Telnet Control of a VLC Daemon

Darryn Anton Jordan
jrddar001@myuct.ac.za

June 25, 2015

Contents

0.1 Introduction

This document details the method used to initiate and terminate recordings of a live feed, streamed from an IP camera using VLC (VideoLAN Client) in telnet interface.

In order to avoid confusion, some terms will now be introduced. Each radar node will be referred to as a server. This server will run VLC in its daemon mode and have a network connection to the IP camera.
The computer used to control and make requests from the server will be referred to as a client.

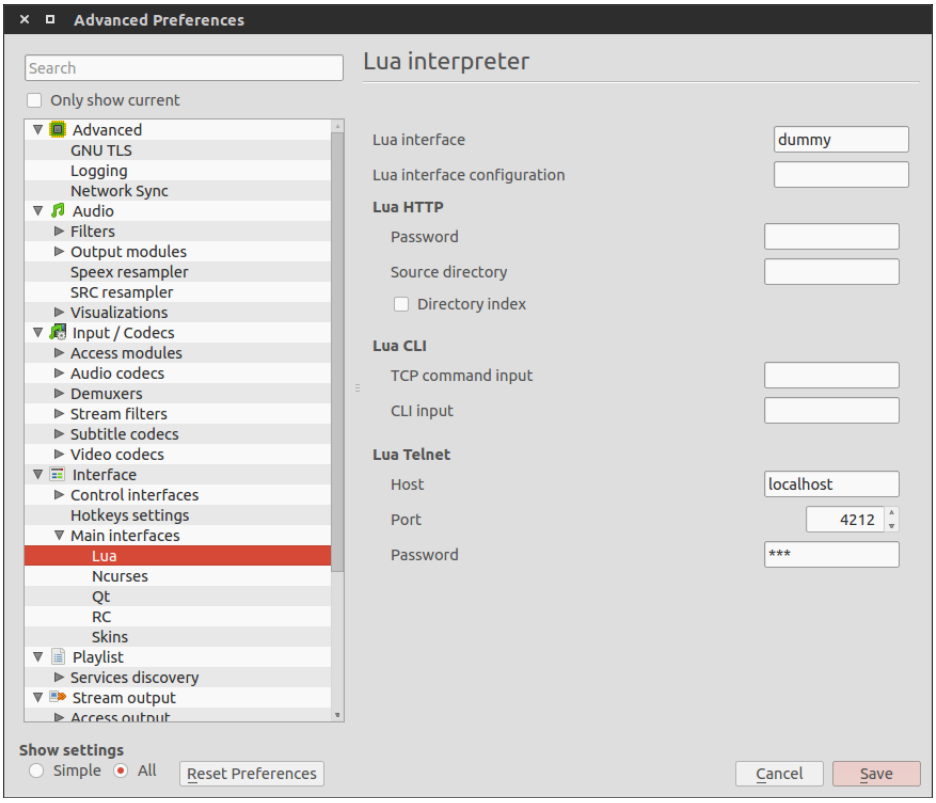
0.1.1 Software and Hardware Used

- Ubuntu 15.04
- VLC 2.2.0 Weatherwax
- Code::Blocks 13.12
- Boost Asio Library
- Wireshark 1.12.1
- AVTECH AVM565A IP Camera

0.2 Preparation and Set-up

0.2.1 Server

The server is required to have VLC installed (easily found in the Ubuntu Software Centre). Once installed, a password and host for telnet connections must be set. This can be found by navigating to Tools>Preferences>Show All Settings>Interface>Main Interfaces>Lua.



Lua interpreter

Lua interface: dummy

Lua interface configuration:

Lua HTTP

Password:

Source directory:

□ Directory index

Lua CLI

TCP command input:

CLI input:

Lua Telnet

Host: localhost

Port: 4212

Password: ***

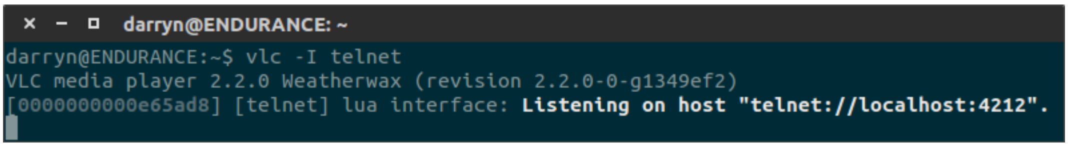
Show settings: ○ Simple ● All

Figure 1: Telnet Settings

As seen in the figure above, the host is set to 'localhost' (or 127.0.0.1), the default port is '4212' and the password is set to 'vlc' for simplicity. In order to launch VLC in its daemon telnet interface, the following command is run in terminal.

```
vlc -I telnet
```

If the response shown in the figure below is returned, then the server is ready to be controlled.



```
darryn@ENDURANCE:~$ vlc -I telnet
VLC media player 2.2.0 Weatherwax (revision 2.2.0-0-g1349ef2)
[0000000000e65ad8] [telnet] lua interface: Listening on host "telnet://localhost:4212".
```

Figure 2: VLC Telnet Server

We now require the IP address of the network camera connected to the server. See Appendix A for one way of finding this.

0.2.2 Software Development

Code::Blocks and the Asio Boost Library were used to develop the C++ application responsible for socket communication to the VLC daemon.

Prerequisites

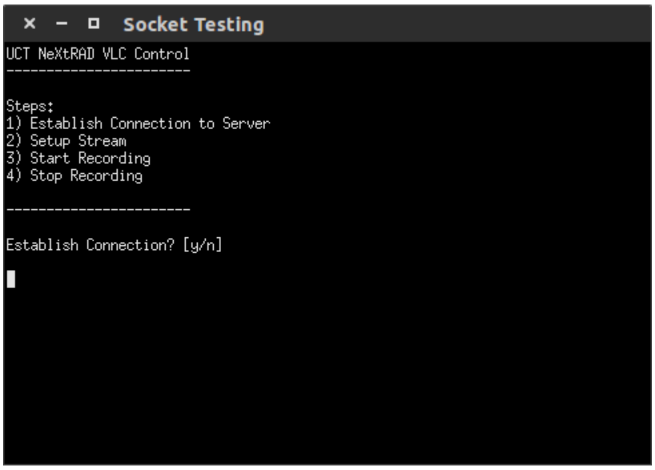
The following terminal commands will ensure all prerequisites for development (g++ compiler and Boost) are present.

```
sudo apt-get install build-essential
sudo apt-get install libboost-all-dev
```

The project can now be opened in Code::Blocks, but **must** be cleaned and rebuilt.

Expected Results

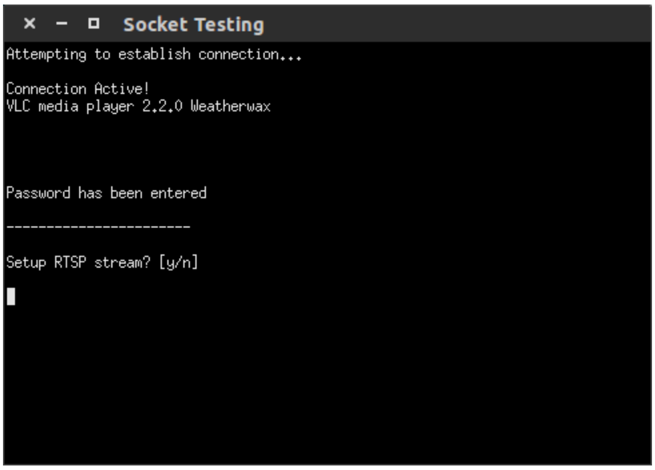
After compiling the program, the user is presented with the following console.



```
UCT NeXtRAD VLC Control
-----------------------

Steps:
1) Establish Connection to Server
2) Setup Stream
3) Start Recording
4) Stop Recording

-----------------------

Establish Connection? [y/n]

▮
```

Figure 3: VLC Socket Controller Welcome

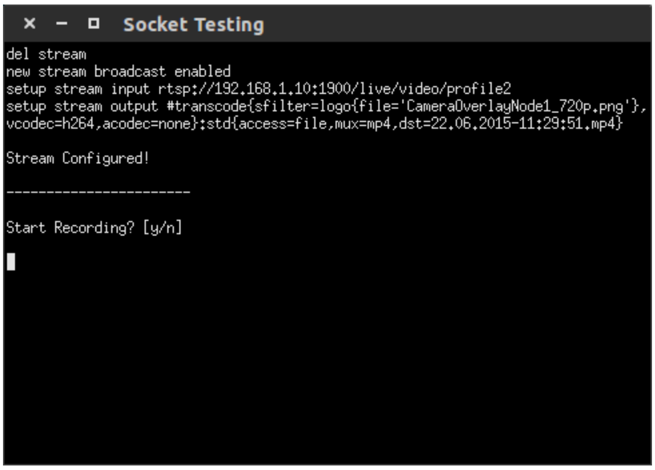
If the server is configured correctly, then the following response is returned. The program is now connected and a stream can be established.



```
Attempting to establish connection...

Connection Active!
VLC media player 2.2.0 Weatherwax



Password has been entered

-----------------------

Setup RTSP stream? [y/n]

▮
```

Figure 4: Connection Established

The following figure shows that the program displays all commands to be sent to the VLC daemon. Parameters such as the IP address, port number and path to overlay image and must obviously be modified.



```
del stream
new stream broadcast enabled
setup stream input rtsp://192.168.1.10:1900/live/video/profile2
setup stream output #transcode{sfilter=logo{file='CameraOverlayNode1_720p.png'},
vcodec=h264,acodec=none}:std{access=file,mux=mp4,dst=22.06.2015-11:29:51.mp4}

Stream Configured!

-----------------------

Start Recording? [y/n]
```

Figure 5: Stream Settings

Once the stream is terminated, the program produces a .mpg file in the home directory with the computers date and time as its filename. Below is a screenshot of a sample output.

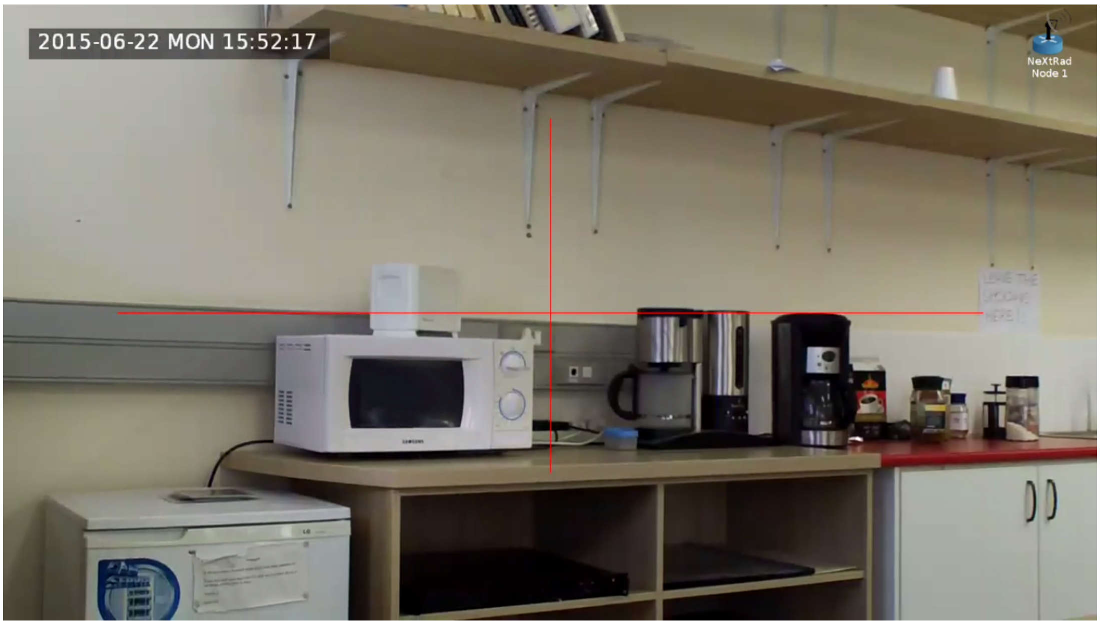


Figure 6: Example Output

Note: in this test, Profile 2 was used (1280x720). The correspondingly sized image overlay (CameraOverlayNode1_720p) had to be used.

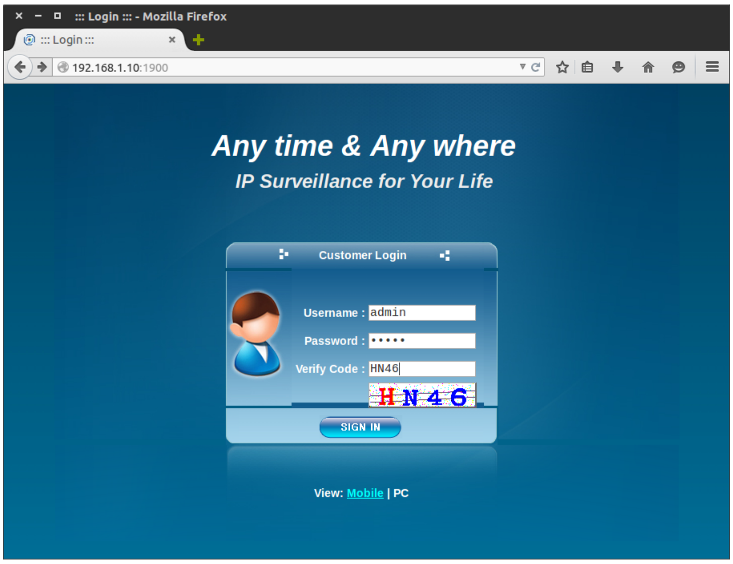
Possible Source of Confusion

Please note that if the camera has been reset, and the default settings are active, then the camera does not allow for anonomous login. This means that in order to access a RTSP stream, the admin username and password must be entered. In this case, **no video will be recorded**.

The program cannot account for this. Please refer to the following section, for a guide on activating anonomous login.

0.3 Camera Streaming Settings

Once the network camera's IP address is known, various settings can be tweaked. Opening `http://camera_ip_address:port_number` allows an admin to alter these settings. By default the username and password are *admin*.



Username : admin

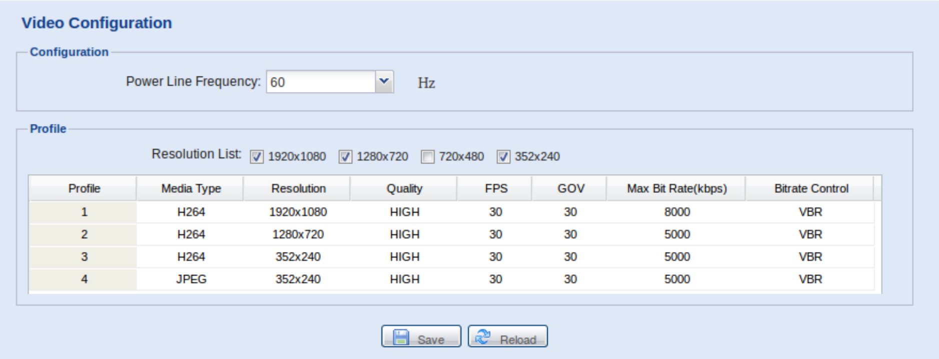
Password : •••••

Verify Code : HN46

Figure 7: Camera Login

Video Profiles

Navigating to Config>Camera>Video allows the admin to configure the streaming profiles. During development it was decided that Profile 2 would be used for 1280x720 recording and that Profile 3 would possibly be a lower resolution for live streaming to the client.



Power Line Frequency: 60 Hz

Resolution List: ☑ 1920x1080 ☑ 1280x720 □ 720x480 ☑ 352x240

Profile	Media Type	Resolution	Quality	FPS	GOV	Max Bit Rate(kbps)	Bitrate Control
1	H264	1920x1080	HIGH	30	30	8000	VBR
2	H264	1280x720	HIGH	30	30	5000	VBR
3	H264	352x240	HIGH	30	30	5000	VBR
4	JPEG	352x240	HIGH	30	30	5000	VBR

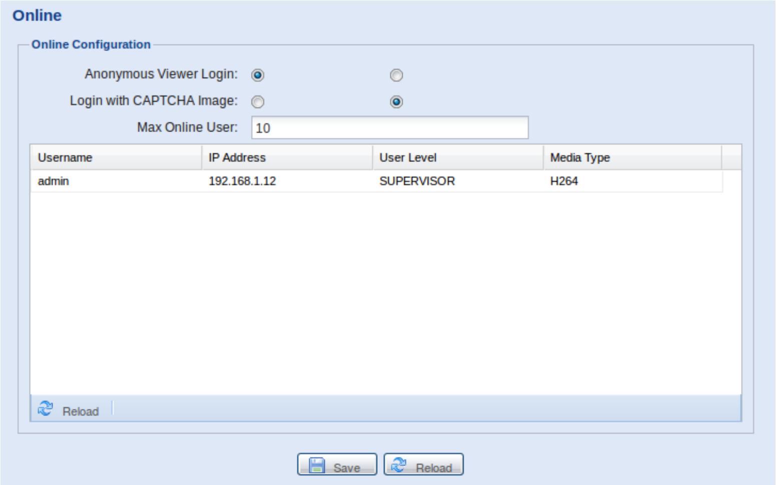
Figure 8: Profile Configuration

These profiles can be individually tested using VLC. Each stream is accessable at
`rtsp://camera_ip_address:port_number/live/video/profile(1/2/3/4)`

e.g. `rtsp://192.168.1.10:1900/live/video/profile2`

Anonomous Login

It is **vital** that anonomous login be enabled for video recording to be possible. Navigate to Config>General>Online and ENABLE anonomous login. This is automatically set to disabled upon camera reset.



Online

Online Configuration

Anonymous Viewer Login: ● ○

Login with CAPTCHA Image: ○ ●

Max Online User: 10

Username	IP Address	User Level	Media Type
admin	192.168.1.12	SUPERVISOR	H264

Figure 9: Online Configuration

0.4 Addtitional Information

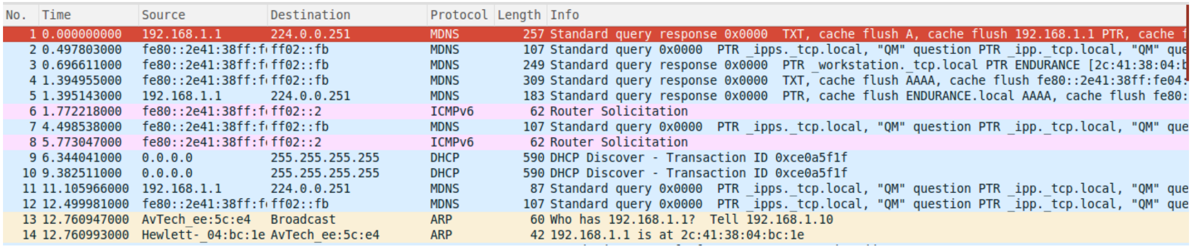
- Basics of the VLC telnet interface:
  `http://www.videolan.org/doc/streaming-howto/en/ch05.html`
- VLC Streaming options:
  `http://www.videolan.org/doc/streaming-howto/en/ch03.html`

Appendix A: Finding the IP Address of an AVTECH Network Camera

- Supply the camera with power (12V, 1A).
- Connect to camera using Ethernet cable.
- Set computer IP address to 192.168.1.1
- Install Wireshark, with some additional tweaking:

```
sudo chgrp myusername /usr/bin/dumpcap
sudo chmod 750 /usr/bin/dumpcap
sudo setcap cap_net_raw,cap_net_admin+eip /usr/bin/dumpcap
```

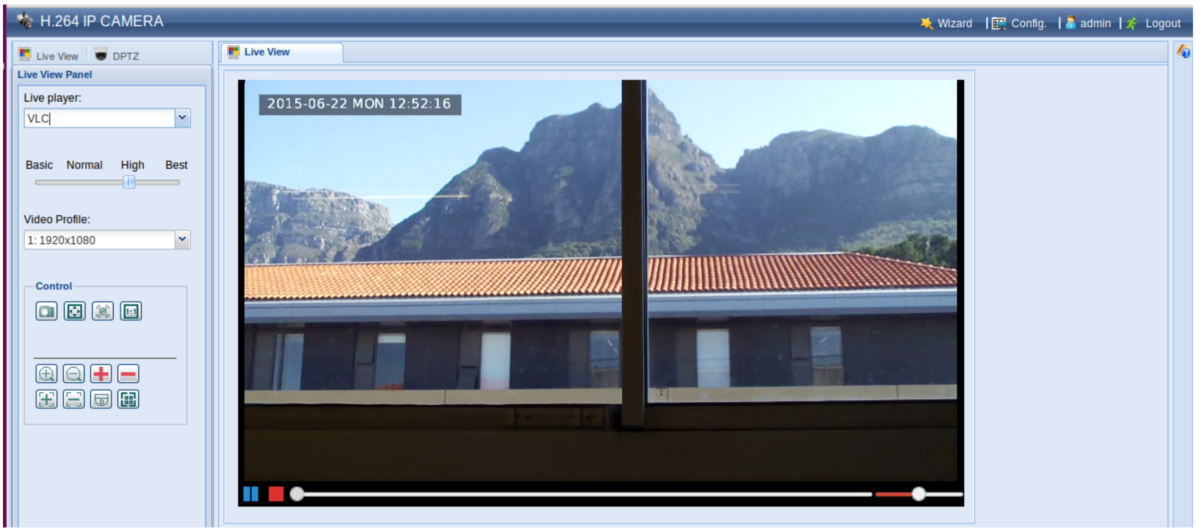
- Monitor the connections with IP Cam over the Ethernet connection. The IP can be found in an AVTECH broadcast. In this example; 192.168.1.10.



```
No. Time         Source             Destination     Protocol Length Info
  1 0.000000000  192.168.1.1        224.0.0.251     MDNS        257 Standard query response 0x0000  TXT, cache flush A, cache flush 192.168.1.1 PTR, cache f
  2 0.497803000  fe80::2e41:38ff:f  ff02::fb        MDNS        107 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" que
  3 0.696611000  fe80::2e41:38ff:f  ff02::fb        MDNS        249 Standard query response 0x0000  PTR _workstation._tcp.local PTR ENDURANCE [2c:41:38:04:b
  4 1.394955000  fe80::2e41:38ff:f  ff02::fb        MDNS        309 Standard query response 0x0000  TXT, cache flush AAAA, cache flush fe80::2e41:38ff:fe04:
  5 1.395143000  192.168.1.1        224.0.0.251     MDNS        183 Standard query response 0x0000  PTR, cache flush ENDURANCE.local AAAA, cache flush fe80:
  6 1.772218000  fe80::2e41:38ff:f  ff02::2         ICMPv6       62 Router Solicitation
  7 4.498538000  fe80::2e41:38ff:f  ff02::fb        MDNS        107 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" que
  8 5.773047000  fe80::2e41:38ff:f  ff02::2         ICMPv6       62 Router Solicitation
  9 6.344041000  0.0.0.0            255.255.255.255 DHCP        590 DHCP Discover - Transaction ID 0xce0a5f1f
 10 9.382511000  0.0.0.0            255.255.255.255 DHCP        590 DHCP Discover - Transaction ID 0xce0a5f1f
 11 11.105966000 192.168.1.1        224.0.0.251     MDNS         87 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" que
 12 12.499981000 fe80::2e41:38ff:f  ff02::fb        MDNS        107 Standard query 0x0000  PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" que
 13 12.760947000 AvTech_ee:5c:e4    Broadcast       ARP          60 Who has 192.168.1.1?  Tell 192.168.1.10
 14 12.760993000 Hewlett-_04:bc:1e  AvTech_ee:5c:e4 ARP          42 192.168.1.1 is at 2c:41:38:04:bc:1e
```

- Open web browser: http://192.168.1.10:1900. Default username (admin) and password (admin).
- (Optional) for live viewing in the browser, install:

```
sudo apt-get install vlc browser-plugin-vlc
```



Live player: VLC

Video Profile: 1: 1920x1080

Appendix B: Code Listing

```
//includes
#include <boost/asio.hpp>
#include <iostream>
#include <string>
#include <stdio.h>
#include <ctime>

//namespaces
using namespace boost::asio;
using namespace std;

//function declarations
void enterPassword(string text);
void setupStream();
void start();
void stop();
void connect();

//global variables
char buff[300];
char option;
io_service service;
ip::tcp::socket sock(service);

//functions
void clearBuffer()
{
    for (int i = 0; i < 301; i++)
    {
        buff[i] = ’ ’; //clear all elements of the array
    }
}

void connect()
{
    system("clear\n"); //clear console
    cout << "Attempting to establish connection..." << endl << endl;
    ip::tcp::endpoint ep(ip::address::from_string("127.0.0.1"), 4212); //
        define the endpoint at the known server address & port

    try
        {
            sock.connect(ep); //attempt to connect to the endpoint, if no
                exception is thrown the connection is successful
            cout << "Connection Active!" << endl;
            enterPassword("vlc\n"); //enter the vlc server password
            cout << "Password has been entered" << endl << endl;
            cout << "------------------------------" << endl << endl;

            cout << "Setup RTSP stream? [y/n]" << endl << endl;
            while(true) //wait for response
            {
                cin >> option;
                if (option == ’y’)
                    {setupStream();}
                else if (option == ’n’)
                    {exit(0);} //close program
                else
                    {cout << "Invalid Response. Please use ’y’ or ’n’" <<
                        endl;}
            }
        }
    catch (boost::system::system_error const& e) //exception was thrown,
        connection failed
        {
            cout << "Warning: could not " << e.what() << endl << endl;

            cout << "Retry Connection? [y/n]" << endl << endl;
            while(true)
            {
                cin >> option;
                if (option == ’y’)
                    {connect();} //restart connection
                else if (option == ’n’)
                    {exit(0);}
                else
                    {cout << "Invalid Response. Please use ’y’ or ’n’" <<
                        endl;}
            }
        }
}

void write(string text)
{
    sock.write_some(buffer(text)); //write to the terminal
    cout << text;                  //echo to console
}

void read()
{
    clearBuffer();
    sock.read_some(buffer(buff)); //read terminal response to the buffer
        array
    cout << buff << endl;         //echo to console
}

void setupStream()
{
    system("clear\n");            //clear console

    /*Could enter address manually if hard coding is unwanted:
        cout << "Enter stream address (e.g rtsp://localhost:8554/stream)" <<
             endl;
        cin >> streamAddress; */

    string streamAddress = "rtsp://192.168.1.10:1900/live/video/profile2";
    write("del stream\n");        //delete any previous instances of ’stream
        ’ if they exist
    write("new stream broadcast enabled\n"); //create a new instance of ’
        stream’ and enable it
    write("setup stream input " + streamAddress + "\n"); //set stream input

    time_t rawtime;                 //
    struct tm * timeinfo;           //
    char buffer[80];                //
                                    //get date & time as a a string
    time (&rawtime);                //
    timeinfo = localtime(&rawtime); //

    strftime(buffer,80,"%d.%m.%Y-%I:%M:%S",timeinfo); //set date & time
        format

    string dateTime(buffer);        //define date & time string
    write("setup stream output #transcode{sfilter=logo{file=’
        CameraOverlayNode1_720p.png’},vcodec=h264,acodec=none}:std{access=
        file,mux=mp4,dst=" + dateTime + ".mp4}\n"); //setup the output type,
        encoding format and filename

    cout << endl << "Stream Configured!" << endl << endl;
    cout << "-----------------------------" << endl << endl;

    cout << "Start Recording? [y/n]" << endl << endl;
    while(true)
    {
        cin >> option;
        if (option == ’y’)
            {start();}
        else if (option == ’n’)
            {exit(0);}
        else
            {cout << "Invalid Response. Please use ’y’ or ’n’" << endl;}
    }

}

void start()
{
    system("clear\n");            //clear console
    cout << "Recording Started" << endl << endl;
    write("control stream play\n"); //start recoring
    cout << endl << "-----------------------------" << endl << endl;

    cout << "Stop Recording? [y/n]" << endl << endl;
    while(true)
    {
        cin >> option;
        if (option == ’y’)
            {stop();}
        else if (option == ’n’)
            {exit(0);}
        else
            {cout << "Invalid Response. Please use ’y’ or ’n’" << endl;}
    }

}

void stop()
{
    system("clear\n");              //clear console
    cout << "Recording Stoped!" << endl << endl;
    write("control stream stop\n"); //stop recording
    cout << endl << "-----------------------------" << endl << endl;

    cout << "Setup New Stream? [y/n]" << endl << endl;
    while(true)
    {
        cin >> option;
        if (option == ’y’)
            {setupStream();}
        else if (option == ’n’)
            {exit(0);}
        else
            {cout << "Invalid Response. Please use ’y’ or ’n’" << endl;}
    }
}
```

```
void enterPassword(string text)
{
    sock.write_some(buffer(text)); //this will not be shown on the console
    read();
}

void welcome()
{
    cout << "NeXtRAD VLC Telnet Controller" << endl;
    cout << "-----------------------------" << endl << endl;
    cout << "Steps:" << endl;
    cout << "1) Establish Connection to Server" << endl;
    cout << "2) Setup Stream" << endl;
    cout << "3) Start Recording" << endl;
    cout << "4) Stop Recording" << endl << endl;
    cout << "-----------------------------" << endl << endl;
}

int main()
{
    welcome(); //display welcome screen

    cout << "Establish Connection? [y/n]" << endl << endl;
    while(true) //wait for response
    {
        cin >> option;
        if (option == 'y')
            {connect();} //chose yes - attempt connection
        else if (option == 'n')
            {break;} //chose no - close program
        else
            {cout << "Invalid Response. Please use 'y' or 'n'" << endl;} //
                request new response
    }
    return 0;
}
```