Small Basic and the Raspberry Pi

By Philip Munts Munts Technologies

Originally published here on the Microsoft TechNet Wiki on 9 March 2015

Introduction

Microsoft Small Basic is a fun and easy to learn programming language. But for the *Embedded Systems* world in which I work and play, Small Basic doesn't offer much. With the LitDev extension (<u>http://litdev.co.uk</u>), you can issue commands to something connected to a serial port or a USB port. However, serial ports can be finicky and there is a steep learning curve for USB devices, even when using a library like LitDev.

There is, however, one facility built into Small Basic that can be used for controlling external devices: The **Network** object. The **Network** object offers two methods for fetching web pages. The **Network**. *DownloadFile(URL)* method stores the web server response to a local file, and the **Network**. *GetWebPageContents(URL)* method returns the web server response as a string.

Digression: A Short History of The Internet

What we now call the Internet started out as research conducted by the U.S. Department of Defense Advanced Research Projects Agency (ARPA), called ARPANET. It began operation in 1969 with 4 nodes. I will call this the *Internet First Generation*: It connected *research facilities*.

The U.S. National Science Foundation began funding other non-military research networks (CSNET and NSFNET) using the same protocols as ARPANET in 1981. These networks interconnected with ARPANET using the Internet Protocol, and were made available to universities. The entire network began to be called *The Internet*. It still connected *research facilities*, but definition of same was increasingly flexible.

In the late 1980's, private network operators also began connecting to The Internet and offering service to ordinary consumers. The primary services those days were email, file transfer services, and discussion forums. By the mid 1990's, it was possible for an ordinary person to obtain an email address, search for and download files, and read and post comments on various online forums. I will call this the *Internet Second Generation*: It connected individual *persons* to *information sources*.

After 1994, three things happened that caused the Internet to grow explosively. First, the WWW (World Wide Web) service was introduced to the Internet. The WWW was simply a new file delivery service, but with files formatted for interactive viewing using a new markup language called HTML, delivered with a new efficient transfer protocol called HTTP, to new interactive client programs called *browsers*.

The second thing that happened was the introduction of so-called *broadband* high speed Internet service, which was much faster than the previous telephone line dial up service. Broadband service can be delivered via specially conditioned telephone lines (DSL), cable TV cables, optic fibers, terrestrial radio channels, and satellite links.

The third thing that happened, just before the new millennium, was the invention of *wireless Ethernet* and the rapid deployment of wireless hotspots. Now it was possible to connect to the Internet without cables. This was still the second generation, though, *people* connecting to *information sources*.

Even the earliest web servers from 1994 on had an interesting capability called CGI (Common Gateway Interface). This allowed a web browser ask the web server to run an external program and return the results. For example, fetching http://someserver.com/cgi-bin/someprogram from your web browser would ask the computer serving http://someserver.com to run the program file called *someprogram* stored in the *cgi-bin* web server file system directory. (*cgi-bin* was recognized by the web server to be a special magic directory containing program files instead of HTML text files). The program file *someprogram* was expected to generate some text response, probably formatted in HTML, which was then delivered back to your web browser.

The CGI facility was intended for generating customized HTML on the fly (dynamic web pages). But we realized very early on that CGI programs could control external devices as well. Interestingly, there is a high level of abstraction here: You cannot determine from the URL whether <u>http://someserver.com/cgi-bin/someprogram</u> is running on a supercomputer in an underground data center, or whether it is running on a single chip microcomputer inside a light bulb. This is the embedded systems restatement of the famous *New Yorker* cartoon quip "*On the Internet, nobody knows you're a dog*". Because of security weaknesses, the original CGI facility is not used much on ordinary web servers anymore. But the principle lives on in smaller devices.

Ubiquitous wireless internet has enabled what I will call the *Internet Third Generation*, also known as the *Internet of Things* or *IoT*. With the Internet of Things, many, if not most, of the nodes on the Internet will be *devices* rather than *people* and *information sources* like the second generation. These devices can act as mini-web servers, providing information interactively to a person, or they may communicate with each other purely in a machine-to-machine fashion. Manufacturers of almost every kind of consumer goods are busy incorporating Internet services of one kind or another into their products.

How to control a device from Small Basic

Imagine we would like to turn on and off a light from a Small Basic program: A classic embedded systems task.

There are several ways to implement such a system. In the old days, one would build a microcomputer with an RS-232 serial port into the light controller. You then connect a cable from the serial port on the device to the serial port on the Windows computer, possibly with a 25-9 pin adapter, crossover cable, null modem adapter, etc. *Then* you would try to figure out the baud rate, parity settings, and which serial port was which on your computer. If you were determined you would eventually get it working. After installing the LitDev extension for serial port support, you can now control the light from Small Basic by sending commands out the serial port to the light controller.

The cable challenges and bit rate limitations of serial ports led to the introduction of USB (Universal Serial Bus), which standardized the cabling and increased the baud rate by orders of magnitude. But USB brings its own peculiar challenges, including the fact that devices can wind up at different addresses depending on which port they are plugged into. Still, one can now build a microcomputer with a USB port into the light controller and eventually get it working from Small Basic using the LitDev extension.

A third option is to build a microcomputer with either wired or wireless Ethernet into the light controller. If the microcomputer runs a mini-web server with something like CGI, one can easily issue commands to it from a web browser, *or* from Small Basic using the *Network* object.

The Raspberry Pi

There are *hundreds* of microcomputer boards available that can be used for such a device server. We can narrow the field by concentrating only on those that are well supported and cost less than \$30. Certainly among the most popular in this range is the Raspberry Pi (<u>http://www.raspberrypi.org</u>), with more than five million sold. The Raspberry Pi has been available in five production models (A, B, A+, B+, 2B). The B and B+ models have two or four USB ports and built-in wired Ethernet. The A and A+ models have only one USB port and no built-in networking, but cost less and consume considerably less power. The original A and B models have a 26-pin expansion header. The A+ and B+ models have a 40-pin expansion header (26 pins of which are the same as the A and B). The very latest Raspberry Pi 2 Model B is mechanically interchangeable with the B+ but offers a faster quad-core processor and twice the RAM.



Figure 1 – Raspberry Pi Model A+

The Raspberry Pi Model A+ is the least expensive and most power efficient member of the family, with a suggested retail price of USD \$20. The actual street price runs \$25 at time of writing. You will need to add a wireless Ethernet adapter (the Edimax EW-7811U is strongly recommended), a micro-SD card for booting the operating system, and a micro-USB power source, probably bringing the total system price over USD \$30.

Interface connectors on the A+ are a micro USB slave port (+5V power supply only), a couple of flat cable connectors for LCD display and camera modules, an HDMI video output jack, a 3.5 mm Audio/Video jack, one USB host port, and the 40 pin expansion header J8. For this article, a 2 gigabyte micro-SD card and an Edimax EW-7811U wireless Ethernet adapter are permanently installed. Figure 1 above shows the Raspberry Pi A+ used for this article. Along the left edge of the board, the micro-SD card is barely visible. The wireless network adapter is plugged into the USB host port on the right edge of the board. The 40-pin expansion header J8 is visible along the top edge of the board.

Raspbian Linux Operating System

The usual operating system for the Raspberry Pi is Raspbian Linux, which is essentially Debian Linux recompiled for the processor in the Raspberry Pi. It must installed onto the micro-SD card, which is then inserted in the Raspberry Pi. The micro-SD card must be reformatted with two partitions: A small DOS FAT file system partition which contains the boot files, and a larger Linux OS file system partition which contains the rest of the operating system.

Raspbian Linux is both the strength and the weakness of the Raspberry Pi. With its Unix heritage, it is unbelievably powerful for such a small computer. However, for a headless (no keyboard or display) embedded system, Raspbian Linux leaves a lot to be desired. Installing, configuring, and administering it is not trivial. The operating system partition can be corrupted if the system is not shutdown in an orderly fashion. Furthermore, the flash memory cells on the micro-SD card are not eternal and will eventually wear out. SD cards are cheap and easy to use, but they were never designed to serve for operating system partitions.

<u>MuntsOS</u>

MuntsOS is an extremely reduced distribution of the Linux operating system. It is delivered as a single Linux kernel image file with the main or root file system built into it. When the kernel boots on a target computer, it unpacks a fresh copy of the root file system into RAM and uses it there. The Raspberry Pi boots from a micro-SD card, but after it has finished booting it no longer accesses the micro-SD card. This has several advantages. First, the RAM file system is very fast, much faster the micro-SD card. Second, it cannot be corrupted by pulling power without performing an orderly shutdown. Every time the Raspberry Pi boots, it unpacks a new and fresh copy of the root file system from the kernel image. Third, it does not routinely write to the micro-SD card, which helps avoid wearing out the flash memory cells in the card.

The Thin Server

For some time I have been experimenting with a system design pattern I call the *Thin Server*. It is essentially an inversion of the *Thin Client* concept that Sun Microsystems and other companies began marketing at the close of the 20th century. The idea behind the JavaStation and other thin client devices was to relocate as much as possible the software complexity and processing power from the client computer to the server computer. The client computer was to be little more than a network interface for the user's mouse, keyboard, display monitor, and audio. All of the user's application software would run on the server, where it could be centrally managed.

In contrast, a *Thin Server* is little more than a network interface for a single I/O device. Ideally, a *Thin Server* will be built from a cheap and ubiquitous network microcomputer like the Raspberry Pi. The software must be easy to install from a user's PC or Mac without requiring any special programming tools. It must be able to run headless, administered via the network. It must be able to survive without orderly shutdowns, and must not write much to flash media. It must provide a network based API (Application Programming Interface) using HTTP as a lowest common denominator.

MuntsOS, with its operating system running entirely from RAM, serves well for the *Thin Server*, and in fact the two concepts have evolved together over the past few years.

<u>GPIO</u>

General Purpose Input/Output pins are connections available on microcomputers like the Raspberry Pi. GPIO pins carry digital signals (on/off or high/low or binary 1/0) between the microcomputer and the external device. A GPIO pin can be configured as an *Input*, meaning the signal flows from the external device to the microcomputer, or as an *Output*, meaning the signal flows from the external device.

The Linux operating system numbers GPIO pins from 0 (GPIO0, GPIO1, ...). Note that the Linux GPIO pin number usually does not correspond to any physical pin number. Some processors may have hundreds of GPIO pins available. Linux treats each pin as a separate I/O device. The Raspberry Pi Model A+ brings out a number of GPIO pins, along with power and ground to the J8 expansion header. See Figure 2.

Pin#	NAME		NAME	Pin‡
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1, I2C)	$\bigcirc \bigcirc$	DC Power 5v	04
05	GPIO03 (SCL1, I2C)	$\bigcirc \bigcirc$	Ground	06
07	GPIO04 (GPIO_GCLK)	\odot	(TXD0) GPIO14	08
09	Ground	00	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	00	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	00	Ground	14
15	GPIO22 (GPIO_GEN3)	00	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	\mathbf{O}	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	$\bigcirc \bigcirc$	Ground	20
21	GPIO09 (SPI_MISO)	\odot	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	\odot	(SPI_CE0_N) GPIO08	24
25	Ground	\mathbf{O}	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	\odot	(I2C ID EEPROM) ID_SC	28
29	GPIO05	00	Ground	30
31	GPIO06	00	GPIO12	32
33	GPIO13	00	Ground	34
35	GPIO19	00	GPIO16	36
37	GPIO26	00	GPIO20	38
39	Ground	00	GPIO21	40

Raspberry Pi B+ J8 Header

Figure 2 – J8 Expansion Header

The Raspberry Pi GPIO pins are 3.3V signals, meaning outputs switch between ground and +3.3V, and inputs must not be connected to voltages below ground or above +3.3V. They cannot deliver very much power, and will almost always need some sort of driver or amplifier for real device loads such as motors and lights. It is barely possible to drive a single LED (Light Emitting Diode) from a GPIO pin, but this cannot be recommended. Use some sort of driver or amplifier such as the ULN2003A.

Note that some of the Raspberry Pi expansion header pins have alternate functions. For example, *GPI014* and *GPI015* can also be used for a serial port. Such pins may not be available for GPIO.

For experimentation, it is convenient to connect the Raspberry Pi to a solderless breadboard with an adapter like the <u>Adafruit T-Cobbler Plus</u> to the J8 expansion header. Figure 3 shows the Raspberry Pi Model A+ connected to a solderless breadboard.



Figure 3 – Breadboard Connection

A Thin Server for GPIO

Setting up a Raspberry Pi GPIO Thin Server is very simple. Just download the zip file at:

http://tech.munts.com/rpi/GPIO_server/muntsos/muntsos-gpio-server-RaspberryPi.zip

and extract it to a freshly FAT32 formatted micro-SD card. It includes support for the Edimax EW-7811U wireless network adapter (and others using similar Realtek chip sets). The zip file contains the MuntsOS kernel image file plus a number of other binary and text files the Raspberry Pi requires for booting. Then edit the text file:

autoexec.d/00-wlan-RaspberryPi

on the micro-SD card to configure the thin server's host name, network SSID and encryption key. Eject the micro-SD card from your PC or Mac, and insert it into the socket on the bottom of the Raspberry Pi board. When you power on the Raspberry Pi, it will boot MuntsOS from the micro-SD card and attach to the wireless network.

Discovering the IP address the Raspberry Pi has acquired may prove tricky. The easiest way to find its address is with the *Apple Bonjour MDNS* (Multicast Dynamic Name Service) service, which for Windows is part of the iTunes package but can be installed separately. The Raspberry Pi GPIO Thin Server will register itself as *hostname.local*, where *hostname* is whatever name you wrote in *00-wlan-RaspberryPi* (or *RaspberryPi-MuntsOS.local* by default). If the DHCP server assigns the Raspberry Pi a host name, the name *newhostname.local* will also be registered, where *newhostname* is the first part of the newly assigned host name.

For example, if you write *foo* as the host name in *00-wlan-RaspberryPi*, and the local DHCP server subsequently assigns the host name *bar.munts.net*, the Raspberry Pi will respond to *both foo.local* and *bar.local*.

Without MDNS finding the IP address will be more difficult. If you have administrative access to the DHCP server for the wireless network, you may be able to retrieve the IP address assignment from it. If worst comes to worst, you can attach a keyboard and display (HDMI) to the Raspberry Pi and log in (user *root*, password *default*) and run the *ifconfig* command to find out what its IP address is.

GPIO Server API

The web server running on the GPIO Thin Server responds to requests of the following forms:

http://servername:8083/GPIO/ddr/n,s
http://servername:8083/GPIO/get/n
http://servername:8083/GPIO/put/n,s

Set data direction (0=input, 1=output)
Get pin (state 0=OFF, 1=ON)
Put pin (state 0=OFF, 1=ON)

where *servername* is the IP address or host name, *n* is the GPIO pin number (as numbered by the Linux operating system), and *s* is the new state 0 or 1. As an example, the following sequence of requests sent from a browser to a Raspberry Pi GPIO Thin Server will configure *GPIO26* (which happens to be pin 37 of the J8 expansion header) as an output and turn it on:

http://foo.local:8083/GPIO/ddr/26,1 http://foo.local:8083/GPIO/put/26,1

Putting It All Together



fritzing



Figure 4 shows a <u>Fritzing</u> diagram for connecting a single LED to the Raspberry Pi A+ J8 expansion header. The *GPIO26* signal drives a ULN2003A low side switch, which serves to amplify the current enough to turn on an LED. The ULN2003A has been around a long time (the particular unit I used for this article was manufactured in 1978!) but is still very useful. It is a low side switch device, meaning it pulls current to ground. A load, such as the LED, must be connected between a positive supply voltage (+3.3V from the Raspberry Pi in this case) and one of the ULN2003A outputs. The flat side of the LED and/or shorter lead connects to ULN2003A pin 10 and the round side and/or longer lead connects to the 220 ohm resistor.

A sample program for turning the LED on and off from Small Basic is available, with publication ID *PCD388-0*. The sample program is divided into three main sections.



Figure 5 – LED Control Program

The first section lays out the GUI (Graphical User Interface), placing a server prompt, a text box for the server name, and a button initially labeled "Connect":

' Lay out GUI

```
GraphicsWindow.Width = 300
GraphicsWindow.Height = 130
GraphicsWindow.Title = "LED Control Program"
GraphicsWindow.Show()
GraphicsWindow.DrawText(40, 30, "Server:")
```

```
ServerBox = Controls.AddTextBox(100, 28)
ConnectButton = Controls.AddButton("Connect", 120, 80)
```

The second section is the button handler subroutine, which contains the bulk of the program logic:

```
' Everything happens in the button handler subroutine
```

```
Controls.ButtonClicked = ButtonHandler
```

```
Sub ButtonHandler
  b = Controls.LastClickedButton
  c = Controls.GetButtonCaption(b)
  If c = "Connect" Then
    servername = Controls.GetTextBoxText(ServerBox)
    pin = 26
    direction = 1
    PutDDR()
    data = 0
    PutGPIO()
    Controls.SetButtonCaption(b, " ON
                                          ")
  EndIf
  If c = " ON
                 " Then
   pin = 26
    data = 1
    PutGPIO()
    Controls.SetButtonCaption(b, " OFF ")
  EndIf
  If c = " OFF " Then
    pin = 26
    data = 0
    PutGPIO()
                                          ")
    Controls.SetButtonCaption(b, " ON
  EndIf
EndSub
```

The last section of the program contains subroutines that wrap the HTTP server API:

```
' Write GPIO data direction bit
   Parameter variables:
     servername -- IP address or host name
     pin -- GPIO pin number (2-26, as numbered by Linux)
     direction -- 0 for input, 1 for output
.
Sub PutDDR
 webpage = Network.GetWebPageContents("http://" + servername + ":8083/GPIO/ddr/" + pin +
"," + direction)
 If Text.StartsWith(webpage, "DDR" + pin + "=") Then
 Else
    GraphicsWindow.ShowMessage(webpage, "GPIO Server Error")
 EndIf
EndSub
' Read GPIO data bit
   Parameter variables:
      servername -- IP address or host name
     pin -- GPIO pin number (2-26, as numbered by Linux)
     data -- Set to 0 or 1
Sub GetGPIO
 webpage = Network.GetWebPageContents("http://" + servername + ":8083/GPIO/get/" + pin)
  If Text.StartsWith(webpage, "GPIO" + pin + "=") Then
   data = Text.GetSubTextToEnd(webpage, Text.GetIndexOf(webpage, "=") + 1)
 Else
    GraphicsWindow.ShowMessage(webpage, "GPIO Server Error")
 EndIf
EndSub
' Write GPIO data bit
    Parameter variables:
      servername -- IP address or host name
     pin -- GPIO pin number (2-26, as numbered by Linux)
     data -- 0=OFF (low), 1=ON (high)
Sub PutGPIO
 webpage = Network.GetWebPageContents("http://" + servername + ":8083/GPIO/put/" + pin +
"," + data)
 If Text.StartsWith(webpage, "GPIO" + pin + "=") Then
 Else
   GraphicsWindow.ShowMessage(webpage, "GPIO Server Error")
 EndIf
EndSub
```