SPCTOR UAV SDRadar Command Protocol

This is a work in progress file describing the SPCTOR UAV-SDRadar Command Protocol. The commands listed below are intended to be used as examples and a foundation for designing the final protocol.

Table of Commands

Command	Description	Arg	Example
help	help message	n/a	--help
status	get status	n/a	--status
date	get date/time	n/a	--date
flightplan	Upload a flight plan	string	--flightplan [lat1,lon1,el1;lat2,lon2,el2;…]
executeflight	Execute flight plan at specified date/time	string	--executeflight YYYYMMDDThhmmss
file	Base name of data output file (must have .dat extension)	string	--file flight1/out.dat
freqs	Vector of RF Center frequencies. Format; f1,f2,… or f1:df:fend	string	--freqs 1e9:5e7:2e9
sweeps	number of frequency sweeps (Default: 1)	int	--sweeps 1000
sri	Frequency Sweep Repition Interval in sec	double	--sri .7
getdata	Get full dataset for a give flight id	string	--getdata fABC001

Example C++ Server Code Snippet

This is example C++ code that will interface with the uavsdradar and respond to requests over TCP/IP

```
void tcp_connection::handle_read_request(const boost::system::error_code& error)
{
    std::string m_message;
    if (error == boost::asio::error::eof){
         std::cout<<"\nEnd Response"<<std::endl;
         //break; // Connection closed cleanly by peer.
     }
     else if (error){
        std::cout << "Error: " << error << "\n";
        throw boost::system::system_error(error); // Some other error.
     }
    else{
      // Check that response is OK.
      std::istream response_stream(&request_);
      std::string udar_version;
      response_stream >> udar_version;

      unsigned int req_code;
      response_stream >> req_code;
      std::string req_str;
      std::getline(response_stream, req_str);
      std::cout<<"Request:"<<udar_version<<" "<<req_code<<" "<<req_str<<std::endl;
      if (!response_stream || udar_version.substr(0, 5) != "UDAR/")
```

```
        {
          std::cout << "Invalid response\n";
          return;
        }
        if (req_code == 001)
        {

            std::string flightplanstr, flighttimestr, freqplanstr, rxfname, flightid;
            int numsweeps;
            double sri;

            std::vector<std::string> tokenvec = tokenize(req_str);
            //setup the program options
            po::options_description desc("Allowed options");
            desc.add_options()
              ("help |"help message")
              ("status |"get status")
              ("date |"get date/time")
              ("flightplan |po::value<std::string>(&flightplanstr), "Upload a flight plan")
              ("executeflight |po::value<std::string>(&flighttimestr), "Execute flight plan at specified date/t
              ("file |po::value<std::string>(&rxfname), "Base name of data output file (must have .dat extensio
              ("freqs |po::value<std::string>(&freqplanstr), "Vector of RF Center frequencies. Format; f1,f2,..
              ("sweeps |po::value<int>(&numsweeps)->default_value(1), "number of frequency sweeps (Default: 1)"
              ("sri |po::value<double>(&sri),"Frequency Sweep Repition Interval in sec")
              ("getdata |po::value<std::string>(&flightid),"Get full dataset for a give flight id")
            ;
            po::variables_map vm;
            try{
                po::store(po::command_line_parser(tokenvec).options(desc).run(), vm);

                if (vm.count("help")){
                    std::stringstream ss;
                    ss << boost::format("UAV SDRadar supported commands: %s.") % desc;
                    m_message = ss.str();
                    boost::asio::async_write(socket_, boost::asio::buffer(m_message),
                      boost::bind(&tcp_connection::handle_write, shared_from_this(),
                        boost::asio::placeholders::error,
                        boost::asio::placeholders::bytes_transferred));
                    return;
                }
                po::notify(vm);
            }
            catch(po::error& e) {
                std::cerr << "ERROR: " << e.what() << std::endl << std::endl;
                std::cerr << desc << std::endl;
                std::stringstream ss;
                ss <<"ERROR: " << e.what() <<"\n";
                ss << boost::format("UAV SDRadar supported commands: %s.") % desc;
                m_message = ss.str();
                boost::asio::async_write(socket_, boost::asio::buffer(m_message),
                  boost::bind(&tcp_connection::handle_write, shared_from_this(),
                    boost::asio::placeholders::error,
                    boost::asio::placeholders::bytes_transferred));
                return;
            }
            if(vm.count("status")>0){
                m_message  = get_status_string();
                boost::asio::async_write(socket_, boost::asio::buffer(m_message),
                  boost::bind(&tcp_connection::handle_write, shared_from_this(),
                    boost::asio::placeholders::error,
                    boost::asio::placeholders::bytes_transferred));
                return;
            }
            if(vm.count("date")>0){
                m_message  = make_daytime_string();
                boost::asio::async_write(socket_, boost::asio::buffer(m_message),
                  boost::bind(&tcp_connection::handle_write, shared_from_this(),
                    boost::asio::placeholders::error,
                    boost::asio::placeholders::bytes_transferred));
                return;
            }

            if (!_usrp->isActive())
                _uavsdradar->refresh();
```

```
std::vector<double> freqvec;
std::string freqvecstr;
if(vm.count("freqs") > 0){
    boost::filesystem::path freqplanp = _config_path / boost::filesystem::path(freqplanstr);
    if (boost::filesystem::exists(freqplanp)){
         freqvecstr = get_text_file_string(freqplanp.string());
     }
     else{
         freqvecstr = freqplanstr;
     }
    if(parse_freqplan(freqvec,freqvecstr)){
        m_message = "[--freqs] Error: Unable to parse frequency plan: " + freqplanstr;
        boost::asio::async_write(socket_, boost::asio::buffer(m_message),
          boost::bind(&tcp_connection::handle_write, shared_from_this(),
            boost::asio::placeholders::error,
            boost::asio::placeholders::bytes_transferred));
        return;
    }
    _uavsdradar->set_freqvec(freqvec);
}
if(vm.count("sri") > 0)
    _uavsdradar->set_sri(sri);

if(vm.count("flightplan") > 0)
    _uavsdradar->upload_flight_plan(flightplanstr);

std::string curr_flightid;
if(vm.count("executeflight") > 0){
    curr_flightid = _uavsdradar->execute_flight_at_time(flighttimestr,numsweeps,rxfname);
    m_message.append("--flightid " + curr_flightid + "\n");
    boost::asio::async_write(socket_, boost::asio::buffer(m_message),
      boost::bind(&tcp_connection::handle_write, shared_from_this(),
        boost::asio::placeholders::error,
        boost::asio::placeholders::bytes_transferred));
    return;
}

if(vm.count("getdata") > 0){
  if (flightid.empty()) // return data from current flight
      flightid = _uavsdradar->get_last_flight_id();

  std::vector<std::string> outfiles;
  _uavsdradar->get_fnames_from_flightid(flightid,outfiles);

  for (size_t i=0; i<outfiles.size();i++){
      boost::filesystem::path p(outfiles[i].c_str());
      if(p.extension().string()==".txt"){
          m_message.append("--outfile " + p.string()+"\n");
          m_message.append(_uavsdradar->get_text_file_string(outfiles[i]));
      }
      else {
        std::string binary_str;
        int nbytes = _uavsdradar->get_binary_file_bytestr(binary_str);
        m_message.append("--binaryoutfile " + p.string()+ " --nB " + std::to_string(nbytes) + "\n")
        m_message.append(binary_str);
        m_message.append("\n--eof\n");

        // Implementation of binary file data stream code:

        // std::stringstream data_ss;
        // std::ifstream ifile;
        // ifile.open(outfiles[i].c_str(), std::ios::binary);
        // int nbytes = 0;
        // if (ifile.is_open()){
        //   ifile.seekg(0,ifile.end);
        //   nbytes = ifile.tellg();
        //   ifile.seekg(0,ifile.beg);
        //   data_ss << ifile.rdbuf();
        // }
        // m_message.append("--binaryoutfile " + p.string()+ " --nB " + std::to_string(nbytes) + "\
        // m_message.append(data_ss.str());
        // m_message.append("\n--eof\n");
```

```
                }
                m_message.append("\n");
            }
        }
      }
      else {
          m_message = make_req_code_help_string();
      }

    boost::asio::async_write(socket_, boost::asio::buffer(m_message /*+"\r\n--eot\r"*/),
        boost::bind(&tcp_connection::handle_write, shared_from_this(),
          boost::asio::placeholders::error,
          boost::asio::placeholders::bytes_transferred));
```

Example Python Client Code

Example python script that can send formatted commands to uavsdradar server

```
import socket
import ssl
import getpass
import getopt,sys

# client
def main(argv):
    HOST = '11.10.10.1'
    PORT = 9912
    logfilename = "client_test.log"

    try:
        opts,args = getopt.getopt(argv,"hi:f:",["help", "ip=", "file="])
    except getopt.GetoptError:
        print("Usage : client_test.py -i <server ip> -f <logfile> or client_test.py --ip <server ip> --file <
        sys.exit(2)
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            print("Usage : client_test.py -i <server ip> -f <logfile> or client_test.py --ip <server ip> --fi
            sys.exit()
        elif opt in ("-i", "--ip"):
            HOST = str(arg)
        elif opt in ("-f", "--file"):
            logfilename = str(arg)

    print("Using Server IP:")
    print(HOST)
    print("Using output logfile:")
    print(logfilename)

    logfile = open(logfilename,"w")

    server_cert = '~/.x300_Client/config/server.crt'
    client_cert = '~/.x300_Client/config/client.crt'
    client_key = '~/.x300_Client/config/client.key'

    # pword = input("Enter Password:")
    pword = getpass.getpass()

    # sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # sock.setblocking(1);
    # sock.connect((HOST, PORT))

    context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
    context.verify_mode = ssl.CERT_REQUIRED

    context.load_verify_locations(server_cert)
    context.load_cert_chain(certfile=client_cert, keyfile=client_key, password=pword)
    context.load_dh_params('~/.x300_Client/config/dh1024.pem')

    print("Enter Command (try --help)")
```

```
    while True:

        request_str = input(">>")
        if (request_str=="quit" or request_str=="q"):
            break;

        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setblocking(1);
        sock.connect((HOST, PORT))

        if ssl.HAS_SNI:
            secure_sock = context.wrap_socket(sock, server_side=False, server_hostname=HOST)
        else:
            secure_sock = context.wrap_socket(sock, server_side=False)

        # secure_sock.connect((HOST, PORT))

        cert = secure_sock.getpeercert()
        # print("getpeercert():")
        # print(cert)

        # verify server
        # if not cert or ('commonName', 'test') not in cert['subject'][3]: raise Exception("ERROR")
        # secure_sock.send(b"UDAR/1.0 200 --status")

        request = "UDAR/1.0 200 " + request_str + "\n\r\n"
        request_b = request.encode('ASCII')

        print("request:")
        print(request_b)

        secure_sock.send(request_b)
        # secure_sock.send(b"UDAR/1.0 200 --status\n\r\n")


        while True:
            try:
                rx_data = secure_sock.recv(4096)
                if len(rx_data) == 0:
                    break
                print(str(rx_data,'ASCII'))
                logfile.write(str(rx_data,'ASCII'))
            except ssl.SSLError as e:
                print(e)
                break

        secure_sock.unwrap()
        # secure_sock.shutdown(socket.SHUT_RDWR)
        secure_sock.close()
        sock.close()

    logfile.close()
    print("Done!")

if __name__ == '__main__':
    main(sys.argv[1:])
```