

EMBEDDED IN THIN SLICES

The Internet of Things (Part 2)

Connect Wirelessly with a Microcontroller

In the second part of this article series, Bob provided tips on choosing a carrier for an 'Net-connected embedded system. In this article he details how to connect simple devices wirelessly to the Internet.

By Bob Japenga (US)

In our first installment, we talked about the various ways we have connected our embedded systems wirelessly to the Internet. I covered the various decisions that need to be made in choosing a carrier for your embedded system. This month we will look in detail at how we have connected simple devices wirelessly to the internet.

Yesterday, we had a new potential client fly in to see us to discuss their need to create an Internet of Things (IoT) device for their very non-electronic company. This company has been in business for over 50 years making and innovating extremely low-tech products. Now they wanted their extremely low-tech product to be wirelessly connected to the Internet. They came to us with a prototype developed by their general manager using an Arduino and a development kit. This was a man who had no electrical engineering background. It was quite impressive. He did not have any complications sending the data to the cloud. He did struggle with designing the sensor to obtain the data he desired. With the feasibility behind them and after becoming knowledgeable enough to know what it takes to join the IoT revolution, they wanted us to

turn it into a product. This shows how easy it is to take a simple microcontroller, talk to a complex cell module, and create a device that will soon join the IoT revolution.

Many of our systems are similar to theirs and use a very simple microcontroller with very little memory. Some cell modules offer a good assortment of options for connecting to the web. Since some of the module manufacturers we use require an NDA to obtain their documentation, we will only talk about our experience with a Swiss company called u-blox who freely publishes their technical documentation on-line. In particular we will talk about the features of the LISA C200, which supports CDMA. Other modules have similar functionality. **Figure 1** shows the basic system architecture that we will be reviewing this month.

AT COMMAND SET

Most of the module manufacturers provide an AT command set to configure the module and initiate communications over the network. Developed by Dennis Hayes for the 300-baud Hayes SmartModem (ah, those were the days!), this simple command and response



protocol is used on a lot of communications devices. This is the primary way a small microcontroller will talk to the cell module. With the AT command set, the microcontroller can easily use: FTP, HTTP, UDP, TCP/IP, and SMS. Let's look briefly at how easy this is to do.

FILE SYSTEM

All of the networking commands can use an on-board flash file system on the cell module. This flash file system has about 1 MB of disk space available for you to use. There are rudimentary file system commands to allow you to read, write, get stats, or delete (ASCII or binary) files from this flash file system. Since the files are read over a serial port it is not like reading from a classic disk controller. Any read over the serial port could be corrupted. Thus we always provide some method of validation that what you have read or written over the serial port is indeed what you intended.

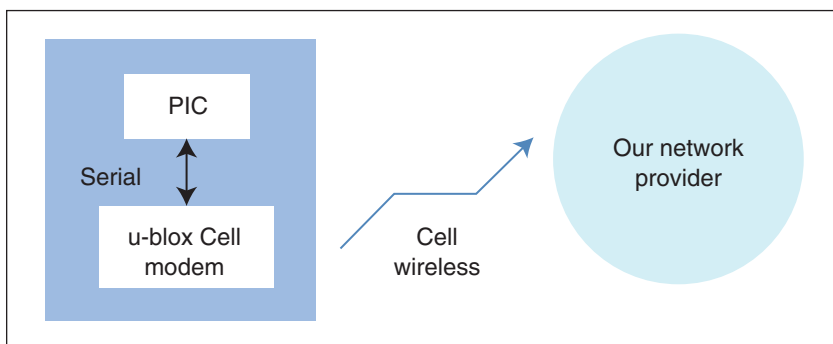
FTP

FTP is a file transfer protocol that allows you to send files unencrypted (including the password) over the Internet to and from your system. If you want to send or receive files over the Internet to or from an FTP server, you need an ftp client on your device. The cell module contains that ftp client for you and provides an AT command interface to that client.

There are just a few AT commands required to send a file. The first one defines the username and password as well as the URL (either named or IP address). Then you log in with another AT command. You then can either obtain one file from the server (get) or send one file to the server (put). Each command provides a response in a file on the file system as to the success or failure of the operation. Additional commands may be required to obtain the status of the network. Once you have completed sending or receiving files, you can log off with another AT command.

The LISA C200 supports an FTP client which supports the FTP commands listed in **Table 1**. A number of standard FTP functions are not supported (like verbose) because they don't apply. There are others that would be nice to have (like the append command or mput and mget for sending and receiving multiple files) but are not provided.

In today's world, FTP should rarely be used because it is so insecure. We had one customer many years ago who, when we offered both FTP and secure FTP (SFTP), wanted us to delete FTP off his devices because he did not want his customers to use



it. If you are on a private network, FTP would work and maintain the security you need.

FIGURE 1
The basic system architecture

HTTP

The cell module supports the following HTTP methods: GET, PUT, POST, HEAD, and DELETE. The process is quite simple and does not require a lot of code to implement on your little microcontroller. You can set up a number of profiles which contain exactly where the data is going with a series of AT commands. Once a profile is set, you can issue another AT command to initiate the specific HTTP method associated with that profile. For example, if you are POSTing to multiple URLs, you can set up a separate profile for each. The response from the server comes back as a file on the local flash file system that you specified. You can read the file using the file system AT read command.

As with FTP, it would be ideal if HTTPS (secure HTTP) was supported. Without HTTPS, on one project we needed to encrypt everything that is included in the payload of the HTTP request and decrypt everything that comes back. For this application we used Advanced Encryption Standard (AES) 128, which uses a symmetric private key algorithm.

AES 128 is a specific instance of an encryption standard established by the National Institute of Standards and Technology

TABLE 1
The FTP client supports several ftp commands

u-blox FTP Command	Windows FTP	Description
0	Quit	Log out
1	User	Log in
2	Delete	Delete file from the server
3	Rename	Rename file on the server
4	Get	Retrieve file from the server
5	Put	Send a file to the server
8	Cd	Change the working directory
10	Mkdir	Make a directory on the server
11	Rmdir	Remove directory on the server
13	N/A	Obtain stats on a file on the server
14	Dir	List the filenames on the server



ABOUT THE AUTHOR

Bob Japenga has been designing embedded systems since 1973. In 1988, along with his best friend, he started Micro-Tools, which specializes in creating a variety of real-time embedded systems. With a combined embedded systems experience base of more than 200 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in many areas of embedded systems and motion control. You can reach him at rjapenga@microtoolsinc.com.

in the United States. It uses a 128-bit key. The encryption and decryption algorithms are public knowledge. There are thus 3.4×10^{38} possible keys. If you tried to decrypt the encrypted data by brute force by trying a different key a trillion times per second, it would still take 1,018 years. To date, no one has found any other way to crack the code other than brute force.

TCP/IP

On one system, we initially we had a diagnostic mode where we were sending the contents of a screen every 10 seconds (the embedded version of Google Hangout) using HTTP. When we specified it, we knew that HTTP and the server could handle the data throughput, but the cell module just could not keep up with sending 1k of data every 10 seconds in separate HTTP POSTs. As a result we implemented the feature using the cell module's TCP/IP capability.

The cell module provides AT commands to create the socket, to open the socket, to send the data, to close the socket, and to set options for the socket. When a message comes back from the socket, an asynchronous response is provided so your microcontroller does not need to poll the server for the results.

The cell module can handle up to 6 TCP/IP sockets open at a time. It also supports sending and receiving binary or ASCII data.

SMS

A Short Message System (SMS)—or what we call “texting”—client can be invaluable if you wish to provide a relatively responsive system while keeping your data plan costs low. For example, let's say that you want to send up data once per day. Occasionally you would also like to be able to command the device to do something and have it respond within a few seconds. Since the overhead of an HTTP POST can be greater than 1 KB, checking the server every minute for some kind of request would use up more than one megabyte of data per day from your data plan. With SMS, the server can just send the embedded system an unsolicited message with specific instructions. The text will be stored in a file and the presence of that file can be polled (at no cost to your data plan).

A couple of AT commands will set up

your cell modem for sending or receiving text messages. The microcontroller can periodically send another AT command to see the file you specified to receive SMS messaging with no cost to your data plan except for the SMS message. If you plan to use this approach, make sure your data plan supports SMS.

PROBLEMS WITH THIS APPROACH

No cell module will implement the full HTTP or FTP specification. Let's review some of the more serious shortcomings that we uncovered.


With the u-blox module, we found that both the FTP and HTTP implementation did not support sending multiple files up with a single command. This actually resulted in causing us to re-write some of our server code. So if you are going to be using your microcontroller to talk to an existing cloud server that may be expecting multiple files, you are out of luck.

Another shortcoming mentioned is that the HTTP and FTP responses are slower than when you connect these cell modems via the Point to Point Protocol (PPP) (if the cell modem supports PPP) or Ethernet (some cell modules simulate an Ethernet connection eliminating the overhead of PPP).

We also found that we wanted to tweak the HTTP headers and this was also not allowed on the u-blox module. We used a serial port (although all of the cell modems we have used have USB) on one of the two u-blox implementations. Even at 115,200 baud, this is a relatively slow interface for accessing 0.5-MB files. If your microcontroller can support USB, then I would recommend that you talk to the modem via USB.

Finally, we mentioned the lack of HTTPS and SFTP support previously. As one of our engineers said, HTTP is being deprecated across the Internet. Even Google search engine ratings are lowered if your web site doesn't support HTTPS. Not that this applies to our M2M world, but it does speak to a growing trend away from HTTP.

ON TO CERTIFICATION

Connecting your device to the Internet has become much simpler with the design of a number of the cell modules on the market. Using simple AT commands over a serial port or a USB port, you can connect your device to the Internet and join the Internet of Things revolution. Next time we will discuss certification options for your embedded systems. Certification is a big topic so I can guarantee that we will approach it in thin slices. 

SOURCES

LISA-C200 CDMA 1xRTT Module
ublox | www.u-blox.com

