

# PPP Driver

The PPP packet driver is a set of libraries in Dynamic C that allows the user to establish a PPP (Point-to-Point Protocol) link over a full-duplex serial line between a Rabbit-based controller and another system that supports PPP. You may also establish PPP links over Ethernet (PPPoE).

A common use of the PPP protocol is the transfer of IP packets between a remote host and an Internet Service Provider (ISP) over a modem connection. The PPP packet driver supports the transfer of Internet Protocol (IP) data and is compatible with all TCP/IP libraries for the Rabbit.

Establishing PPP links has become easier and more flexible. You can have as many different PPP interfaces as you have available serial (and Ethernet) ports. You can also run PPP (over serial and/or Ethernet) at the same time as ordinary non-PPPoE Ethernet.

## 1.0 PPP Libraries

The PPP driver is in three library files.

### PPP.LIB

Contains routines to handle the link negotiation (LCP), authentication (PAP) and IP negotiation (IPCP). These are the three main sub-protocols of PPP. PPP.LIB calls routines in the other two libraries to handle the lower level (physical) layer. There are no application-callable functions in any of the PPP libraries; PPP is mainly controlled via the `ifconfig()` function and friends.

### PPPLINK.LIB

Contains handlers for the asynchronous serial physical layer, namely the interrupt service routine for transmitting and receiving characters over the serial link. It also handles the insertion and detection of escape characters and CRC generation and checking.

### PPPOE.LIB

Contains handlers for the PPPoE physical layer, which is mainly the access concentrator discovery mechanism, and the addition of the PPPoE header to Ethernet packets. This library calls the Ethernet driver library to handle the Ethernet physical layer.

A fourth library, `MODEM.LIB`, contains functions for controlling an external modem through a full RS232 link. `MODEM.LIB` should not be required for most ISP connections, since most ISPs these days auto-detect the use of PPP and do not require any special logon screen navigation. Basic dial-up to an ISP is handled directly by `ifconfig()` settings, which do not require any special modem control (providing that your modem has a Hayes-compatible interface).

`MODEM.LIB` is not directly related to PPP. It allows ASCII strings to be sent to and received from the modem. Typically, these strings are AT commands and modem responses.

If you have special requirements for establishing communications with an ISP that cannot be handled by the default PPP library methods, you will need to explicitly include `MODEM.LIB` and write a program to establish the communications link. The program will typically need to command the modem to dial out; wait for a valid connection; send a user ID and password to the ISP and validate the response. After this has completed successfully, PPP can be started using the `ifup()` function. For a complete description of `ifup()` and other Dynamic C TCP/IP functions, please see the Dynamic C TCP/IP User's Manual..

**NOTE:** `MODEM.LIB` is currently limited to controlling a single modem. The modem serial port and control lines are defined using macro constants, which should match with the definitions of the PPP interface.

The sample program `Samples\PPP\modem_test.c` shows the general idea for using `MODEM.LIB`.

## 2.0 Operation Details for PPP over Serial

The first step is to configure whatever transport medium will be used for the PPP connection. For directly connecting a serial line to the peer, the two serial data lines TX and RX may be adequate. The most common situation, however, will be some sort of modem.

### 2.1 The Modem Interface

The interface between a modem and a controller is either a true RS232 interface or a variation on RS232 that uses TTL voltage levels for all of the signals. The latter are used by board-mounted modem modules. If an external modem is used, an RS232 transceiver chip is needed to convert RS232 voltages to logic signals and vice versa. A full RS232 connection has 3 outputs and 5 inputs from the controller's point of view. In RS232 terminology, the controller is referred to as the DTE (Data Terminal Equipment). Modems and other peripherals are referred to as DCE's (Data Communications Equipment).

The specifics of a dial-up PPP connection are dependent on the modem hardware and the ISP.

### 2.1.1 Rabbit Pin Connections to Modem

The modem control library, `MODEM.LIB`, defines default connections to the Rabbit as follows:

**Table 1. Rabbit Pin Assignments for Modem Connection**

| RS232 Signal | Rabbit Pin | Direction |
|--------------|------------|-----------|
| DTR          | PB6        | out       |
| RTS          | PB7        | out       |
| CTS          | PB0        | in        |
| DCD          | PB2        | in        |
| RI           | PB3        | in        |
| DSR          | PB4        | in        |
| TD           | PC2        | out       |
| RD           | PC3        | in        |

### 2.2 Flow Control

Hardware flow control is implemented for the Rabbit PPP system. It follows the RS232 convention of using Ready To Send (RTS) and Clear To Send (CTS) lines.

Flow control is not required for speeds up to and including 115200 bps. The internal character processing is fast enough that the controller does not have to throttle incoming data flow. However, the modem or peer may need to throttle transmitted data. It is recommended that the RTS (modem to controller) line be connected for modems that cannot handle a continuous data stream at the specified rate. You can also connect the CTS (controller to modem) line, but the controller will merely assert this line continuously. To enable or disable hardware flow control, call `ifconfig()` with the `IFS_PPP_FLOWCONTROL` parameter identifier. You should also specify `IFS_PPP_RTSPIN` and `IFS_PPP_CTSPIN` parameter identifiers.

### 3.0 Operation Details for PPPoE

PPPoE avoids most of the complexities of PPP over serial. This is because the hardware (Ethernet) is easy to set up, and no modems are involved. Actually, you might have something called a DSL modem (or similar), but this type of modem does not have to do “dial-up” in the usual sense.

PPPoE is selected by defining the symbol `USE_PPPOE` to be a non-zero value. Currently, the only value supported is ‘1’ with ‘2’ reserved for future controller boards that have a second Ethernet chip. If you define `USE_PPPOE`, then you should also define `IFCONFIG_PPPOE0` to contain initialization options passed to `ifconfig()`. When PPPoE is specified, the interface is referred to by `IF_PPPOE0`. (`IF_PPPOE1` is reserved for future boards.)

## 4.0 Link Control Protocol Options

Link Control Protocol is the first sub-protocol used on a PPP link. The following LCP options are supported by the Rabbit PPP system:

**Table 2. Configuration Options**

| LCP Configuration Option Type Field | Meaning of Option Type                       |
|-------------------------------------|--|
| 01                                  | MRU (Maximum-Receive-Unit)                   |
| 02                                  | ACCM (Async-Control-Character-Map)           |
| 03                                  | Auth (Authentication-Type): PAP only         |
| 05                                  | Magic Number                                 |
| 07                                  | PFC (Protocol-Field-Compression)             |
| 08                                  | ACFC (Address-and-Control-Field-Compression) |

For more information on these options, refer to RFC 1661: The Point-to-Point Protocol (PPP) at:

<http://www.faqs.org/rfcs/rfc1661.html>

## 5.0 Configuring PPP

Since multiple interfaces are supported, your application should call `ifconfig()` to change PPP interface parameters at run-time, or define suitable `IFCONFIG_PPP*` macros for boot-time configuration of each PPP interface (both serial and PPPoE).

You select serial port hardware to use with PPP by defining `USE_PPP_SERIAL` before including `dcrtcp.lib`. Similarly, you select PPPoE by defining `USE_PPPOE`.

### 5.1 Serial Port Selection

PPP over asynchronous serial requires a suitable Rabbit serial port to be selected. You can use any of the available ports, since they all support asynchronous communications.

The serial port selection is entirely dynamic, however there is a fixed mapping between interface numbers and serial port hardware. `IF_PPP0` always represents serial port A. `IF_PPP1` is always serial port B, and so on.

The serial port hardware to use is determined by the `USE_PPP_SERIAL` macro, which your application defines in order to specify PPP serial interfaces. `USE_PPP_SERIAL` is set to a bitwise OR combination of numbers representing the desired serial port(s). Ports are assigned according to the following table.

**Table 3. Bitmap Values for USE\_PPP\_SERIAL**

| Interface Number | Serial Port | Bitmap Value |
|------------------|-------------|--------------|
| IF_PPP0          | SERA        | 0x01         |
| IF_PPP1          | SERB        | 0x02         |
| IF_PPP2          | SERC        | 0x04         |
| IF_PPP3          | SERD        | 0x08         |

The Rabbit 3000 supports an additional two serial ports, SERE and SERF, however the TCP/IP library does not fully support use of these ports for PPP.

If multiple PPP serial interfaces are required, use (for example)

```
#define USE_PPP_SERIAL 0x0C
```

which, as the bitwise combination of 0x04 and 0x08, specifies SERC (IF\_PPP2) and SERD (IF\_PPP3).

## 5.2 PPPoE Port Selection

Since all Rabbit-based controller boards currently have at most a single Ethernet driver chip, only a single PPPoE interface is available (however it can be shared with non-PPPoE Ethernet over the same hardware - non-PPPoE Ethernet will use interface `IF_ETH0` while PPPoE will use `IF_PPPOE0`).

## 5.3 ifconfig() Options for PPP

The `ifconfig()` parameter identifiers described in this section pertain to any PPP interface, whether serial or Ethernet. There are a considerable number of options (detailed in the *Dynamic C TCP/IP User's Manual*) pertinent to PPP over asynchronous serial. PPPoE does not, as yet, require any special configuration options because of its relative simplicity.

The parameter identifiers listed here are passed to the `ifconfig()` function. They can also be used in the appropriate `IFCONFIG_PPP*` macro definitions, to ensure that the interface(s) are initialized correctly at boot time. For example, a run-time change to the userid and password might be coded as follows:

```
ifconfig (IF_PPP2,
          IFS_PPP_REMOTEAUTH, "myUserid", "myPassword",
          IFS_END);
```

The same definition, for boot-time initialization, might be coded as

```
#define IFCONFIG_PPP2 \
    other parameters \
    IFS_PPP_REMOTEAUTH, "myUserid", "myPassword", \
    other parameters
```

The general PPP properties set during initialization are:

**Table 4. Macros for PPP Initialization (Serial and Ethernet)**

| Macro Name           | Macro Description   | Data Type(s) for Macro Params |
|----------------------|---|-------------------------------|
| IFS_PPP_ACCEPTIP     | Accept peer's idea of our local IP address.                   | bool                          |
| IFS_PPP_REMOTEIP     | Try to set IP address of peer.                                | longword                      |
| IFS_PPP_ACCEPTDNS    | Accept a DNS server IP address from peer.                     | bool                          |
| IFS_PPP_REMOTEDNS    | Set DNS server IP addresses for peer (primary and secondary). | longword, longword            |
| IFS_PPP_AUTHCALLBACK | Called when a peer attempts to authenticate.                  | int (*)()                     |
| IFS_PPP_INIT         | Sets up PPP with default parameters.                          | none                          |
| IFS_PPP_REMOTEAUTH   | Sets username and password to give to peer.                   | char *, char *                |
| IFS_PPP_LOCALAUTH    | Required username and password for incoming peer              | char *, char *                |

All of these IFS\_PPP\_\* macros (except the initialization and callback) have IFG\_PPP\_\* versions that allow an application to look at the properties that have been set.

## 5.4 ifconfig() Options for Serial PPP

The `ifconfig()` parameter identifiers described in this section pertain to serial PPP interfaces only. (You can specify these options for PPPoE interfaces, but they will be quietly ignored.) They may also be used in the appropriate `IFCONFIG_PPP*` macro definitions for boot-time initialization.

**Table 5. Macros for PPP Initialization (for Serial)**

| Macro Name                              | Macro Description   | Data Type(s) for Macro Parm |
|---|---|-----------------------------|
| <code>IFS_PPP_SPEED</code>              | Set serial PPP speed (bps)  | longword                    |
| <code>IFS_PPP_RTSPIN</code>             | Define the RTS pin.   | int, char *, int            |
| <code>IFS_PPP_CTSPIN<sup>f</sup></code> | Define the CTS pin.   | int, int                    |
| <code>IFS_PPP_USEPORTD</code>           | Use parallel port D instead of parallel port C for serial ports A and B.  | bool                        |
| <code>IFS_PPP_FLOWCONTROL</code>        | Turn hardware flow control on or off  | bool                        |
| <code>IFS_PPP_HANGUP</code>             | An optional string to send to the modem after PPP shuts down.   | char *                      |
| <code>IFS_PPP_MODEMESCAPE</code>        | When enabled, sends modem escape sequences before send/expect or hangup sequence is: '<delay>+++<delay>' This is recognized by almost all modems to force them into command mode. | bool                        |
| <code>IFS_PPP_SENDEXPECT</code>         | A formatted send and expect sequence for dialing and shell login.   | char *                      |
| <code>IFS_PPP_USEMODEM</code>           | Specify whether to use modem dialout string.  | bool                        |

All of these `IFS_PPP_*` macros have `IFG_PPP_*` versions that allow an application to look at the properties that have been set.

The parameter for the `IFS_PPP_SENDEXPECT` option is a string containing a send/expect script to run when the PPP connection comes up. It is a series of tokens separated by spaces, alternating between a string to transmit, and a string to expect back.

For example:

```
SEscript = "ATDT5551212 CONNECT '' ogin: 'Joe User' word: secret  
PPP";
```

The sequence is:

1. Send ATDT5551212 - dials up an ISP.
2. Wait for the word CONNECT.
3. An empty send string, "" means don't send anything and wait for the next expect string.
4. Wait for "login:" or "Login:" By leaving off the 'L' either one will match.
5. Send 'Joe User' Note that this token is contained in single quotes, because it contains a space within it.
6. Wait for "password:" or "Password:"
7. Send the password.
8. Wait for the sequence 'PPP' This indicates a PPP session has started.

#### 5.4.1 Additional Rules for Send/Expect Scripts

- A carriage return character (ASCII 13) is automatically sent after each send token
- An ampersand(&) at the start of an expect token indicates that the driver should wait indefinitely for that token to be received. This is useful when waiting to answer a call, e.g., To set the modem to answer and wait indefinitely for a connection "ATS0=1 &CONNECT"
- As mentioned above, an empty token "" is immediately skipped. This allows for a chain of expect tokens to be used.
- The macro PSS\_MODEM\_CONNECT\_WAIT determines the total time for the script. If this is exceeded, a timeout failure will occur and the interface will fail to come up. Using the ampersand modifier resets this timeout.

Note that the IFS\_PPP\_USEMODEM specifies that PPP assumes that it is talking to a modem. When the interface is being brought up, it will first run through the send/expect script. After the script completes, PPP will assume that it can launch straight into LCP. If this is not appropriate, do not use IFS\_PPP\_SENDEXPECT or IFS\_PPP\_USEMODEM. Instead, use the facilities of MODEM.LIB to perform an appropriate login to the ISP. Only when this is complete should you call `ifup()`.

Use of MODEM.LIB entails some limitations:

- Only one PPP serial interface can use MODEM.LIB.
- You need to configure MODEM.LIB to match the serial port you are using for PPP.
- Ensure that you specify an IFCONFIG\_PPP\* default such that the interface remains "down" at boot-time. In other words, do *not* append IFS\_UP to the IFCONFIG\_PPP\* definition.

## 5.5 Starting and Stopping PPP Interfaces

The details of establishing and tearing down PPP links are handled by `sock_init()` and `tcp_tick()`, as are all other TCP/IP functions.

To start a PPP interface `ifup()` is used, just as it is for non-PPPoE Ethernet interfaces. One difference that you should note is that the interface will not usually be up after `ifup()` returns. `ifup()` only sets the process in motion, which takes much longer for PPP than it does for non-PPPoE Ethernet.

Your application should be aware of this, since you will not be able to open sockets on an interface that is not fully enabled. If necessary, you can poll the interface to wait for it to come up. While polling, you *must* call `tcp_tick()` regularly. This is because it is actually the processing driven from `tcp_tick()` that drives the whole PPP negotiation machinery.

The correct way to poll an interface is given by the following code fragment. This code includes tests for the possibility that the interface may not be able to come up (e.g., because of a time-out).

```
ifup(IF_PPP2);
while (ifpending(IF_PPP2) == 1) tcp_tick();
if (!ifstatus(IF_PPP2))
    printf("Failed!\n");
```

A similar consideration applies for bringing the interface down:

```
ifdown(IF_PPP2);
while (ifpending(IF_PPP2) == 3) tcp_tick();
```

Note that there is no need to test for an interface “failing to come down,” however the tear-down process may take a short time. If you wait for the interface to come down before restarting it then there is a better chance that the link will come back up successfully, since the peer will have been notified properly.

**NOTE:** For PPP links with `IFS_PPP_USEMODEM` in effect, the process of bringing the interface up and down will include the modem dial-out and hang-up procedure. If you had `USEMODEM` in effect when connecting, but turned it off during the connection, then `ifdown()` will *not* perform modem hang-up. You will need to “manually” hang up the modem (or possibly just renegotiate from the LCP phase, if this is what you intended, by calling `ifup()`).

## 6.0 API Functions

This section describes the functions for modem control.

---

---

### ModemClose

---

---

```
void ModemClose( void );
```

#### DESCRIPTION

Closes the serial driver down.

#### LIBRARY

MODEM.LIB

---

---

### ModemConnected

---

---

```
int ModemConnected( void );
```

#### DESCRIPTION

Returns true if the DCD line is asserted, meaning the modem is connected to a remote carrier.

#### RETURN VALUE

1: DCD line is active.  
0: DCD inactive (nothing connected).

#### LIBRARY

MODEM.LIB

---

---

## ModemExpect

---

---

```
int ModemExpect( char *send_string, unsigned long timeout );
```

### DESCRIPTION

Listens for a specific string to be sent by the modem.

### PARAMETERS

**send\_string** A NULL-terminated string to listen for.

**timeout** Maximum wait in milliseconds for a character.

### RETURN VALUE

1: The expected string was received.

0: A timeout occurred before receiving the string.

### LIBRARY

MODEM.LIB

---

---

## ModemHangup

---

---

```
int ModemHangup( void );
```

### DESCRIPTION

Sends "ATH" and "ATZ" commands

### RETURN VALUE

1: Success.

0: Modem not responding.

### LIBRARY

MODEM.LIB

---

---

## ModemInit

---

---

```
int ModemInit( void );
```

### DESCRIPTION

Resets modem with AT, ATZ commands.

### RETURN VALUE

1: Success.  
0: Modem not responding.

### LIBRARY

MODEM.LIB

---

---

## ModemOpen

---

---

```
int ModemOpen( unsigned long baud );
```

### DESCRIPTION

Starts up communication with an external modem.

### PARAMETERS

**baud**                    The baud rate for communicating with the modem.

### RETURN VALUE

1: External modem detected  
0: Not connected to external modem

### LIBRARY

MODEM.LIB

---

---

## ModemReady

---

---

```
int ModemReady( void );
```

### DESCRIPTION

Returns true if the DSR line is asserted.

### RETURN VALUE

1: DSR line is active.  
0: DSR inactive (nothing connected).

### LIBRARY

MODEM.LIB

---

---

## ModemRingin

---

---

```
int ModemRingin( void );
```

### DESCRIPTION

Returns true if the RI line is asserted, meaning that the line is ringing.

### RETURN VALUE

1: RI line is active.  
0: RI inactive (nothing connected).

### LIBRARY

MODEM.LIB

---

---

## ModemSend

---

---

```
void ModemSend( char *send_string );
```

### DESCRIPTION

Sends a string to the modem.

### PARAMETERS

**send\_string** A NULL-terminated string to be sent to the modem.

### LIBRARY

MODEM.LIB

---

---

## ModemStartPPP

---

---

```
void ModemStartPPP( void );
```

### DESCRIPTION

Hands control of the serial line over to the PPP driver.

### LIBRARY

MODEM.LIB

---

---

## PPPactive

---

---

```
int PPPactive( void );
```

### DESCRIPTION

Returns boolean value indicating if there is currently an active link to a peer.

### RETURN VALUE

>0: Active link to peer.  
0: No active link.

### LIBRARY

PPP.LIB

---

---

## PPPnegotiateIP

---

---

```
void PPPnegotiateIP( unsigned long local_ip, unsigned long remote_ip  
);
```

### DESCRIPTION

Sets PPP driver to negotiate IP addresses for itself and the remote peer. Otherwise, the system will rely on the remote peer to set addresses.

### PARAMETERS

|                        |  |
|------------------------|--|
| <code>local_ip</code>  | IP number to use for this PPP connection.        |
| <code>remote_ip</code> | IP number that the remote peer should be set to. |

### LIBRARY

PPP.LIB

---

---

---

## PPPSerialGetErrors

---

---

```
word PPPSerialGetErrors( void );
```

### DESCRIPTION

Gets a bit field with flags set for any errors that occurred. These flags are then cleared, so that a particular error will only cause the flag to be set once.

### RETURN VALUE

A bit field with flags for various errors. The errors along with their bit masks are as follows:

- PPP\_NOBUFFER 0x01
- PPP\_RXOVERRUN 0x02
- PPP\_BUFFEROVERFLOW 0x08

The high byte of the return value contains the number of CRC errors since the last call to this function (0-255).

### LIBRARY

PPPLINK.LIB

---

---

## PPPsetAuthenticatee

---

---

```
void PPPsetAuthenticatee( char *username, char *password );
```

### DESCRIPTION

Sets the driver up to send a PAP authentication message to a peer when requested.

### PARAMETERS

|                 |   |
|-----------------|---|
| <b>username</b> | The username to send to the peer. The argument string is not copied, so the argument string must stay constant. |
| <b>password</b> | The password to send to the peer. The argument string is not copied, so the argument string must stay constant. |

### LIBRARY

PPP.LIB

---

---

## PPPsetAuthenticator

---

---

```
void PPPsetAuthenticator( char *username, char *password );
```

### DESCRIPTION

Sets the driver up to require a PAP authentication message from a peer. Negotiation will fail unless the peer sends the specified username/password pair. This function is generally used when the Rabbit is acting as a dial-in server.

### PARAMETERS

|                 |   |
|-----------------|---|
| <b>username</b> | The user name that the peer must match for the link to proceed. |
| <b>password</b> | The password that the peer must match for the link to proceed.  |

### LIBRARY

PPP.LIB

---

---

## PPPshutdown

---

---

```
int PPPshutdown( unsigned long timeout );
```

### DESCRIPTION

Sends a Link Terminate Request packet. Waits for link to be torn down.

### PARAMETERS

|                |  |
|----------------|--|
| <b>timeout</b> | Number of milliseconds to wait before giving up on a response from the peer. |
|----------------|--|

### RETURN VALUE

1: Shutdown succeeded.  
0: Shutdown timed-out.

### LIBRARY

PPP.LIB