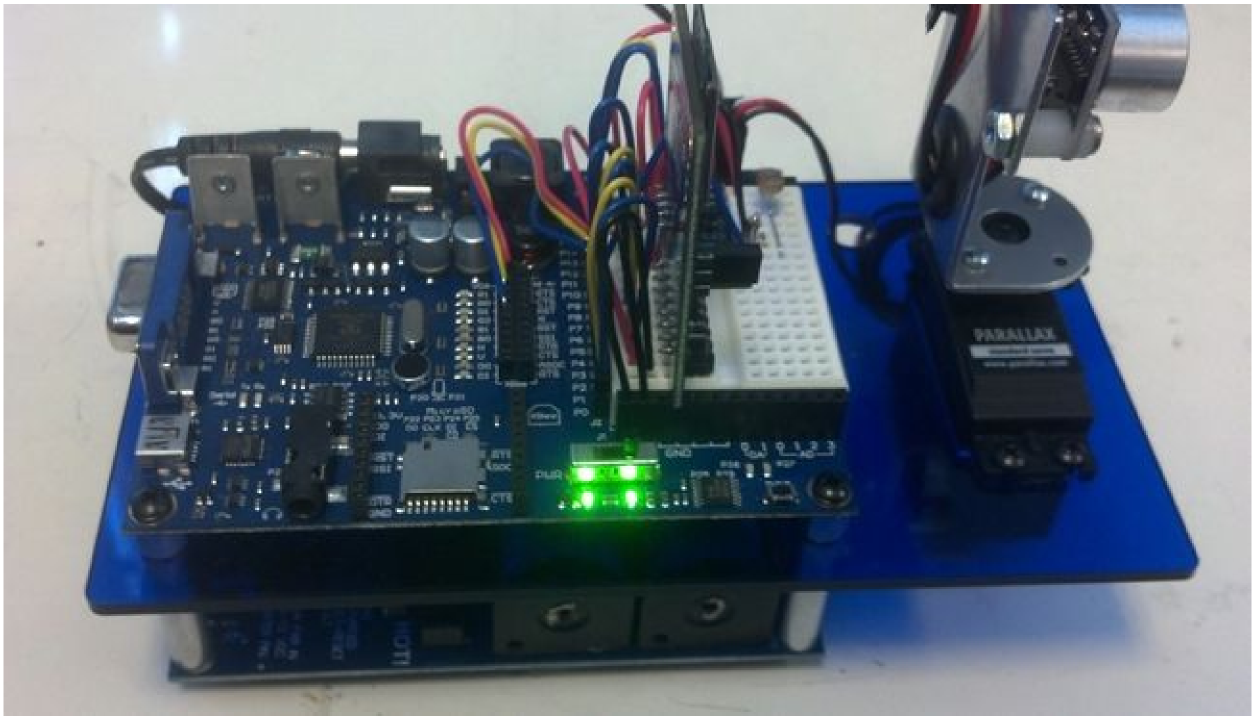


- ✓ The PING))) Sensor connects to the servo header P18 while the servo connects to P19.
- ✓ Be sure the voltage selection jumper for these ports are set to 5V and not VIN.
- ✓ Also be sure when the servo turns that the bracket doesn't hit anything on the board.



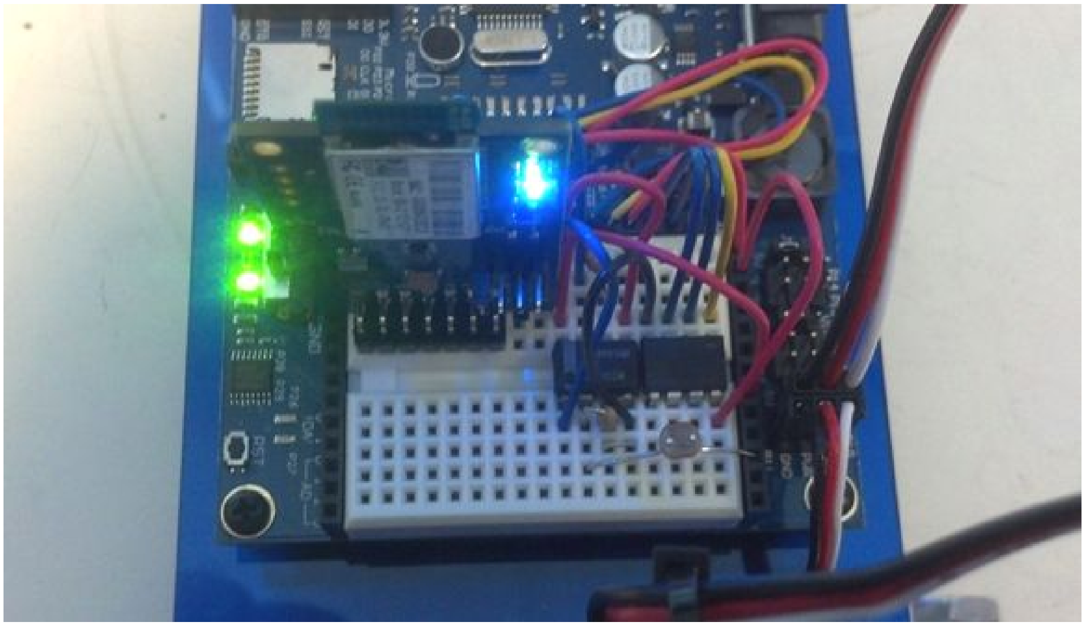
- ✓ P12 connects to R1 on the VGA socket, P11 connects to G1 and P10 connects to B1.

In this manner you have control of three different color LEDs which can be selectively turned on/off.

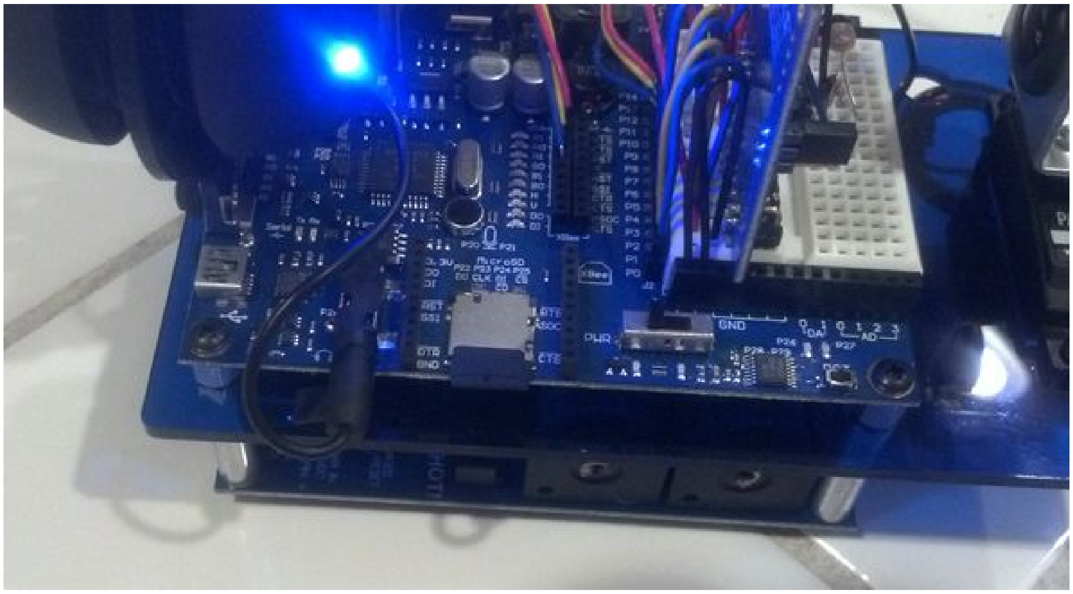


The photoresistor is connected to a 10K resistor forming a voltage divider which is fed into the input of the ADC0831.

- ✓ See the schematic at the beginning of the demo SPIN code for the connections to the ADC0831 and the DS1620.



- ✓ You will need a USB adapter or some other means to download the WAV files onto the microSD card.
- ✓ Once the files are on the card, install it into the microSD card socket on the Propeller BOE.
- ✓ After installing the Veho Speaker Stand and Speaker, plug the 1/8" phone plug into the audio jack on the Propeller BOE.
- ✓ Be sure the card is plugged in all the way and that the Veho Speaker is turned on (blue LED will light up).



Communicating with the Demo Unit

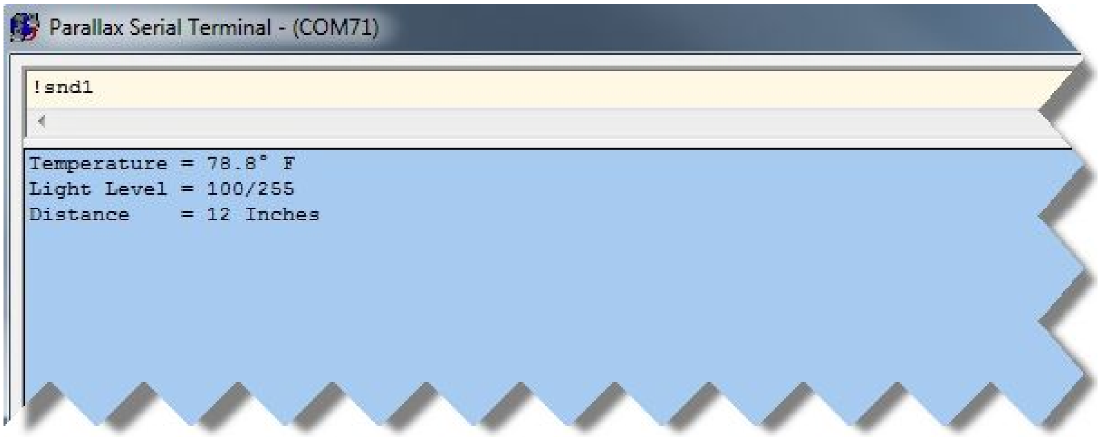
Communication with the demo unit is done by connecting via Bluetooth from your PC.

- ✓ Before this can be done you will have to pair the RN-42 Bluetooth Module with your PC.

How this is done varies with your Bluetooth driver and operating system and goes beyond the scope of this project.

- ✓ Load the Bluetooth PC_Prop-BoE Demo V1.0 program to your Propeller BoE.
- ✓ Once connected to Bluetooth you can open the Parallax Serial Terminal (or any terminal) to the Bluetooth COM Port created by your O/S Bluetooth driver. The port speed should be set to 9600 bps. Be sure the Propeller BOE power switch is in the second position (2) so the PING))) Sensor and servo are powered up.

Once connected the blue LINK LED on the RN-42 Bluetooth Module will light up. At this point you should be getting telemetry from the demo unit as shown in the example screenshot. Temperature, light level and distance as determined by the PING))) Sensor.



```
!snd1
```

```
Temperature = 78.8° F
Light Level = 100/255
Distance    = 12 Inches
```

You can also send commands to the demo unit. The demo unit understands the following commands sent from the terminal:

- !clr - clear the terminal screen, and refresh the telemetry data
- !svo1 – move the servo to position one (90 degrees right)
- !svo2 – move the servo to position two (center position)
- !svo3 – move the servo to position three (90 degrees left)
- !snd# - where # is a number from 0-9 corresponding to a WAV file to play
- !inr0 – turn Red LED off
- !inr1 – turn Red LED on
- !ing0 – turn Green LED off
- !ing1 – turn Green LED on
- !inb0 – turn Blue LED off
- !inb1 – turn Blue LED on

Each command starts with an exclamation point. The next three letters are the command. Three letters were chosen to keep things simple for the parser code. Some commands have a parameter that follows. Once the full command is understood it is executed. No carriage return and/or linefeed are required. All commands are lowercase.

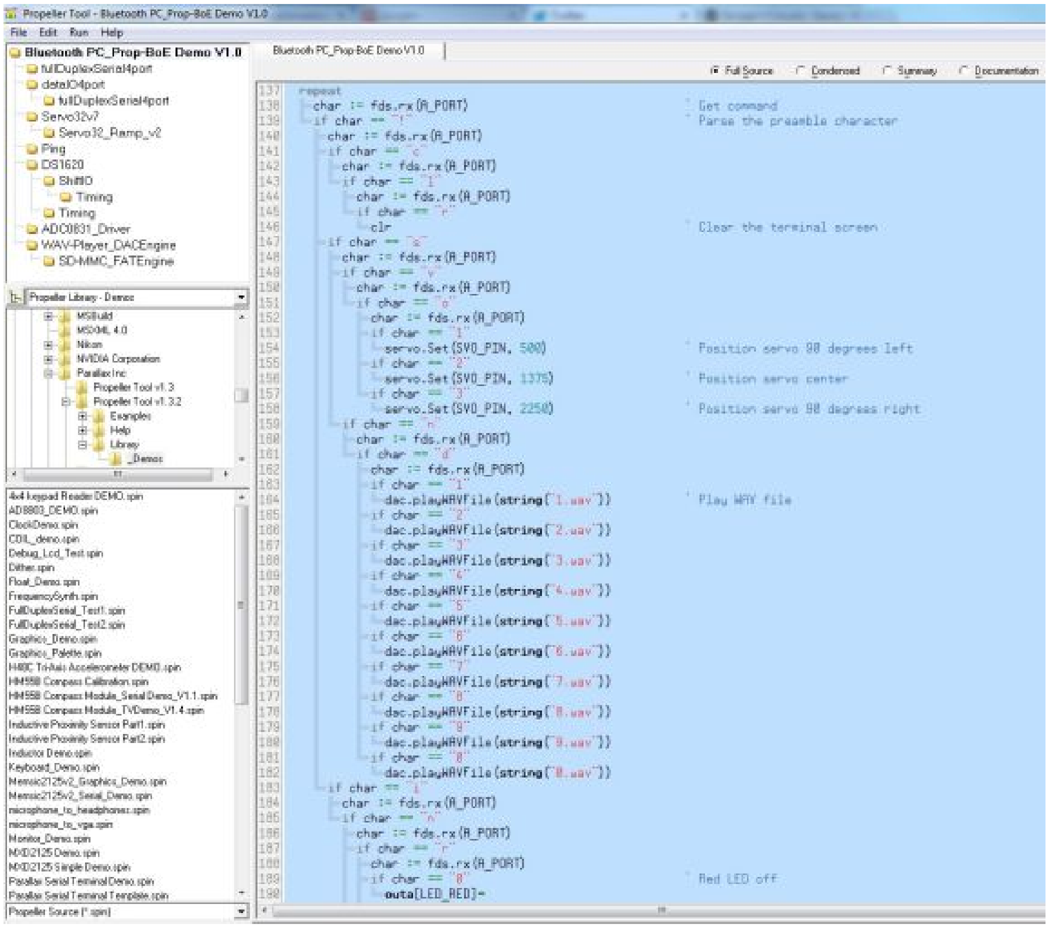
The only command that would affect the telemetry is the "svo" command, since the direction the PING))) Sensor faces may change the distance seen. Watch the video on the main page of this project for an example of how everything works. The video shows all commands as well as disconnecting and reconnecting.

The Details

All the objects used for the various functions were obtained from the Propeller Library (built into the IDE), the OBEX (Object Exchange) or the Parallax Forums. Everything works together with very little work to balance out resources. All required objects are included in the code ZIP file. The telemetry code is launched into a new cog. Its function is to read the sensors and send the data, wait one second and repeat. It runs completely independent of the other functions. Each of the three telemetry sections calls a method in the appropriate object, setting a variable and then formatting that data out to the terminal.

The command parser evaluates incoming commands and executes them when a complete command is detected. It does this by first evaluating every character starting with the “!” character. Once this character is received the next character is checked against possible matches. If at any point the next character is not a match the whole evaluation resets. In the parser section the various levels are indented in such a way that commands are evaluated on through until completion or until a wrong character.

For example, once the “!” is received the next character is compared against “c”, “s” or “i”. If “s” is received the next character is compared against “v” or “n”. If “v” is received the next character is compared against “o”. If it does not match the parser won’t evaluate again until “!” is received again. This level-nested evaluation is an easy way to parse commands, even if they have similar characters at the beginning as the “svo” and “snd” commands do.



```
repeat
  char := fds.rx(A_PORT)                              ' Get command
  if char == "!"                                      ' Parse the preamble character
    char := fds.rx(A_PORT)
    if char == "c"
      char := fds.rx(A_PORT)
      if char == "l"
        char := fds.rx(A_PORT)
        if char == "r"
          clr                                         ' Clear the terminal screen
    if char == "s"
      char := fds.rx(A_PORT)
      if char == "v"
        char := fds.rx(A_PORT)
        if char == "o"
          char := fds.rx(A_PORT)
          if char == "1"
            servo.Set(SVO_PIN, 500)                   ' Position servo 90 degrees left
          if char == "2"
            servo.Set(SVO_PIN, 1375)                  ' Position servo center
          if char == "3"
            servo.Set(SVO_PIN, 2250)                  ' Position servo 90 degrees right
      if char == "n"
        char := fds.rx(A_PORT)
        if char == "d"
          char := fds.rx(A_PORT)
          if char == "1"
            dac.playWAVFile(string("1.wav"))          ' Play WAV file
          if char == "2"
            dac.playWAVFile(string("2.wav"))
          if char == "3"
            dac.playWAVFile(string("3.wav"))
          if char == "4"
            dac.playWAVFile(string("4.wav"))
          if char == "5"
            dac.playWAVFile(string("5.wav"))
          if char == "6"
            dac.playWAVFile(string("6.wav"))
          if char == "7"
            dac.playWAVFile(string("7.wav"))
          if char == "8"
            dac.playWAVFile(string("8.wav"))
          if char == "9"
            dac.playWAVFile(string("9.wav"))
          if char == "0"
            dac.playWAVFile(string("0.wav"))
    if char == "i"
      char := fds.rx(A_PORT)
      if char == "r"
        char := fds.rx(A_PORT)
        if char == "r"
          char := fds.rx(A_PORT)
          if char == "0"                              ' Red LED off
            outa[LED_RED]~
```

When there is no connection via Bluetooth the telemetry won't be sent due to flow control. Likewise, since no commands can be received everything remains as it was until a connection is established and valid commands received.

For more information about the RN-42 Bluetooth Module please go to Parallax.com and search "30086".

Did You Know?

This project is made up by adding code from more than 7 different objects and modified portions of the demos to get the desired results. To make things easier, constants for each object were placed in multiple `CON` sections, keeping them isolated. The start methods were placed as needed and initialization code was placed at the beginning to ensure everything started up properly. Despite of the apparent complexity of having over 7 objects running in the demo code, there are only 3 `PUB` and one `PRI` method, and two of these contain only one line of code.

Try This

- ✓ Change the commands used to activate various functions.

You can do this by studying the structure of the parser. Indentation is important to how things are executed (and evaluated) in SPIN, so watch the level of indentation of each sub-section of the parser evaluations.

- ✓ You could also add commands, or add/change the WAV files loaded on the microSD card.

The Test_microSD.spin file makes use of a command to detect when a WAV file has stopped playing. You could use this to enhance the "snd" commands in this demo.