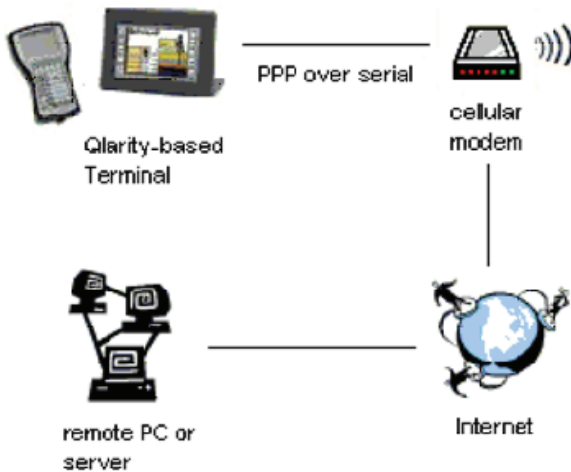


Introduction

What is PPP? PPP is the standard for transmission of IP packets over serial lines. This allows TCP/IP and UDP communication over a Qlarity-based terminal's serial port.

Why use PPP? When PPP communication is established in Qlarity, data can be transmitted or received using TCP/IP or UDP on a network accessed by a modem. If the modem has wireless Internet access, data can be sent or received on-the-go with a Qlarity-based terminal. This is limited by the Internet provider's area-of-service, just like a cellular phone.

Diagram of using a Qlarity-based terminal to communicate with a remote device over PPP.



It is helpful to have a basic knowledge of Qlarity Foundry® for this tutorial. Free copies of Qlarity Foundry, Qlarity tutorials, documentation and other resources are available at www.qlarity.com.

The PPPComm Object

First, launch Qlarity Foundry and start a new workspace. When the “Select the Type of Workspace to Create” window appears, select the “Standard workspace” option. This will automatically include the libraries needed such as DataProc.qlib. Select *PPPCommV2* (green icon)

from the *Object Palette* and create an instance of this object. Rename the object *PPPComm*.

Select the *PPPComm* object in the Object Tree and look at the properties of this object in the Object Properties window in the lower right of the screen. Properties that need to be configured before attempting PPP communication are the *comport* and properties in the *Modem Initialization* category. With most modems you must send over a sequence of AT commands before you can negotiate a PPP connection. These commands are different for each brand of modem. With the *Modem Initialization* properties a sequence of commands can be specified to be sent to the modem automatically before opening a PPP connection. A PPP connection cannot be established in “Simulation View” of Qlarity Foundry, so the testing will need to be done on the terminal.

Here is a sample configuration for talking to a CDM-819s Modem from Land Cellular.

▼ Modem Initialization	
modemcommand1	"AT\r\n"
modemresponse1	"OK"
timeperiod1	5000
retries1	6
modemcommand2	"AT+IFC=0,0\r\n"
modemresponse2	"OK"
timeperiod2	5000
retries2	2
modemcommandx	"ATDT#777\r\n;"
modemresponsex	"CONNECT;"
timeperiodx	"15000;"
retriesx	"4;"

From this configuration the terminal must successfully send three commands and receive a valid response from each command before attempting PPP communication. A successful sequence would include these events.

1. Send the string “AT\r\n”.
2. Receive the string “OK” within 5000 milliseconds (ms).

3. Send the string “AT+IFC=0,0\r\n”.
4. Receive the string “OK” within 5000 ms.
5. Send the string “ATDT#777\r\n”.
6. Receive the string “CONNECT” within 15 seconds.

The PPP object handles this communication.

Serial Port Settings

Before attempting to communicate with a modem, make sure the terminal’s serial settings such as Baud Rate, Parity, Flow Control and Stop Bits are configured correctly to match the modem. These settings can be configured in Power On Setup as described in the “Qlarity-based Terminal Hardware Manual” or by using the built-in function *SetSystemSetting* (see F1 in Qlarity Foundry for documentation).

Verify the wiring is correct to the modem. See the section “Wiring from the Terminal to the Modem” in this document, as well as the “Qlarity-based Terminal Hardware Manual” for more information.

PPPComm Events

After configuring the *PPPComm* object, try communicating with the modem using the terminal. The *PPPComm* object includes many events to indicate the status of a connection. See the included example *PPP_test.qly* for an example of handling the different events to verify communication with the modem. One very useful event is *ConnectionEstablished()*. This event will occur when PPP is negotiated and ready for Ethernet communication. This would be a good place to open a TCP/IP connection using an *EthernetClientV2* object.

Example event for PPPComm object:

```
func ConnectionEstablished()
    ethernetClient.openConnection()
endfunc
```

PPPComm Functions

A useful function of the *PPPComm* object is *GetState()*. *GetState* returns the status of the PPP connection. When opening an Ethernet connection outside of the *ConnectionEstablished* event this function should be called. The *EthernetClientV2* has a similar function called *GetConnectionState* that can be used to determine

the status of an Ethernet connection.

Example functions placed in Globals:

```
`open connection if possible
func OpenEthConnection()
    if PPPComm.GetState() <> pppcomm_ppp_connection then
        return
    else
        if ethernetclient.getConnectionState() == ec_notconnected then
            ethernetClient.openConnection()
        endif
    endif
endfunc

`determine if connection is established
func IsEthConnection() returns boolean
    if pppcomm.GetState() <> pppcomm_ppp_connection then
        return false
    endif
    if ethernetclient.getConnectionState() == ec_connectionmade then
        return true
    else
        return false
    endif
endfunc
```

Wiring from the Terminal to the Modem

When connecting a QTERM® terminal to a modem, a standard null modem connector does not work reliably, so the connector will depend on the type of modem being used. The modems QSI has tested have shown problems when using a null modem adapter. This document attempts to describe what is happening and why.

The terms Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) are used to describe the assignment of pins on devices which use an EIA-232 as a serial protocol. A PC is said to be a DTE device while a modem is said to be a DCE device.

The serial port connections of a QSI terminal are wired as DCE devices. This makes connecting them to a PC very straight forward. They can simply be connected to the PC using a straight-thru cable (because the connection is from DTE to DCE). Unfortunately, this makes

connecting a QSI terminal to a modem a challenge. Instead of connecting DTE to DCE (as is the case for PC to a QSI terminal), or DTE to DTE (as is the case for PC to PC where a simple null modem connection can be utilized), The connection is DCE to DCE.

To understand the problem, first consider the serial connector of a PC. To connect two PC serial ports together and have them communicate, a straight-thru cable cannot be used as they would both send out signals on the same wire to each other, but neither would be able to receive any signal (consider pins 2 and pins 3). The solution is to cross-over some of the signals. Doing so removes the modem, thus this type of connection is called a “null modem” connection. There are commercially-available products to perform this function.

The standard null modem connection (commercially available from various vendors) is shown in Figure 1. This connection works well for connecting two PC serial ports together (DTE to DTE).

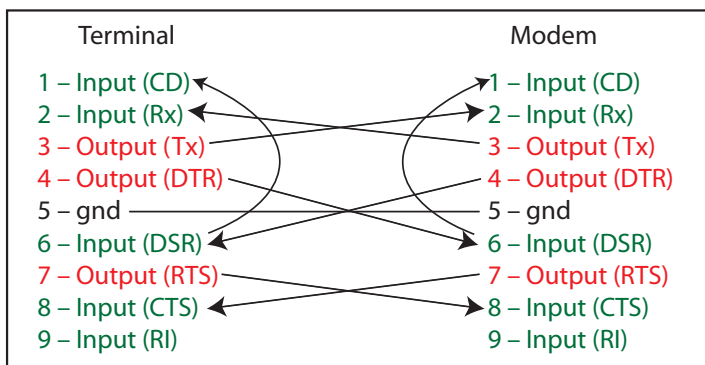


Figure 1

Using the same Null modem connector, as was shown previously, to connect the QTERM and a modem which are DCE devices would result in the following.

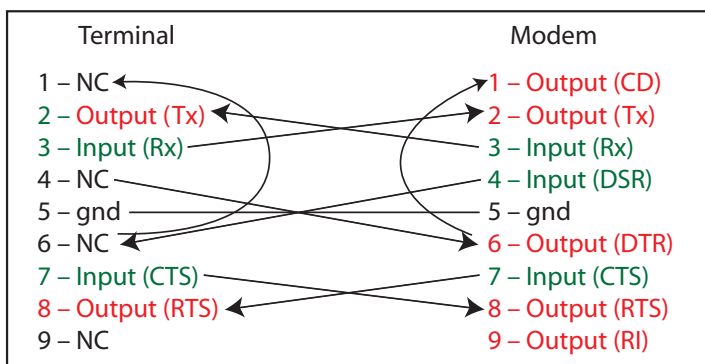


Figure 2

There are two different problems with this connection. The first problem is two outputs on the modem have been connected together (pins 6 and 1). For DTE these were both inputs, but for DCE devices they are outputs. Connecting two outputs together can have negative effects such as damaging the output circuitry or causing excessive current draw by the device resulting in overheating.

The second problem is that an input with no connection exists (pin 4 of the modem). QSI’s investigation of issues between the QTERM and the modem found that leaving this input unconnected causes the modem to misbehave. This especially becomes a problem when a long cable is used between the modem and the QTERM. In such a case, the input will appear to be connected to a “long antenna” which couples in switching noise from the transmit/receive signals in the same cable. It is possible to tell the modem to ignore hardware handshake lines (such as this input, DSR) using an AT command, but was not always reliable with the modems QSI tested. In particular, QSI noticed that during the time when the modem would switch from AT mode to PPP mode, the modem seemed to react to transitions on the DSR line and drop bytes from the QTERM. The only reliable method found is to ensure the input signal DSR is connected. In testing, the following connection was made which resulted in reliable communications (see Figure 3).

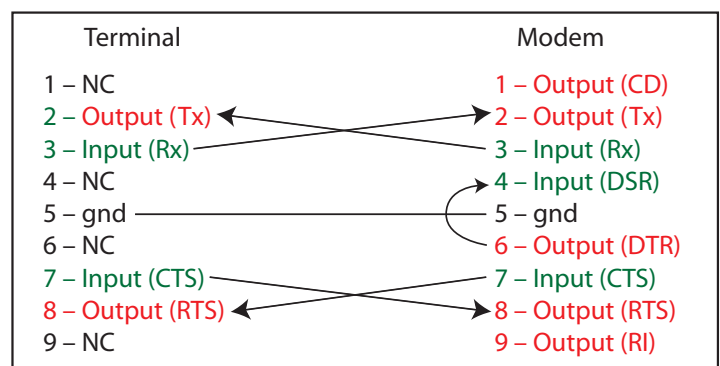


Figure 3

While the connector illustrated worked for the modems tested, there is no guarantee it will work for all modems. For reliable communications, if a modem is DCE and the QTERM is DCE, a standard null modem connector should not be used. Instead, a cable wired specifically for your modem will have to be used.

Further Reading

Qlarity FAQ: DataRecieved,
MSG_COMM_RECEIVED and you
<http://www.qsiforums.com/viewtopic.php?t=128>

Qlarity FAQ: Can I change serial settings in my pro-
gram? <http://www.qsiforums.com/viewtopic.php?t=43>

Links

Qlarity Terminal Hardware Manual
<http://www.qsicorp.com/support/files/>

Land-Cellular: <http://www.calamp.com/home/landcell.html>