Application Note

Sample Programs for Using the xPico® Wi-Fi® Pi Plate

Part Number 920-931
Revision A October 2014

Intellectual Property

Contacts

Lantronix, Inc.
167 Technology Drive
Irvine, CA 92618, USA
Toll Free: 800-526-8766
Phone: 949-453-3990
Fax: 949-453-3995

Technical Support
Online: www.lantronix.com/support

Sales Offices
For a current list of our domestic and international sales offices go to the Lantronix web site at www.lantronix.com/about/contact.

Disclaimer

All information contained herein is provided "AS IS." Lantronix undertakes no obligation to update the information in this publication. Lantronix does not make, and specifically disclaims, all warranties of any kind (expressed, implied or otherwise) regarding title, non-infringement, fitness, quality, accuracy, completeness, usefulness, suitability or performance of the information provided herein. Lantronix shall have no liability whatsoever to any user for any damages, losses and causes of action (whether in contract or in tort or otherwise) in connection with the user's access or usage of any of the information or content contained herein. The information and specifications contained in this document are subject to change without notice.

Lab Setup

The following lab provides an example of how to use the Lantronix® xPico® Wi-Fi® Pi Plate to connect an xPico Wi-Fi embedded device server to a Raspberry Pi® microcomputer board. This example uses the Raspberry Pi B+ microcomputer board, however, other Raspberry Pi platforms may be utilized.

In order to use this sample application, copy the relevant html file onto the xPico Wi-Fi device server, and copy three shell scripts into the same directory on the Raspberry Pi unit.

All the files required for this sample can be downloaded from the Lantronix website.

The following link will lead you to a set of videos that will take you through the set-up and execution of this sample application for the Raspberry Pi using the xPico Wi-Fi Pi Plate:

https://www.youtube.com/playlist?list=PLd5kQjx7Qlxxmy4mxnkQycFbQV-w0c_9H

To get the full benefit of all the capabilities of the xPico Wi-Fi module, it is recommended that you reference the user guide. This document can be downloaded from the Lantronix website (http://www.lantronix.com/pdf/xPico-Wi-Fi_UG.pdf).

Jumper Settings

This sample application does not require changing or modifying the xPico Wi-Fi Pi Plate jumpers from their default settings. The default jumper settings for the xPico Wi-Fi Pi Plate are described in *Table 1* below. For more information about these jumpers, refer to the *xPico Wi-Fi Pi Plate User Guide* (part number 900-710-R).

Table 1 Jumper Settings

JP	Position	Label	Function	Default
JP1	1-2	UUT PWR	Connects to 0.301 ohm current sense resistor R1. Measure voltage on JP1 to calculate module power consumption	Not installed
JP17	1-2	WLAN LED	Install to use WLAN LED	Installed
JP17	3-4	WAKE	Install to use wake-up input and button,SW1	Installed
JP17	5-6	RXD2	Install to route xPico Wi-Fi module second serial port to J9 via the on board USB to serial converter	Installed
JP17	7-8	TXD2	Install to route xPico Wi-Fi module second serial port to J9 via the on board USB to serial converter	Installed
JP17	9-10	DEFAULTS	Install to use Defaults input and button, SW2	Installed
JP17	11-12	RESET	Install to use Hardware Reset input and button, SW3	Installed
JP3	1-2	TX	Install position 1-2 to connect xPico module TXD1 to Raspberry Pi computer board serial RX.	Installed
JP5	1-2	RX	Install position 1-2 to connect xPico module RXD1 to Raspberry Pi computer board serial TX.	Installed
JP2	1-2	CP1	Breakout header for CP1	Installed
JP2	3-4	CP2	Breakout header for CP2	Installed
JP2	5-6	CP3	Breakout header for CP3	Installed
JP2	7-8	CP4	Breakout header for CP4	Installed
JP2	9-10	CP5	Breakout header for CP5	Installed

JP	Position	Label	Function	Default
JP2	11-12	CP6	Breakout header for CP6	Installed
JP2	13-14	CP7	Breakout header for CP7	Installed
JP2	15-16	CP8	Breakout header for CP8	Installed
JP2	17-18	RTS1	Header for RTS1, pin 18 does not connect anywhere else on the board.	Installed
JP2	19-20	CTS1	Header for CTS1, pin 20 does not connect anywhere else on the board.	Installed
JP6	1-2	POWER	Install pins 1-2 to power plate board from Raspberry Pi computer board	Installed
JP10	1-2	3.3V	3.3V power generated by the on board regulator	Not installed
JP11	1-2	GND	Board signal ground.	Not installed

Sample Application

This sample application demonstrates the use of a Wi-Fi enabled mobile device to control an HVAC simulator running on a Raspberry Pi B+ and then exports the results into the Bug Labs Cloud Server.

The output of the simulation updates the web page hosted on the xPico Wi-Fi module and also pushes the data to a cloud server. The sample application demonstrates the real life capabilities of an on-board web server, simultaeous Soft Access Point and client, with data being communicated to and from a mobile device and a network access point all over a simple serial port that connects the Raspberry Pi and the xPico Wi-Fi. *Figure 1* below illustrates this sample application.

The files and scripts necessary to reproduce this sample application are available and can be downloaded from the Lantronix website at (http://www.lantronix.com/resources/appnotes.html)

Figure 1

Setup of the Raspberry Pi Plate

The Lantronix xPico Wi-Fi Pi Plate communicates with the Raspberry Pi through the serial port interface. By default, the Raspberry Pi is configured to run *getty* (login) on the serial port. To successfully run the demo, you will need to disable *getty* from running on the port. This step is performed within the demo script, which implies that you must run the script as root (sudo).

1. Disable the *getty* from running on the serial port interface.
2. Download the `doXPWdemo.sh`, `mux.sh`, and the `hvacSim.sh` files to the Raspberry Pi using FTP or another copy mechanism. (***Note:*** *You can use a browser to the xPico Wi-Fi IP address if it is associated with the infrastructure network and you have Ethernet access to that network from your Raspberry Pi.*)
3. Place these three files into a common directory and set the file permissions to allow them to be executed.
   - `chmod 755 doXPWdemo.sh`
   - `chmod 755 hvacSim.sh`
   - `chmod 755 mux.sh`

Setup the xPico Wi-Fi Unit

1. Gain access to the xPico Wi-Fi web manager.
2. Navigate to ***File System -> Browse*** in the web manager.
3. If not already present, create an **http** directory.
4. Select the **http** directory that you created.
5. Upload the files extracted from the `PiPlateDemo1.0.zip` file to the **http** directory (see *Figure 2*).
6. By default, the Soft AP mode is enabled with a default SSID of **xPicoWiFi_xxxxxx**, where **xxxxxx** are the last six characters of the unique xPico Wi-Fi serial number. This number is available on the module label.
7. Connect your device to the xPico Wi-Fi device's Soft AP SSID. The default password is **XPICOWIFI**.
8. Open a standard browser and in the address field of the browser enter the following URL: xpicowifi.lantronix.com or alternatively use 192.168.0.1 as the IP Address.
9. When prompted enter the username, **admin** and the password, **PASSWORD** to access the configuration and management web pages.

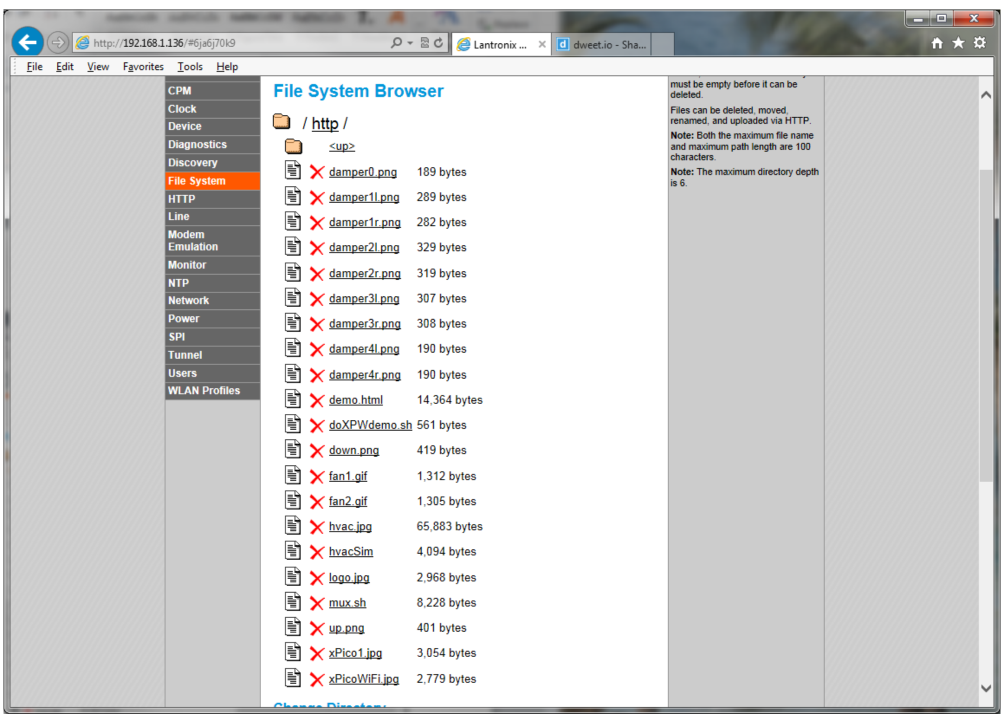


Figure 2

Configuring the Serial Channel Changes

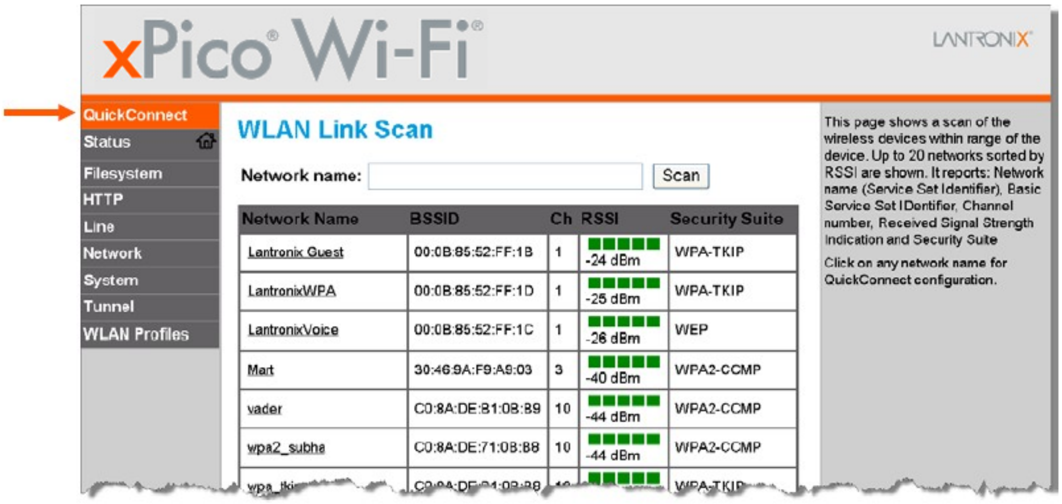
1. Gain access to the xPico Wi-Fi web manager.
2. Navigate to ***Line -> Line 1*** *(or **Line 2**)* ***-> Configuration*** in the web manager.
3. Configure the device for Mux. The remaining serial settings should be 9600, 8 data bits, no parity, 1 stop bit, and no flow control (see *Figure 3*).
4. Click **Submit** to save the settings.

Figure 3

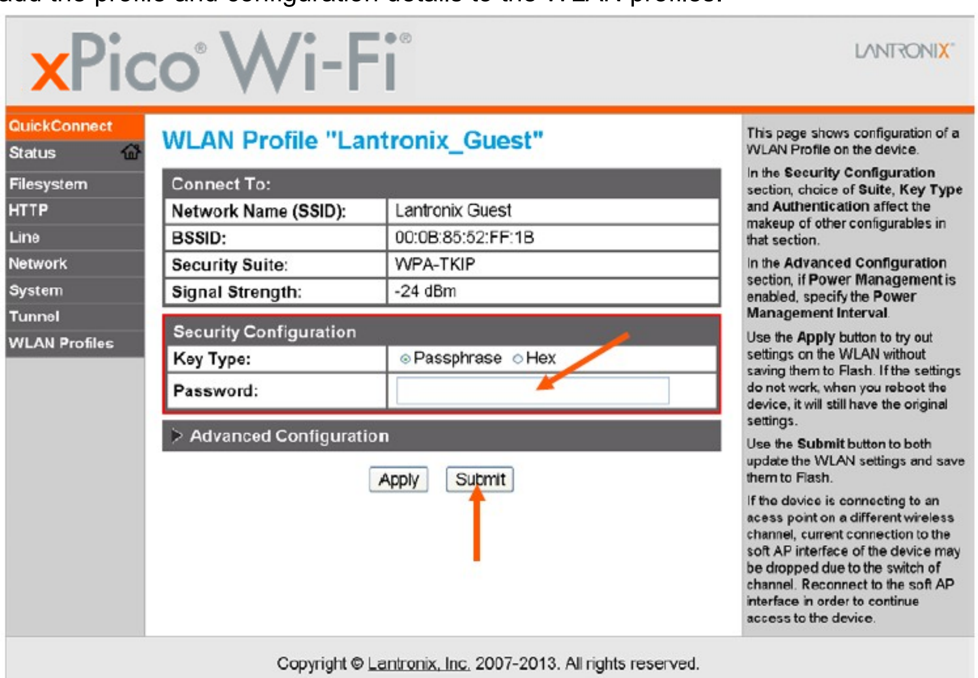
xPico Wi-Fi Network Configuration

Configure the WLAN Profile and Network wlan0 interface to allow the xPico Wi-Fi Pi Plate to associate with your Wi-Fi infrastructure network. This is **required** to perform cloud posting (via dweet.io).

1. Gain access to the xPico Wi-Fi web manager.
2. Navigate to **QuickConnect** on the left side menu bar of web manager.



3. Upon selection of the **QuickConnect** option, the xPico Wi-Fi device scans and displays up to 20 wireless devices in order of strongest signal strength at the top.
4. Click on a network name to view the connection to the desired Access Point. The Access Point will display.
5. Enter the password and click **Submit** to directly connect to the Access Point and to add the profile and configuration details to the WLAN profiles.



6. Click on **Status** on the left menu bar of web manager, then look for the IP address under Interface (wlan0) to see the IP address assigned to the xPico Wi-Fi device by the infrastructure network. You may now connect to the xPico Wi-Fi device from the infrastructure side and the Soft AP side.

Demo Execution

Follow the steps below to start the demo.

1. Create your own DweetName for this example in order to ensure that you avoid conflicts when reviewing the posted data. (Use `control-C` to exit the doXPWdemo, or send the SIGHUP signal.)
2. Execute the `doXPWdemo.sh` file. The demo will open the serial port and communicate with the Lantronix xPico Wi-Fi Pi Plate.

```
sudo ./doXPWdemo [DweetName]
```

   where: DweetName is optional, but if supplied, the demo will also “dweet” the current HVAC simulation to dweet.io. The name supplied here is the device name to follow at http://dweet.io/follow/<DweetName>. See *xPico Wi-Fi Network Configuration* above.

3. While using your browser, surf to the demo.html file on xPico Wi-Fi Pi Plate.

```
http://<IP Address of xPico Wi-Fi Pi Plate>/demo.html
```

Once the web page loads, there are three adjustable settings (see *Figure 4*). You may change the **Outside Air Temperature**, the **Room Air Temperature**, and the **Occupancy** of the room. The air temperatures will affect damper movement and the water supply valve openings. Room Occupancy will affect the CO_2 levels, which affect damper opening of outside air. The recirculating fan should continue to run until the current values match the set points or if the CO_2 level is too high. The simulation takes about 45 seconds to complete per “set point” change.

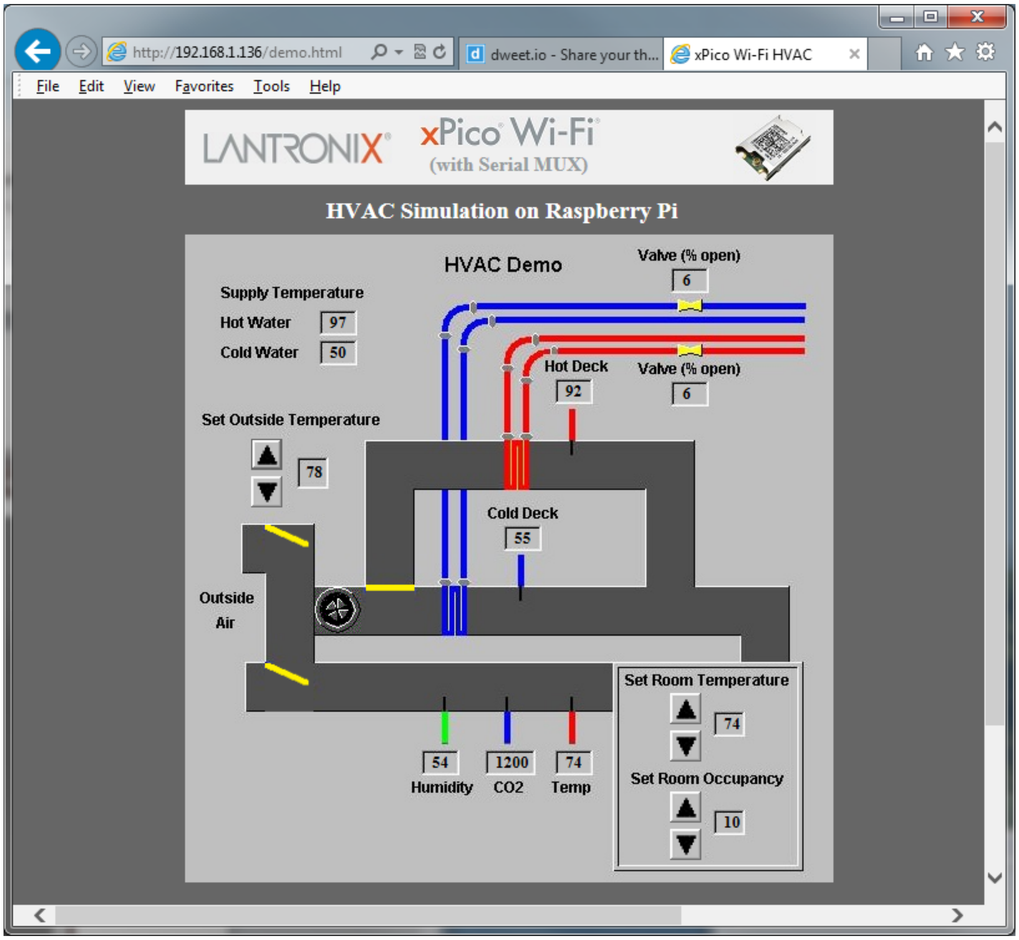

http://192.168.1.136/demo.html
dweet.io - Share your th...
xPico Wi-Fi HVAC
File
Edit
View
Favorites
Tools
Help
LANTRONIX
xPico Wi-Fi
(with Serial MUX)
HVAC Simulation on Raspberry Pi
HVAC Demo
Valve (% open)
6
Supply Temperature
Hot Water
97
Cold Water
50
Hot Deck
92
Valve (% open)
6
Set Outside Temperature
78
Cold Deck
55
Outside
Air
Set Room Temperature
74
Set Room Occupancy
10
54
Humidity
1200
CO2
74
Temp

Figure 4

APPENDIX: Sample Code

DoXPWdemo.sh

This is the top level code for the demo.

```
#!/bin/bash
#
# RAM based temp storage directory
rbTmp=/run/shm
# Serial port speed
baudRate=9600
# Serial port
ttyPort=ttyAMA0
# debug log file.  Set to /dev/null to disable logging
logfile=xpwlog
# device name for the cloud post. Set to "" to disable post to dweet.io
if [ $# -ne 0 ]; then
      deviceName=$1
else
      deviceName=""
fi
#
function cleanExit()
{
      echo "Restoring inittab"
      if [ -f $rbTmp/inittab$$ ]; then
            cp $rbTmp/inittab$$ /etc/inittab
      fi
      echo "Restarting getty on $ttyPort"
      kill -HUP 1
      exit 0
}
#
trap cleanExit 1 2 3
#
echo "Backing up inittab"
cp /etc/inittab $rbTmp/inittab$$
echo "Changing inittab"
sed 's/^T0/#T0/' < /etc/inittab > $rbTmp/inittab1$$
cp $rbTmp/inittab1$$ /etc/inittab
echo "Stopping getty on $ttyPort"
kill -HUP 1
echo "Starting xPico Wi-Fi Pi Plate Demo on $ttyPort"
./mux.sh $baudRate $rbTmp $logfile $deviceName <> /dev/$ttyPort >&0
#
cleanExit
#
```

Mux.sh

This code receives data from the webserver and pushes it up to the cloud; all over single serial connection.

```
#!/bin/bash
# ./mux.sh baudRate tmpDirectory logfile deviceName <> /dev/ttyAMA0 >&0
#
if [ $# -ne 4 ]; then
      echo "usage: mux.sh baudRate tmpDirectory logfile deviceName"
      exit 0
fi
# debugging file.  Set to /dev/null to disable logging
if [ "$3" == "/dev/null" ]; then
      logfile=$3
else
      logfile=$2/$3
fi
#
# temporary file to hold the HTTP POST to the cloud
tmpfile=$2/dweet$$
#
# temporary file to hold the cloud response
rdfile=$2/dweetRsp$$
#
# device name for the cloud post
deviceName=$4
#
# a few global script variables
c=""
readIn=""
postOut=""
postRunning=0
lcnt=0
#
# destroy the mux instances before exit
#
cleanExit()
{
      echo " "
      read -t 1 junk
      echo "1k"
      read -t 1 junk
      echo "2k"
      read -t 1 junk
      stty 9600 sane
      exit 0
}
#
# Build the cloud GET message header (for posting data to the cloud) &
send it
#
doDweet()
{
      echo -n "GET /dweet/for/$deviceName?" > $tmpfile
      echo -n "$1" >> $tmpfile
```

```
        echo -n " HTTP/1.1" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -n "Host: dweet.io" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -n "User-Agent: xPicoPiPlate/1.0" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -n "Accept: text/html,application/xhtml+xml" >> $tmpfile
        echo -n ",application/xml" >> $tmpfile
        echo -n ";q=0.9,*/*;q-0.8" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -n "Accept-Language: en-US,en;q=0.5" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -n "Accept-Encoding: identity" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -n "Connection: keep-alive" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -n "Cache-Control: max-age=0" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
        echo -e "\x0D" >> $tmpfile
                # connect to the cloud service
        if [ $postRunning -eq 0 ]; then
                lcnt=0
                postRunning=1
                doCmd "2cdweet.io:80"
                if [ "$cmdRsp" != "K" ]; then
                                # failure, kill instance
                        postRunning=0
                        doCmd 2k
                fi
        else
                lcnt=`expr $lcnt + 1`
                if [ $lcnt -ge 3 ]; then        # missed
                        postRunning=0
                        doCmd 2k
                fi
        fi
}
#
# read response data from cloud service (upto 1024 bytes)
# and stuff it in a file ($2)
#
muxReadBinaryFile()
{
        cat /dev/null  > $2
        doCmd "$1rb*1024"
        if  [ "$cmdRsp" == "K" ]; then
                while [ 1 ]
                do
                        c=
                        read -t 1 -n 1 c
                        if [ $? -eq 0 ]; then
                                if [ "$c" == "*" ]; then
                                        c1=
                                        read -t 1 -n 1 c1
                                        if [ "$c1" != "*" ]; then
                                                break
                                        fi
```

```
                        fi
                        echo -n "$c" >> $2
                else
                        break
                fi
        done
        endRead $1
    fi
}
#
# read POSTed data from web browser (upto 256 bytes)
# and stuff it in a variable readIn
#
rdEvent()
{
    doCmd "$1rb*256"
    if  [ "$cmdRsp" == "K" ]; then
#               read POST message (luckily, it terminates with
newline)
        read -t 5 readIn
        endRead $1
    fi
}
#
# terminate the mux read operation
endRead()
{
#           send command byte to end mux operation
    echo -n "$1"
#               read should be "*\n"
    read -t 1 junk
#           send command terminator
    echo ""
#               read should be "K" or "W"
    c=
    read -t 1 -n 1 c
}
#
# write binary data to the mux
muxWriteBinary()
{
    echo "$1sb~"
    incnt=
    c=
#           Get the mux buffer size
    while [[ "$c" != "K" ]]
    do
        read -n 1 c
        case "$c" in
            E)    # error message
                read -t 1 junk
                break
                ;;
            K)
                break
                ;;
            *)
```

```
                        incnt=$incnt$c
                        ;;
                esac
        done
        if [ "$c" == "K" ]; then
#                       calculate buffer size needed
                if [ "$1" == "1" ]; then
                        dataBytes=${#2}
                elif [ "$1" == "2" ]; then
                        dataBytes=$(wc -c "$2" | cut -f 1 -d ' ')
                fi
                if [ $incnt -gt $dataBytes ]; then
                        if [ "$1" == "1" ]; then
                                echo -n $2
                        elif [ "$1" == "2" ]; then
                                cat $2
                        fi
                        echo "~"
                        read -t 5 -n 1 c
                fi
        fi
}
#
# Open a new listen socket
# $1 - instance
# $2 - close & open options
muxListen()
{
        case "$2" in
                0)      doCmd $1h
                        doCmd W$1r
                        ;;
                1)      doCmd $1k
                        doCmd $1h
                        doCmd W$1r
                        ;;
                2)      doCmd $1p
                        doCmd $1f
                        doCmd $1e
                        doCmd $1h
                        ;;
        esac
}
#
# wait for a mux event
waitEvents()
{
        kcnt=0
        while [ 1 ]
        do
                eList=""
                evt=""
                read -t 5 eList
                if [ $? -ne 0 ]; then
                        if [ $postRunning -eq 1 ]; then
                                doCmd 2k
                                postRunning=0
```

```
                        doCmd W1r
                  else
#                             check channel
                        doCmd 1
                        if  [ "$cmdRsp" == "D" ]; then
                              muxListen 1 0
                              kcnt=0
                        elif  [ "$cmdRsp" == "E" ]; then
                              muxListen 1 1
                              kcnt=0
                        elif  [ "$cmdRsp" == "K" ]; then
                              kcnt=`expr $kcnt + 1`
                              if [ $kcnt -ge 3 ]; then
                                    muxListen 1 1
                                    kcnt=0
                              fi
                        fi
                  fi
            else
                  evt=`echo $eList | cut -b1-2`
            fi
            case "$evt" in
                  1r)   # read ready on channel 1
                        kcnt=0
                        rdEvent 1
                        resp1=`./hvacSim.sh -r "$readIn" $logfile`
                        muxWriteBinary 1 "$resp1"
                        muxListen 1 2
                        if [ ${#deviceName} -ne 0 ]; then
                              postOut=`./hvacSim.sh -p "$readIn"
$logfile`
                              doDweet $postOut
                        fi
                        if [ $postRunning -eq 1 ]; then
                              doCmd W1r2s
                        else
                              doCmd W1r
                        fi
                        ;;
                  2r)   # read ready on channel 2
                        muxReadBinaryFile 2 "$rdfile"
                        doCmd 2e
                        postRunning=0
                        doCmd W1r
                        ;;
                  2s)   #send ready on channel 2
                        muxWriteBinary 2 "$tmpfile"
                        doCmd 2p
                        doCmd W2r
                        ;;
            esac
      done
}
#
# simple send command and parse response
# $1 - command
# if not in sync, this read operation will block - that's bad
```

```
doCmd()
{
      echo "$1"
      junk=
      cmdRsp=
      while [ "$cmdRsp" == "" ]
      do
        read -n 1 cmdRsp
        if  [ "$cmdRsp" != "" ]; then
          case "$cmdRsp" in
            E)    read -t 1 junk
                  ;;
            K)    ;;
            D)    ;;
            W)    ;;
            *)
                  read -t 1 junk
                  cmdRsp="U"
                  ;;
          esac
        fi
      done
}
#
# sync up with mux inteperter
syncToStart()
{
# we don't know if stale data is available
#                 read until there is nothing
      echo "1"
      while [ 1 ]
      do
            read -t 1 -n 1 junk
            if [ $? -ne 0 ]; then
                  break
            fi
      done
#                 send status command to sync with mux parser
      doCmd 1
#                 kill both instances - just in case we are mid
#operation
      doCmd 1k
      doCmd 2k
}
#
# Main script entry
#
# open the log, and disable tty echo and output processing
#
trap cleanExit 1 2 3
echo "open log $$" > $logfile
echo "HTTP mux" >> $logfile
stty $1 -echo -opost -icanon
                  # clear & sync the communication channel
syncToStart
                  # wait for events - never returns
waitEvents
```

```
                  # clean up and exit
cleanExit
#
```

HvacSim.sh

This code simulates HVAC operations.

```
#!/bin/bash
if [ $# -ne 3 ]; then
      echo "usage: hvacSim.sh -mode[p,r] postBody logfile"
      exit 0
fi
logfile=$3
# parameter list
sparam[0]=0
sparam[1]=0
sparam[2]=0
sparam[3]=0
sparam[4]=0
sparam[5]=0
sparam[6]=0
sparam[7]=0
sparam[8]=0
sparam[9]=0
sparam[10]=0
sparam[11]=0
sparam[12]=0
sparam[13]=0
sparam[14]=0
sparam[15]=0
sparam[16]=0
sparam[17]=0
#
# sane input in case of error
function defInput()
{
      sparam[0]=78
      sparam[1]=72
      sparam[2]=10
      sparam[3]=97
      sparam[4]=50
      sparam[5]=6
      sparam[6]=6
      sparam[7]=92
      sparam[8]=55
      sparam[9]=54
      sparam[10]=1200
      sparam[11]=74
      sparam[12]=1
      sparam[13]=335
      sparam[14]=335
```

```
        sparam[15]=180
        sparam[16]=74
        sparam[17]=0
}
#
# parse POST body into needed parameters
function parseEM()
{
        ref='^[0-9]+$'
        list1=`echo $1 | tr '\n' ' ' | sed 's/ //g' `
        list1=`echo $list1 | tr "\&" " "`
        n=0
        for tag in $list1
        do
                sparam[n]=`echo $tag | cut -d= -f2`
                if ! [[ ${sparam[n]} =~ $ref ]]; then
                        if [ $n -lt 17 ] ; then
                                defInput
                        fi
                fi
                n=`expr $n + 1`
        done
}
#
# main script starts here
#parseEM
"oSP=85&rSP=72&pSP=10&hs=97&cs=50&hv=6&cv=6&hd=92&cd=55&hu=54&co=1200&r
a=74&fan=0&dp1=335&dp2=335&dp3=180&rat=74123&dT=00000"
# parse the variable list
parseEM $2
#
# assign them to better variable names
outAirT=${sparam[0]}
rs=${sparam[1]}
ro=${sparam[2]}
hs=${sparam[3]}
cs=${sparam[4]}
hv=${sparam[5]}
cv=${sparam[6]}
hd=${sparam[7]}
cod=${sparam[8]}
hu=${sparam[9]}
co=${sparam[10]}
rt=${sparam[11]}
fan=${sparam[12]}
damp0=${sparam[13]}
damp1=${sparam[14]}
damp2=${sparam[15]}
retAirT=${sparam[16]}
deltaT=${sparam[17]}
#
inT=$rt
hv=6
cv=6
#
if [ $deltaT -eq 0 ]; then
        deltaT=`expr $rs - $rt`
```

```
      deltaT=`expr $deltaT \* 1000 / 10`
fi
#
if [ $fan -ne 0 ]; then
      dp=`expr $co - 400`
      dp=`expr $dp / 10`
      if [ $dp -gt 90 ]; then
            dp=90
      fi
      damp0=`expr $dp + 270`
      co=`expr $co - $dp \* 4 / 9`
fi
co=`expr $co + $ro`
dp=`expr $damp0 - 270`
dp=`expr $dp \* 10 / 9`
inT=`expr 100 - $dp`
inT=`expr $outAirT \* $dp + $rt \* $inT`
inT=`expr $inT + 50`
inT=`expr $inT / 100`
if [ $rs -lt $inT ]; then
      cv=`expr $inT - $rs`
      cv=`expr $cv \* 13`
      damp2=180
      fan=1
elif [ $rs -gt $inT ]; then
      hv=`expr $rs - $inT`
      hv=`expr $hv \* 13`
      damp2=270
      fan=2
else
      damp2=203
      fan=0
fi
if [ $hv -gt 100 ]; then
      hv=100
elif [ $hv -lt   0 ]; then
      hv=0
fi
if [ $cv -gt 100 ]; then
      cv=100
elif [ $cv -lt   0 ]; then
      cv=0
fi
if [ $outAirT -lt 35 ]; then
      cv=5
fi
if [ $ro -eq 0 ]; then
      if [ $co -gt 400 ]; then
            if [ $damp0 -gt 270 ]; then
                  if [ $fan -ne 0 ]; then
                        co=`expr $co - 10`
                  fi
            fi
      fi
fi
if [ $co -lt 400 ]; then
      co=400
```

```
elif [ $co -gt 500 ]; then
      if [ $damp0 -lt 300 ]; then
            damp0=300
      fi
      fan=1
fi
if [ $fan -eq 0 ]; then
      damp0=270
fi
damp1=$damp0
# scale them for integer math
rs1=`expr $rs \* 1000`
hd1=`expr $hd \* 1000`
cod1=`expr $cod \* 1000`
if [ $retAirT -ne $rs1 ]; then
      retAirT=`expr $retAirT + $deltaT`
fi
if [ $retAirT -lt $cod1 ]; then
      if [ $fan -ne 0 ]; then
            retAirT=$cod1
      fi
elif [ $retAirT -gt $hd1 ]; then
      if [ $fan -ne 0 ]; then
            retAirT=$hd1
      fi
fi
rat=`expr $retAirT / 1000`
#echo "retA $retAirT $deltaT"
#
# output desired result format
if [ "$1" == "-r" ]; then
      echo -n "\$$hs, $cs, $hv, $cv, $hd, $cod, $hu, $co, $rat, "
      echo "$fan, $damp0, $damp1, $damp2, $retAirT, $deltaT*"
elif [ "$1" == "-p" ]; then
      echo -n "oSP=$outAirT&rSP=$rs&pSP=$ro"
      echo -n
"&hs=$hs&cs=$cs&hv=$hv&cv=$cv&hd=$hd&cd=$cod&hu=$hu&co=$co"
      echo -n "&ra=$rat&fan=$fan&dp1=$damp0&dp2=$damp1"
      echo    "&dp3=$damp2&rat=$retAirT&dT=$deltaT"
else
      echo "error: bad mode, -p or -r must be supplied"
fi

exit 0
```