DOS Stamp Internet Made Easy (DIME) v1.0b
User's Manual
revision 1.0

This page intentionally left blank

DIME v1.0b User's Manual

Overview

This document explains how the Internet works and how to interface an embedded computing application to it. The examples given show how to use Bagotronix products with the Internet, but the principles are similar for many other computing devices. Previous knowledge of Internet protocols is not necessary, but is helpful.

The DOS Stamp Internet Made Easy (DIME) software makes embedded Internet easy, but there are several concepts you need to understand to apply DIME to your embedded Internet application. These concepts are explained in this document.

This document is designed to be read sequentially from beginning to end. The concepts are presented in a linear fashion, with the most fundamental concepts first.

Prerequisites

It is assumed that you are already familiar with using the Internet on your PC. That is, how to dial into your Internet Service Provider (ISP), start up a browser (Internet Explorer, Netscape, etc.), visit websites, and send and receive e-mail.

You must also be familiar with how to upload programs and interact with the DOS Stamp. This topic is covered in the DOS Stamp User's Manual, which you should read before you read this document.

To customize DIME for your embedded Internet application, you need to possess the following skills:

- C programming language
- How to use your C compiler
- DOS usage and application development
- Reading simple wiring diagrams

An addition, the following items are required:

- DOS Stamp with the standard second serial port configuration (RS-232) and the DiskOnChip option
- Cables to connect the DOS Stamp to your PC and a modem
- External modem with RS-232 interface
- Borland/Turbo C compiler v3.0 or higher. Other DOS-targeting C compilers may be used, but source code changes may be required.
- Two ISP Internet accounts, at least one of which is dial-up access
- A browser (for testing)
- An e-mail account (for testing)
- eTCP/WATTCP manual
- The book uC/OS-II The Real-Time Kernel by Jean J. Labrosse

Serial Ports, RS-232, and Modem Interfacing

Most DIME applications will connect to the Internet using dial-up phone service with regular modems, or over the air with radio modems. To use a modem with DIME, it must be interfaced to the DOS Stamp's COM2 serial port.

RS-232 signals swing both positive (+12V) and negative (-12V) with respect to ground. PC serial ports have the following RS-232 signals:

Name	Purpose	Details
Transmit Data (TX)	Outgoing data	Output. Positive voltage represents a "0". Negative voltage represent a "1".
Receive Data (RX)	Incoming data	Input. Positive voltage represents a "0". Negative voltage represent a "1".
Clear To Send (CTS)	A signal from the other machine (or our modem) that says it is OK for this machine to transmit	Input. Not used by DIME. Loop back to modem's RTS output
Request To Send (RTS)	A signal to the other machine (or our modem) that says it is OK for it to transmit to this machine	Not used by DIME. Loop back to modem's CTS input
Data Terminal Ready (DTR)	A signal to the other machine (or our modem) that says this machine is ready	Output. Not used by DIME
Data Set Ready (DSR)	A signal from the other machine (or our modem) that says it is ready	Input. Not used by DIME
Data Carrier Detect (DCD)	A signal from the modem that says it has detected a valid carrier	Input. Positive voltage when a carrier is present. Negative voltage when no carrier is present.
Ring Indicator (RI)	A signal from the modem that pulses each time phone line rings	Input. Not used by DIME.
Ground (GND)	Ground reference for all other signals	Connect DOS Stamp GND to modem GND.

Table 1: RS-232 Serial Port Signals

Notice that the DOS Stamp COM2 port supports only two signals: TX and RX. To use a modem, we need a third signal, DCD. To accommodate DCD, use one of the DOS Stamp's general-purpose I/O pins (PIO) to monitor DCD. See the connection diagram in Figure 1.

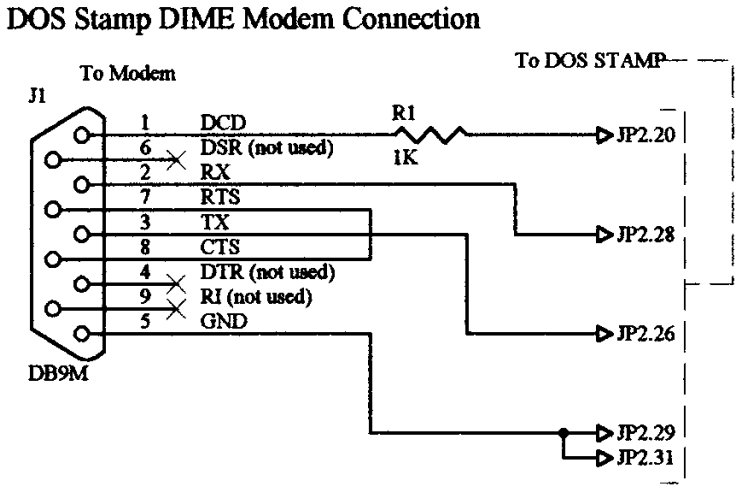

Figure 1: DOS Stamp DIME Modem Connections

Since the modem's DCD signal is RS-232, it swings way beyond the voltage limits of the PIO pin. We need to limit its voltage range before passing it on to the PIO pin. Resistor R1 works in conjunction with the PIO pin's internal chip protection diodes (not shown) to "clamp" the voltage swing to a safe range. DIME does not use hardware handshaking, so the modem's need for a CTS signal is satisfied by looping back its RTS output.

Modems are typically controlled using the "AT" command set. When you initialize the modem, be sure to include commands for turning off all types of handshaking (both hardware and software). They are not necessary for DIME.

Internet Standards and RFCs

The Internet owes its incredible growth in large part to the open standards upon which it is based. In fact, the main reason the Internet came to be was the mandate from the U.S. government Defense Advanced Research Projects Agency (DARPA) to unify all the proprietary information systems the U.S. military had acquired over the years. This required the design of standard protocols for networking of computers. Internet standards are now presided over by the Internet Engineering Task Force (IETF). The standards are called Request for Comments (RFC) and are available for free download from http://www.ietf.org. There are thousands of RFCs covering a wide variety of networking topics, but here we are interested in only the ones that directly affect how we connect an embedded system to the Internet.

Internet Protocol (IP) and IP Addresses

The most fundamental part of the Internet Protocol (IP) is the concept of how a machine is identified on the Internet. The IP address is a way of uniquely identifying individual machines. When your PC connects to the Internet, it uses an IP address to identify itself to other computers on the Internet. Why can't your PC be anonymous, you ask? The IP address is used by other machines on the Internet (routers) to route data to and from your PC. Without knowing the IP address of your PC, the routers would not know where to send the data. For the same reason, your embedded Internet application must have an IP address to connect to the Internet.

An IP address is a 32-bit unsigned binary number. When IP addresses are written, they are expressed as a decimal number with periods written between digit groups. For example, the IP address 192.168.1.9 . The largest value of any of the digit groups is 255, and the smallest value is 0. For example, the IP address 255.0.34.127. Each digit group represents the decimal equivalent of eight of the binary bits in the IP address. For example, the IP address 192.168.1.9 is equivalent to the 32-bit binary form 11000000.10101000.00000001.00001001, which is also equivalent to the hexadecimal form C0.A8.01.09 .

There are 2^{32}, or 4,294,967,296 (over 4 billion!) possible IP addresses. That may seem like a lot, but as the Internet continues to grow, the supply of unused IP addresses shrinks. Eventually, it may be necessary to use a new type of IP address with more digits. But for now, we can use 32-bit IP addresses if we are not wasteful.

How do we conserve the limited supply of IP addresses? How are we assigned an available IP address? The single answer to both questions: dynamic IP addressing. The concept of dynamic IP addressing is that when your machine establishes a connection to the ISP, it is assigned an available IP address for the duration of the connection. When the connection is terminated, that IP address is taken back by the ISP and handed out to another machine as needed. Your ISP "reserves" a block of IP addresses, and hands them out as its customer's call in. For example, XYZ ISP Inc. reserved 256 IP addresses, but has 2000 customers. As customers call in, each one is assigned an IP address taken from the pool of reserved IP addresses. When a customer's connection is terminated, the IP address is taken back and returned to the pool of reserved IP addresses. In this way, 2000 Internet users can get by with only 256 IP addresses.

What if we don't like dynamic IP addressing? What if we want to reserve our own IP address? Then ask your ISP to rent you a "static" IP address. You will pay more for a static IP address, and it is usually not necessary. For most embedded Internet applications, a dynamic IP address will do just fine. The only reason you might need a static IP address is if your embedded Internet application will be referred to by a hyperlink from some other web page. The DIME demo web page has a static IP address, since it is referred to by a hyperlink from the Bagotronix website.

TCP/IP Protocol

TCP/IP is an acronym for Transmission Control Protocol / Internet Protocol. The "/IP" indicates that it uses the IP addressing scheme. It is frequently referred to as just "TCP". The software that provides TCP/IP capability is frequently referred to as a "TCP/IP stack". It is not a stack in the same sense of a CPU stack, but instead refers to the layering of protocols looking like a stack.

TCP/IP is a method for reliably transmitting data across IP networks. When you bring up a website in your browser, the browser sends data to the website, and the website sends data back to your browser. This exchange of data is done using TCP/IP. The data is placed in packets with information about the data. This information goes into the packet "header". The actual data is placed after the header.

Why go through all this trouble making packets out of the data? Why not just send the data and be done with it? The routers on the Internet need to know where to send the data, and who sent it. They also need to know when it was sent, in what order, and if the data is still valid or has been corrupted. To accommodate these needs, the packet header contains the destination and source IP addresses, a sequence number, a checksum, and other fields. Some of these header areas are part of IP, and others are part of TCP. The low-level details are not important here. WATTCP takes care of these details for you.

TCP is considered a reliable protocol, because if packets are lost or corrupted, they are retransmitted (within reason). Obviously, a TCP connection cannot be expected to eternally retransmit packets, because the entire Internet would become bogged down with old packets still trying to get through. So there is a Time-To-Live (TTL) field in each TCP/IP packet that tells TCP when to give up.

It is possible to have multiple TCP connections concurrently over a single Internet connection. This is done with "ports". Ports are used to steer the data to various network applications that may be running concurrently. For example, the DIME demo unit has a web server that runs on TCP Port 80, and an e-mail sender that runs on TCP Port 25. Both of these TCP applications run concurrently, and the port numbers are used to steer the data from the Internet connection to each application.

UDP/IP Protocol

UDP/IP is an acronym for User Datagram Protocol / Internet Protocol. The "/IP" indicates that it uses the IP addressing scheme. It is frequently referred to as just "UDP". The software that provides UDP/IP capability is frequently referred to as a "UDP/IP stack". It is not a stack in the same sense of a CPU stack, but instead refers to the layering of protocols looking like a stack.

UDP/IP is an alternative method for transmitting data across IP networks. Like TCP, the data is placed in packets with information about the data. This information goes into the packet "header". The actual data is placed after the header.

UDP is considered an unreliable protocol, because it is one-way. A UDP sender sends a packet, and no confirmation of reception is expected or accepted. If the UDP packet got lost, discarded, or corrupted, the sender would not know. Why would we want to use such an unreliable protocol? There exist some network services that are best served with one-way transmission. For example, a time server would broadcast the time over UDP periodically, and would not care if any machine were listening. Also, the processing overhead for UDP is much less than TCP. This can make a significant difference in real-time applications.

Like TCP, UDP also uses ports to steer the data to various network applications.

Physical, Upper, and Lower Layers

Protocols are arranged in "layers". Each layer has a purpose. A layer either does something with the data, or establishes and maintains the link.

"Upper" and "lower" are relative terms. A layer is the "upper layer" if it is built on the layer you are talking about. For example, if you are talking about IP, TCP is an upper layer because it is built on IP. A layer is the "lower layer" if it underlies the layer you are talking about. For example, if you are talking about TCP, IP is the lower layer.

The actual hardware and low-level software drivers that interact with the hardware are called the "physical layer", and is the lowest layer of all. The UART, modem, cables, phone lines, and physical layer software are all part of the physical layer, as far as the protocols are concerned. When someone says "the physical layer is down", it means that any of the following could be true:

- The modem is not ready
- The modem is not connected to the remote modem (the carrier was lost or was never established)
- The UART is not working or has not been properly initialized
- The cables are not connected or improperly connected

The physical layer software is low-level software that initializes, configures, and directly interacts with the hardware. For example, software that sends an "AT" dialing command to the modem, looks for the response, and signals the upper layer (the protocols) that the hardware is ready to use.

Point-To-Point Protocol (PPP) [RFC 1661]

Point-To-Point Protocol (PPP) is actually a name for a suite of protocols. It does not have an "/IP" after it because it is not part of the Internet Protocol. PPP is an additional layer of encapsulation that facilitates transmission of data over error-prone serial data links. If your PC connects to your ISP over a Plain Old Telephone System (POTS) regular phone line, your PC is probably using PPP. Most likely your embedded Internet application will need to use PPP to connect to an ISP. POTS modems, radio modems, and RS-232 serial data links are subject to noise and electromagnetic interference (EMI), which can corrupt data and produce extraneous data. PPP has built-in mechanisms for validating data. PPP does not have mechanisms for retransmitting corrupted data; that is the responsibility of the IP transport protocol used (i.e. TCP/IP). In the event of a data error, PPP discards the erroneous data. PPP also ignores any data that is not encapsulated in a valid PPP frame. PPP cannot work with 7-bit serial data links. The link must be "8-bit clean", so do not try to use 7 data bits in your UART configuration. You must also have full-duplex (separate transmit and receive) lines, since PPP does not support half-duplex (shared transmit and receive) serial data links.

To establish a PPP connection, it is essential to understand how PPP works. PPP is peer-oriented. This means that both machines are equal in status; they are not master and slave. Although technically both machines are peers, the "other" machine is frequently referred to as the "peer". For example, when the DOS Stamp negotiates with the peer, the DOS Stamp is negotiating with the machine on the other end of the link.

PPP Link Phases

There are five phases the PPP link can be in:

- Dead
- Establish
- Authenticate
- Network
- Terminate

The Dead phase is when the PPP link is not ready to send or receive data. This is usually because the physical layer does not have a connection established. Once the physical layer has established a connection, it must signal the upper layer (LCP) that the physical layer is up (ready). When LCP receives a signal that the lower layer (physical) is up, the PPP link moves to the Establish phase.

The Establish phase is when the PPP link negotiates with the peer over the configuration of the link. Both machines must come to an agreement on the configuration before the link can do useful work. These negotiations are complex and are carried out by the Link Configuration Protocol (LCP), which runs during the Establish phase. If LCP negotiation is successful, LCP moves the PPP link into the Authenticate phase by signaling the upper layer (authentication protocol) that LCP is open. If both machines cannot agree on the configuration, LCP moves the PPP link back to the Dead phase by signaling the lower layer (physical) with a command to break the connection.

The Authenticate phase is when the PPP link authenticates itself to the peer, using the chosen authentication protocol. This is how the username and password are transmitted to the peer. If the username and password are valid, authentication moves the PPP link to the Network phase by signaling the upper layer (network protocols) that authentication was successful. If authentication fails, it moves the PPP link to the Terminate phase by signaling the lower layer (LCP) with a command to close the link.

The Network phase is when the actual data transfer to and from the Internet happens. Each network protocol (IP, IPX, AppleTalk, etc.) must configure itself with the peer before data encapsulated in that protocol can be transferred. This configuration is done using Network Control Protocol (NCP). The PPP link stays in the Network phase until either: (1) LCP receives a signal to close the link, or (2) a signal is received from the physical layer that the connection has been broken. In the case of (1), a "terminate" packet may have been received from the peer, or our own application may have sent the signal. In the case of (2), the modem may have lost the carrier, or other some other hardware problem may have occurred. The PPP link then moves to the Terminate phase.

The Terminate phase is when the PPP link is closed gracefully. During Termination, LCP is used to close the link by exchanging "terminate" packets with the peer. Upon receipt of these packets, LCP signals the lower layer (physical) that LCP is down (closed). The physical layer should then disconnect (hang up). The PPP link then moves to the Dead phase. If the physical layer is already down (not ready), LCP will not bother to send terminate packets because there is no connection available to carry them.

Link Configuration Protocol (LCP) – (Part of PPP)

Link Configuration Protocol (LCP) is a way for peers to automatically agree upon many characteristics of the link, such as packet size limits, which authentication protocol to use, header field compression, etc. LCP can also detect if the serial link is looped back (echoing the data it receives).

If you are having trouble establishing a PPP link, but the modem is dialing correctly, it could be due to an unsupported LCP option required by one peer but not supported by the other. LCP can usually establish a compatible link with almost any peer. In difficult situations, the resulting link is the lowest common denominator that each peer can accommodate. In impossible situations, the link will terminate after futile negotiations.

During LCP, the authentication protocol is negotiated. If the peers cannot agree on the same authentication protocol, or if the username and password are not valid, the link will be terminated immediately.

Password Authentication Protocol (PAP) – (Part of LCP)

Password Authentication Protocol (PAP) is the most commonly used PPP authentication protocol. PAP is a simple protocol that sends the username and password to the peer in clear text (not encrypted). At this time, DIME supports only PAP.

Challenge Handshake Authentication Protocol (CHAP) – (Part of LCP)

Challenge Handshake Authentication Protocol (CHAP) is an authentication protocol that sends the username and password in encrypted form to the peer. At this time, DIME does not support CHAP.

Network Control Protocol (NCP) – (Part of PPP)

Network Control Protocol (NCP) is actually not a protocol. NCP is a generic name given to any control protocol that configures PPP links for use with a network protocol. For example, if we want to use IP on the PPP link, we need to configure the link using an NCP called IPCP. If we want to use IPX on the PPP link, we need to configure the link using an NCP called IPXCP. IPCP and IPXCP are both classified as NCPs. There are many other network protocols, each one with its own NCP. It is possible to use more than one network protocol concurrently on the same PPP link. Since we are only using IP on the link, we are only interested in IPCP, and ignore all other NCPs.

Internet Protocol Control Protocol (IPCP) – (Part of PPP) [RFC 1332]

Internet Protocol Control Protocol (IPCP) is used to configure the PPP link for IP. IPCP negotiates options with the peer, just like LCP does. There are few options for IPCP. Van Jacobson (VJ) compression is one of the options, and is supported by DIME. Even if the peer does not support VJ, DIME can still fall back to non-VJ operation through negotiation.

During IPCP, the IP address is assigned by the peer. In most cases, the IP address is dynamically assigned. This means it will be different each time the link is established. If you have chosen a static IP address for your ISP account, the IP address will be the same each time.

Asynchronous High-level Data Link Control (AHDLC) Framing - (Part of PPP) [RFC 1662]

When data is transmitted over asynchronous serial links using PPP, it is transmitted in AHDLC frames. A frame consists of a start flag (also known as a frame delimiter), an address field, a control field, a variable length of data, and a CRC. The start flag is a single byte, 7E. The address field is a single byte, FF. The control field is a single byte, 03. Note that since 7E is used as the start flag, it cannot be used elsewhere in the AHDLC frame. Any bytes in the data or CRC with the value 7E must be converted to an equivalent two-byte representation. This is known as "escaping". The escaped representation is the value is 7D, followed by another byte that is the original value exclusive-OR'ed with the value 20. This is illustrated below:

```
 1   2   3   4           5
7E  FF  03  68  32  90  7E  34  61 (original byte sequence)
```

where (1) is the start flag, (2) is the address field, (3) is the control field, and (4) is where the data begins. The first data byte that requires escaping is (5), but we are also going to escape (3) for reasons explained later. After escaping, the byte sequence becomes:

```
 1   2   3       4           5
7E  FF  7D  23  68  32  90  7D  5E  34  61 (resulting byte sequence)
```

Notice how the 03 control field was changed into the two-byte sequence of 7D 23, and the 7E data was changed into the two-byte sequence of 7D 5E. The resulting byte sequence is what actually gets transmitted over the link. The 7D value is the escape flag, and tells the peer that the next byte is to be converted back to its original value. The peer does this by discarding the 7D value and exclusive-OR'ing the escaped value with the value 20.

Of course, the escape flag 7D also cannot appear elsewhere in the data stream. So, to send a value of 7D, a two-byte sequence of 7D 5D is sent.

It is also possible to escape other values. By default, PPP escapes the values 00 - 1F, 7D, and 7E. This explains why we escaped the value 03 in the example above. Values in the 00 - 1F range can be individually escaped or not escaped by setting the Asynchronous Control Character Map (ACCM) parameter. ACCM is negotiated with the peer during LCP. It is desirable to escape as few values as the peer will support, because sending escape flags lowers the effective throughput of the link.

The CRC is calculated based on the original (un-escaped) values beginning with the address field. The resulting CRC is itself escaped if necessary.

When data passes over the link, it has been subjected to encapsulation at each layer. For example, data sent by TCP/IP over a PPP link looks like this:

```
AHDLC frame start flag (PPP)
AHDLC address field (PPP)
AHDLC control field (PPP) {
        IP header (IP) {
                TCP header (TCP) {
                        Actual data (Application, i.e. HTTP, SMTP)
                }
        }
}
AHDLC CRC (PPP)
```

DIME PPP Implementation

DIME implements PPP using a modified version of DOSPPPD v0.6, which is a DOS port of the Linux pppd program. Bagotronix has modified DOSPPPD to:

- Make it work on the DOS Stamp
- Make it capable of automatic unattended operation

DSPPPD is the modified version of DOSPPPD that runs on the DOS Stamp. DSPPPD uses the packet driver software interface, which is an open standard for interfacing network applications with network physical layer drivers. The packet driver specification is available for free download from http://www.crynwr.com.

DSPPPD differs from most packet drivers in two important ways:

1) Packet drivers typically perform only physical layer functions. DSPPPD also performs the higher layer functions of PPP (LCP, authentication, IPCP, etc.).
2) Packet drivers typically are solely responsible for managing the physical layer. With DSPPPD, management of the physical layer is shared with your application.

The only part of the physical layer that DSPPPD manages is the UART. DSPPPD initializes the UART and receives interrupts from it. The remainder of the physical layer is managed by your application. This division of labor makes it easy to accommodate different physical layer hardware, such as POTS modems, radio modems, direct serial connections, etc.

DSPPPD initializes the UART with a configuration specified on the DOS command line when DSPPPD is invoked. DSPPPD then goes resident and waits for your application. When your application decides to actually establish the connection, it sends the modem initialization string to DSPPPD, which passes the string directly through to the modem. Your application then queries DSPPPD for any response strings from the modem. Once your application detects a modem carrier signal, it then signals DSPPPD that the physical layer is up. DSPPPD then begins the PPP process. Once PPP has entered the Network phase, DSPPPD signals your application that the IP layer is up. Your application can then begin using network applications, and must monitor the status of the physical layer concurrently. When the physical layer goes down, your application must signal DSPPPD so that it can move the PPP link into the Dead phase.

DSPPPD is a Terminate and Stay Resident (TSR) program. It is invoked once at the DOS prompt before your application is invoked. Command line options are required when DSPPPD is invoked. These options are explained in the pppd.man file on the DIME distribution disk.

How to Establish a PPP Link

Establishing a PPP link involves the following process (refer to source code in DIMEDEMO.C):

1) Invoke the PPP packet driver at the DOS prompt, with the necessary command line options. For example:

dspppd 38400 local –crtscts asyncmap a0000 user <username> passwd <password>

The first option (38400) is the baud rate to use. The maximum recommended baud rate is 38.4K. The second option (local) specifies what type of physical link is used: direct connection (local), or modem (modem). Use the "local" option even if you are using a modem, because we do not want DSPPPD to monitor the modem DCD signal (your application does that). The third option (-crtscts) specifies that hardware handshaking is not to be used. DIME does not currently support hardware handshaking, which is not necessary since the protocols are so robust. The fourth option (asyncmap a0000) specifies which values are to be escaped over the PPP link by setting the ACCM during LCP negotiation. An ACCM of 000A0000 is commonly used by most ISPs. The fifth option (user <username>) is your ISP account username. The sixth option (passwd <password>) is your ISP account password.

When DSPPPD is invoked it initializes the UART, goes resident, returns to the DOS prompt, and waits for a signal from your application that the physical layer is up.

2) Invoke your application at the DOS prompt.

Your application establishes a connection by bringing the physical layer up. The physical layer is managed by ConnectionTask( ), which should be inside your application:

3) Send a dialing command to the modem (in ConnectionTask( )):

```
case CONN_NO:
        // see if we are signaled to establish a connection
        if (openconn) {
                // send modem dialing commands
                WriteSerialString ("ATZ\r\n");
                OSTimeDlyHMSM(0, 0, 1, 0);      // wait 1 s for modem to reset
                // put your ISP's phone number in here
                // and any special modem commands
                WriteSerialString ("AT&KDT5761242\r\n");
                tick time = OSTimeGet();                // begin our carrier wait
                connstate = CONN_CONNECT_WAIT;
        }
        break;
```

4) Wait for the modem's DCD signal, then signal DSPPPD that the physical layer is up (in ConnectionTask( )):

```
case CONN_CONNECT_WAIT:
        // wait for CD
        if (PIOget(PIO_CD) && openconn) {
                OSTimeDlyHMSM(0, 0, 0, 250);    // delay 250 ms so modem is really ready
                // signal packet driver that lower layer is up
                asyfuncs.asyf_lowerup();
                ticktime = OSTimeGet();     // begin our network wait
                connstate = CONN_NETWORK_WAIT;
        }
```

```
else {
        if (OSTimeGet() > ticktime + MODEM_WAIT * OS_TICKS_PER_SEC
                || !openconn)
                connstate = CONN_CLOSING;
}
break;
```

5) Wait for DSPPPD to move to the Network phase (in ConnectionTask( )):

```
case CONN_NETWORK_WAIT:
        // wait for packet driver's network phase
        if (asyfuncs.asyf_phase() == PHASE_NETWORK && openconn) {
                <omitted for clarity>
                // At this point, we should be able to use the connection
                connstate = CONN_YES;
        }
        break;
```

At this point, the PPP link is established and is available for use by the sockets, network layers, and your application.

Sockets

Before you can use TCP or UDP to transfer data, you need to open a socket or listen to a socket. This is done with the tcp_open( ), udp_open( ), or tcp_listen( ) functions. These are documented in the WATTCP manual.

What is a socket? The definition varies slightly depending on whom you ask, but for our purposes, WATTCP defines a socket as a combination of the transport type, IP address, and port number. This completely describes which IP format the packets are in, who the packets are coming from or going to, and for which application the packets are intended. DIME uses a slightly modified version of WATTCP, which is a public domain open source sockets library created by Erick Engelke.

WATTCP must be configured before you can use sockets in your application. WATTCP reads the configuration file WATTCP.CFG when the sock_init( ) function is called. In the WATTCP.CFG file, you must specify these options:

- IP address assignment method (BOOTP, DHCP, or a static IP address)
- Gateway IP address
- Net mask
- Name server IP address (at least one)

There are other options for WATTCP, but they are usually not necessary.

Boot Protocol (BOOTP) [RFC 951]

Boot Protocol (BOOTP) is a way for a machine to find out what IP address it should use. When a machine sends a BOOTP request, a BOOTP server hands out an IP address to the machine. When using PPP, the BOOTP request is actually serviced by the PPP packet driver; it is not sent to the peer. This works because PPP has already established the link and has been assigned an IP address by the peer. Since PPP already has the IP address, there is no need to pass the BOOTP request on to the peer. This is done transparently. All your application has to do is make a BOOTP request after the physical layer is up and PPP is in the Network phase (refer to ConnectionTask( ), in DIMDEMO.C):

```
case CONN_NETWORK_WAIT:
        // wait for packet driver's network phase
```

```
if (asyfuncs.asyf_phase() == PHASE_NETWORK && openconn) {
        <omitted for clarity>
        _dobootp();
        <omitted for clarity>
        // At this point, we should be able to use the connection
        connstate = CONN_YES;
}
else {
        <omitted for clarity>
}
break;
```

In the WATTCP.CFG file, specify IP address assignment via BOOTP like this:

my_ip = bootp

Dynamic Host Configuration Protocol (DHCP) [RFC 1112, 1122, 1123, 1534]

Dynamic Host Configuration Protocol (DHCP) is another way for a machine to find out what IP address it should use. When a machine sends a DHCP request, a DHCP server hands out an IP address to the machine. DHCP works like BOOTP, with one important difference: DHCP "leases" IP address for a limited time. When the lease expires, the IP address is taken back by the DHCP server, and another request must be made to obtain a new one. When using PPP, the DHCP request is actually serviced by the PPP packet driver; it is not sent to the peer. Although WATTCP supports DHCP, it has not yet been tested with DSPPPD. In the WATTCP.CFG file, specify IP address assignment via DHCP like this:

my_ip = dhcp

There is no compelling reason to choose DHCP over BOOTP for PPP links. The PPP link keeps the same IP address for the duration of the connection, so DHCP leases are unnecessary.

Static IP Address

If you will be using a static IP address with PPP, you may still use the BOOTP and DHCP methods. This works because PPP has already established the link and has been assigned the static IP address by the peer. If you prefer to not use BOOTP or DHCP, you may specify a static IP address in the WATTCP.CFG file, like this:

my_ip = 192.168.1.9 (substitute your actual IP address)

Gateway IP Address

A good definition for gateway is "the machine nearest to your machine in the chain that relays your IP traffic to the network". When you use an ISP, the ISP is your gateway to the Internet. The ISP's gateway has a static IP address. WATTCP must know the gateway's IP address. In the WATTCP.CFG file, specify the gateway IP address like this:

gateway = 192.168.34.1 (substitute the gateway's actual IP address)

If you are connecting to the Internet through a local gateway in your facility, you must specify its IP address for the gateway, not your ISP's gateway.

Net Mask

A net mask (also known as subnet mask) is a way of qualifying IP addresses over a range. Any IP address not matching the pattern of the net mask is considered to not be on the subnet. Subnets are used to manage

network traffic by confining the traffic on a subnet to only those machines on that subnet. Subnetting is not really necessary for point-to-point links, but the subnet mask mechanism is built into WATTCP, so it must be configured properly. In the WATTCP.CFG file, you can just use this:

```
netmask = 255.255.255.0
```

Name Server IP Address

A good definition for name server (also known as Domain Name Server (DNS)) is "a machine that turns a domain name into an IP address". When you use a browser and type in a domain name (i.e. www.bagotronix.com), the browser sends a DNS request to the name server. The name server looks up the actual IP address of that domain and sends it to the browser. This is how the browser finds out which IP address to send a web page request to. You will probably want to use DNS services in your embedded Internet application. Therefore, WATTCP must know the name server IP address. In the WATTCP.CFG file, specify the name server IP address like this:

```
nameserver = 192.168.34.1          (substitute the name server's actual IP address)
```

You may specify more than one name server assignment, each on a separate line. It is a good idea to have at least two, since any given name server may be unavailable at certain times.

How to Get DIMEDEMO.C Up and Running

You should get DIMEDEMO running as is before you start making changes for your application. At this point you have learned enough concepts to configure DIMEDEMO and get it up and running. Starting from the beginning:

1) Connect the modem to the DOS Stamp according to the diagram shown in the "Serial Ports, RS-232, and Modem Interfacing" section.

2) Edit DIMEDEMO.C to make the following changes (in boldface):

In ConnectionTask( ):

```
    // send modem dialing commands
    WriteSerialString ("ATZ\r\n");
    OSTimeDlyHMSM(0, 0, 1, 0);          // wait 1 s for modem to reset
    // put your ISP's phone number in here, and any special modem commands
    WriteSerialString ("AT&KDT5551234\r\n");
```

In SmtpTask( ):

```
    // Substitute the name of your SMTP e-mail server here
    hostip = resolve ("mail.yourisp.com"); // get IP address of SMTP server
    <omitted for clarity>
    if (hostip) {   // if we got the IP address OK, send the message
        <omitted for clarity>
        // Substitute the name of your e-mail account here
        merr = SMTP_SendHello (pSmtpSock, tmpmsg, sizeof tmpmsg,
                "somebody@someplace.com");
        <omitted for clarity>
        // Substitute the name of your e-mail account here
        merr = SMTP_SendMailFrom (pSmtpSock, tmpmsg, sizeof tmpmsg,
                "somebody@someplace.com");
    }
    <omitted for clarity>
    // Substitute the name of your e-mail account here
    merr = SMTP_SendMsgHeader (pSmtpSock, now,
```

"**somebody@someplace.com**", wvMailRecipient.wvval,
wvMailSubject.wvval);

3) Compile DIMEDEMO.C and link it with these object files:

- dosstamp.obj
- os_cpu_a.obj
- os_cpu_c.obj
- ucos_ii.obj
- ds.obj
- dimedemo.obj

There is no need to recompile the source files for these object files if you are using a linker that is compatible with Turbo C++ v3.0. If you experience linker problems, you should recompile all of the source files except os_cpu_a.obj. This is an 80186 assembly language source file that has been assembled with TASM v3.0. If you are using Turbo C++ v3.0 or Borland C++ v3.x, you may use the makefile DIMEDEMO.MAK to automate this step. You may also use the project file DIMEDEMO.PRJ to compile and link within the IDE.

When compiling, make sure the compiler options are set for:

- Large memory model
- No floating point math
- Standard stack frame
- No debugging information
- Generate 80186 instructions

4) Upload these files to the DOS Stamp:

- DIMEDEMO.EXE
- DIMEDEMO.HTM
- DSPPPD.EXE

5) On the DOS Stamp, create the WATTCP.CFG configuration file as described in the "Sockets" section. At the DOS prompt, type:

copy con wattcp.cfg

Type each line of the WATTCP configuration information. When finished, type <CTRL-Z> to close the file.

Now test your DIMEDEMO setup:

6) Follow step 1 in the "How to Establish a PPP Link" section.

7) When the DOS prompt returns, run DIMEDEMO.

The modem should dial within a few seconds. After the modems negotiate, the PPP exchange will occur. If the ISP hangs up, it is probably because the username or password is incorrect. If the ISP does not hang up, your connection is successful. After about 60 seconds, press the space bar to refresh the terminal window. You should then see the IP address assigned to this link session. If the IP address is still 0.0.0.0, something went wrong with the PPP exchange, resulting in no IP address assignment from the ISP.

You may now try to bring the DIMEDEMO.HTM web page up in your browser by typing

http://<assigned IP address>/dimedemo.htm

into your browser. If it does not work, check to make sure you have entered the IP address and the filename correctly. Refresh the web page to see the BIOS tick counter advance.

You may now try interacting with the web page buttons and fields. Try sending an e-mail message to yourself. Try changing the state of GPIO pins by clicking on the GPIO buttons.

Real Time Operating System (RTOS)

A Real Time Operating System (RTOS) is software that enables multiple tasks to run concurrently on a single CPU. Each task is given a time slice on the CPU. When that time slice is up, the RTOS:

1) Interrupts the current task
2) Saves the state of the current task
3) Decides which task to run next
4) Loads the state of the next task
5) Makes the next task into the current task
6) Runs the current task

The DOS Stamp already has DOS on it, so why do we need an RTOS? Because DIME must run multiple tasks concurrently, but DOS can only run one task at a time. The protocol layers and sockets must be serviced in a regular, timely manner. However, the non-Internet parts of your embedded application need to run concurrently with the protocol layers and sockets. The RTOS allows us to schedule the CPU's time between the protocol layers, sockets, and the application. The RTOS does not replace DOS, but adds the ability to multi-task to your application. The RTOS does not add the ability to multi-task to DOS itself, so there still can be only one DOS-using task running concurrently.

DIME uses the uC/OS-II real-time kernel as the RTOS. The uC/OS-II kernel is pre-emptive. This means the kernel can interrupt a task in progress. Cooperative kernels cannot do this, and are not appropriate for our needs. For more information on uC/OS-II, refer to the book uC/OS-II The Real-Time Kernel by Jean J. Labrosse.

The Application Layer

Above the TCP and UDP transport layers, there is the application layer. This is where data originates and is consumed. In this context, "application" does not necessarily refer to the entire application program, but to the parts of it that interact with the network layer. So, even though DIME is an application, it actually has two "network applications" built into it: a HTTP web server, and a SMTP e-mail sender. A network application uses sockets to send and/or receive data over the TCP or UDP transport layer.

Hyper Text Transport Protocol (HTTP) – (A Network Application) [RFC 1945]

Hyper Text Transport Protocol (HTTP) is a network application. HTTP uses the TCP/IP transport layer to send and receive web pages. A HTTP server (also known as a web server), listens for incoming information requests on TCP Port 80. When you use a browser, you are sending HTTP requests to a HTTP server. When a request is received, the server parses the Uniform Resource Identifier (URI) string, and then performs the requested action.

There are three actions (also known as methods) a HTTP server can perform:

- GET method
- POST method
- HEAD method

GET Method – (Part of HTTP) [RFC 1945]

The GET method sends the requested information to the requester. Browsers send GET commands to HTTP servers to get web pages. For example, in your browser, if you type:

http://www.bagotronix.com/dimedemo.htm

the browser sends this request to the HTTP server:

GET /dimedemo.htm HTTP/1.0

This tells the server to get the data in the file dimedemo.htm and send it to the browser. Notice that "http://www.bagotronix.com" is not in the browser request. That information is not necessary for the server, but the browser uses it to determine:

- That the request is to be sent in the form of a HTTP request
- The domain name www.bagotronix.com is where the request is to be sent.

The GET method can also be used to send information to the HTTP server. This is done with a query string appended to the GET request. For example, in your browser, if you type:

http://www.bagotronix.com/dimedemo.htm?WhatsUpDoc

the browser sends this request to the HTTP server:

GET /dimedemo.htm?WhatsUpDoc HTTP/1.0

The query string is everything after the "?", up to the separating space before the "HTTP/1.0". What the server does with the query string depends on how the server is designed.

POST Method – (Part of HTTP) [RFC 1945]

The POST method is used to send information to a HTTP server. Browsers frequently use the POST method to send the results of on-screen forms to a server. When you buy something from a website, you are asked to enter your name, address, credit card information, etc. into an on-screen form. When you click on the "send" or "submit" button, the contents of the form are sent to the server.

For example, the browser might send this request to the HTTP server:

POST /someinfo.txt HTTP/1.0 <the info follows>

What the server does with the information depends on how the server is designed. Typically, the information is appended to an existing file of the same name. If the file does not already exist, it is created. Some servers use the information to invoke CGI scripts, which cause other actions to occur on the server. The DIME web server does not support CGI, but does create files and appends to existing files.

Either GET or POST methods can be used to send information to a HTTP server. Which method is better? For interactive web pages, you will probably want to use the GET method. Web pages can be easily designed with forms that send form data using the GET method, appending the form data as a query string. Also, the GET method automatically refreshes the browser window after sending the form data, therefore the results are immediately viewable. The POST method can send more data than a query string, but it does not automatically refresh the browser window. The POST method is capable of creating a file and appending new form data to it. The GET method cannot do this. This can be a liability, however. You would not want a website user to crash your embedded system by filling up the disk with appended data from repeated POSTs. For this reason, the DIME web server has the POST method disabled by default.

HEAD Method – (Part of HTTP) [RFC 1945]

The HEAD method is identical to GET, except that the HTTP server does not actually send the requested information. The HEAD method is frequently used for testing the validity of hyperlinks. DIME does not implement HEAD, and returns the HTTP error code "501 Not Implemented" if a HEAD request is received.

Hyper Text Markup Language (HTML)

Web pages are typically written in Hyper Text Markup Language (HTML). HTML combines displayed text with hidden formatting codes. When your browser shows a web page, it interprets the HTML formatting codes and positions the displayed text accordingly. To see what the HTML representation of a web page looks like, click on View->Source in your browser. Writing HTML manually is tedious. The easiest way to write HTML web pages is to use HTML editing software.

Static vs. Dynamic Web Pages

A static web page is one whose content does not change. A dynamic web page (i.e. the DIME demo) is one whose content may be updated in real time due to real-world events, user input, or program control. Static web pages are not adequate for embedded Internet applications, because they cannot be updated in real time. To support dynamic web pages, a mechanism for selectively updating the HTML content in real time is needed. This may be done by embedding in the HTML references to variables that exist in the application program. These variables are known as Web Variables.

Web Variables (WV)

Web Variables (WV) are actual variables that reside in your application program. Each WV is given a unique name (a text string) and can be assigned a value (a text string). Text is the only data type supported for the WV value. Using a WV is a three-step process:

1) Create a WV, like this

```
static WV wvSomeWebVar;
WVCreate (&wvSomeWebVar, "SomeWebVar", 10, "SomeValue", TRUE);
```

where the text name of the WV is "SomeWebVar", 10 bytes are allocated to store the text value, the initial value of the WV is "SomeValue", and the initial value of the webchanged flag is TRUE.

The text name of a WV does not have to be the same as its C declaration name. However, similar names do make the source code easier to follow.

2) Update the WV value inside the program if the WV represents something that is affected by real-world events or program control:

```
OSSemPend (WVSem, 0, &semerr);          // wait for exclusive access to WV's
sprintf (wvSomeWebVar.wvval, "%08lX", BiosTicks);      // update WV
OSSemPost (WVSem);                      // release WV's
```

If the WV value is to be changed by the web user with a browser, the program must poll the WV to see if it has changed, and act accordingly:

```
OSSemPend (WVSem, 0, &semerr);          // wait for exclusive access to WV's
if (wvSomeWebVar.webchanged) {
        <act accordingly>
        wvSomeWebVar.webchanged = 0;          // reset flag
}
```

```
OSSemPost (WVSem);                    // release WV's
```

It is essential to guarantee exclusive access to the WV. Otherwise, a higher priority task that uses the WV could interfere with a WV update already in progress in an interrupted lower priority task.

3) Create a HTML web page with references to the WV name embedded in text, forms, or controls:

```
<p>The current BIOS tick counter shows  <b>##SomeWebVar##</b>h  ticks
since midnight.</p>
```

This is a fragment of the DIMEDEMO.HTM web page. The WV is delimited by "##". The "##" is a signal to the DIME HTTP server that a WV value is to be substituted in place of everything between the "##" delimiters. The HTTP server is case sensitive and white space sensitive. Be sure to use the exact WV name here, with no leading or trailing spaces.

Your application is responsible for converting WV value text from/to suitable data types. There are several suitable C library functions for this purpose (i.e. atoi( ), sprintf( )) . After data type conversion, your application must also validate the data to ensure it is within reasonable limits.

DIME HTTP Server Implementation

The DIME HTTP server implements a subset of the HTTP/1.0 standard and supports:

- GET method
- POST method with file creation and append
- Query strings
- HTTP/0.9 simple requests
- Proper content length for fixed-length information responses (i.e. static web pages)

The DIME HTTP server supports these features that are not part of the HTTP/1.0 standard:

- Dynamic HTML pages
- Web variables

The DIME HTTP server does not support:

- Time and date stamps
- HEAD method
- Content length for variable-length information responses (content length is omitted)
- CGI

More than one instance of the DIME HTTP server can run concurrently. This allows several users to access a web page on the same DOS Stamp simultaneously. Browsers often make multiple requests to the same server concurrently. For example, a web page containing text and a graphic can cause a browser to make two simultaneous requests: one for the text, and one for the graphic. If there is no available server socket, the browser may not wait for one, and put the "broken graphic" icon where the graphic should be.

Simple Mail Transfer Protocol (SMTP) – (A Network Application) [RFC 821]

Simple Mail Transfer Protocol (SMTP) is a network application. SMTP uses the TCP/IP transport layer to send (but not receive) e-mail messages. A SMTP sender sends e-mail on TCP Port 25. When you use an e-mail program on your PC to send e-mail, it talks to the ISP's SMTP server. When you use the e-mail program on your PC to receive e-mail, it talks to the ISP's POP3 server, not their SMTP server. This is an important distinction. When you set up DIME to talk to your ISP, make sure you specify the ISP's SMTP server, not its POP3 server.

The DIME SMTP sender implements the following SMTP commands:

- HELO (Hello)
- MAIL
- RCPT (Recipient)
- DATA
- QUIT

These commands must be sent in this order. Between the DATA and QUIT commands, you send the actual e-mail message. See SmtpTask( ) in DIMEDEMO.C for an example.

Recommended for Further Reading

Sams Teach Yourself TCP/IP in 24 Hours, Joe Casad and Bob Willsey, First Printing, ©1998 Sams Publishing, ISBN 0-672-31248-4

uC/OS-II: The Real-Time Kernel, Jean J. Labrosse, ©1999 R&D Books, ISBN 0-87930-543-6

Illustrated TCP/IP, Matthew Naugle, ©1999 John Wiley & Sons, Inc., ISBN 0-471-19656-8

PPP Design and Debugging, James Carlson, ©1998 Addison Wesley Longman, ISBN 0-201-18539-3