RasWIK - The Raspberry Pi Wireless Inventors Kit Part 2

Category: Radio projects using XRF, ERF, URF & SRF's
Last Updated on Monday, 12 August 2013 08:13

Raspberry Pi Wireless Inventors Kit - Part 2

The Raspberry Pi Wireless Inventors Kit (RasWIK) is an **exciting and affordable** way to build your own wireless devices with the Raspberry Pi. The Wireless Inventors Kit is a low cost and easy entry into everything wireless.

In Part 1 of this guide you will have set up the kit, learned about digital and analog input and output, and got an understanding of LLAP, the message protocol we use between the Raspberry Pi and the wireless Arduino Uno based XinoRF controller. If you have not gone through Part 1, please do so before going further with this Part 2.

In this part of the guide we are going to take you through the steps that will allow you to write your own Python programs to remotely control the XinoRF. That will enable you to really let your imagination go and build almost anything you want, without the safe learning environment provided by the RasWIK application.

Table of contents

Python programming and real world examples

Activity 18 - An introduction to LLAP (the wireless messages)

<< back to contents page

What will we learn?

You will learn the meaning of the messages that are sent between the WIK Basic interface and the XinoRF. They keep appearing in the two white boxes at the bottom of WIK's basic interface and we have ignored them so far. Let's take a look.

How long will this take?

It takes as long as it takes to read this text and play with some of the previous examples. You can be done in 10 minutes if you want.

How difficult is this project?

Very easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

Let's start!

Make sure your Pi is installed and running as detailed in activity 01.

The WIK Basic Interface has two panels at the bottom, both white, with a grey bar over the top with a few buttons. In the left hand white panel WIK tells you what is going on on the serial connection in "plain English". In the right hand panel, you can see the actual messages that are exchanged between the WIK Basic interface on your Raspberry Pi and the XinoRF. Now press the reset button on the XinoRF and observe that a message string:

    a--STARTED--

is received on your Raspberry Pi. Received messages are shown in blue. WIK explains that:

    Received LLAP from -- with DATA: STARTED

Now press the send button on the grey bar over the two white message monitors. Observe that WIK explains that

    Sending LLAP to -- with DATA: HELLO

and the actual message in the right hand pane:

    a--HELLO----

Sent messages are shown in red, so they are easy to distinguish from received ones, shown in blue. You will see that the XinoRF has responded with a similar message:

    a--HELLO----

and that WIK explains:

    Received LLAP from -- with DATA: HELLO

This round trip message exchange is called a 'handshake' and it lets both sides know that they are alive and talking.

Now turn to the Basic's tab, as in Activity 1.Let's turn on the LED on D13 by pressing the 'HIGH' button. Observe the messages being sent and received:

    a--D13HIGH--

This message is sent to the XinoRF, and received back in acknowledgement that D13 is high and the LED has been lit. The message

    a--D13LOW---

is exchanged when we press the D13 LOW button in the interface.

All the activities that we have been doing earlier use a similar simple message exchange to make the XinoRF do what we want, or to read values from its inputs.

All LLAP messages are exactly 12 characters long and start with a lower case letter 'a'. The next two characters tell us which device we are talking to. The XinoRF in your kit comes up with the default address '--'. The data in the message is always 9 characters long. If we have something shorter to send, we simply pad the message out with dashes.

You can go over any of the previous examples and explore what messages are being sent back and forth. The messages have all been designed to be easy to read for humans.

Sending LLAP messages is as easy as sending a text or a tweet!

Activity 19 - Sending a LLAP wireless message

<< back to contents page

What will we learn?

In this activity we will show you what it takes to send a LLAP wireless message using Python and walk you through the code so can you can have a go yourself.

How long will this take?

No more than 10 minutes

How difficult is this project?

Very easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

Let's walk through the code below and see how it works. The very first thing we do is import the libraries we need (line 17): one to drive the serial port, the other to allow us to wait for a short time (sleep).

Next we set up the serial port itself (line 20 to 25). It needs to know which device to address and at what speed. Slice of Radio's serial port has been configured on /dev/ttyAMA0 and its speed is 9600 baud, so we define two variables with these values (line 21 & 22). The values port and baud are then used to open the the serial connection to the radio. It is good practice to wait a very short while and let the serial port settle down before doing anything with it (line 28).

Now we can write to the port by simply sending it a string of ASCII characters (line 33). In order for us to communicate correctly with the XinoRF, we need to send a correctly formatted LLAP message in that text string. In this case we address device '--' and tell it to pull D13 high:

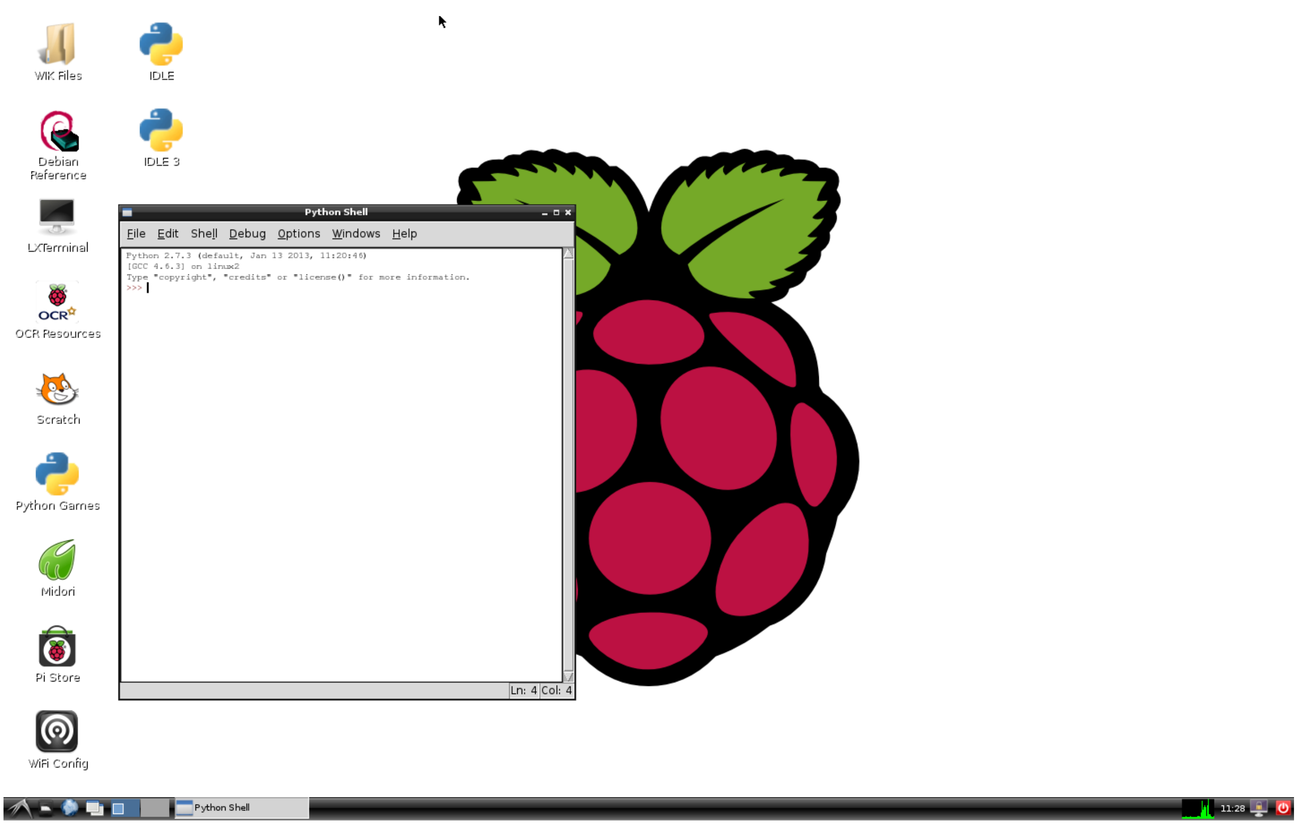
    a--D13HIGH--

Since this is all we do in this simple example, we can then close the serial connection and exit the script (line 39).

```
01. #!/usr/bin/env python
02. # -*- coding: utf-8 -*-
03. """ Wireless Inventors Kit Python Example 01Send.py
04.     Ciseco Ltd. Copyright 2013
05.
06.     This basic Python example just open a serial port and send one LLAP message
07.
08.
09.     Author: Matt Lloyd
10.
11.     This code is distributed in the hope that it will be useful,
12.     but WITHOUT ANY WARRANTY; without even the implied warranty of
13.     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
14.
15. """
16. #import the PySerial library and sleep from the time library
17. import serial
18. from time import sleep
19.
20. # declare to variables, holding the com port we wish to talk to and the speed
21. port = '/dev/ttyAMA0'
22. baud = 9600
23.
24. # open a serial connection using the variables above
25. ser = serial.Serial(port=port, baudrate=baud)
26.
27. # wait for a moment before doing anything else
28. sleep(0.2)
29.
30. # write a--D13HIGH-- out to the serial port
31. # this should turn the XinoRF LED on
32. # changing this to a--D13LOW--- will turn the LED off
33. ser.write('a--D13HIGH--')
34.
35. # wait for a moment before doing anything else
36. sleep(0.2)
37.
38. # close the serial port
39. ser.close
40.
41. # at the end of the script python automatically exits
```

To run the code above yourself, open IDLE (there should be a link on the desktop). You will see a Python console as shown in the screen shot below:

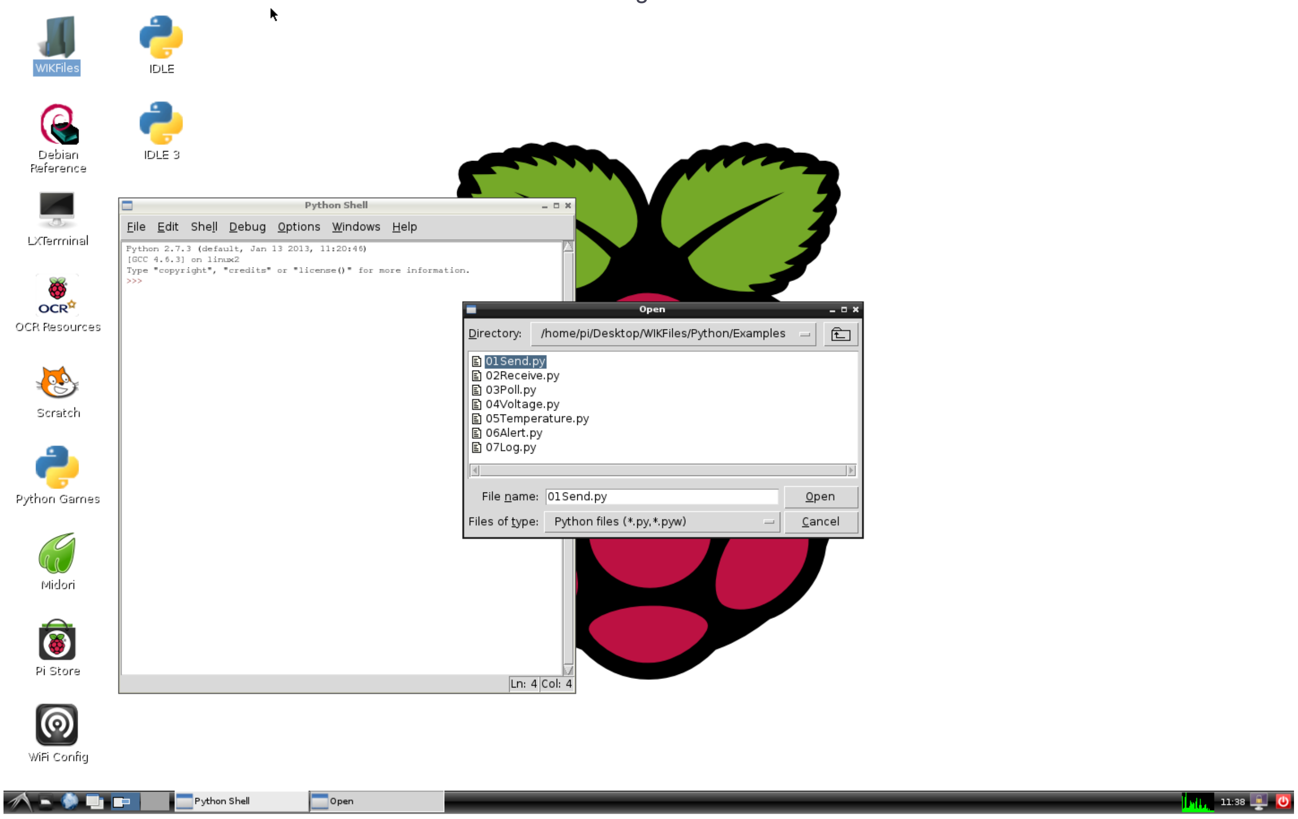
Image



```
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>>
```

To open a copy of the code click File->Open and browse to 'WIK Files/Python/Examples' folder. Then open 01Send.py

Image



```
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>>
```

Directory: /home/pi/Desktop/WIKFiles/Python/Examples

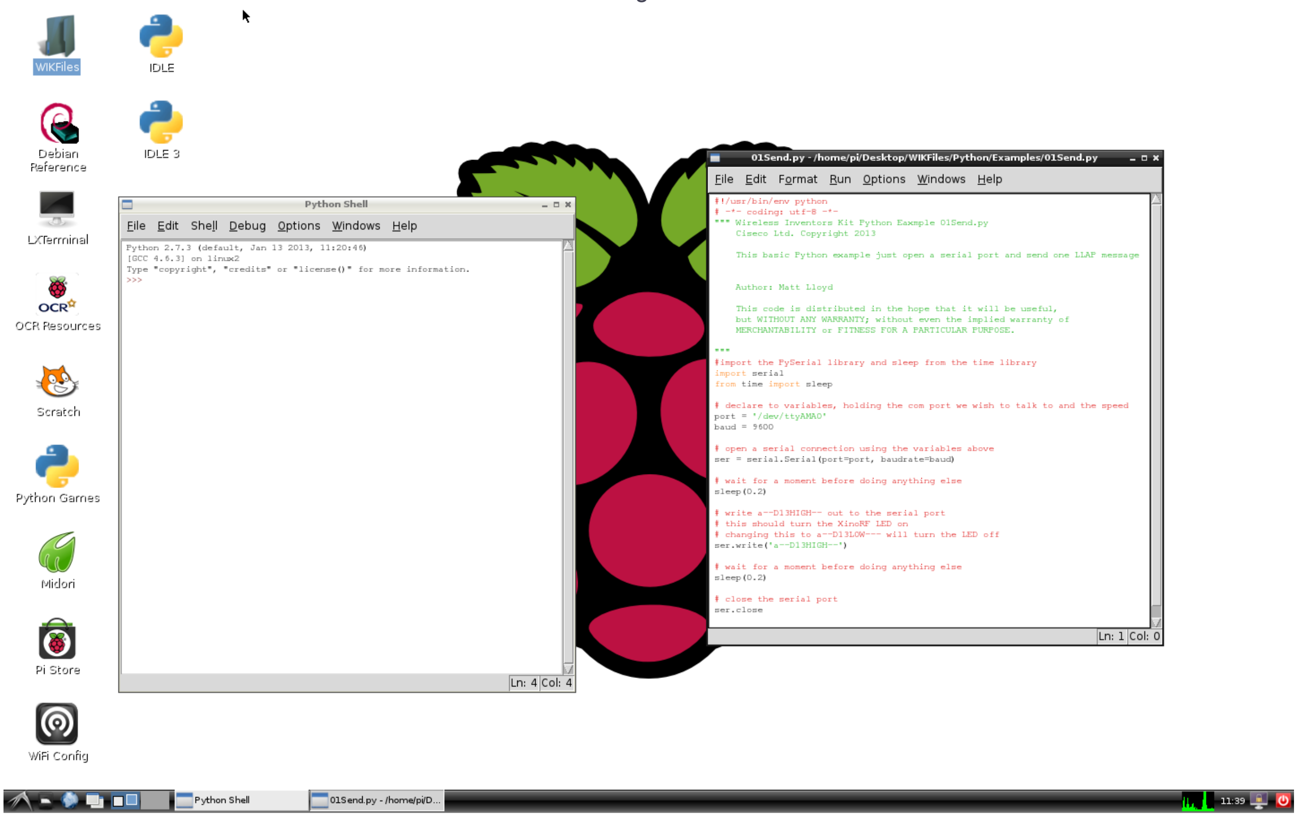
01Send.py
02Receive.py
03Poll.py
04Voltage.py
05Temperature.py
06Alert.py
07Log.py

File name: 01Send.py

Files of type: Python files (*.py,*.pyw)

This will open the file in a new window as shown below

Image

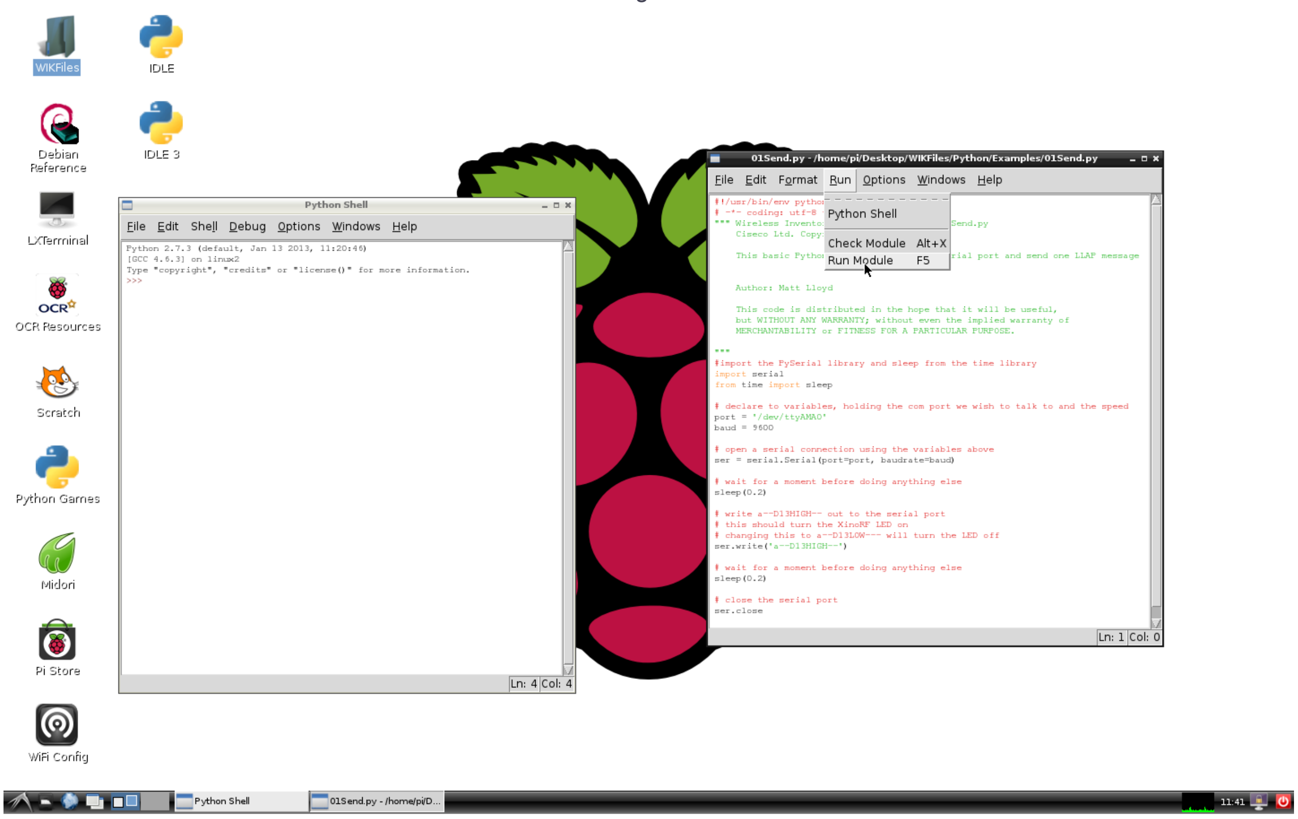

WIKFiles
IDLE
Debian Reference
IDLE 3
LXTerminal
OCR Resources
Scratch
Python Games
Midori
Pi Store
WiFi Config
Python Shell
File Edit Shell Debug Options Windows Help
01Send.py - /home/pi/Desktop/WIKFiles/Python/Examples/01Send.py
File Edit Format Run Options Windows Help
Ln: 4 Col: 4
Ln: 1 Col: 0
Python Shell
01Send.py - /home/pi/D...
11:39

```
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" Wireless Inventors Kit Python Eaxmple 01Send.py
    Ciseco Ltd. Copyright 2013

    This basic Python example just open a serial port and send one LLAP message


    Author: Matt Lloyd

    This code is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

"""
#import the PySerial library and sleep from the time library
import serial
from time import sleep

# declare to variables, holding the com port we wish to talk to and the speed
port = '/dev/ttyAMA0'
baud = 9600

# open a serial connection using the variables above
ser = serial.Serial(port=port, baudrate=baud)

# wait for a moment before doing anything else
sleep(0.2)

# write a--D13HIGH-- out to the serial port
# this should turn the XinoRF LED on
# changing this to a--D13LOW--- will turn the LED off
ser.write('a--D13HIGH--')

# wait for a moment before doing anything else
sleep(0.2)

# close the serial port
ser.close
```

To run the example click Run->Run Module

Image



The first time you run this example, you will see the the LED connected to D13 light up. On each subsequent run of the program you won't be able to observe anything as we just send it another message to pull D13 high and switch the LED on, when it is already on anyway. Now press the XinoRF reset button and see the LED go off. Then when you run the python script again, it comes back on. If you edit line 33 and change the 'a--D13HIGH--' to 'a--D13LOW---' and run the script again it will turn off the LED.

Activity 20- Receiving a LLAP wireless message

[<< back to contents page]

What will we learn?

In this activity we explain what it takes to receive LLAP wireless messages using Python and walk you through the code that you can run yourself.

How long will this take?

No more than 10 minutes

How difficult is this project?

Very easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

The code for this example is very similar to the code we walked through for sending LLAP (Activity 20). We need to import the same libraries and set up the serial port in exactly the same way. We have kept the code that sends the message to the XinoRF, telling it to pull D13 high.

When the XinoRF receives this message, it carries out the command and then sends back a message. By simply reading this message from the serial port (line 40) and then printing it, we can see what it sends back. As LLAP messages are 12 characters long, we can read that into the variable called 'reply' and then print this to the Python console (line 43).

```
01. #!/usr/bin/env python
02. # -*- coding: utf-8 -*-
03. """ Wireless Inventors Kit Python Example 02Receive.py
04.     Ciseco Ltd. Copyright 2013
05. 
06.     This basic Python example just open a serial port and send one LLAP message
07.     and receive a reply and prints it to the console
08. 
09. 
10.     Author: Matt Lloyd
11. 
12.     This code is distributed in the hope that it will be useful,
13.     but WITHOUT ANY WARRANTY; without even the implied warranty of
14.     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15. 
16. """
17. #import the PySerial library and sleep from the time library
18. import serial
19. from time import sleep
20. 
21. # declare to variables, holding the com port we wish to talk to and the speed
22. port = '/dev/ttyAMA0'
23. baud = 9600
24. 
25. # open a serial connection using the variables above
26. ser = serial.Serial(port=port, baudrate=baud)
27. 
28. # wait for a moment before doing anything else
29. sleep(0.2)
30. 
31. # write a--D13HIGH-- out to the serial port
32. # this should turn the XinoRF LED on
33. # changing this to a--D13LOW--- will turn the LED off
34. ser.write('a--D13HIGH--')
35. 
36. # wait for a moment before doing anything else
37. sleep(0.2)
38. 
39. # read 12 characters from the serial port
40. reply = ser.read(12)
41. 
42. # print the replay
43. print(reply)
44. 
45. # close the serial port
46. ser.close
47. 
48. # at the end of the script python automatically exits
```

As in activity 18 you can use IDLE to open the 02Receive.py file from the Examples folder.
The first time you run this example, you may see the message

    a--STARTED--

printed out, instead of the response to your command:

    a--D13HIGH--

This is because Linux may buffer messages incoming on /dev/ttyAMA0. The first 12 bytes it reads after the XinoRF has been reset is the startup message. The message we expect to read after having set D13 high are in the next 12 bytes.

Activity 21- Periodically polling devices

<< back to contents page

What will we learn?

In this activity we will send repeated requests out and receive responses. You'll have the opportunity to see the code and run it yourself, as well as make some modifications.

How long will this take?

No more than 10 minutes to get going, then any length of time you wish experimenting.

How difficult is this project?

Very easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

In this example, we periodically send a message to the XinoRF and receive its response. We call the process of repeatedly asking for a response 'polling'.

You will see that we use the same libraries as before and we set up the serial port in exactly the same way.

To create a loop that polls multiple times we use a counter and the 'while' statement (line 31 & 35). At the end of each loop we increase the counter by one (line 61). The code used for each loop is indented by four spaces or one TAB (line 36 to 61).

In the loop of the program, we ask the XinoRF to read analog input 0 (A0 line 38) and print out the value we receive. If A0 is not connected to anything the readings will vary quite a bit as the input "floats".
Instead of printing the raw LLAP message 'a--A0+532---' we do some formatting to present the ADC value in a nice readable manner. First we split off the last 5 characters of the message (line 50). Next we remove any trailing '-' (line 53). We format the final string using the '.format()' function before printing to the console (line 58)

```
01. #!/usr/bin/env python# -*- coding: utf-8 -*-
02. """ Wireless Inventors Kit Python Example 03Poll.py
03.     Ciseco Ltd. Copyright 2013
04.
05.     Polled (or repeated) send and receive of LLAP messages
06.
07.
08.     Author: Matt Lloyd
09.
10.     This code is distributed in the hope that it will be useful,
11.     but WITHOUT ANY WARRANTY; without even the implied warranty of
12.     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
13.
14. """
15. #import the PySerial library and sleep from the time library
16. import serial
17. from time import sleep
18.
19. # declare to variables, holding the com port we wish to talk to and the speed
20. port = '/dev/ttyAMA0'
21. baud = 9600
22.
23. # open a serial connection using the variables above
24. ser = serial.Serial(port=port, baudrate=baud)
25.
26. # wait for a moment before doing anything else
27. sleep(0.2)
28.
29. # setup a counter starting at 0
30. count = 0
31.
32. # loop over the block of code while the count is less than 4
33. # when the count = 4 the loop will break and we carry on with the rest
34. while count < 4:
35.     # write a--A00READ-- out to the serial port
36.     # this will return the current ADC reading for Pin A0
37.     ser.write('a--A00READ--')
38.
39.     # wait for a moment before doing anything else
40.     sleep(0.2)
41.
42.     # read 12 characters from the serial port
43.     reply = ser.read(12)
44.
45.     # at this point reply should contain something like 'a--A01+532--'
46.     # the numbers after the + are the ADC reading we interested in
47.
48.     # take just the last part of the message
49.     value = reply[7:]
50.
51.     # strip the trailing '-'
52.     value = value.strip('-')
53.
54.     # print the ADC Value
55.     # here we are doing a little formatting of the output
56.     # the {} inside the quotes is replaced with the contents of value
57.     print("ADC: {}".format(value))
58.
```

```
59.     # increase the count by 1 at the end of the block
60.     count += 1
61.
62. # close the serial port
63. ser.close()
64.
65. # at the end of the script python automatically exits
```

As in activity 18 you can use IDLE to open the 03Poll.py file from the Examples folder. Running this should give an output similar to the one below.

Image

You can modify the code to poll at greater intervals, by adding a sleep statement in the loop. You can also increase the number of polls by changing the while condition on line 35

Connect something to the analog input you are reading, such as a thermistor or light dependent resistor (LDR) as we did in some of the activities earlier, so you get some real readings.

Activity 22 - Converting ADC to Voltage

[<< back to contents page]

What will we learn?

As in the previous activity we will send repeated requests out and receive responses to read an analogue value. Once received, we will convert that value into a voltage. You'll have the opportunity to see the code and run it yourself, as well as make some modifications.

How long will this take?

No more than 10 minutes

How difficult is this project?

Easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

1 x 10K potentiometer

1 x Breadboard

3 x Jumper wire

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

In this example, we follow the pattern from the previous example, where we periodically poll for an analogue value. This time we convert the value we read into an actual voltage read.

First wire up the XinoRF as described in Activity 8 in Part 1 of this guide. This will allow you to vary the voltage on pin A0 of the XinoRF.

You will recognise the code upto line 51 as being the same as in the previous example. Instead of just printing the raw ADC value, we want to convert it, and to do so, we need to convert the string value into an integer, so we do maths with it (line 55). The maths is quite simple: for 5V we get an ADC value of 1023, for 0V we get a value of 0. So if we get a value of 512, we'd have 2.5V. Simply dividing adc by 1023 and multiplying by 5 thus gives us the measured voltage (line 58). At line 66, we print both the ADC value and the voltage we calculated, after doing appropriate formatting.

```
01. #!/usr/bin/env python# -*- coding: utf-8 -*-
02. """ Wireless Inventors Kit Python Example 04Voltage.py
03.     Ciseco Ltd. Copyright 2013
04. 
05.     Conversion of a ADC Value to Voltage
06. 
07.     Author: Matt Lloyd
08. 
09.     This code is distributed in the hope that it will be useful,
10.     but WITHOUT ANY WARRANTY; without even the implied warranty of
11.     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
12. 
13. """
14. #import the PySerial library and sleep from the time library
15. import serial
16. from time import sleep
17. 
18. # declare to variables, holding the com port we wish to talk to and the speed
19. port = '/dev/ttyAMA0'
20. baud = 9600
21. 
22. # open a serial connection using the variables above
23. ser = serial.Serial(port=port, baudrate=baud)
24. 
25. # wait for a moment before doing anything else
26. sleep(0.2)
```

```
27.
28. # setup a counter starting at 0
29. count = 0
30.
31. # loop over the block of code while the count is less than 4
32. # when the count = 4 the loop will break and we carry on with the rest
33. while count < 4:
34.     # write a--A00READ-- out to the serial port
35.     # this will return the current ADC reading for Pin A0
36.     ser.write('a--A00READ--')
37.
38.     # wait for a moment before doing anything else
39.     sleep(0.2)
40.
41.     # read 12 characters from the serial port
42.     reply = ser.read(12)
43.
44.     # at this point reply should contain something like 'a--A01+532--'
45.     # the numbers after the + are the ADC reading we interested in
46.
47.     # take just the last part of the message
48.     adc = reply[7:]
49.
50.     # strip the trailing '-'
51.     adc = adc.strip('-')
52.
53.     # adc is currently a string, we need to convert this to an integer
54.     # before we can do any maths
55.     adc = int(adc)
56.
57.     # to calculate the voltage we use the following formula
58.     volts = (adc / 1023.0 * 5.0)
59.
60.     # print the ADC Value and the Volts value
61.     # here we are doing a little formatting of the output
62.     # the first {} inside the quotes is replaced with the contents of value
63.     # the second {} is replaced with our calculated Volts
64.     # the :0.2f inside the second {} limits the
65.     # floating point number to two decimal places
66.     print("ADC: {} Volts: {:0.2f}V".format(adc, volts))
67.
68.     # increase the count by 1 at the end of the block
69.     count += 1
70.
71. # close the serial port
72. ser.close()
73.
74. # at the end of the script python automatically exits
```

As in activity 18 you can use IDLE to open the 04Voltage.py file from the Examples folder. Running this should give an output similar to the one below.

image

You can modify the code to poll at greater intervals, by adding a sleep statement in the loop. You can also increase the number of polls by changing the while condition on line 35.

Activity 23- Converting ADC to Temperature

<< back to contents page

What will we learn?

As in the previous activities we will send repeated requests out and receive responses to read an analogue value. Once received, we will convert that value into a temperature. You'll have the opportunity to see the code and run it yourself, as well as make some modifications.

How long will this take?

No more than 10 minutes

How difficult is this project?

Easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

1 x 10K resistor

1 x Thermistor

1 x Breadboard

3 x Jumper wire

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

In this example, we follow the pattern from the previous examples, where we periodically poll for an analogue value. This time we convert the value we read into an actual temperature read.

First wire up the XinoRF as described in Activity 10 in Part 1 of this guide. The thermistor (temperature sensitive resistor) will vary the voltage on pin A0 of the XinoRF, and we will convert that reading to temperature.

You will recognise the code upto line 54 as being the same as in the previous example. The only difference is that we also import the math library so have access to routines that we need for the conversion to temperature. Instead of just printing the raw ADC value, we want to convert it, and to do so, we need to convert the string value into an integer, so we do maths with it (line 58).

The maths for temperature conversion is a bit more difficult than the one we have seen so far. The formula we use to determine the temperature in Kelvin is on line 75. If you want to find out more about it, have a look at http://en.wikipedia.org/wiki/Thermistor for a detailed explanation.

Before we can do the conversion, we need to define some constants, as done in lines 60 to 65. The code in line 67 to 69 is here to make sure that we never divide by 0, even if the ADC value we read is zero. To ensure we get an accurate reading of the temperature, we convert the string value adc to a float, which gives us a decimal point. We convert he temperature from kelvin to Celcius by subtracting 273.15, and then go through some formatting to print both the ADC value and the temperature we calculated.

```
01. #!/usr/bin/env python# -*- coding: utf-8 -*-
02. """ Wireless Inventors Kit Python Example 05Temperature.py
03.     Ciseco Ltd. Copyright 2013
04.
05.     Conversion of a ADC Value to Temperature
06.
07.
08.     Author: Matt Lloyd
09.
10.     This code is distributed in the hope that it will be useful,
11.     but WITHOUT ANY WARRANTY; without even the implied warranty of
12.     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
13.
14. """
15. #import the PySerial library and sleep from the time library
16. import serial
17. from time import sleep
18. # for temperature we also need the math library
19. import math
20.
21. # declare to variables, holding the com port we wish to talk to and the speed
22. port = '/dev/ttyAMA0'
23. baud = 9600
24.
25. # open a serial connection using the variables above
26. ser = serial.Serial(port=port, baudrate=baud)
27.
28. # wait for a moment before doing anything else
29. sleep(0.2)
30.
31. # setup a counter starting at 0
32. count = 0
33.
34. # loop over the block of code while the count is less than 4
35. # when the count = 4 the loop will break and we carry on with the rest
36. while count < 4:
37.     # write a--A00READ-- out to the serial port
38.     # this will return the current ADC reading for Pin A0
39.     ser.write('a--A00READ--')
40.
41.     # wait for a moment before doing anything else
42.     sleep(0.2)
43.
44.     # read 12 characters from the serial port
45.     reply = ser.read(12)
46.
47.     # at this point reply should contain something like 'a--A01+532--'
48.     # the numbers after the + are the ADC reading we interested in
49.
50.     # take just the last part of the message
51.     adc = reply[7:]
```

```
52.
53.     # strip the trailing '-'
54.     adc = adc.strip('-')
55.
56.     # adc is currently a string, we need to convert this to an integer
57.     # before we can do any maths
58.     adc = int(adc)
59.
60.     # to calculate the temperature we use a more complex formula
61.     # here we store some of the fixed numbers in variables
62.     BVAL = 3977              # default beta value for the thermistor
63.     RTEMP = 25.0 + 273.15    # reference temperature (25C expressed in Kelvin)
64.     RNOM = 10000.0           # default reference resistance at reference temperature; adjust to
           calibrate
65.     SRES = 10000.0           # default series resister value; adjust as per your implementation
66.
67.     # to catch a divide by zero error we check the value of adc and fake it if needed
68.     if adc == 0:
69.         adc = 0.001
70.
71.     # value of the resistance of the thermistor
72.     Rtherm = (1023.0/float(adc) - 1)*10000
73.
74.     # see http:#en.wikipedia.org/wiki/Thermistor for an explanation of the formula
75.     kelvin = RTEMP*BVAL/(BVAL+RTEMP*(math.log(Rtherm/RNOM)))
76.
77.     # convert from Kelvin to Celsius
78.     temperature = kelvin - 273.15
79.
80.     # print the ADC Value and the Temperature value
81.     # here we are doing a little formatting of the output
82.     # the first {} inside the quotes is replaced with the contents of value
83.     # the second {} is replaced with our calculated Volts
84.     # the :0.2f inside the second {} limits the
85.     # floating point number to two decimal places
86.     print("ADC: {} Temperature: {:0.2f}C".format(adc, temperature))
87.
88.     # increase the count by 1 at the end of the block
89.     count += 1
90.
91. # close the serial port
92. ser.close()
93.
94. # at the end of the script python automatically exits
```

As in activity 18 you can use IDLE to open the 05Temperature.py file from the Examples folder. Running this should give an output similar to the one below.

image

You can modify the code to poll at greater intervals, by adding a sleep statement in the loop. You can also increase the number of polls by changing the while condition on line 36.

Activity 24 - Filtering LLAP messages

<< back to contents page

What will we learn?

So far, we have always assumed that the message we expect to receive is in fact in the message buffer. However, when there are multiple devices around and different types of data being sent over the radio network, this may not always be true.

In this example we show you how to deal with multiple incoming messages, and make sure we only read in real LLAP messages from the source we expect. You will be able to run the code and make modifications.

How long will this take?

No more than 5 minutes

How difficult is this project?

Easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

With reference to the code below, the code to line 33 should look quite familiar by now. As you know too, when we send a LLAP message to the XinoRF, it will generally respond with a message, either saying it has carried out a command, or reurning a reading when asked.

In this activity we are sending a series of messages to the XinoRF, one after the other, with a short pause in between (lines 38 to 49). Each time a message is received, the XinoRF will carry out some instructions and then send a response. These responses will get buffered (stacked up) in the Raspberry Pi, waiting for our program to read them out. Starting at line 52, we read in characters from the buffer for as long as they are available (ser.inWaiting() is true).

Each time we read, we check to see if it is a lower case 'a' and if it is, we assume to be at the start of a LLAP message. We can then read in the next 11 characters to form the full message (line 63).

In line 71 we check that the message comes from a device with address "--". Any other devices are thus ignored. If the data part starts with "A00" we have data from an analog pin and print the data part after some formatting. Else, if the data part starts with a "D", we have data about a digital pin and print a different message.

From line 82 we loop back up within the while loop to read the next set of bytes from the buffer and loop round till the input buffer is empty.

```
01. #!/usr/bin/env python# -*- coding: utf-8 -*-
02. """ Wireless Inventors Kit Python Example 06Filtering.py
03.     Ciseco Ltd. Copyright 2013
04. 
05.     Filtering incoming LLAP messages
06.     In this example we will filter incoming messages for a set devID
07.     Then filter different replies to be handled differently
08. 
09. 
10.     Author: Matt Lloyd
11. 
12.     This code is distributed in the hope that it will be useful,
13.     but WITHOUT ANY WARRANTY; without even the implied warranty of
14.     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15. 
16. """
17. #import the PySerial library and sleep from the time library
18. import serial
19. from time import sleep
20. # for temperature we also need the math library
21. import math
22. 
23. # declare to variables, holding the com port we wish to talk to and the speed
24. port = '/dev/ttyAMA0'
25. baud = 9600
26. 
27. # open a serial connection using the variables above
28. ser = serial.Serial(port=port, baudrate=baud)
29. 
30. # wait for a moment before doing anything else
31. sleep(0.2)
32. 
33. # this time we are going to send 4 commands one after the other
34. # the replies will get buffered by Python until
35. # we later call ser.read()
36. # after each ser.write() we print to the console and sleep() to give the
37. # XinoRF time to reply before sending the next
38. ser.write('a--D13HIGH--')
39. print("Sent a--D13HIGH--")
40. sleep(1)
41. ser.write('a--A00READ--')
42. print("Sent a--A00READ--")
43. sleep(1)
44. ser.write('a--A01READ--')
45. print("Sent a--A01READ--")
46. sleep(1)
47. ser.write('a--D02READ--')
48. print("Sent a--D02READ--")
49. sleep(0.2)
50. 
51. # loop until the serial buffer is empty
52. while ser.inWaiting():
53.     # read a single character
54.     char = ser.read()
55. 
56.     # check we have the start of a LLAP message
57.     if char == 'a':
58.         # start building the full llap message by adding the 'a' we have
59.         llapMsg = 'a'
60. 
61.         # read in the next 11 characters form the serial buffer
62.         # into the llap message
```

```
        llapMsg += ser.read(11)

        # now we split the llap message apart into devID and data
        devID = llapMsg[1:3]
        data = llapMsg[3:]

        # check the devID is correct for our device
        # (WIK ships as -- be default)
        if devID == '--':
            # check to see if the message is about A00
            if data.startswith('A00'):
                # split out the pin & the return value and print to the console
                print("Got Analog reading for {} of {}".format(data[0:3],
                                                               data[4:].strip('-')))

            # check to see if the message relates to a digital pin
            elif data.startswith('D'):
                # split out the pin & the return value and print to the console
                print("Got digital reading for {} of {}".format(data[0:3],
                                                                data[3:].strip('-')))


# close the serial port
ser.close()

# at the end of the script python automatically exits
```

As in activity 18 you can use IDLE to open the 06Filtering.py file from the Examples folder.

Running this should give an output similar to the one below.

Image

Activity 25- A simple battery monitor

<< back to contents page

What will we learn?

That we can measure a voltage on one pin and then depending on the reading show a red, orange, green status indicator.

How long will this take?

No more than 10 minutes

How difficult is this project?

Very easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

5 x jumper wire

1 x breadboard

3 x 470 ohm resistor

1 x LED red

1 x LED yellow

1 x LED green

1 x 10K potentiometer

Let's start!

This activity is based on two previous activities: the traffic light (activity 4) and activity 8, where we used a variable resistor to measure a voltage.

First use the breadboard to build up the circuit you built earlier for the traffic lights in activity 4. Include all the wiring to the XinoRF pins D09, D11 and D13 for the green, yellow and red LEDs respectively.

Then, on the same breadboard wire up the variable resistor, which we shall use as a surrogate for a battery with variable charge left in it. Wire one end to ground, the other to 3V3, and the middle pin to A0, just like you did in Activity 8.

The code below periodically sends a message to the XinoRF, asking it for the value of the input on A0. This value will lie between 0 and 1024, and we have divided this range in three equal parts. When the value is in the lower third, we instruct the XinoRF to light up the red LED, in the middle third, the yellow LED is lit up, and in the top third, the green LED comes on.

There are a few subtle differences between this code and the polling activity we did earlier.

First of all, we have defined a value "maxcount" to define how many times the loop is executed. Secondly, each time we go through the loop, we flush the input buffer. This is because each time we tell the XinoRF to turn an LED on or off, it sends back a message and these messages clog up the input buffer unless they are cleared out.

After sending the request for Ao, and reading the return message, we check it actually contains a--A. If it does not, there could be another message in the buffer we need to flush out. Only if we are sure that we have a correct response do we process it and make the decision regarding what LED should be lit.

```
#!/usr/bin/env python# -*- coding: utf-8 -*-
""" Wireless Inventors Kit Python Example 07Feedback.py
    Ciseco Ltd. Copyright 2013

    Polled (or repeated) send and receive of LLAP messages,
    measuring Voltage on A0 and setting red/yellow/green charge light

    Authors: Matt Lloyd & Rob van der Linden

    This code is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

"""
#import the PySerial library and sleep from the time library
import serial
from time import sleep

# declare to variables, holding the com port we wish to talk to and the speed
port = '/dev/ttyAMA0'
baud = 9600

# open a serial connection using the variables above
ser = serial.Serial(port=port, baudrate=baud)

# wait for a moment before doing anything else
sleep(0.2)

# setup a counter starting at 0
count = 0
# number of times round the loop
maxcount = 20

# clear out the serial input buffer to ensure there are no old messages lying around
ser.flushInput()

# loop over the block of code while the count is less than maxcount
# when the count = maxcount the loop will break and we carry on with the rest
while count < maxcount:
    # write a--A00READ-- out to the serial port
    # this will return the current ADC reading for Pin A0
    ser.write('a--A00READ--')

    # wait for a moment before doing anything else
    sleep(0.2)

    # loop until the serial buffer is empty
    while ser.inWaiting():
        # read a single character
        char = ser.read()

        # check we have the start of a LLAP message
        if char == 'a':
            # start building the full llap message by adding the 'a' we have
            llapMsg = 'a'

            # read in the next 11 characters form the serial buffer
            # into the llap message
            llapMsg += ser.read(11)

            # now we split the llap message apart into devID and data
            devID = llapMsg[1:3]
            data = llapMsg[3:]

            # check the devID is correct for our device
            # (WIK ships as -- be default)
            if devID == '--':
                # check to see if the message is about A00
                # if not we skip the section of code
                if data.startswith('A00'):
                    # take just the last part of the message
                    value = data[4:]

                    # strip the trailing '-'
                    value = value.strip('-')

                    # convert the string in value to an integer
                    v = int(value)

                    # dividing the range of the ADC (1024) into 3 areas:
                    # if v lies below 342 light up the red LED only
                    # if v is between 342 and 684 light up the yellow LED
                    # if v is above 684 light up the green LED

                    if v < 342:
                        # print("RED")
                        ser.write("a--D09LOW---")
                        sleep(0.2)
                        ser.write("a--D11LOW---")
                        sleep(0.2)
                        ser.write("a--D13HIGH--")
                    elif v >=342 and v <= 685:
                        # print("YELLOW")
                        ser.write("a--D09LOW---")
                        sleep(0.2)
                        ser.write("a--D11HIGH--")
                        sleep(0.2)
                        ser.write("a--D13LOW---")
                    elif v > 685:
                        # print("GREEN")
                        ser.write("a--D09HIGH--")
                        sleep(0.2)
                        ser.write("a--D11LOW---")
                        sleep(0.2)
                        ser.write("a--D13LOW---")

                    # print the ADC Value
                    # here we are doing a little formatting of the output
                    # the {} inside the quotes is replaced with the contents of value
                    print("ADC: {}".format(value))

                    # increase the count by 1 at the end of the block
                    count += 1

# close the serial port
ser.close()

# at the end of the script python automatically exits
```

As in previous examples, you can load the code into IDLE by opening 07Feedback.py from the Examples folder.

You can modify the code to poll at greater intervals, by adding a sleep statement in the loop. You can also increase the number of polls. Experiment with all these and try to understand the response time of the monitor to you changing the position of the potentiometer.

Activity 26- Logging temperature data to a text file

<< back to contents page

What will we learn?

Would't it be nice if you could collect a series of analog readings so you can look at them and analyse them later? To do this, we need to save the data we get from sensors somewhere we can later access.

In this activity we are going to store a series of temperature readings in a file. We'll make it a text file with comma separated values (csv), so that we can load it into a spreadsheet program later.

How long will this take?

About 10 minutes to set up, then any amount of time experimenting.

How difficult is this project?

Easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

1 x 10K resistor

1 x Thermistor

1 x Breadboard

3 x Jumper wire

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

Next use the components to build the same system you did in Activity 10 and Activity 22.

The code below should look quite familiar to those who have run Activity 22, where we converted an ADC value from a thermistor to temperature. As in the previous example, we have set up variables for the number of times we read the temperature (maxcount in line 34), which of course you are free to change.

In line 37 we open a file called log.csv for writing to. Because it is a csv file, it is handy to have some headings on the columns, and we write these first in line 40.

We then enter a loop in which we read the ADC value from A00 and wait for the response from the XinoRF. Note how we validate the messages we receive to make sure we get a valid LLAP message from the source we require, as we did in the filtering example earlier.

The code upto line 102 is identical to that in Activity 20. In line 106 we write the data to the log.csv file.

Because we cannot see that writing to log.csv is actually happening, line 113 writes data to the standard output, so that we can see that the program is actually working.

Once the loop has been executed maxcount number of times, the program exists, but before doing so it closes the file and the serial connection.

```
001. #!/usr/bin/env python# -*- coding: utf-8 -*-
002. """ Wireless Inventors Kit Python Example 08Logging.py
003.     Ciseco Ltd. Copyright 2013
004.
005.     Logging incomming LLAP messgaes to a text file
006.     include conversion to temperature
007.
008.     Authors: Matt Lloyd & Rob van der Linden
009.
010.     This code is distributed in the hope that it will be useful,
011.     but WITHOUT ANY WARRANTY; without even the implied warranty of
012.     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
013.
014. """
015. #import the PySerial library and sleep from the time library
016. import serial
017. from time import sleep, asctime
018. # for temperature we also need the math library
019. import math
020.
021. # declare to variables, holding the com port we wish to talk to and the speed
022. port = '/dev/ttyAMA0'
023. baud = 9600
024.
025. # open a serial connection using the variables above
026. ser = serial.Serial(port=port, baudrate=baud)
027.
028. # wait for a moment before doing anything else
029. sleep(0.2)
030.
031. # setup a counter starting at 0
032. count = 0
033. # number of times round the loop
034. maxcount = 20
```

```

# open a file to log the data to
file = open('./log.csv', 'w')

# we are going to output data in a csv format so this adds a heading row
file.write("\"Time\",\"LLAP\",\"Temperature\"\n")

# clear out the serial input buffer to ensure there are no old messages lying around
ser.flushInput()

# loop over the block of code while the count is less than maxcount
# when the count = maxcount the loop will break and we carry on with the rest
while count < maxcount:
    # write a--A00READ-- out to the serial port
    # this will return the current ADC reading for Pin A0
    ser.write('a--A00READ--')

    # wait for a moment before doing anything else
    sleep(0.2)

    # loop until the serial buffer is empty
    while ser.inWaiting():
        # read a single character
        char = ser.read()

        # check we have the start of a LLAP message
        if char == 'a':
            # start building the full llap message by adding the 'a' we have
            llapMsg = 'a'

            # read in the next 11 characters form the serial buffer
            # into the llap message
            llapMsg += ser.read(11)

            # now we split the llap message apart into devID and data
            devID = llapMsg[1:3]
            data = llapMsg[3:]

            # check the devID is correct for our device
            # (WIK ships as -- be default)
            if devID == '--':
                # check to see if the message is about A00
                # if not we skip the section of code
                if data.startswith('A00'):
                    # take just the last part of the message
                    # strip the traling -'s
                    # and convert to an int
                    adc = int(data[4:].strip('-'))

                    # to calculate the temperature we use a more complex formula
                    # here we store some of the fixed numbers in variables
                    BVAL = 3977              # default beta value for the thermistor
                    RTEMP = 25.0 + 273.15    # reference temperature (25C expressed in Kelvin)
                    RNOM = 10000.0           # default reference resistance at reference temperature; adjust to calibrate
                    SRES = 10000.0           # default series resister value; adjust as per your implementation

                    # to catch a divide by zero error we check the value of adc and fake it if needed
                    if adc == 0:
                         adc = 0.001

                    # value of the resistance of the thermistor
                    Rtherm = (1023.0/float(adc) - 1)*10000

                    # see http:#en.wikipedia.org/wiki/Thermistor for an explanation of the formula
                    kelvin = RTEMP*BVAL/(BVAL+RTEMP*(math.log(Rtherm/RNOM)))

                    # convert from Kelvin to Celsius
                    temperature = kelvin - 273.15

                    # now log the time, orignal llap messgae and temperature
                    # to the file in a csv format
                    file.write("\"{}\",\"{}\",\"{:0.2f}\"\n".format(asctime(),
                                                                   llapMsg,
                                                                   temperature
                                                                   )
```

```
                                )

                    # little bit of user feed back so we know the script is working
                    print("Logged {}".format(count))

                    # increase the count by 1 at the end of the block
                    count += 1

# close the file
file.close()

# close the serial port
ser.close()

# at the end of the script python automatically exits
```

As in activity 18 you can use IDLE to open the 08Logging.py file from the Examples folder.

If the program runs and there is no output on the Python Shell saying "Logged .." then press the reset button on the XinoRF to get things started properly. Once you have run the program, you can open the log.csv file with another program such as a spreadsheet program and deal with the data, create graphs etc. The log file can be found in the WIK FILES folder under Python/Examples.

Before you run the program a second time it is a good idea to either rename the file log.csv to something else or delete it. If you don't the program will simply write a new header line in betweenthe data you already have, making the log.csv file less useful.

Activity 27- Sending temperature data to the Xively IoT cloud

[<< back to contents page]

What will we learn?

In the previous activity, we stored a series of temperature readings in a log file. That's great if we collect this data only for our own local use. But what if we want to access this data when we are away from our loved raspberry Pi? And what if we want to share this data with other people?

In this activity we will show you how to store temperature data on a real Internet of Things service in the cloud so that we can remotely access it, look at graphs and share data with others.

How long will this take?

About 30 minutes

How difficult is this project?

Easy

Requirements:

1 x XinoRF

1 x USB cable or DC jack to power the XinoRF (6v to 12v is ideal)

1 x Raspberry Pi (Model A or Model B) + keyboard + mouse + monitor

1 x Slice of Radio

1 x SD card as supplied with your kit

1 x 10K resistor

1 x Thermistor

1 x Breadboard

3 x Jumper wire

Let's start!

Make sure your Pi is installed and running as detailed in activity 01. However, do not run the RasWIK application software, as we will run our own Python program to do the work.

Next use the components to build the same system you did in Activity 10, Activity 22, and Activity 25.

Much of the code below will look familiar if you ran through the previous activities. You will recognise the sending of the request to obtain the ADC value on A00 (line ??) and the code needed to receive the response and convert the ADC value to temperature (ending in line ??).

But other things are new, in particular the way we log the data to Xively.

Xively is a service that allows you to post data for storage and later retrieval. It is a so called "web service" which is accessed via an Application Programming Interface or API. Of course Xively do not want anyone to be arbitrarilly writing data on their service, as it would all become a jumble. To use Xively, we need two things: an account, and an API key that is to be used each time we post (or read) data.

First we set up an account with Xively (assuming you don't have one already, of course, else, just log in):

Go to http://www.xively.com

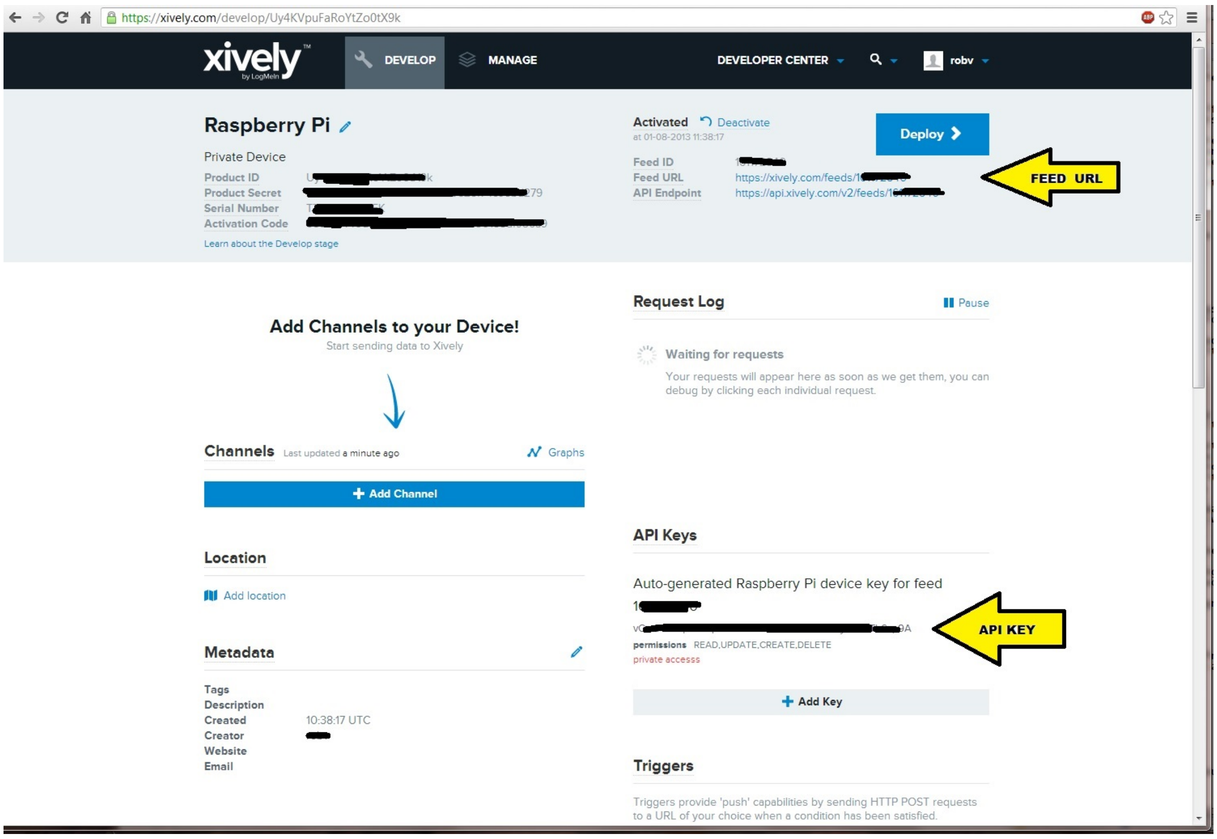
Create an account by clicking "Sign Up" on the home page.

Click on the DEVELOP tab and wait for the page to load

Now click "Add Device"

Give your device a name, e.g. Raspberry Pi and a description if you want.

Make it a Private Device under the privacy settings for now.
Click Add Device.
You should now see a screen as below:



Raspberry Pi

Private Device

Product ID	[redacted]
Product Secret	[redacted]
Serial Number	[redacted]
Activation Code	[redacted]

Learn about the Develop stage

Activated Deactivate
at 01-08-2013 11:38:17

Feed ID	[redacted]
Feed URL	https://xively.com/feeds/[redacted]
API Endpoint	https://api.xively.com/v2/feeds/[redacted]

Add Channels to your Device!
Start sending data to Xively

Channels Last updated a minute ago

Location

Metadata

Tags	
Description	
Created	10:38:17 UTC
Creator	[redacted]
Website	
Email	

Request Log

Waiting for requests
Your requests will appear here as soon as we get them, you can debug by clicking each individual request.

API Keys

Auto-generated Raspberry Pi device key for feed
[redacted]
[redacted]
permissions READ,UPDATE,CREATE,DELETE
private accesss

Triggers

Triggers provide 'push' capabilities by sending HTTP POST requests to a URL of your choice when a condition has been satisfied.

On this screen you have everything you need to insert in the code on lines 51 and 52.
The FEED_ID is just above the Feed URL. This needs to be copied into the Python code.
The API_KEY is on the right hand side (see arrow). This also needs to be inserted in the Python code.
Keep the develop page open and you will see requests from the Raspberry Pi appear in the Request Log section of the page once we run the program.
Now, let's walk through the rest of the code before we run it and finalise our example.
In line 56 we open an interface to the Xively API. In line 72-76 we set up the feed and create a datastream and open the feed to Xively.
In line 138 and 139 we create a record of the temperature and the time, and then in lines 142 to 145 we push the data out to Xively. We'll print an error message if it does not work.
As before, we provide local feedback in line 148 before looping back for the next mesurement.
When all measurements are done, we clode the serial port and exit.

```
001. #!/usr/bin/env python# -*- coding: utf-8 -*-
002. """ Wireless Inventors Kit Python Example 09Xively.py
003.     Ciseco Ltd. Copyright 2013
004.
005.     Logging incoming LLAP temperatures to a Xively datastream
006.
007.     Note the Example requires you to create a Xively account and obtain an API key
008.
009.     Authors: Matt Lloyd & Rob van der Linden
```

```python

    This code is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

"""
#import the PySerial library and sleep from the time library
import serial
from time import sleep, asctime
# for temperature we also need the math library
import math
# we need the Xively Python module and a few others
import xively
import requests
from datetime import datetime

# function to be called later in the script
def open_datastream(feed):
    # the function returns a Xively datastream handle
    # if the datastream is not present on Xively within your feed one is created
    try:
        datastream = feed.datastreams.get("Temperature")
        print("Datastream for Temperature found")
        return datastream
    except:
        # no datadtream was found so we create a new Temperature based stream
        datastream = feed.datastreams.create("Temperature", tags="temp",
                                             unit=xively.Unit(label='Celsius',
                                                              type='basicSI',
                                                              symbol='C'))
        print("Datastream for Temperature created")
        return datastream


# declare to variables, holding the com port we wish to talk to and the speed
port = '/dev/ttyAMA0'
baud = 9600

# declare variables for Xively
# you need to copy in you API_KEY and FEED_ID paste them between the quotes
# the WIK documentation includes details on how to obtain these
FEED_ID = "***YOUR FEED ID***"
API_KEY = "***YOUR API KEY***"


# initialise api client
api = xively.XivelyAPIClient(API_KEY)

# open a serial connection using the variables above
ser = serial.Serial(port=port, baudrate=baud)

# wait for a moment before doing anything else
sleep(0.2)

# setup a counter starting at 0
count = 0
# number of times round the loop
maxcount = 20

# clear out the serial input buffer to ensure there are no old messages lying around
ser.flushInput()

feed = api.feeds.get(FEED_ID)

datastream = open_datastream(feed)
datastream.max_value = None
datastream.min_value = None

# loop over the block of code while the count is less than maxcount
# when the count = maxcount the loop will break and we carry on with the rest
while count < maxcount:
    # write a--A00READ-- out to the serial port
    # this will return the current ADC reading for Pin A0
    ser.write('a--A00READ--')

    # wait for a moment before doing anything else
    sleep(0.2)

    # loop until the serial buffer is empty
    while ser.inWaiting():
        # read a single character
        char = ser.read()

        # check we have the start of a LLAP message
        if char == 'a':
            # start building the full llap message by adding the 'a' we have
            llapMsg = 'a'

            # read in the next 11 characters form the serial buffer
            # into the llap message
            llapMsg += ser.read(11)

            # now we split the llap message apart into devID and data
            devID = llapMsg[1:3]
            data = llapMsg[3:]

            # check the devID is correct for our device
            # (WIK ships as -- be default)
            if devID == '--':
                # check to see if the message is about A00
                # if not we skip the section of code
                if data.startswith('A00'):
                    # take just the last part of the message
                    # strip the trailing -'s
                    # and convert to an int
                    adc = int(data[4:].strip('-'))

                    # to calculate the temperature we use a more complex formula
                    # here we store some of the fixed numbers in variables
                    BVAL = 3977              # default beta value for the thermistor
                    RTEMP = 25.0 + 273.15    # reference temperature (25C expressed in Kelvin)
                    RNOM = 10000.0           # default reference resistance at reference temperature; adjust to calibrate
                    SRES = 10000.0           # default series resister value; adjust as per your implementation

                    # to catch a divide by zero error we check the value of adc and fake it if needed
                    if adc == 0:
                        adc = 0.001

                    # value of the resistance of the thermistor
                    Rtherm = (1023.0/float(adc) - 1)*10000

                    # see http:#en.wikipedia.org/wiki/Thermistor for an explanation of the formula
                    kelvin = RTEMP*BVAL/(BVAL+RTEMP*(math.log(Rtherm/RNOM)))

                    # convert from Kelvin to Celsius
                    temperature = kelvin - 273.15

                    # set a new dataint with the temperature and time
                    datastream.current_value = temperature
                    datastream.at = datetime.utcnow()

                    # now we push the update to Xively, and create an error if it fails
                    try:
                        datastream.update()
                    except requests.HTTPError as e:
                        print "HTTPError({0}): {1}".format(e.errno, e.strerror)

                    # little bit of user feed back so we know the script is working
                    print("Logged {}".format(count))

                    # increase the count by 1 at the end of the block
                    count += 1

                    # sleep a while before logging another temp
                    sleep(10)

# close the serial port
ser.close()

# at the end of the script python automatically exits
```

Now open the file 09Xively.py in IDLE and insert your FEEDID and your API_KEY in line 51 and 52, save the file and then run it while the Xively develop desktop is still open in your browser (see image above). In the request log area you should see various requests appearing, and in the Channels section you will see that temperature measurements are being uploaded and become visible.

If all you see in the Python Shell is the message "Datastream for Temperature found" and nothing else, press the reset button on the XinoRF and you will start seeing messages like "Logged 0", "Logged 1", etc. appearing. You will also see the temperature graph in the Channels section emerge.

BTW, there is no need to deploy the device in Xively, as it will remain available on the develop workbench, as they call it.

Activity 28 - WIK Launcher Advanced

<< back to contents page

What will we learn?

!! NOTE: the interface to WIKLauncher Advanced will be changed -- this section will be amended once that has been done.

So far we have only used the Main menu of WIK Launcher. In this section we will explain how to use the functions under the Advanced tab, which allow us to update software, firmware and reset radios to their factory settings.

How long will this take?

About 10 minutes

How difficult is this project?

Moderate

Requirements:

1 x XinoRF

1 x USB cable

1 x Raspberry Pi (model A or model B)

1 x SD card as supplied with your kit

Let's start!

All the components in the kit come ready configured, including the Raspberry Pi operating system and all the programs and files we need for the Wireless Inventors Kit.

As with all computer and software projects, software is often updated to fix errors, add extra functionality and so on. The software, firmware and radio settings can sometimes become corrupted, especially when you start experimenting in a bit more depth.

The functions under the Advanced menu of WIK Launcher will help you restore everything to the latest version and/or factory settings:

- Update from Zip File: allows you to manually update the files in WIK FILES.
- Reset Slice of Radio Settings: allows you to reset the radio settings on the Raspberry Pi to their defaults
- Reset XinoRF Radio Settings: allows you to reset the radio settings on the XinoRF to their defaults
- Update XinoRF Firmware: allows you to reload the firmware on the XinoRF
- Update Arduino IDE WIKSketch Files: allows you to update the WIKSketch files resident on the Raspberry Pi

We will go over each of these in a bit more detail next:

Update from Zip File

The WIK FILES folder lives on the desktop of your Raspberry Pi. When you run RunMe.py from this folder (to start the WIK Launcher) we usually check to see if there is a new version available. If so, you are asked whether you want to update the files, so you will normally always have the latest working set.

There are situations when you may want to make sure you have a fully working set of files, for instance when you have accidentally or intentionally made changes to the files and things have not worked out.

The first thing to do is to manually download the zip file with the WIK FILES in it. With the NetSurf browser (don't use Midori) go to http://ciseco.co.uk/files/WIK. Click on the version you want (usually the latest) and save the zip file on the desktop for now (select "Save As"). Wait long enough for the whole zip file to be downloaded (The Pi s rather slower than other computers you are used to!).

Next in WIK Launcher > Advanced select "Update from Zip File" and browse to and select the zip file you have just downloaded and saved on the Desktop. Click Open and the files will be extracted after which WIK Launcher restarts.

You now have a fresh new set of files in WIK FILES and you're ready to go.

Reset Slice of Radio Settings

This function has not yet been implemented. Once it is, you can reset the radio settings on the Raspberry Pi end to their defaults.

Reset XinoRF Radio Settings

This function has not yet been implemented. Once it is, you will be able to simply reurn the radio settings on the XinoRF to their default settings.

Update XinoRF Firmware

This function has not yet been implemented. Once it has, you will be able to automatically reset the Firmware on the XinoRF to the factory default. The firmware is a standard Arduino sketch, called WIKSketch. Activity 28 shows you how to load the sketch manually.

Update Arduino IDE WIKSketch Files

Before you can manually update the XinoRF firmware (see next activity) you need the WIKSketch and associated library files to be available in the Arduino sketchbook folder that belongs to the Arduino IDE installed on your Raspberry Pi.

This function makes sure the files are in the right place.

When selecting this option you will be asked to identify the Arduino sketchbook folder. On the Raspberry Pi image we ship with the kit, this will be /home/pi/sketchbook. It is likley already selected, so you can press OK if that is so. Else select the Arduino IDE sketchbook location and press OK.

You will see a message saying that the latest WIKSketch can be found

under
Examples > LLAPSerial > WIKSketch.

This sets you up for uploading the firmware to the XinoRF, described in the next activity .

We may add further functions under the Advanced tab as the kit develops.

Activity 29 - Reloading or updating the XinoRF manually

[<< back to contents page]

What will we learn?

The software that ships on your XinoRF is a standard Arduino sketch. In this activity we will show you how to manually reload or update it using the Arduino IDE on the Raspberry Pi.

How long will this take?

About 10 minutes

How difficult is this project?

Moderate

Requirements:

1 x XinoRF

1 x USB cable

1 x Raspberry Pi (model A or model B)

1 x SD card as supplied with your kit

Let's start!

The XinoRF that ships with the RasWIK kit has its software preloaded, so you can get started right away. There are situations in which you may like to reload this software. For instance, if you have a XinoRF without the preloaded software, or in the unlikely event of the software on the XinoRF becoming corrupted.

Fortunately it is really simple to load software onto the XinoRF from the Raspberry Pi. Here is how to do it.

First connect the XinoRF directly to the Raspberry Pi via the USB cable. You may connect it via a powered hub.

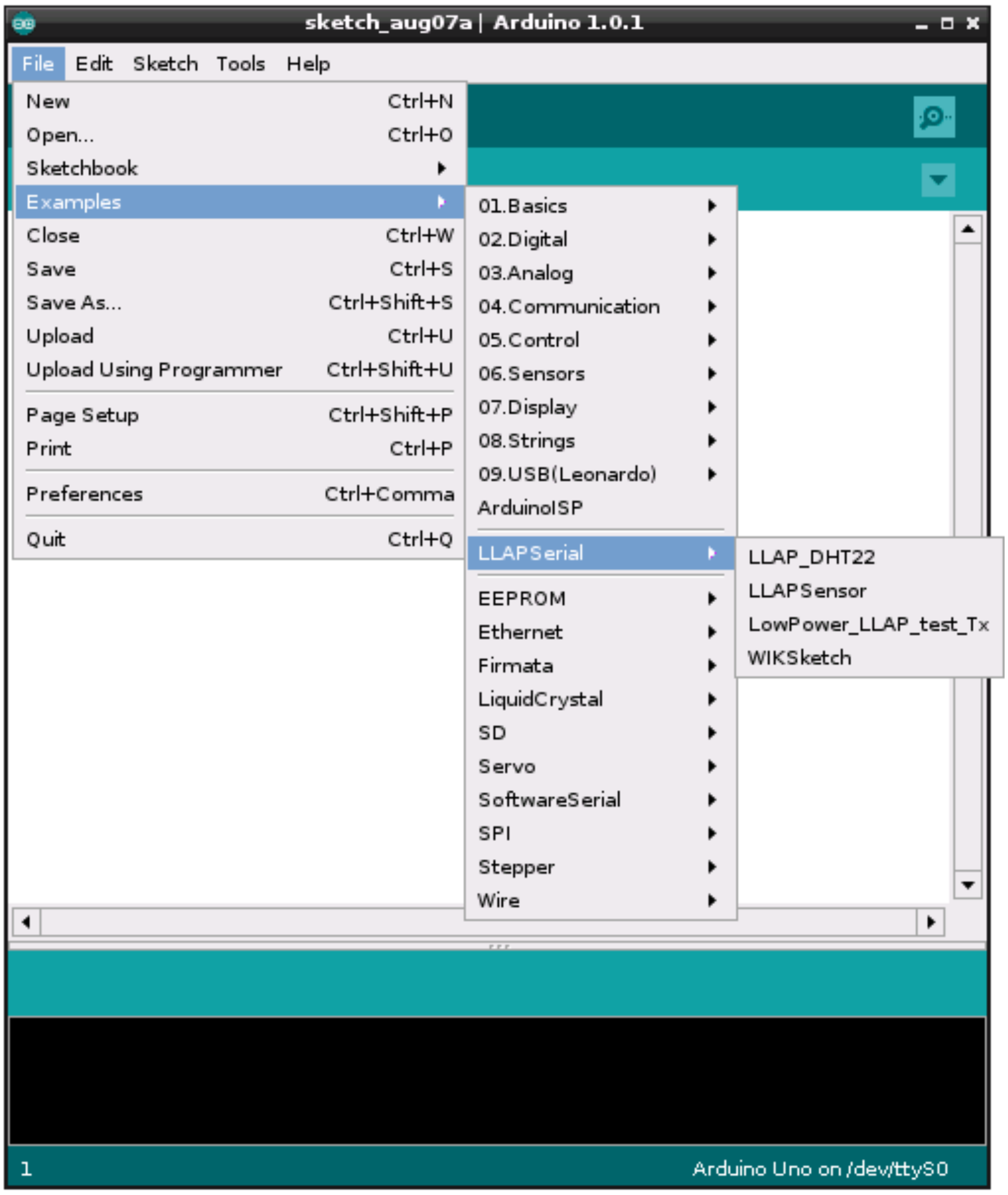
Then go to the Advanced tab on WIK Launcher and select "Update Arduino IDE WIKScetch Files". See the previous activity for details. When done you can close WIK Launcher.

Then on the Raspberry Pi and double click the icon called "Arduino IDE" on your desktop. Alternatively, go to the start menu and under "Programming" you will see "Arduino IDE". Click this to start the Arduino IDE.

Click "File" and in the dropdown select "Examples", then "LLAPSerial" and "WIKSketch".

This will open the the sketch that is on your XinoRF when it shipped.

image



In the blue bar under the menu items "File", "Edit", "Sketch" ... etc. you see a row of icons.

The first icon is used to check a sketch for errors, the second icon (with the right arrow in it) is used to upload.

Click on the second icon (the one with the right arrow) and wait for the sketch to compile and upload. In the status field at the bottom of the window you will see what is happening. Wait until you see the message:

"done with auto-reset"

and nothing else, to indicate that the upload was successful. If you do get other messages, then perform the upload again, until you get a successful upload.

APPENDIX

<< back to contents page

A1 - Following these projects without having the full kit

If you bought a kit you can ignore this section.

If you are replicating these projects with a standard XinoRF you will need to install SandyWare onto your XinoRF. You will need the Arduino IDE installed and the LLAPSerial library copying from our GitHub https://github.com/CisecoPlc/LLAPSerial to your machine, mine needed located at C:\Users\Miles\Documents\Arduino\libraries\LLAPSerial. The 2 files you need are in a directory called LLAPSerial the files are **LLAPSerial.h** and **LLAPSerial.cpp**. Once they are there you will need to restart your Arduino IDE. You can then upload the WIK sketch.

A2 - Using different radios to those than in the kit

It is possible to use any transparent radio link and Arduino compatible device. We cannot as you would expect know exactly how best to other peoples hardware but we know it can be done. If you have tried any of the activities with devices such as the XBee, RFBee, LPRS or other radio. Please do let us know and we'll let people know by blogging about it.

A3 - Where to get support

It is possible to use any transparent radio link and Arduino compatible device. We cannot as you would expect know exactly how best to other peoples hardware but we know it can be done. If you have tried any of the activities with devices such as the XBee, RFBee, LPRS or other radio. Please do let us know and we'll let people know by blogging about it.

A4 - Useful links

1. Reviews of the RasWIK kit

2. The RasWIK on our shop
3. Further adventures in wireless on our shop (this includes: 1 x Servo, 1 x Motor, 1 x Speaker, 1 x PIR sensor, 1 x TCRT 5000 (reflection sensor), 1 x Tilt switch, 1 x Reed switch and magnet)
4. The Slice of Radio on our shop
5. The XinoRF on our shop
6. The Slice of Radio on our shop
7. Download the Arduino IDE
8. The main Arduino forum
9. Other Arduino forums
10. Fritzing part available for the XinoRF, it's what we sued to make the circuit images **http://openmicros.org/index.php/articles/88-ciseco-product-documentation/275-fritzing-parts**

Other projects we have found using our radios and the Pi

1. Birmingham university freezer monitor

A5 - XinoRF pinouts

<< back to contents page

		SCL	
		SDA	
		AREF	
		GND	
		D13	OUTPUT D13
	IOREF	D12	INPUT D12
	RESET	D11	OUTPUT D11 / PWM
	3V3	D10	INPUT D10
	5V	D09	OUTPUT D09 / PWM
	GND	D08	Reserved for radio control
	GND	D07	INPUT D07
	VIN	D06	OUTPUT D06 / PWM
ANALOG A00	A00	D05	SERVO
ANALOG A01	A01	D04	COUNT
ANALOG A02	A02	D03	INTERRUPT D03
ANALOG A03	A03	D02	INTERRUPT D02
ANALOG A04	A04	D01/TX	Reserved for radio comms

ANALOG A05	A05	D00/RX	Reserved for radio comms

A6 - List of commands available on the Sandy Sketch

<< back to contents page

The supported commands are:

OUTPUT

aXXD06HIGH--

aXXD06LOW---

aXXD09HIGH--

aXXD09LOW---

aXXD11HIGH--

aXXD11LOW---

aXXD13HIGH-- (D13 is also the onboard XinoRF LED)

aXXD13LOW---(D13 is also the onboard XinoRF LED)

DIGITAL INPUT (any input not pulled to ground (ie floats) will always read high due to the internal chip pull ups being turned on)

aXXD07READ--

aXXD10READ--

aXXD12READ--

ANALOG INPUT (any input pin left floating (ie not connected) will give fairly random readings)

aXXA00READ---

aXXA01READ---

aXXA02READ---

aXXA03READ---

aXXA04READ---

aXXA05READ---

PWM OUTPUT (The last 3 numbers set the PWM duty between 000 (off) and 255 (fully on))

aXXD06PWM128

aXXD09PWM255

aXXD11PWM000

SERVO (set a servo between 0 and 180 degrees on pin D05)

aXXSERVO000- (set to 0)

aXXSERVO121- (set to 121 degrees)

COUNT (If you connect D4 to ground you will increase the counter by 1 each time, at 9999 the next number will be 0000)

aXXCOUNT----

aXXCOUNT0001 (set the counter to 1)

aXXCOUNT4523 (set the counter to 4523)

INTERRUPT (this is not a command you can send wirelessly, it detects a change in state (low/high) of either D02 or D03 and sends out a message to indicate the change. Of note is that a floating pin will always read high due to the internal chip pull ups being turned on)

aXXD03LOW--- (pin D03 was just connected to ground)

aXXD02HIGH-- (pin D2 was disconnected from ground)

Troubleshooting

<< back to contents page

Q1. I can't find any mention or I don't understand something written about in this document.

A1. We do our best to put ourselves in your shoes when writing docs, if you let us know what should have covered or explained differently, we'll change this document.

Q2. I see wrong characters being sent wirelessly when I replace the RasWIK sketch with an Arduino sketch

A2. The XinoRF board runs at 115,200bps rather than the more common 9600bps, this is because the Uno bootloader we use, also runs at 115,200bps.

Q3. Where can I find out more technical info on the XinoRF and Slice of Radio?

A3. Click here to jump to the section that has links to our shop products

Q4. If I over write the SD card can I download the image afterwards.

A4. No, however you can download our standard Pi image and the WIK Files zip files, this is what we do to make the RasWIK SD image here.

END OF DOCUMENT