Wi-Fi Orb Sees All

By Tom Cantrell, Convergence Promotions LLC
10/10/2012

Wi-Fi has played a critical role in the growth of PC networking by enhancing mobility and extending reach in those places (many businesses and most homes) that are not outfitted with wall-to-wall CAT-10 cable.

In recent years, Wi-Fi has experienced explosive growth in new applications such as telephony (smartphones) and audio/video distribution (Internet-connected DVRs). As the number of 'hotspots' and the coverage area grows, Wi-Fi is taking on aspects of a 'utility' that goes well beyond its humble cable replacement roots. It's not hard to imagine that soon practically every phone and TV will have Wi-Fi built in as a standard feature.

Do your own thing

So what does Wi-Fi do for an encore? Well, how about an 'Internet of Things' (aka 'IoT') that connects virtually every gadget with an electron moving to the Internet? As shown in Figure 1, first generation 'IoT' devices are already hitting the market, and that is just the start. It is impossible to imagine all the applications that will evolve once everything is connected to the 'cloud,' but no doubt some of them will be killer indeed.

Figure 1: Internet-enabled 'Things' you can buy today include thermostats (Courtesy of Nest, FitBit and QuickSmart, respectively).

Seeing the future

I was over at a friend's office and noticed he had a spherical lamp, kind of like a crystal ball, emitting a soft red glow on his desk. He said it was a stock market 'Orb,' and sure enough as our meeting progressed, the lamp cycled to green as the Dow Jones Industrial Average (DJIA) turned positive.

His 'Orb' was from a company called Ambient Devices. Their Ambient Stock Orb (see Figure 2) relies on a proprietary data-casting service over a cellphone network. While some data is free (e.g., the DJIA), other data monitoring options require a paid subscription.

Figure 2: Ambient Stock Orb (Courtesy Ambient Devices).

Let's build our own 'Wi-Fi Orb', one that lets you monitor anything you can find on the web, free of charge. Thanks to the march of silicon, all it takes is an MCU, Wi-Fi interface and a bit of software. Figure 3 shows **Parallax, Inc.'s** **Propeller™ Board of Education® (BoE)** on the left and the WIZnet WizFi210 EVB on the right. The large boards make prototyping easy but overstate the silicon and real estate required. A real 'Wi-Fi Orb' would need only the Propeller brain (MCU, serial EEPROM and crystal near just below the red LED) and the postage stamp-size WizFi210.

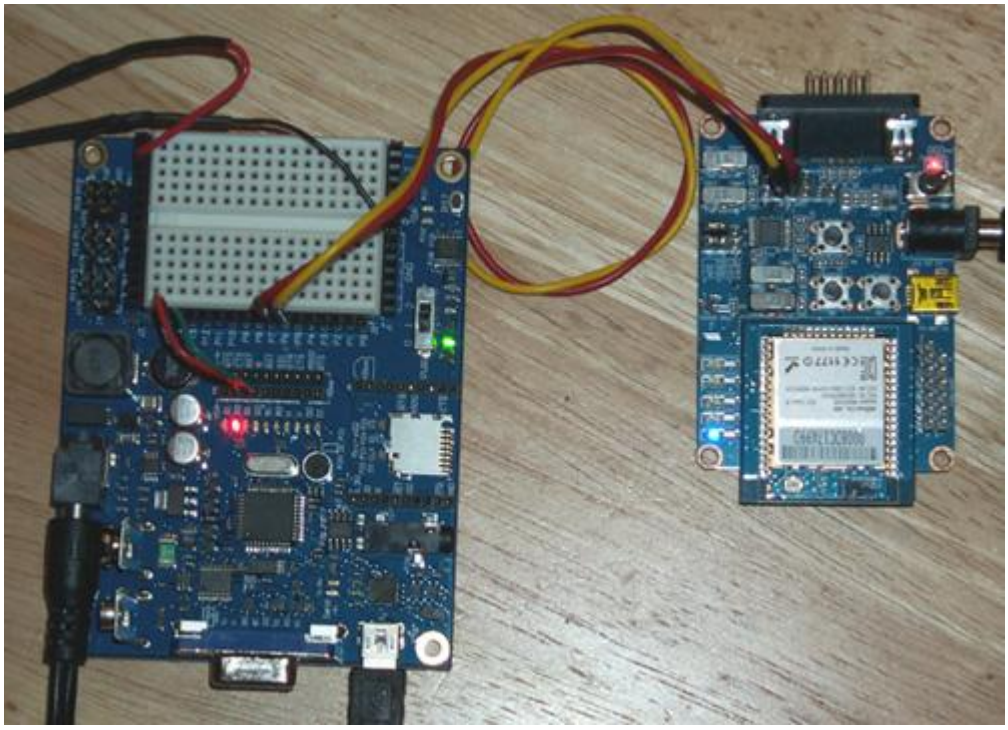

Figure 3: Propeller Board of Education (BoE) on the left and the WIZnet WizFi210 EVB on the right.

Parallax's **Propeller** is a unique multicore flash MCU that is especially useful for quick development. A fast turnaround IDE means edit-compile-download cycles take just seconds. There is also a full library of pre-defined software 'objects', and more in the

Parallax online ‘Object Exchange.’ Objects really help leverage and focus your coding power on what is unique about your application, instead of the boilerplate.

The Wi-Fi part of the equation is handled by a WIZnet WizFi210 embedded Wi-Fi module. It has a standard serial interface that makes for a simple connection to the Propeller. But it’s the module’s built-in intelligence that really makes things easy. Getting online is just a matter of issuing a few high-level commands.

Like any other Wi-Fi device, the first step is provisioning the WizFi210 so that it can access your wireless network. That means setting it up with your WLAN name (i.e., SSID), security regime (WEP, WPA-PSK, and passwords), gateway and DNS server addresses, etc. The WizFi210 has a built-in webpage for wireless provisioning, or it can be done by entering commands via the serial port. Once provisioned, the settings profile is saved to flash inside the WizFi210 module and used for subsequent power-ups.

Data deluge

No doubt you can find the DJIA a million places on the web, but I see it all the time on **www.yahoo.com** (see Figure 4) so why not just grab it from there?

Figure 4: DJIA at www.yahoo.com.

The WizFi210 has dozens of built-in commands that make it easy to navigate the web. They are called ‘AT’ commands and trace their roots back to the dial-up modem days. One nice aspect is the WizFi210 commands and responses are all ASCII, so using a terminal emulator, such as a HyperTerminal, you can simply type in commands and see what happens. Assuming the WizFi210 is provisioned and can reach the access point, here’s all you need to do to get Yahoo’s homepage (commands sent to the WizFi210 shown in bold).

The first step is to associate with the access point using your WLAN name (i.e., SSID).

```
at+wa=your_ap_ssid
        IP              SubNet          Gateway
  10.0.0.20:     255.255.255.0:     10.0.0.1
[OK]
```

Now you can access your DNS server to find the current IP address for Yahoo.

```
at+dnslookup=www.yahoo.com
IP:72.30.2.43
```

[OK]

Define a TCP client connection to Yahoo's IP address, port 80.

at+nauto=0,1,72.30.2.43,80
[OK]

Now open the connection to Yahoo and switch the WizFi210 from command mode to data transfer mode.

ata2
[OK]

At this point, the WizFi210 quits accepting 'AT' commands and turns into a simple serial-to-Wi-Fi translator. Whatever comes in the WizFi210 serial port will be sent via Wi-Fi to Yahoo and vice-versa. Carefully type in (it won't be echoed) GET / (with a space (ASCII $20) between GET and /) followed by a linefeed (CTRL-J on your keyboard i.e., ASCII $0A). After a moment the HTML for Yahoo's homepage will come streaming across the terminal screen. Grabbing the DJIA is as simple as having the Propeller search for the 'Dow' label that marks the location (at a fixed offset) of the DJIA data (i.e., 12,861.2) as shown in Figure 5.

```
    <a class="" href="_ylt=Av_ylG5DT9nyUjKPdMOb85qbvZx4/SIG=11pfmfokb/EXP=
1336584173/**http%3A//finance.yahoo.com/q%3Fs=%5EDJI"><span>Dow: </span>
    <span class="normal-quote y-txt-2">12,861.2 </span>    <em
```

Figure 5: Yahoo homepage HTML.

Less is more

I wrote a short test program to have the Propeller get the DJIA from Yahoo and it worked like a charm (see Figure 6).

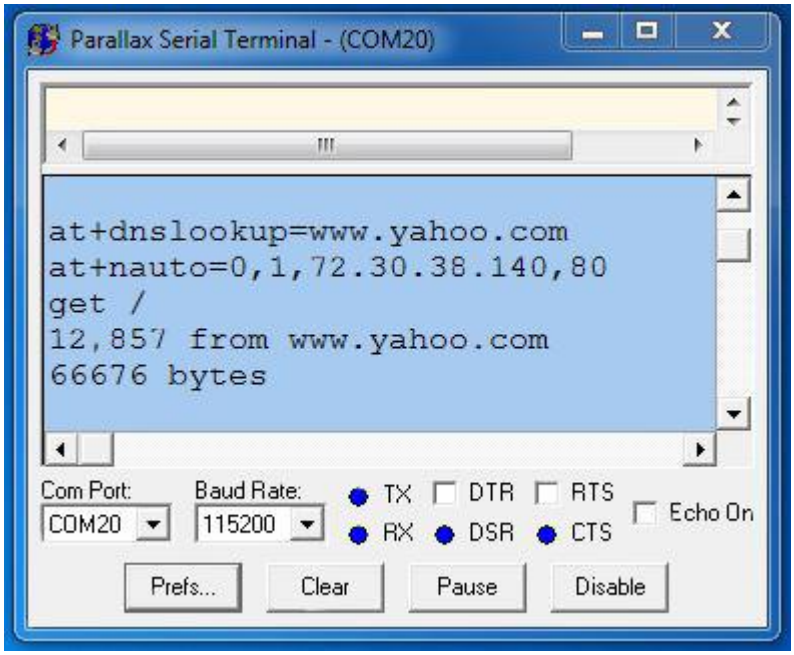

Figure 6: Getting the DJIA by searching for 'Dow' on the Yahoo homepage.

Unfortunately, you have to go through more than 66,000 bytes of Yahoo homepage just to get to the few bytes you want!

While searching for a more practical solution, I came across another Orb design (the 'Tannenbaum Ticker' - **http://www.gadgetgangster.com/news/54/520**) that in turn led me to a finance data source hosted by Google (see Figure 7). If you type 'http://www.google.com/ig/api?stock=.dji' into your browser you will receive a short and sweet summary of the DJIA, or indeed any stock you choose.

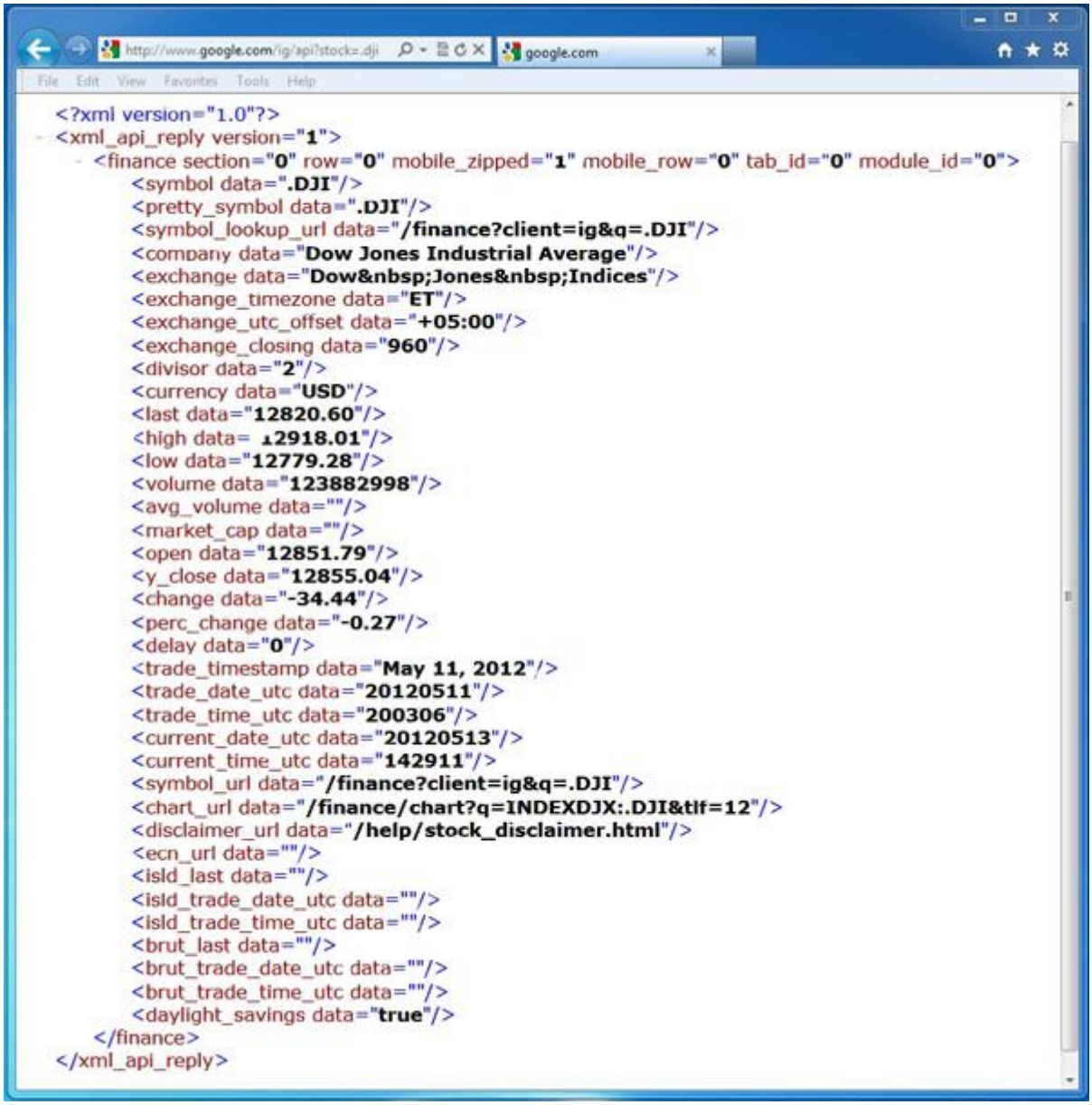

Figure 7: Google Finance API.

I tweaked my test program to use the Google finance data and that cut the transfer to less than 1,000 bytes (see Figure 8). That's still quite a bit of overhead, but a fact of web life. Whenever you get data from a webpage there is a lot of extra HTTP and HTML hidden behind what you see on the screen.

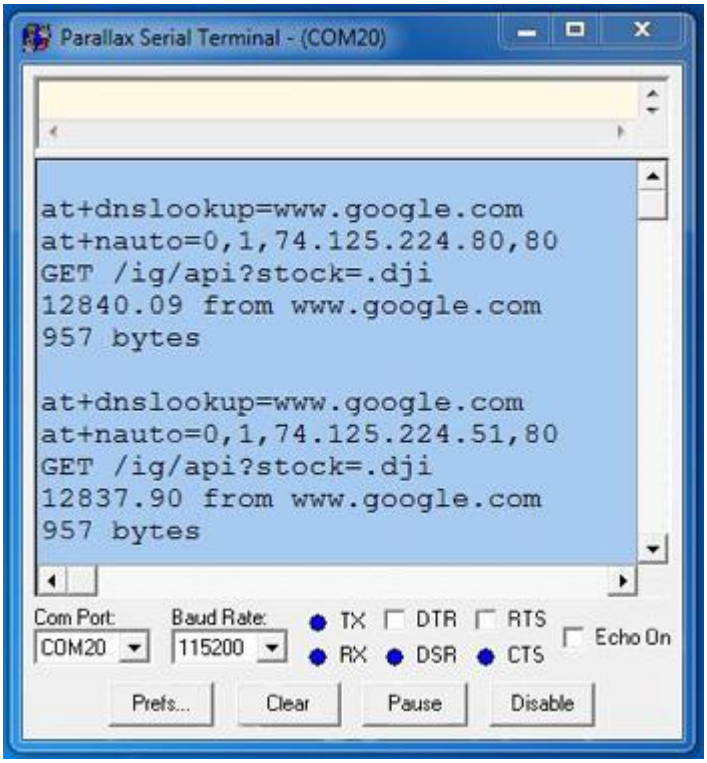

Figure 8: Getting the DJIA using the Google finance API.

Under-the-hood

Let's go through the Propeller program that gets financial data from Google. Be advised, the program makes cursory attempts at error handling but is by no means bulletproof. If it cannot find the access point, or the website is down, or you type something in wrong (SSID, website address, GET request, search key, etc.) the program will hang. Consider it a reminder that every IoT design needs a strategy for dealing with network problems and must avoid the tendency to be overly dependent on the network.

The program starts by bringing in a couple of OBJ(ects) from the Propeller library. Debug talks to the Parallax terminal emulator and WizFi implements the **UART** that connects the WizFi210. Next, variables are defined for storing the target website IP address (ipaddr) and the retrieved data (xmls). Now comes the key ingredient, a DAT(a) structure that defines the path to the desired information such as WLAN SSID, target website, GET command, symbol and xml search keys. Then execution begins and, after initializing the Debug and WizFi objects, the program enters the main loop.

```
OBJ
Debug : "Parallax Serial Terminal"
WizFi : "FullDuplexSerial"
CON
lenipaddr = 18 'max length ip address string incl. cr/lf
lenxml = 50 'max length xml field data
VAR
```

```
byte i,j,k,ipaddr[lenipaddr],xmls[lenxml]
DAT
ssid byte "your_ap_ssid",$d,0 'replace with your wlan ssid
website byte "www.google.com",$d,0 'using Google
get byte "GET /ig/api?stock=" 'finance data api '
symbol byte ".dji",$a,0 'stock symbol ex:.dji,aapl,ibm
numkeys byte 4
keys byte "last",0,”perc”,0,”trade_date”,0,”trade_time”,0'xml keys
PUB WizFiOrb 'program gets xml data item from a webpage
Debug.start(115_200) 'start debug terminal
WizFi.start(6,7,0,57600) 'start uart connected to WizFi
REPEAT ‘forever
```

The first step is to wake up the WizFi210 (it is put to sleep at the end of the loop) and then initiate and confirm successful association (i.e., at+wa) with the WLAN. After that, it is as easy as stepping through the rest of the command sequence shown in the Yahoo example (i.e., at+dnslookup, at+nauto, ata2) and then sending the GET request.

I am tempted to write a function to consolidate the code that handshakes each command (for example: issue command, check response, retry until WizFi says ‘[OK]’). But notice that some commands need special handling (for example: at+psdpsleep returns nothing, +++ needs a delay). Also, it is easier to add debug code for individual commands by keeping things separate. A universal AT command handler will need to accommodate a lot of options including expected response (if any), data and command timeouts, retry count and interval, debug echo on/off, etc.

```
REPEAT
WizFi.str(string("at",$d)) 'dummy command to wakeup WizFi
REPEAT
UNTIL WizFi.rx == $5B 'look for [
UNTIL WizFi.rx=="O" 'retry until WizFi says [OK]
REPEAT
WizFi.str(string("at+wa="))
WizFi.str(@ssid) '(re)try to associate
REPEAT
UNTIL WizFi.rx == $5B 'look for [
UNTIL WizFi.rx == "O" 'associated when WizFi says [OK]
WizFi.str(string("at+dnslookup=")) 'lookup website ip address WizFi.str(@website)
REPEAT
UNTIL WizFi.rx == ":" ': marks start of returned ip address
REPEAT i from 0 to lenipaddr-1
ipaddr[i] := WizFi.rx 'save ip address digits
IF ipaddr[i] == $d 'and terminating carriage return
quit
REPEAT
WizFi.str(string("at+nauto=0,1,"))'create tcp client connection
i:=0
REPEAT UNTIL ipaddr[i]==$d
```

```
WizFi.tx(ipaddr[i]) 'send IP address digits
i++
WizFi.str(string(",80",$d)) 'port 80
REPEAT
UNTIL WizFi.rx == $5B 'look for [
UNTIL WizFi.rx=="O" 'retry until WizFi says [OK]
REPEAT
WizFi.str(string("ata2",$d)) 'connect to website
REPEAT
UNTIL WizFi.rx == $5B ‘look for [
UNTIL WizFi.rx=="O" ‘connected when WizFi says [OK]
WizFi.str(@get) ‘send get request to website
```

After the GET request is sent, the webpage will come streaming into the Propeller. The program searches the incoming data for a match with each of the keys defined in the DAT(a) section and builds a response string (xmls) which contains the retrieved data items, each separated by a carriage return. The keys themselves need only be long enough to be unique, but they can be longer (up to the ‘=’ sign). For example, you could use ‘trade_time’ for readability or ‘de_time’ to save space. Note that the keys in the DAT(a) section must be listed in the order they appear on the webpage. This restriction is imposed so that the search can be done on the fly in a single pass (usually less depending on location of the last search key) without having to store the webpage in RAM. With the query complete, the retrieved data is displayed on the debug terminal.

```
k:=0 'k is offset for each key in keys
j:=0 'j is pointer to build xml string
REPEAT numkeys 'for each key in keys
i:=0 'i counts consecutive key char matches
REPEAT 'search incoming webpage for key
CASE WizFi.rx == keys[k+i] 'count consecutive key char matches
TRUE : i++ 'increment if match
FALSE : i:=0 'clear otherwise
UNTIL keys[k+i+1] == 0 'until key matched
k:=k+i+2 'point to next key
REPEAT 'find opening doublequote
UNTIL WizFi.rx == $22
REPEAT 'gather desired data
xmls[j]:=WizFi.rx
j++
UNTIL xmls[j-1] == $22 'until closing doublequote
xmls[j-1]:=$d 'put carriage return after data

k:=0 'k displays key chars
j:=0 'j displays data chars
Debug.str(string($d,"Symbol = "))
Debug.str(@symbol)
REPEAT numkeys 'for each key
Debug.str(string(" "))
```

```
REPEAT 'display key
Debug.char(keys[k])
k++
UNTIL keys[k] == 0
Debug.str(string(" = "))
REPEAT 'display data
Debug.char(xmls[j])
j++
UNTIL xmls[j-1] == $d
```

Now it is time to go to sleep until the next sample. The +++ followed by a one second delay is a special escape sequence recognized by the WizFi210 as a request to exit data mode and return to command mode. Then AT commands are issued to disassociate from the WLAN (ath) and enter sleep mode (at+psdpsleep). Finally, the Propeller puts itself to sleep until the next sample. The Google data generally seems to update every couple of minutes, but sometimes more frequently, so 90 seconds seems like a reasonable compromise.

```
REPEAT
WizFi.str(string("+++")) 'request WizFi to command mode
waitcnt(clkfreq + cnt) 'wait for WizFi to command mode
WizFi.str(string("at",$d)) 'confirm WizFi to command mode
REPEAT
UNTIL WizFi.rx == $5B 'look for [
UNTIL WizFi.rx=="O" 'retry until WizFi says [OK]
REPEAT
WizFi.str(string("ath",$d)) 'hang-up i.e. disassociate
REPEAT
UNTIL WizFi.rx == $5B 'look for "["
UNTIL WizFi.rx=="O" 'retry until WizFi says [OK]
WizFi.str(string("at+psdpsleep",$d)) 'put WizFi to sleep
waitcnt((clkfreq*45)+cnt) 'put MCU to sleep
waitcnt((clkfreq*45)+cnt) 'put MCU to sleep
```

Figure 9 shows the program in action. Using different keys you can gather any, or all, of the information you find useful.

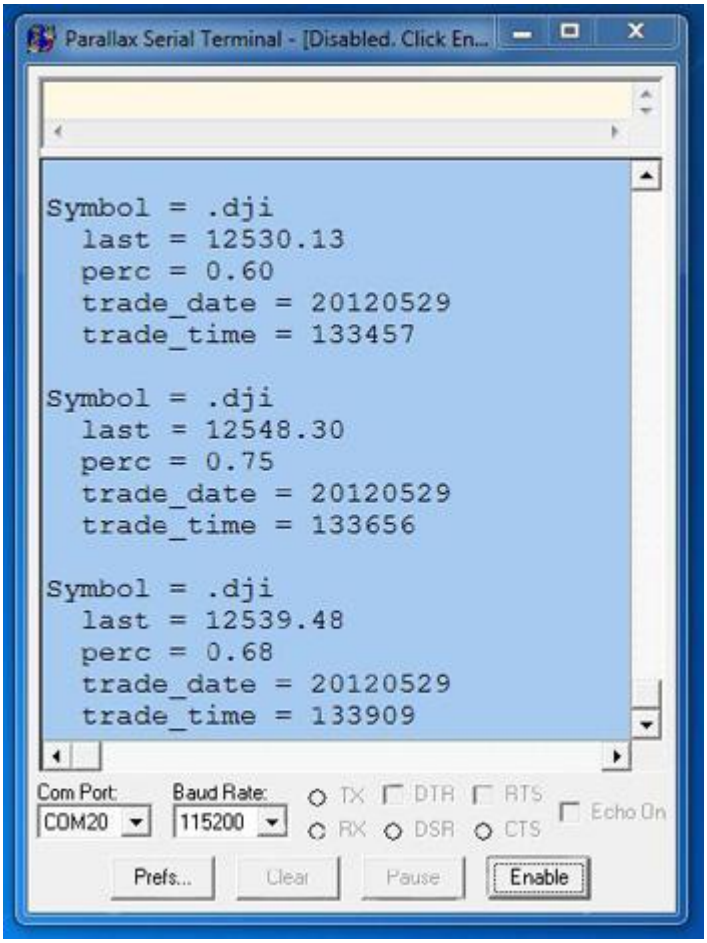

Figure 9: Financial data courtesy of Google.

Now comes the fun part, which is deciding how to ‘ambientize’ the data. The Propeller BoE has some red and green LEDs, so by setting the xml key to <perc and adding a bit of code like...

CASE xml[0] == “-“ 'percent change negative?
TRUE : OUTA[15..16] := %01 ‘red led on, green off
FALSE : OUTA[15..16] := %10 ‘green led on, red off

…voila, you have got a mini-me Orb!

The BoE offers quite a few more interesting options. There is a VGA port so you could hook it to a screen or TV. That might seem like it is defeating the purpose of ‘ambience’ (i.e., not having to stare at a screen). But instead of the typical numbers and graphs, you could have a colorful DJIA ‘light show’ conveying the market vibe in a more creative way.

Also intriguing are the BoE audio options including a microphone and stereo headphone jack. When the market is up, a few bars of Wagner could sing its praises. The BoE has an SD Card slot, so I could grab a .WAV file on the PC for playback by the Propeller through the headphone jack. Or I can use the microphone to record my

own sound effects, an amusing prospect indeed.

Semper Wi-Fi

At the reservoir in Los Angeles in 1913, William Mulholland, before a crowd of thousands, turned to one of his engineers at the last pump and gave the indication to turn the great wheels, open the floodgates, and down the water came, dancing and sparkling in the sun, and started spilling into the reservoir, and Mulholland turned to the dignitaries and said, 'There it is, gentlemen, take it.'
T. H. Watkins (**http://www.pbs.org/weta/thewest/program/episodes/eight/takeit.htm**)

You can see that the basic hardware building blocks for the 'IoT' are here today. Now comes the exciting part. What can formerly isolated devices do with the full knowledge of the web at hand? Who owns the information? Who guarantees it will be accurate and available (and who is liable if it isn't)? What about security and privacy? As the IoT gadgets go online, the answers will become clear.

Like water for a thirsty city, the fact Wi-Fi is already in most homes and businesses is a compelling advantage. No great wheels need be turned, the floodgates are already open. There Wi-Fi is, take it.