PPP®
User's Guide

July 2004

U S SOFTWARE®
EMBEDDED EXCELLENCE

Copyright and Trademark Information

Lantronix, Inc.
15353 Barranca Parkway
Irvine, CA 92618
(949)453-3990
Fax (949) 453-3995

For Support Contact:
Micro Digital Associates, Inc.
2900 Bristol Street, #G204
Costa Mesa, CA 92626
(714) 437-7333
support@smxinfo.com
www.smxinfo.com

Contents

I) Introduction

This guide describes the use of the Point to Point Protocol for USNET. USNET/PPP provides support for connecting via serial links to or from remote hosts in embedded environments. As the purpose of this guide is not to teach how PPP works but to teach how to use PPP, we recommend a formal text for those who wish to pursue in depth knowledge of the actual protocol specification. Also, this guide is a supplement to the standard USNET guide and should be read only after reading the standard manual and after gaining a general understanding of USNET.

II) Overview

The Point to Point Protocol (PPP) is used to establish a link to a single remote host. This is commonly used in data acquisition, internet connectivity and vast other arenas.

For dial-up purposes (that is, using a modem and telephone line), a dialer is included. It is also an option to use a personal or vendor specific dialer in place of our mechanism, though we cannot support this. All code is written in C and will work with any supported USNET processor/compiler combination. USNET dialing does require the use of a precompile-time interpreter, provided in DOS executable format with source code. A compiler/linker for the development OS should not have problems turning this into an executable file. It is written in ANSI C.

At the time of this document's creation, RFC 1661 is the most current specification of PPP. Other RFCs used include:

1332	IPCP
1334	PAP
1662	HDLC framing
1990	MP
1994	CHAP
2433	MS-CHAP

III) PPP in Theory

The Point to Point Protocol is not a server/client system. It is commonly used that way, but only because it is convenient to do so. We will use those terms in the following manner: the side who initiates communications is the client and the side who is waiting to be contacted is the server. The host is the side of reference (i.e. this" side); the peer is the side opposite the reference (i.e. "that" side). So the server can be peer or host and the client can be peer or host (or vice versa for either). The peer may also be called a remote host.

There are two necessary phases within PPP: Link Control Protocol phase and Network Control Protocol phase. These are referred to as LCP and NCP respectively. The LCP used by PPP is most recently specified in RFC1661. The NCP phase is specified by the network layer protocols used. In USNET, the Internet Protocol (IP) is used in our network layer, so we use the Internet Protocol Control Protocol (RFC1332). A third phase, commonly included at the end of the LCP phase and before the NCP phase, is authentication. Generally, the Password Authentication Protocol (PAP from RFC1334) or Challenge Handshake Authentication Protocol (CHAP from RFC1994 or MS-CHAP from RFC2433) is used.

A) LCP Phase

The LCP phase determines the requirements and capabilities of both sides of a PPP link before actual communications begin. Let us refer to the client as the host and the server as the peer. Typically, the client (host) sends a configure-request packet ("conf-req") to the server (peer) to initiate communications. This packet contains a list of options that the host would prefer to use in the future. The peer should respond with either a configure-acknowledge ("conf-ack") or a configure-negative-acknowledge ("conf-nak") according to its satisfaction with the options within the conf-req. Also, when a peer receives a conf-req, it will send a conf-req back with the options it would like to use, so the process is mutual. If the host receives a conf-nak, then the peer was dissatisfied with the options enabled and the host must reconfigure itself and send a new set of options corresponding to the wishes/abilities of the peer in a new conf-req. If the options nak'd (negatively-acknowledged) are necessary for correct functioning of the host, the host must terminate the link negotiations.

If a conf-ack is received by the host, the host must wait to receive the peer's conf-req. If the host gets the conf-req and the options requested are acceptable, the host must send a conf-ack. At this point, the LCP phase is Open and the next phase should be initiated. If a timeout occurs before the conf-req is received, the host must re-send its conf-req and restart its half of the negotiations.

B) Authentication Phase (PAP/CHAP)

Authentication is used to decide what level of access the authenticatee should have to the authenticator. This is usually a "all- or-nothing" sort of thing. Using the same pair from LCP as an example, we continue on to authentication. Let us assume that the peer (server) requested PAP in its conf-req. This would require the host (client) to now send an authentication-request ("auth-req"). This packet includes a user ID and a password. USNET does not encrypt these. If the peer finds the user ID and the password acceptable, the host should receive an authentication-acknowledge ("auth-ack") and authentication would be completed. If the peer finds the user ID and password unacceptable, the host should receive an authentication-negative-acknowledge ("auth-nak") and the link should be terminated by the peer (this is not necessarily true, however).

Let us go back to the end of LCP and assume that the peer had requested CHAP in its conf-req instead of PAP. The peer (server) would then send a challenge (some unique value to be hashed). The host (client) would then tag on its password (secret) to the challenge and hash it with MD5. It would place this hashed value in a response and send it back. The peer would hash what should be the same thing on their side and compare it to the original. If they match, the peer would send a success packet and authentication would be concluded;

otherwise, it would send a failure packet and the link should terminate (although it may continue on). There are two distinct advantages about CHAP over PAP. Primarily, the raw password is never sent over the network (this does mean that both sides must maintain a copy of the password). Secondarily, the authenticator authenticates the authenticatee (i.e. sends the first packet) rather than forcing the authenticatee to authenticate itself to the authenticator.

MS-CHAP is different than CHAP. It makes use of the MD4 algorithm to hash the password.

Mutual authentication is appropriate, and often suggested as a means of increasing security, though most "servers" will not allow this. USNET will allow this, though some work may need to be done for its role as an authenticator. USNET has no pre-configured mechanism for storing a table of User IDs and secrets (passwords) for potential peers, though the structure to access that table is in place.

C) NCP Phase

Once the LCP is finished (and authentication if necessary), the NCP phase(s) must start. We use IPCP, as mentioned earlier. The behavior is nearly identical to the LCP phase, but its purpose is not to set up link layer communications but to set up network layer communications for the IP protocol, including the IP address.

Optionally, USNET allows a host to use Van Jacobson TCP/IP header compression. It is negotiated during IPCP. Throughput should increase slightly when using this.

IV) PPP in Practice

A) Usage

In config.mak, make sure that the line that enables the ppp to be built is uncommented: "PRODLIST += ppp".

Set up the netdata[] table in netconf.c to an appropriate state. Here is an example table for use with PPP:

```
/* There is an explicit route between all hosts */
GLOBALCONST const struct NETDATA netdata[] =
{
    "remote1", "eth1", C, {192,168,1,15}, EA0, 0, 0, 0, 0, 0,
    "peer0", "eth1", C, {192,168,1,10}, EA0, 0, 0, 0, 0, 0,

    "peer0", "ppp0", C, {0,0,0,0}, EA0, 0, 0, 0, 0, 0,
    "host0", "ppp0", C, {0,0,0,0}, EA0, 0, PPP, NS16550, 0,
            "IRNO=3 PORT=0x2F8 CLOCK=115200 BAUD=9600",
    "hostd0", "ppp0", C, {0,0,0,0}, EA0, DIAL, PPP, NS16550, 0,
            "IRNO=4 PORT=0x3F8 CLOCK=115200 BAUD=9600",

    "peer1", "ppp1", C, {206,251,94,250}, EA0, 0, 0, 0, 0, 0,
    "hostd1", "ppp1", C, {206,251,94,242}, EA0, DIAL, PPP, NS16550, 0,
            "IRNO=3 PORT=0x2F8 CLOCK=115200 BAUD=9600",
};
```

"host0" -- A USNET host that connects to other hosts through a null modem. It has no IP address assigned statically so it is assumed that the peer will provide one during IPCP. If this host wishes to connect to the immediate peer, it must open a connection to "peer0" by name. To use the peer as a router, open a connection to "remote1" by name.

"hostd0" -- A USNET host that connects to other hosts through a modem. It has no static IP address so it is assumed that the peer will assign one during IPCP. If this host wishes to connect to the immediate peer, it must open a connection to "peer0" by name. To use the peer as a router, open a connection to "remote1" by name. The only difference between this entry and "host0" is the flags entry where the word, DIAL, is placed. The macro, DIALD, will need to be configured to 1 in include\pppconf.h in order to use this entry.

"hostd1" -- A USNET host that connects to other hosts through a modem. This host has an IP address. If a peer dials into it, this host will be able to assign the peer the IP address from "peer1". The macro, DIALD, will need to be configured to 1 in include\pppconf.h in order to use this entry.

```
/* An implied route is specified for the host */
GLOBALCONST const struct NETDATA netdata[] = {
    "peer0", "ppp", C, {0,0,0,0}, EA0, ROUTER, 0, 0, 0, 0,
    "hostd0", "ppp", C, {0,0,0,0}, EA0, DIAL, PPP, NS16550, 0,
          "IRNO=3 PORT=0x2f8 CLOCK=115200 BAUD=9600",
/* This interface can be undefined if needed
    "hostd0", "eth", C, {206,251,94,236}, EA0, 0, Ethernet,
          I82557, 0, "IRNO=10 PORT=0xd400",
*/
};
```

"hostd0" -- hostd0 will use the router, peer0, no matter what IP address he attempts to contact. Note that no IP addresses are required in this table. If necessary, the host may have other interfaces to which subnetting still applies. If anything is not in that subnet, the default router, peer0, will be used.

In the pppsrc\save diretory is the file, prefrmt.exe. This is in DOS executable format. The source code for this file is in the pppsrc directory under the name prefrmt.c. If the development machine cannot execute DOS applications, prefrmt.c should be compiled for the appropriate OS. The source code's only dependancy is having script.h and script2.h in the include path. If script2.h does not exist, make an empty file in the same directory as script.h called "script2.h" (it is normally generated during the standard build process). Make sure the resulting executable file ends up in the pppsrc\save directory.

If scripted dialing will be used (DIALD == 1), the script files may require modifications to interact more correctly with the modem being used.

dial-in.scr -- This is used to allow a remote host to dial into USNET. It uses manual answer mode but may be changed to use auto answer.

dial-out.scr -- This is used to dial out to a remote host over a line. At least the phone number will have to be changed along with any special considerations for flow control or other modem or line specific properties.

dial-dwn.scr -- This is made to de-initialize a modem after a session has ended. This is not absolutely necessary, but it makes it easier to bring the modem up the next time.

See the "Scripting" section of this manual for assistance with the function of these files (Section IV C).

There is also a dialing test program in the appsrc directory, scrtest. It is a test of the dialing functionality only and is not meant to be an example of the correct way to use PPP.

When first starting or if scripts or pppconf.h options are changed, consider turning PPP_DEBUG to 1. This will make changes and their effects more readily apparent. It will also reveal areas that may need adjustment.

B) Configuration

All PPP related macro values are defined in include\pppconf.h. They are quite extensive and some of them interact with each other, so it is important to understand what they do when changing them. In the state it is shipped in, PPP should be able to establish a link with most implementations using a null modem.

PPP_DEBUG

USNET PPP comes with a module called pppdebug.c which can parse and print out -- in English -- with Nprintf() the frames that are sent and received by the link. This macro enables/disables this capability. It is useful to set this macro to 1 while configuring the PPP link. Once the link is behaving appropriately, this can be set to 0 and only warnings and errors will be printed out with Nprintf(). NTRACE takes precedence over this value. The USNET manual explains the usage of NTRACE for debugging.

DIALD

This specifies whether PPP will use the dialer automatically. See Part C of Section IV of this document for further information.

DBUFFER

PPP starts negotiations when the application forces the link up explicitly or when the first datagram is transmitted. This option tells PPP to buffer datagrams while the link comes up. By default this is on.

DBUFFER_SZ

This tells PPP how many buffers to queue up while waiting for the link to become established. The default value is NBUFFS/NNETS so that PPP doesn't starve the rest of USNET out of buffers but has enough to effectively perform the function of dial-on-demand.

IDLE_TOUT

This value specifies the amount of vacant time in seconds (TimeMS()/1000) in the link before it is closed manually. As delivered, it is disabled with a value of 0.

ECHO_TOUTMS

This value specifies the amount of time (in TimeMS() milli-seconds) in an open link between echo-request packets being sent. This can be used to check the link quality or to check if the peer has disappeared (if the peer loses connectivity without warning).

ECHO_RETRIES

This value specifies the number of echo-request packets sent without a reply before the link is deemed bad and is set to close. Setting ECHO_TOUTMS to a positive non-zero value enables this.

PPP_USERID

Because the PPP authentication user ID may differ from the application level user ID, we provide this value. It defaults to the application layer user ID. This value is set in Portinit() and can be changed thereafter through the ioctl routine (see the APPENDIX).

PPP_PASSWD

Because the PPP authentication password may differ from the application level password, we provide this value. It defaults to the application layer password. This value is set in Portinit() and can be changed thereafter through the ioctl routine (see the APPENDIX).

AUTHENT

We support PAP, CHAP and MS-CHAP authentication. This macro specifies which of those we will allow a peer to use on us. For client-oriented applications, this will usually be set to allow all three. For server-oriented applications, most people turn this off to save code space. All three are enabled by default.

USE_NT

Set this to one to use NT style challenge response. Set to zero for Lan Manager style challenge response. It is best to leave this on unless the remote host is a Lan Manager or an old Windows machine.

REQAUTH

This specifies which authentication will be requested by theUSNET host. For CHAP/MS-CHAP, AUTH_ALG must also be set (see below). For PAP, it is what it is.

AUTH_ALG

For MS-CHAP, this value must be set to CHAPalg_MD4; for normal CHAP, the value must be set to CHAPalg_MD5.

TOUTMS

This is the ms time elapse (TimeMS()) before time out. Our default is 2.5 seconds (2500) though RFC 1661 sets the default at three seconds. It has been noted that race conditions occur more frequently with smaller values, though every link is different. Links that come up slowly may need a smaller timeout period. Links that do not come up at all may require a longer timeout period.

TOUT_GROW

This specifies whether or not the restart timer should start small and grow to the maximum timeout value (TOUTMS) as link quality is assessed to be poor. It is off by default. When on, this may cause more retransmissions than necessary at the start of negotiations.

MAXCONF

This is the value in the restart counter for both LCP and IPCP. It should default to ten. The configuration packet will be resent this many times without response before the link is set to close.

MAXTERM

This is the value in the restart counter for LCP when closing. It should default to three. The terminate request packet will be resent this many times without acknowledgement before the link is forced closed.

COMPRESSION

This can be set to request and support protocol field and address/control field compression and/or VJ TCP/IP header compression. It is generally best to leave this at 3 to support both types as this will increase your throughput slightly. If code size is favored, it is best to leave this at either 1 (for address/control/protocol field compression) or 0 (for no compression). VJ compression requires a great deal of code, but the others do not.

MAXSLOTS

Maximum slots for TCP/IP (VJ) header compression. See RFC 1144 for more information or leave them at their default values. They basically correspond to the number of TCP connections coexisting on the link.

PPP_MRU

Specifies whether or not the host will negotiate the MRU for USNET. This value is the Maximum Receive Unit and is equivalent to (MAXBUF - MESSH_SZ - LHDRSZ) in USNET. Unless you are planning on reducing buffer size, this is not necessary.

MAGICNUM

Specifies whether or not the host will use Magic number with LCP. Unless you really want to save on the amount of data sent, leave this on. It is standard for almost all PPP links.

ASYNC

RFC 1662 tells of HDLC framing and the character escaping mechanism. This option will request that the peer use the RACCM value as its character map when sending to us. This option is enabled by default.

RACCM

This is the Remote Asynchronous Control Character Mapping. This option is only negotiated if ASYNC is enabled (see above). It is a 32 bit field where each bit corresponds to a character < 0x20. If the bit is true, the character must be escaped by PPP HDLC. Therefore, a value of 0x00000000 increases throughput the most but decreases reliability. A value of 0xffffffff escapes all characters and decreases throughput. The default value is 0x00000000.

IPCP_DNS

RFC 1877 includes extensions for PPP that allow configuration of DNS addresses during IPCP. This is not recommended except for dedicated devices with minimal application functionality and is disabled by default.

There are two parts to this option. The active configuration and the passive configuration. When the active portion is enabled (by setting bit 0 to 1), the host will send a configure-request with the current DNSiid. Typically, this will not have been configured by the application and will be 0.0.0.0 for both primary and secondary addresses. When the passive portion is enabled, the host will do nothing unless a configure-request is received for either the primary or secondary DNS in which case the host will reply with a Nak of the address if it does not match the host's DNSiid as configured by the application through netconf.c.

MP

The Multilink Protocol (RFC 1990) is enabled by a new Maximum Reconstructed Receive Unit. The endpoint discriminator is negotiated along with the MRRU. This option is not tested and is not considered a supported feature.

MPBUF
The number of buffers MP packets can occupy. Reasonably, no host should fragment any packet into more pieces than the number of physical connections.

AUTH_ACK_REPLY
When the peer passes authentication, this string is sent. It does not matter what it is, though the peer application may see it.

AUTH_NAK_REPLY
When the peer fails authentication, this string is sent. In MS-CHAP, a result code and retry flag is sent instead. If NTRACE >= 5 and PPP_DEBUG is on, the Message field of this packet will appear as strange characters because of the MS-CHAP result code.

QUALITY
This allows the peer to use link quality report monitoring. Very few implementations support this so you will want to leave this off. If you do wish to use this, contact USSW PPP support.

C) Scripting

In order for PPP to function over a modem, there are three non-error cases that must be handled:

1. If PPP needs to actively establish a link to a remote host over a modem line, the modem needs to dial out to the remote host prior to the initiation of PPP.

2. If PPP is waiting for a remote host to establish a link over a phone line, it must configure the modem to wait for such an event and perform some actions when the event occurs.

3. If PPP is terminated by either the host or the peer, the modem should be configured to a default state to wait for further action.

In order to facilitate this operation, there are several scripts used by USNET dial-on-demand:

1. pppsrc\dial-out.scr -- This is written to configure the modem to dial a phone number to a remote host for active links. Once it is completed successfully, PPP is initiated. It basically involves checking the present condition of the modem, changing it to off-line mode if necessary and dialing the phone number. If the connection is not made because of a modem error condition (e.g. no dialtone), the script will try a few more times. If the modem is not responding, the script will attempt to bring the modem to off-line mode (see dial-dwn.scr below). The phone number is defined as a global variable that can be changed in your application. There is more information on variables in scripts in the "Commands" section following number 3 below.

2. pppsrc\dial-dwn.scr -- This is written to configure the modem to be in terminal mode after being in on-line mode. It hangs up the line. In order to force a modem to off-line mode, most require that the string "+++" be sent surrounded by a guard time of 1 or 2 seconds of silence. The default script will successively increase the guard time from 1 to 5 seconds if the modem is not responding. At that point the script will fail. If the modem returns to terminal mode, the script succeeds.

3. pppsrc\dial-in.scr -- This is written to configure the modem to wait for an incoming call and answer it when it comes. PPP will wait passively once the script finishes with success. With the current setup, this script is executed while PPP is down for all links. If you will only be dialing out, an empty file (null script) can be used in its place in order to save a little CPU time. The current script should never finish unless an incoming call is received.

There are three additional scripts for logging into Windows based machines when using a physical null modem instead of a conventional modem.

1. pppsrc\ms-out.scr -- This is written to send the string CLIENT to the passive machine. The string, CLIENTSERVER, is expected in response after which the script is successfully completed and PPP data can flow.

2. pppsrc\ms-dwn.scr -- This is written to disconnect from a directly connected Windows machine. It sends the string, None, and terminates.

3. pppsrc\ms-in.scr -- This script waits forever for the string, CLIENT. If received, the string CLIENTSERVER is sent followed by some carriage returns and the script completes successfully so that PPP data can flow.

Commands used by scripts:

```
FILE name
    # Name the file being used ('name' in this case).

GOTO 3
    # Go to tag 3 unconditionally.

:3
    # This marks the position for tag 3

CHECK 1 3
    # If the internal status is good
    #    go to tag 1,
    # Else
    #    go to tag 3.

INIT
    # Make the internal status good.

%retries 5
    # Set the initial value of %retries to 5.  Future
    #   references to %retries will become an integer value
    #   at run time.  The name is arbitrary (like any
    #   variable name).  Any characters can be in a variable
    #   name except for '\r', '\n' or ' '.
    #      %bad\rvariablename
    #      %bad variable name
    #      %good_variable_name
    #      %bad_variable_name_this_is_just_too_long
    #
    #   The default variable name length is 30 characters.

-- %retries
    # Decrement the value of %retries by one.  Include a
    #   space between the "--" and the '%'.

++ %retries
    # Increment the value of %retries by one.  Include a
    #   space between the "--" and the '%'.
```

```
(%retries < 0) 1 2
    # If the value of %retries is less than 0,
    #   goto 1,
    # Else
    #   goto 2.
    #
    # >, <, >=, <=, == are supported operators.
    #
    # Either of the two comparison values can be an integer
    #   variable or integer value.  Any variables must be
    #   locally defined in a script.

$USERID test
    # Declare USERID as a pointer to "test".  Future
    #   references to $USERID will become "test".  Note that
    #   USERID is a global that can be referenced by your C
    #   application.  It is a char *.  Don't be afraid to
    #   point it to a new location.
    #
    #   #include "..\pppsrc\script.h"
    #   char *new_string = "new_string";
    #   void func(void)
    #   {
    #       USERID = new_string;
    #   }

SEND ATD 1 $PHONENUM \r
    # Send 8-bit data string to modem.  This particular
    #   sequence will dial the phone number prefixed with
    #   a 1.  The "\r" is necessary for most modems to
    #   signal the end of a command sequence.  NOTE: a ' '
    #   (space character) is required between a variable
    #   name and any other form of data.  This makes
    #   variables distinct while parsing the script.

EXPECT 5 OK
    # If "OK" is not received in 5 seconds, make the internal
    #   status bad and go on.
    # If "OK" is received in 5 seconds or less, just go on.

PAUSE 5
    # Yield control for five seconds.

DEBUG 3 Hello, World!
    # If NTRACE >= 3,
    #   print "Hello, World!" to stdout.
    # Else,
    #   preprocess this command out.

>logArray 20
    # Create an array of 20 bytes to log with

LOG >logArray
    # Start putting incoming data into array
```

```
NOLOG
    # Stop saving incoming data
```

These scripts may need modification for any particular environment. The rules for doing so are mentioned below. While going though them, it is recommended that a dial script be at hand to make references tangible. It is important to remember that the script is parsed into a series of arrays at build time. During execution of the application the arrays are stepped through to discover their outcomes.

Read time rules

- Each command must be on its own line.
- Anything following '#', the comment character, on a single line will be ignored.
- A file line will be truncated after 80 characters.
- prefrmt always expects the scripts to be in the order dial in, dial out and dial down.

Run time rules

- The status flag is internal. The status becomes good when INIT is performed. The status becomes bad when an EXPECT or SEND times out (i.e. the expected data is not received or sent data never sent). Until the status flag is reset, SEND and EXPECT operations will be skipped.
- Except for GOTO, CHECK and the if-else operation, all of which move the script index to the appropriate TAG value, the commands are performed in the order of the script.

Variable usage

- Variables must be declared before they are used. String and log variables have global scope, however, so they can be declared in a prior module and used in a later one:

  dial-in declares ussDialVar

  dial-out uses ussDialVar without declaring it

- Do not define a single variable multiple times. Remember, string and log variables have global scope.
- Variables must be surrounded by space characters so that they can be differentiated from other strings. Upon transmission or reception, the spaces will be disregarded.
- For SEND or EXPECT, data is concatenated at run time as if there were no space or tab characters. Carriage returns can be placed in a string by using "\r" or "^M". These can be used without spaces between them and other strings; however, there must be a space between them and variable references.

Suggestions

- We recommend that SEND and EXPECT are always performed in pairs. If a DEBUG statement or other time consuming command is performed in between, chances are that the EXPECT will not be installed early enough to receive the data provided from the modem as a response to the SEND information.
- Loops that may require a significant amount of time should have a SEND, EXPECT, PAUSE or DEBUG statement inside of them. All other commands are concatenated in execution and will block. For example, the following should never be done because it will starve the rest of USNET.

```
INIT
:1
%temp 30000
(%temp >= 0) 1 2
-- %temp
:2
```

Adding the following just after tag 1 will remedy the situation.

```
PAUSE 0
```

D) Notes on Special Cases

1) Dial On Demand

Although it is good to be able to send/receive data regardless of the state of the link, our present implementation has no direct access to the driver. This means that certain important functions (like checking the wire status) must be performed with timeouts and assumptions.

A second concern with dial-on-demand is that the link often requires a great deal of time to come up (configuring the modem, dialing out, bringing LCP, authentication and NCP up). If the upper layers require timely feed back from the remote host, replies may not arrive quickly enough. For purposes such as forwarding segments periodically over an open connection, it may serve a valuable purpose. There is a method to force the link to be either up or down. See the appendix for details.

One last area of concern is the IP address negotiations used by PPP. If the dial-up server requires that the host change its IP address, the connection may become invalid. Therefore, make sure that the original IP address remains through each demand dial session. One method of gaining this information is through the PPP ioctl() function and the PPP signal functions. Modify the following macro in pppsig.h:

```
 extern Iid previous_ip;
#define PPPSIG_IPCP_UP(netno) \
        do { \
            Iid ip; \
            ussPPPTable.ioctl(&nets[netno], \
                ussPPPHostAddressGetE, &id, 4); \
            if (id.l != previous_ip.l) { \
                Nprintf("Warning: IP address changed!\n"); \
                abort_application_connections(); \
            } \
        } while (0)
```

The PPPSIG_IPCP_UP() macro is chosen because it always procedes the IPCP layer where IP addresses are negotiated. The do { ... } while(0) phrase is used to encapsulate the command as a single expression.

2) Null Modem Links to Window Machines

Three special scripts were created to support this operation more efficiently. They are ms-in.scr, ms-out.scr, and ms-down.scr in the pppsrc directory. The file, pppsrc\dial.mak, has a configuration macro called, MS_NULL_MODEM, that must be configured to 1 to force the use of those scripts in prefernance of the normal modem command scripts (dial-xxx.scr).

Special considerations:

Windows 95 -- This operating system would not behave as a dial-up server so USNET was always the passive host when connecting to Win95 machines.

Windows 98 -- We were not able to install a null modem into the operating system and were forced to seek a third party solution. There was no problem using this operating system with conventional telephone modems. Contact U S Software PPP support for information about the driver used if the null modem cannot be configured.

Windows NT -- A special null modem cable or adapter was required to allow a physical connection between the hosts. The Microsoft web site has ample information about this issue. Otherwise, everything worked as expected.

3) MS-CHAP Authentication

This form of authentication is commonly performed when establishing a link with a Microsoft system. An important note to remember about it is that the domain must be prepended to the user name. Here at USSW, one NT server has the domain USSOFTWARE so a person's user ID is "USSOFTWARE\\name". Authentication will fail without this. Of course, if USNET is the authenticator rather than the authenticatee, the user ID is whatever USNET says it is. Also, NT supports passwords of up to 256 Unicode characters, but we recommend that you not make use of this feature and use conventional passwords.

4) Routing and IP Addresses with PPP Interfaces

IPCP has the ability to change an interface's IP address. Many client-oriented interfaces will not know either their own IP address, the peer's IP address or both. The host address is changed if it is 0.0.0.0 and the remote host can provide a new one. If the host's entry of the peer has an address of 0.0.0.0, the host will fill in that value with the address negotiated by the peer. In order to bring about the features mentioned, there are two cases to consider. Each case can be configured statically in netconf.c in the netdata table. For both cases, the port name for host and remote must be the same.

CASE 1:

If USNET is a server, the remote host can be represented statically by providing an extra entry in netdata in netconf.c. This entry must have the same port name and subnet mask and a valid IP accordingly to the mask. When the peer suggests that its address be 0.0.0.0, USNET will respond with a hint to use the one statically defined and the peer should take the hint. If there are multiple entries satisfying the constraints for a PPP host on a particular interface, the first one will be picked.

CASE 2:

If USNET is running as a client with a PPP interface, it is recommended that (unless the remote and host IP addresses are known to not change) at least the host address be defined as 0.0.0.0. The subnet mask must be set to a value corresponding to what will be true of the future IPs and the rest of the network in general. This means that some information must be known about the remote network. For those situations where there is only one interface on your USNET host, subnet masks do not matter because the remote host will always be the default router. If you are using the remote host as a router instead of a destination, no entry is needed in netdata as you will never be sending data to them. However, if you wish to communicate with the peer and do not know the IP address. Add an entry to netdata with an IP address of 0.0.0.0, but the same subnet and port name as your host's entry. In your application, open your connection to the host by name and USNET will take care of the rest.

Another way to gain access to the IP addresses is through the PPP ioctl() function. The following options are available:

ussPPPHostAddressGetE -- Get the host address
ussPPPHostAddressSetE -- Set the host address

ussPPPRemoteAddressGetE -- Get the remote address
ussPPPRemoteAddressSetE -- Set the remote address

Note that the 'Get' ioctl() operations can only yield useful information after IP address negotiations (i.e. when PPP is open). The 'Set' ioclt() operations can only be used prior to link negotiations (i.e. when PPP is closed).

5) Renogotiation of IP Address

PPP always negotiates the host and peer IP addresses. In the case where the host address is unknown, PPP will request a 0.0.0.0 address and the peer will reply with a valid one to use instead. Once PPP is established, the new IP address is permanently assigned to the host entry (in netconf[]). If PPP goes down and then renegotiates a new link, the last negotiated host address will be requested. Usually the peer will assign a new address if the requested one is not valid, but it may be the case that the peer cannot handle this address and will either abort the link or let network layer operations fail by administering incorrect IP addresses. For cases such as this, the USNET PPP must be configured to reset the host address to 0.0.0.0 between PPP negotiation sequences. The best way to do this is to assign the host address once LCP completes. The macro, PPPSIG_LCP_UP(netno), in pppsig.h can be defined to a value to modify the host IP address. The host address is stored in the nets[netno].haddr field. Here is an example macro definition:

```
#define PPPSIG_LCP_UP(netno) \
        do { \
           Iid id; \
           ussPPPTable.ioctl(&nets[netno], \
              ussPPPHostAddressSetE, &id, 4); \
        } while (0)
```

The PPPSIG_LCP_UP() macro is chosen because it always precedes an IPCP request.

The do { ... } while(0) phrase is used to encapsulate the command as a single expression.

6) Handling Loss of Carrier

Sometimes modems go off-line or a line breaks without warning. This situation can be dealt with in a variety of ways. The following cases examine a few examples and their relative advantages and disadvantages.

CASE 1 Application level timeout

Protocols such as TCP or other application level protocols often have timeout periods for expected data. If the PPP link becomes disestablished, there is a good chance that the upper-level protocols will be able to detect this error and restart the application when they notice that their data is not arriving in a timely manner.

Advantages

requires no initial engineering effort

requires no extra code

Disadvantages

requires a lot of time (depending on the application)
to notice that data is not arriving.

CASE 2   PPP echo-request detect loss of link

PPP has a feature similar to the ping operation of IP level systems called the LCP echo-request packet. USNET PPP can be configured to transmit this echo-request packet periodically. If a certain number of packets are sent in a row without any reply, the link is automatically terminated.

As an example, define the following in pppconf.h:

```
#define ECHO_TOUTMS 2500
#define ECHO_RETRIES 3
```

Given the above definitions and a broken link, PPP will recognize the failed link in a maximum of

```
ECHO_TOUTMS * (ECHO_RETRIES + 1)
```

milliseconds.

The application will probably need to be informed of the condition. The PPP_LINK_DOWN(netno) macro from pppsig.h can be used for this purpose. See the appendix for further information about this macro.

Advantages

requires minimal engineering effort
relatively quick response time

Disadvantages

requires a timeout period with retransmission
to detect a physical line break

does not detect looped-back condition

application must be signalled of link failure

CASE 3   PPP echo-request detect modem looped-back

If a Hayes modem is being used, the modem may enter echo mode automatically when it goes off-line. This feature enables PPP to depend on the magic number within the echo-request packet to detect a looped-back link. To use this feature, MAGICNUM, CHECK_LOOPED_BACK and ECHO_TOUTMS must all be enabled in pppconf.h. Once enabled, PPP will transmit echo-request packets periodically. It will notice that the packets contain the same magic number and will ignore them which will eventually force PPP to conclude that the peer is no longer available.

As an example, define the following in pppconf.h:

```
#define MAGICNUM 1
#define CHECK_LOOP_BACK 1
#define ECHO_TOUTMS 2500
#define ECHO_RETRIES 3
```

The above options will for PPP to send an echo-request packet every 2.5 seconds. If the modem goes off-line it will start echoing back all data. PPP will ignore the echo-request/reply packets and will force a link termination after 3 retransmissions of the echo-request packet. This will take a maximum of

```
ECHO_TOUTMS * (ECHO_RETRIES + 1)
```

milliseconds.

The application will probably need to be informed of the condition. The PPP_LINK_DOWN(netno) macro from pppsig.h can be used for this purpose. See the appendix for further information about this macro.

Advantages

requires minimal engineering effort

relatively quick response time

discovers looped-back condition

Disadvantages

requires a timeout period with retransmission
to detect a physical line break

application must be signalled of link failure

CASE 4 Application detect and handle link break

If the driver or application software can detect a change in line status corresponding to either modem loss of carrier or physical line break, the PPP link can be instantly forced shut.

Write the following code in a new module:

```
#include "net.h"
#include "local.h"
#include "support.h"
#include "ppp.h"
#include "dialapi.h"

static char ppp_kill_flag[NNETS];

void LOST_CARRIER_OR_LINE_BREAK_FOR_PPP(int netno)
{
        ppp_kill_flag[netno] = 1;
}

void nettask_ifkillflag(int netno) {
    if (ppp_kill_flag[netno]) {
        BLOCKPREE();
        nets[netno].state = PPPclsd;
#if DIALD
        MODEM_DIALIN(netno);
#endif
        pppDQ(netno); // Make non-static inside ppp.c!!!
        ppp_kill_flag[netno] = 0;
        RESUMEPREE();
    }
}
```

In net.c, insert the following:

```
void nettask_ifkillflag(int netno); // New prototype
void nettask(...)
{
        ...
        if (netp->protoc[0] == PPP) {
                pppTimeout(netno)
                nettask_ifkillflag(netno);  // New Call
        }
                ...
        }
```

In the event that the application detects loss of carrier or a line break in an ISR, the applicatoin can call the LOST_CARRIER_OR_LINE_BREAK_FOR_PPP(netno) function. When nettask() runs next (within a second), the PPP link will reset itself to start again. The connection level application will probably need to be informed of the condition. This should be done by the ISR that initiates the transaction.

Advantages

requires less than a second from detection of loss of carrier to resolution of PPP link.

does not need to detect looped-back condition

Disadvantages

invasive into the stack

application must be signalled of link failure

Appendix A) PPP ioctl Routines

1) Description

The PPP ioctl() funtion allows an application to dynamically configure or manage various parts of the PPP protocol layer. The PPP ioctl() operations pertain to specific interfaces on the host and therefore each require a handle to the network interface structure, nets.

Here is the function prototype.

```
int ussPPPTable.ioctl(
    void *netp,
    enum ioctlreq req,
    void *arg,
    size_t size);
```

If no error occurs, 0 is returned. Otherwise, applicable errors may be returned. At this time, only the ussErrInval return code is provided for cases when the ioctlreq option is invalid.

The netp parameter must be a USNET (struct NET *) data type.

The arg parameter data type varies from on ioctl option to another.

2) Option Listing

These options are defined in net.h as part of the ioctlreq enumeration.

i) ussLinkIsUpE

Upon return, *(int *)arg will be true if the link is up and false if the link is down.

ii) ussLinkIsDownE

Upon return, *(int *)arg will be true if the link is down and false if the link is down. If the dialer is enabled, this means that the dialer is in the passive state waiting for an incoming call.

iii) ussLinkBringUpE

Attempt to force the link up for the time in seconds specified by the size parameter. If the link becomes established prior to the passage of the full amount of time, the function will return early.

iv) ussLinkBringDownE

Attempt to force the link down for the time in seconds specified by the size parameter. If the link becomes fully disestablished prior to the passage of the full amount of time, the function will return early.

v) ussPPPUserIdSetE

Set the host userid to be negotiated by authentication protocols (PAP, CHAP, MS-CHAP).

The userid is passed through the (char *)arg parameter.

Note that the function stores the pointer, not the actual data (NO strcpy()!). The userid must remain allocated for the entire PPP session.

The userid must be a null terminated string.

vi) ussPPPUserIdGetE

Get the host userid to be negotiated by authentication protocols (PAP, CHAP, MS-CHAP).

The userid is copied into the (char arg[15]) parameter.

Note that the function copies the actual data into the arg parameter so at lease 15 bytes must be allocated to accomidate the potential maximum userid size.

vii) ussPPPPasswordSetE

Set the host password to be negotiated by authentication protocols (PAP, CHAP, MS-CHAP).

The password is passed through the (char *)arg parameter.

Note that the function stores the pointer, not the actual data (NO strcpy()!). The password must remain allocated for the entire PPP session.

The password must be a null terminated string.

viii) ussPPPPasswordGetE

Get the host password to be negotiated by authentication protocols (PAP, CHAP, MS-CHAP).

The password is copied into the (char arg[15]) parameter.

Note that the function copies the actual data into the arg parameter so at lease 15 bytes must be allocated to accomidate the potential maximum userid size.

ix) ussPPPDialEnableE

Enable the dialer on the interface.

This is only valid if DIALD is defined in pppconf.h.

x) ussPPPDialDisableE

Disable the dialer on the interface.

This is only valid if DIALD is defined in pppconf.h.

xi) ussPPPHostAddressGetE

Get the Host IP address as negotiated by IPCP.

Before IPCP is completed, this value is the IP address that USNET PPP will attempt to negotiate on the interface. After IPCP has completed, this value is the actual IP address USNET is communicating as from the PPP interface.

xii) ussPPPHostAddressSetE

Set the Host IP address to negotiate during IPCP.

This is only valid if it is performed prior to link establishment. Also, it does not guarantee that the IP address specified will be the one chosen because PPP must negotiate addresses. If the peer recommends that a different address be used, then USNET will use that one instead.

xiii) ussPPPRemoteAddressGetE

Get the Peer IP address as negotiated by IPCP.

Before IPCP is completed, this value is the IP address of the peer host that USNET PPP was last connected to. After IPCP has completed, this value is the actual IP address of the peer host to which USNET is directly connected to.

xiv) ussPPPRemoteAddressSetE

Set the Peer IP address to negotiate during IPCP.

This is only valid if it is performed prior to link establishment. Also, it does not guarantee that the IP address specified will be the one chosen because PPP must negotiate addresses. If the peer recommends that a different address be used, then USNET will use that one instead.

3) Using PPP ioctl() routines

In all examples below, it is assumed that Ninit() and Portinit() have been called prior to the execution of any ioctl() procedure.

i) Forcing the link up

If your application requires the link layer to be up at a particular point in time, use the following:

```
#include "net.h"
#include "local.h"
#include "support.h"
void func(int netno)
{
    struct NET *netp;
    int i1;

    netp = &nets[netno];
    /*
    ** The last parameter to ioctl when using
    ** ussLinkBringUpE represents the time in seconds
    ** to block while waiting for the condition to
    ** occur.  The function will return early if the
    ** link conclusively fails or succeeds.
    */
    ussPPPTable.ioctl(netp, ussLinkBringUpE, 0, 30);
    ussPPPTable.ioctl(netp, ussLinkIsUpE, &i1, 0);
    if (!i1)
        Nprintf("Error: could not force PPP up!\n");
    else
        Nprintf("PPP is up\n");
}
```

ii) Forcing the link down

If your application requires the link layer to be down at a particular point in time, use the following:

```
#include "net.h"
#include "local.h"
#include "support.h"
```

```
void func(struct CONNECT *conp)
{
   struct NET *netp;
   int i1;

   netp = &nets[conp->netno];
   /*
   ** The last parameter to ioctl() when using
   ** ussLinkBringDownE represents the time in
   ** seconds to block while waiting for the
   ** condition to occur.  The function will return
   ** early if the link is closed.
   */
   ussPPPTable.ioctl(netp, ussLinkBringDownE, 0, 10);
   ussPPPTable.ioctl(netp, ussLinkIsDownE, &i1, 0);
   if (!i1)
      Nprintf("Error: could'nt force PPP down!\n");
   else
      Nprintf("PPP is down\n");
}
```

iii) Capturing the link status

If the status of the link must be known, use the following:

```
#include "net.h"
#include "local.h"
#include "support.h"
void func(int netno)
{
   struct NET *netp;
   int i1;

   netp = &nets[netno];
   ussPPPTable.ioctl(netp, ussLinkIsUpE, &i1, 0);
   if (i1)
      Nprintf("PPP is up\n");
   else {
      ussPPPTable.ioctl(netp, ussLinkIsDownE, &i1, 0);
      if (i1)
         Nprintf("PPP is down\n");
      else
         Nprintf("PPP is negotiating\n");
   }
}
```

iv) Configuring the username and the password

If the host user ID or password must be set prior to link negotiations, use the following:

```
#include "net.h"
#include "local.h"
#include "support.h"

char *uid = "new userid", *pw = "new password";
```

```
void func(int netno)
{
   struct NET *netp;
   char tuid[15], tpw[15];

   netp = &nets[netno];
   ussPPPTable.ioctl(netp, ussPPPUserIdGetE, tuid, 0);
   ussPPPTable.ioctl(netp, ussPPPPasswordGetE, tpw, 0);

   /* Change userid if unmatched */
   if (!strcmp(tuid, uid))
      ussPPPTable.ioctl(netp, ussPPPUserIdSetE, uid, 0);

   /* Change password if unmatched */
   if (!strcmp(tpw, pw))
      ussPPPTable.ioctl(netp, ussPPPPasswordSetE, pw, 0);
}
```

v) Switching between modem and null modem links

If the host wants to connect through a null modem and a conventional modem with only one entry in netdata in netconf.c, use the following:

```
#include "net.h"
#include "local.h"
#include "support.h"
#include "ppp.h"

extern int host_using_modem;

void func(int netno)
{
   struct NET *netp;

   netp = &nets[netno];
#if DIALD
   if (host_using_modem)
      ussPPPTable.ioctl(netp, ussPPPDialEnableE, 0, 0);
   else
      ussPPPTable.ioctl(netp, ussPPPDialDisableE, 0, 0);
#else
   Nprintf("No modem configured\n");
#endif
}
```

vi) Configuring and Capturing the host and peer IP addresses

```
#include "net.h"
#include "local.h"
#include "support.h"


static void getIP(Iid *id)
{
   int i1;
   Nprintf("Enter IP as four single typelings\n    > ");
   for (i1 = 0; i1 < 4; i1++) {
      id->c[i1] = Ngetchr();
```

```
        Nprintf(" %02x (%c)",
            id->c[i1],
            id->c[i1] > 0x19 && id->c[i1] < 0x7f ? id->c[i1] : '.');
    }
    Nprintf("\n");
}

void func(int netno)
{
    int i1;
    Iid hid, rid;

    /* Loop waiting for user request */
    for ( ; ; ) {
        /* Yield to USNET and PPP */
        YIELD();
        /* Check for user input */
        if (Nchkchr()) {
            /* Get user input */
            i1 = Ngetchr();
            if (i1 == 0x1b) {
                break;  /* Exit function */
            }
            else {
                /*
                ** Switch on the user request.
                **  1 -- Get host address
                **  2 -- Set host address
                **  3 -- Get peer address
                **  4 -- Set peer address
                **  ? -- Print addresses
                */
                switch (i1) {
                case '1':
                    ussPPPTable.ioctl(&nets[netno],
                            ussPPPHostAddressGetE, &hid, 4);
                case '2':
                    if (i1 == '2')
                        getIP(&hid);
                    ussPPPTable.ioctl(&nets[netno],
                            ussPPPHostAddressSetE, &hid, 4);
                case '3':
                    ussPPPTable.ioctl(&nets[netno],
                            ussPPPRemoteAddressGetE, &rid, 4);
                case '4':
                    if (i1 == '4')
                        getIP(&rid);
                    ussPPPTable.ioctl(&nets[netno],
                           ussPPPRemoteAddressSetE, &rid, 4);
                    break;
                default:
                    /* Force the link up (non-blocking) */
                    ussPPPTable.ioctl(&nets[netno],
                            ussLinkBringUpE, 0, 0);
                    continue;
                }
                Nprintf(" H %u.%u.%u.%u R %u.%u.%u.%u\n",
                        hid.c[0], hid.c[1], hid.c[2], hid.c[3],
                        rid.c[0], rid.c[1], rid.c[2], rid.c[3]);
            }
        }
    }
    Nprintf("User terminated\n");
}
```

Appendix B) PPP dialapi Routines

1) Description

The dial API routines are defined in include\dialapi.h. For the most part they are used within the PPP core module. In some cases, it may be beneficial to use them from the application to improve configurability. In other cases, it may be useful to modify the dial API to map onto an already defined dialing layer for smoother integration with USNET PPP.

2) Definitions of API

i) MODEM_PROCESS(netno)
Execute modem operations in a non-blocking fashion for a particular interface.

ii) IS_MODEM_DONE(netp)
Boolean condition. True when the currently executing modem process associated with an interface structure is completed.

iii) IS_MODEM_NONE(netp)
Boolean condition. True when the currently executing modem process associated with an interface structure is ready for another operation.

iv) MODEM_DIALIN(netno)
Install or start the dial-in procedure for a particular interface.

v) IS_MODEM_DIALIN(netp)
Boolean condition. True when the currently executing modem process associated with an interface structure is the dial-in process.

vi) MODEM_SET_DIALIN(name)
This function is specific to the USNET PPP implementation. It is used to define the dial-in script.

vii) MODEM_DIALOUT(netno)
Install or start the dial-out procedure for a particular interface.

viii) IS_MODEM_DIALOUT(netp)
Boolean condition. True when the currently executing modem process associated with an interface structure is the dial-out process.

ix) MODEM_SET_DIALOUT(name)
This function is specific to the USNET PPP implementation. It is used to define the dial-out script.

x) MODEM_DIALDOWN(netno)
Install or start the dial-down procedure for a particular interface.

xi) IS_MODEM_DIALDOWN(netp)
Boolean condition. True when the currently executing modem process associated with an interface structure is the dial-down process.

xii) MODEM_SET_DIALDOWN(name)
This function is specific to the USNET PPP implementation. It is used to define the dial-out script.

3) Dynamically Configuring USNET PPP Dial Scripts

In order to change a dial script without recompiling, a few function calls can be made at run-time. At compile-time, the pppsrc\dial.mak file must be configured with the appropriate dial scripts.

It is not necessary, but is recommended, that PPP be not negotiating at the time of the change. This means that the state ought to be closed and the dialer ought to be inactive (dial-in).

Here is an example function that replaces the dial-in script with the ms-in.scr script already provided:

```
void install_ms_in(void)
{
   MODEM_SET_DIALIN(ms_in_scr);
   MODEM_DIALIN(netno);
}
```

Here is an example function that replacess both the dial-in and dial-out scripts using the Hayes modem type scripts:

```
void install_dial_in_and_dial_out(void)
{
   MODEM_SET_DIALIN(dial_in_scr);
   MODEM_SET_DIALOUT(dial_out_scr);
   MODEM_DIALIN(netno);
}
```

Note in the above case that MODEM_DIALIN() is called after installing the new script. This ensures that no ghost script is left executing.

Appendix C) PPP pppsig Routines

1) Description

The ppp signalling routines can be used to signal the application layer of events normally hidden within the PPP stack without the requirement of OS signal handlers. These events can be both informational and functional. They are defined in pppsig.h.

2) Definition of Signals Available

PPPSIG_PPP_UP(netno)
The entire PPP layer is up on the interface and is ready for network traffic.

PPPSIG_PPP_DOWN(netno)
The PPP layer is completely down on the interface and is ready for configuration or complete restart.

PPPSIG_IPCP_UP(netno)
The network layer is up (for IP) on the interface and is ready for traffic. This may also mean that IP addresses have been configured for perhaps the host and/or the peer.

PPPSIG_IPCP_DOWN(netno)
The network layer is down (for IP) on the interface and can no longer handle network traffic. This may also mean that IP addresses are no longer valid.

PPPSIG_HAUTH_UP(netno)
The host is authenticated to the peer on the interface.

PPPSIG_HAUTH_DOWN(netno)
The host failed authentication with the peer on the interface.

PPPSIG_PAUTH_UP(netno)
The peer passed authentication with the host on the interface.

PPPSIG_PAUTH_DOWN(netno)
The peer failed authentication with the host on the interface.

PPPSIG_LCP_UP(netno)
The link layer is established and ready for network level configuration and authentication on the interface.

PPPSIG_LCP_DOWN(netno)
The link layer is down completely and is ready for restart on the interface.

PPPSIG_DIALOUT_UP(netno)
The dial-out proceedure completed on the interface.

PPPSIG_DIALOUT_DOWN(netno)
The dial-out procedure failed on the interface.

PPPSIG_DIALIN_UP(netno)
The dial-in proceedure completed on the interface.

PPPSIG_DIALIN_DOWN(netno)
The dial-in procedure failed on the interface.

PPPSIG_DIALDOWN_UP(netno)
The dial-down proceedure completed on the interface.

PPPSIG_DIALOUT_DOWN(netno)
The dial-down procedure failed on the interface.

Note that this is not considered an error condition to the PPP layer so it will proceed to set up the dial-in procedure.

PPP_STATE_HOLD(netno) -- From ppp.h
Stop PPP at the current state on the interface.

PPP_STATE_RELEASE(netno) -- From ppp.h
Let PPP continue at the current state on the interfaces

3) Using PPP Signaling Routines

See Section IV.D.5 for a useful example of these macros.

It is important to remember that the signalling functions are meant to produce signals. The code that gets executed must not block and ought not to bloat the stack.