The AT+P Protocol: An extension of the Hayes AT command set for embedded Internet connectivity

By Alan Singer

Many embedded systems could benefit from using the Internet for communication and for data collection. Unfortunately, many of these systems were not originally designed with this in mind. They lack the processing power and the memory needed to support an Internet protocol stack, and they may lack a modem or an Ethernet interface. In addition, many embedded system developers don’t currently have the expertise to design the hardware and software needed to provide Internet connectivity. But what if a simple, cost-effective solution could be found for adding Internet connectivity to existing embedded systems? In this article Alan describes an extension to the Hayes AT protocols that facilitates embedded connectivity.

There are several methods that you could use to add an Internet interface to an existing embedded system:

- If there are adequate processing and memory resources in the system, you could purchase and add Internet Protocol software and email or http software to the existing system software.
- If there is no additional processing power available, you could redesign the system to employ a faster MPU (or an Internet-enabled MPU) and then buy (or develop) the required software. However, this would probably require a major redesign of the printed board(s) and might even require the use of a new operating system and debugging environment. It would also require the purchase (or development) of software libraries costing tens of thousands of dollars. You would also have to add more memory to accommodate the Internet protocol software.
- As an alternative to choosing a different MPU, you could add an intelligent controller that works in tandem with the existing MPU, taking commands from the system software and executing all of the Internet-related protocols. This setup has the advantage that you can continue using the same MPU, RTOS, and development tools, and it requires adding only a few lines of code to the existing system software.

Regardless of which of these methods you choose, the cost of implementing the changes should not be underestimated. Any new networking hardware or software must be:
- prototyped
- debugged
- integrated
- tested
- deployed in beta sites to prove interoperability and robustness
- maintained to keep the protocols up to date

For production runs in the thousands of units (say 5,000 units), the per-unit cost of this hardware and software development can add more than $50 per unit. This is without considering the risks of internally developing a new technology, which can cost untold dollars if the project does not come in on time, and on budget.

Many embedded systems developers recognize the costs (and the risks) of developing their own Internet interface, and would gladly purchase an off-the-shelf solution if it were available. That allows them to stay focused on the development of their product’s core technology.

Providing an off-the-shelf solution
Unfortunately, any solution that involves changes to the system software is not very portable. That makes it difficult for suppliers to provide off-the-shelf solutions. One way to provide portability is to provide an Internet interface in the form of an intelligent peripheral device that can be added to an existing embedded system, thus minimizing changes to the architecture of the system.

Such a device, which we will call an Internet controller, would include:
- a processor
- an Internet protocol stack, in firmware

This Internet controller would establish and maintain a physical connection to an ISP through…
- a dial-up connection
- a LAN connection
- a DSL modem
- a cable modem

…and would execute the Internet protocol firmware, relieving the embedded system processor of all the Internet overhead. This Internet controller might be packaged as a single component that is installed directly on the printed circuit board of the embedded system.

Creating a driver is expensive
Internet protocol stacks are complex, and require a lot of configuration parameters. Peripherals have traditionally been configured with control registers, and monitored with status registers. Creating the I/O drivers needed to configure and monitor these devices can be very time consuming and expensive to design, especially if the peripheral is complex, which an Internet controller most certainly will be. Much time will be spent:
- studying manuals that define the various register bits and fields
- writing the driver software needed to configure the registers
The Hayes “AT” command set allows an application programmer to configure the physical layer (to set the baud rate, number of stop bits, even/odd parity, etc.) prior to establishing a communication link. However, it does not include commands for configuring higher protocol layers, such as those used on the Internet.

Fortunately, there is no reason why the command set cannot be extended to include higher-layer configuration commands. If such an extended version of the AT command set were developed (and put into the public domain) it would permit the design of standardized Internet modems. The availability of such modems would then encourage embedded system designers to equip their systems for connection to the Internet. The resulting Internet connectivity would benefit the users of millions of embedded systems.

The AT+i protocol
The AT + internet (AT+i) protocol is an extended version of the Hayes AT protocol that accepts simple, printable ASCII command strings that allow application programmers to configure and use Internet protocols. This command set is designed to be:

- hardware-independent
- operating system-independent
- extendable

The primary reason that the Internet protocols are difficult to use is that there is a myriad of parameters associated with each layer of the protocol stack. Each of these parameters must be assigned a value before the protocol stack can be used.

However, this complexity can be hidden from the application programmer by assigning a default value to each parameter. The application programmer can still change any of these default values by sending assignment statements (in the form of ASCII strings) to the modem, as follows:

<par> = value

Figure 1 lists the Internet-related parameters that can be configured with the AT+i command set, along with a standard default value for each.

Establishing the Internet connection
If an embedded system is always connected to the Internet through a LAN, there is no need to issue commands to establish an Internet connection. However, many embedded systems (especially those in remote locations) will be connecting to the Internet through a dial-up modem. To support this mode

The limitations of the Hayes command set
The Hayes “AT” command set allows an
of operation, the AT+i command set includes commands for dialing an Internet Service Provider and establishing a PPP connection.

## Communicating through an Internet connection

Once the application program in a modem-based system has established a dial-up connection, it conducts Internet transactions with other nodes on the Internet using the same commands that are used for a LAN → Internet connection. The AT+i command set includes commands to support standard Internet transactions, such as:

- Email text/MIME messaging
- HTML page retrieval
- Fax-over-IP
- voice-over-IP
- security
- encryption

Email is the most practical and effective way to use the Internet for messaging. To support this, an Internet peripheral device should contain all the Internet protocols needed to log onto an internet service provider and to send and receive messages as email, including PPP, UDP, TCP, IP, SMTP, and POP3. Figure 2 on page 30 shows the AT+i commands that are used to handle email messaging.

In addition to email, the AT+i protocol can be used to retrieve web pages, and to open and close socket interfaces. This latter feature is particularly handy when:

- sending streaming audio or video
- using proprietary transmission protocols
- using a secure server

The AT+i protocol can be further extended, allowing an embedded system designer to add commands to support a specific application. Commands to support additional protocols, such as JINI or Bluetooth, could eventually be added as these new protocols are standardized.

### Notes:

- ">" is a prompt that is sent by the Internet modem to the application software to indicate that the AT+i command interpreter is ready to accept a command.
- "AT+i" is a command prefix that is sent to the Internet modem to tell its command interpreter that an AT+i command string follows.
- "I/OK" is a response from the command interpreter, indicating to the application software that it has finished executing the previous command.

#### An example of a typical AT+i send session

<table>
<thead>
<tr>
<th>AT+i</th>
<th>I/OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT+iTOA: =<a href="mailto:adresseee@domain.com">adresseee@domain.com</a></td>
<td>Define the addressee.</td>
</tr>
<tr>
<td>I/OK</td>
<td></td>
</tr>
<tr>
<td>AT+iSBJ: &quot;Device Sample Message&quot;</td>
<td>Define the email subject line.</td>
</tr>
<tr>
<td>I/OK</td>
<td></td>
</tr>
<tr>
<td>AT+iEMA: These are the lines of the Email body that will be sent to the address that is currently in the AT+i TOA parameter.</td>
<td>Send the following string as email.</td>
</tr>
<tr>
<td>I/OK</td>
<td>Period indicates the end of the email text.</td>
</tr>
</tbody>
</table>

---

**Figure 3**

An example of a typical AT+i send session

Figure 3 shows how a sequence of AT+i commands can be used to create and send an email message to a specified email address. When this sequence is executed, the message represented by the email text is sent through the Internet to addressee@domain.com.

#### The iModem Internet engine: An example of an Internet modem

Figure 4 shows the embedded version of the iModem from Connect One. Inside the iModem is an Internet engine called the iChip – a low-cost processor that runs the PPP, UDP, TCP/IP, SMTP, POP3, and MIME protocols. Future versions will also support FTP and HTTP. All of the iModem’s protocol software resides in flash memory, and can be remotely updated after the iModem has been deployed.

Application software communicates with the iModem using the AT+i extensions to the Hayes AT command set. The iModem operates in either of two modes. In Internet mode, the iChip Internet engine intercepts and executes commands from the application program that run the Internet protocols, such as commands to dial, login, authenticate, communicate, and sign off the ISP. In standard mode the iChip Internet engine is bypassed, and the commands from the application program are passed directly to the modem.

The iModem handles data rates from 2,400 to 56 Kbytes/sec, and is available in two versions:

- A stand-alone version
- An embedded version

### The stand-alone version

The stand-alone version of the iModem includes:

- the iChip
- a Conexant SocketModem
- LEDs
- a loudspeaker
- a DC power supply
- an RJ-11C connector
- a DB-9 connector for an RS-232 serial connection
- an enclosure

### The embedded version of the iModem

The embedded version of the iModem
includes the iChip Internet engine on a mezzanine board, which has a pin-out identical to a Conexant SocketModem. The iChip is plugged into a SocketModem site on a host carrier board, and the SocketModem is then plugged into the top of the iChip. (See Figure 4.) The iChip is interfaced to the host processor board through a TTL-level or RS232-level serial interface. It also provides:

- control signals for LEDs
- output signals to drive a loudspeaker
- TIP and RING signals for connection to a phone line

Alan Singer is vice president, sales and marketing, at Connect One Ltd. He combines a unique cross-cultural business background with more than 20 years of experience in sales and marketing in the global communications industry. Prior to joining Connect One, Alan served as director of marketing and marketing communications, a networking and Internet software manufacturer. Prior to that he was the product marketing manager and corporate training manager at Fibronics Ltd., a data communications hardware manufacturer. Alan also served as European sales manager for Better On Line Solutions, a developer of AS/400 communications products, and was vice president of Singer Products, an Export Management Company. Alan received a BS degree from Cornell University and an MBA from Columbia University.

If you have any questions about this article, or if you would like to know more about Connect One’s products, you can contact the author at:

Connect One Semiconductors, Inc.
4677 Old Ironsides Drive, Suite 280
Santa Clara, CA 95054
Tel: 408-986-9602
Fax: 408-986-9604
Email: alan@connectone.com
Web site: www.connectone.com

Trademarks:
Connect One, iChip, iModem, iWare, and AT+i, are trademarks of Connect One Semiconductors, Inc.

Connect One develops and manufactures chips, software, and hardware that enable industrial, commercial and consumer devices to connect to the Internet. Connect One’s low-cost solutions enable manufacturers to incorporate Internet connectivity in their products without attaching a desktop PC or a gateway server, or requiring expensive processors.

Connect One’s products include: iChip, which is an embedded TCP/IP peripheral chip that works in tandem with an existing host processor to Internet-enable an embedded system iModem, which is a stand-alone or embedded Internet modem replacement for a standard telephone modem, and iWare, which is embedded software for embedded systems with sufficient processing power.
### Figure 2: AT+i commands used for email messaging

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Parameters/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong></td>
<td><strong>Function</strong></td>
<td><strong>Parameters/Description</strong></td>
</tr>
<tr>
<td>AT+i</td>
<td>Command Prefix</td>
<td>Required to precede all commands</td>
</tr>
<tr>
<td><strong>Host Interface</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| En | Echo Mode | \( n = 0 \rightarrow \) Do not echo host characters.  
\( n = 1 \rightarrow \) Echo all host characters.  
This command is equivalent to and interchangeable with ATEn. |
| **Parameter Database Maintenance** |  |  |
| \(<par>=value\) or \(<par>=value\) | Set Parameter | value stored in parameter \(<par>\) in nonvolatile memory. \(<par>\) will retain set value across power down, indefinitely. For description of all available parameters see section 4.3 |
| \(<par>=value\) | Assign single session parameter value | value is assigned to parameter \(<par>\) for the duration of a single Internet session. Following the session the original value is restored. |
| \(<par>\) | Read parameter | Parameter value is returned |
| \(<par>=?\) | Parameter what? | Returns the allowed values for this parameter. |
| FD | Factory Defaults | Restores all parameters to Factory Defaults. |
| **Status Report** |  |  |
| RP<i> | Request Status report | Returns a status report value or sets a status report mode based on \(<i>\) |
| **Email Send Immediate** |  |  |
| \([!]EMA:<text>\) | Send textual Email immediate | Defines the textual contents of the Email body. Following this command several text lines may be sent in sequence. Sending a CR,CR (line containing only a period) terminates the text body. After termination the Email is sent automatically. Total \(<text>\) size is limited to 96K. |
| \([!]EMB:<sz>,<data>\) | Send Binary Email immediate | Prefixes a binary data stream. The data is encapsulated as a base64 encoded MIME attachment. Following this prefix, exactly \(<sz>\) bytes are streamed to the iChip. For values of \(<sz>\) greater than 256, software flow control applies. |
| \([!]E*\) | Terminate Binary Email | Terminates a Binary (MIME Attachment) Email |
| **Retrieve Email from Mailbox** |  |  |
| \([!]RML\) | Retrieve Mail List | Retrieves an indexed, short form, list of all qualifying messages in mailbox. |
| \([!]RMH[:<i>]\) | Retrieve Header | Retrieves only the Email header part from the \(<i>\)'th Email in the mailbox. |
| \([!]RMM[:<i>]\) | Retrieve Email | Retrieves all Email contents (header + body) of the \(<i>\)'th Email in the mailbox. |
| **HTTP Client Interface** |  |  |
| \([!]RLNK[:<URL>]\) | Retrieve Link | Retrieve a file from a URL on a web server. If \(<URL>\) is not specified, use the URL stored in the URL parameter. |
| \([!]SLNK:<sz>:<strm>\) | Send Link | Send (upload) a byte stream of size \(<sz>\) to the URL specified by the URL parameter. |
| \([!]S*\) | Terminate Send | Terminates an HTTP upload to Link. |
| **Socket Interface** |  |  |
| STCP:<host>,<port> | Socket TCP | Open and connect a TCP socket. If the iChip is not online, the ISP shall be connected. The responding system is assumed to be a server “listening” on the specified socket. Responds with a handle to the socket. |
| SUDP:<host>,<port> | Socket UDP | Open and connect a UDP socket. If the iChip is not online, the ISP shall be connected. Responds with a handle to the socket. |
| SSND[\%]:<hn>,<sz>:<stream> | Socket Send | Send a byte stream of size \(<sz>\) to the socket identified by handle \(<hn>\). \(^\%\) flags autoflush socket. |
| SRCV:<hn>[,<max>] | Socket Receive | Receive a byte stream from the socket identified by handle \(<hn>\). Accept maximum \(<max>\) bytes. If \(<max>\) is not specified, all available bytes are retrieved. |
| SDMP:<hn> | Socket Dump | Dump all buffered inbound data, that has arrived in the socket identified by handle \(<hn>\). |
| SFSH:<hn> | Socket Flush | Flush all buffered outbound data, sent to the socket identified by handle \(<hn>\). |
| \([!]SCLS:<hn>\) | Socket Close | Close the socket identified by handle \(<hn>\). |
| **Connection** |  |  |
| Down | Disconnect from Internet | This command forces the iChip to terminate an Internet session and go offline. |
| **Remote Firmware Update** |  |  |
| FU | Firmware update | In a modem configuration, enters firmware update mode. |