Subject:	**Using Python to Stream Commands to Applied Motion Products Drives**
Applies to:	Drives with Ethernet, RS-232 or RS-485 communication ports
Date:	February 6, 2019
Authors:	Chris Dickens, Lane Andrews

Overview

Why Make Custom Software Scripts to Command Our Drives?

Applied Motion Products drives and integrated motors with streaming command functionality have a robust set of commands that operators can utilize for anything from basic single-axis motion to multi-axis control. Applied Motion Products offers free software utilities to stream commands to these drives, such as *SCL Utility* and *Q Programmer*, but there are advantages to building a custom software application for streaming commands to our drives:

- You can communicate to multiple drives without manually reconnecting from one drive to the next, or design in your own interface for switching. These kinds of customizations allow you to send commands to multiple drives and coordinate motion without relying on any external inputs or timing.
- As our streaming command language allows for immediate commands, you can build a software application that actively monitors and records data - functionality not available in *Q Programmer*.
- Utilizing streaming commands rather than dedicating I/O pins to pulse and direction signals offloads much of the heavy lifting of controlling the motion profiles from the application software to the drive.

Why Use Python to Create a Custom Script?

We chose Python for these examples as it is easy to learn and use, even for users with little or no programming experience. Python has extensive support documentation, sample code, and an active online community for troubleshooting. Its integration into the Raspberry Pi platform also opens a world of possibilities, from hobby projects to large-scale production operations.

Required Equipment

Hardware

To begin, you will need an Applied Motion drive or integrated motor with streaming command capabilities. You will also need an appropriate motor (if non-integrated), power supply, and communication cables. Follow the appropriate Quick Setup Guide for your drive or integrated motor to configure and set up the axis.

In this application note we will be using a Windows PC for configuration and development.

Note: As with any test program, please uncouple your motor from the load so it can spin freely on the work surface before attempting to run any of these programs. Attach the load only after ensuring the program is performing the desired motion profile.

Software

Configure the drive or integrated motor for SCL Mode using the software configuration tool indicated in the relevant Quick Setup Guide. Exit all software and cycle power to the drive or integrated motor before attempting to run your custom software script.

If not already available on your system, download Python and an appropriate IDE or text editor. Python has a helpful getting starting guide that will walk you through the basics: https://www.python.org/about/gettingstarted/

If working with RS-232 or RS-485 communications, you will need pySerial. Documentation including download instructions can be found here: https://pythonhosted.org/pyserial/

Program Requirements

Getting Started

We recommend reading relevant parts of our Host Command Reference to become familiar with our streaming commands and communication protocol. For RS-232/485 connections, reference Appendices B and C. For Ethernet connections read Appendix G.

Program Flow

The program steps required for drive communication can be summarized as follows:

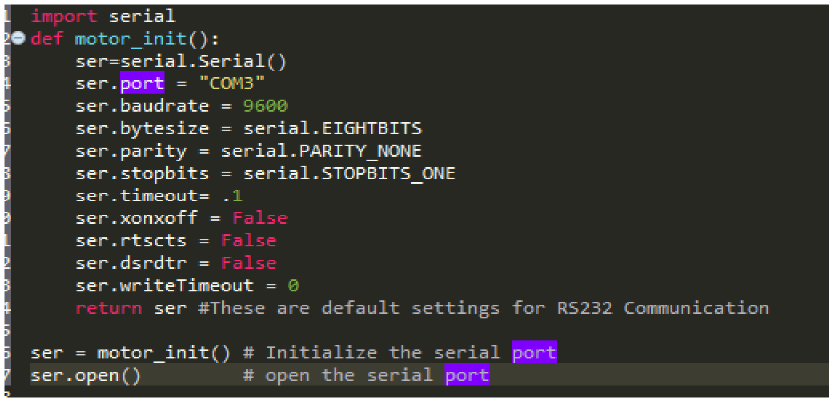
1. Establish communications with the drive.
2. Send a command to the drive.
3. Wait for and handle the response from the drive.

The following two sections show sample code to take the above steps with both RS-232/485 and Ethernet connections. The basic steps outlined below represent only a fraction of your final program. You may choose to embed these steps in functions, utilize interrupts, or simply hard-code a fixed number of commands.

Python is a language best learned by doing. After reviewing the following sample code, check out the Sample Scripts section and start coding!

Program Flow – RS-232/485 Connections

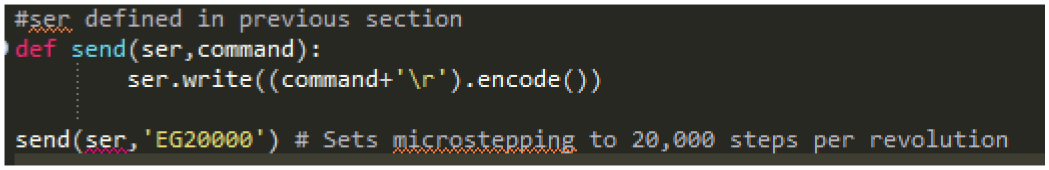
1. Establish communications with the drive:



```
import serial
def motor_init():
    ser=serial.Serial()
    ser.port = "COM3"
    ser.baudrate = 9600
    ser.bytesize = serial.EIGHTBITS
    ser.parity = serial.PARITY_NONE
    ser.stopbits = serial.STOPBITS_ONE
    ser.timeout= .1
    ser.xonxoff = False
    ser.rtscts = False
    ser.dsrdtr = False
    ser.writeTimeout = 0
    return ser #These are default settings for RS232 Communication

ser = motor_init() # Initialize the serial port
ser.open()         # open the serial port
```

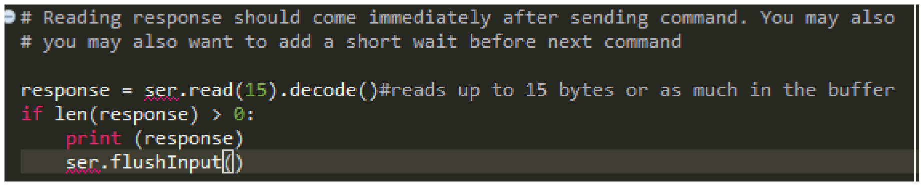
2. Send a command to the drive:



```
#ser defined in previous section
def send(ser,command):
        ser.write((command+'\r').encode())

send(ser,'EG20000') # Sets microstepping to 20,000 steps per revolution
```

3. Wait for and handle the response from the drive:

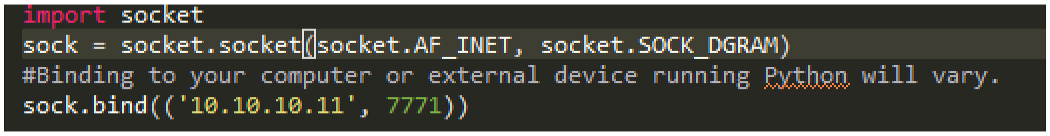


```
# Reading response should come immediately after sending command. You may also
# you may also want to add a short wait before next command

response = ser.read(15).decode()#reads up to 15 bytes or as much in the buffer
if len(response) > 0:
    print (response)
    ser.flushInput()
```

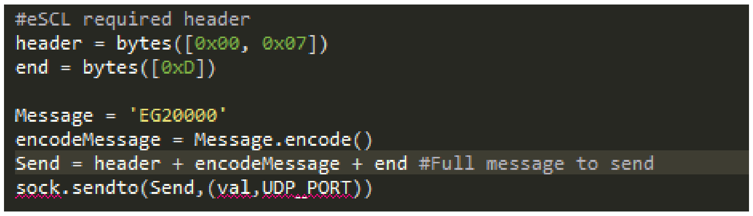
Program Flow – Ethernet Connections

1. Establish communications with the drive:



```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
#Binding to your computer or external device running Python will vary.
sock.bind(('10.10.10.11', 7771))
```

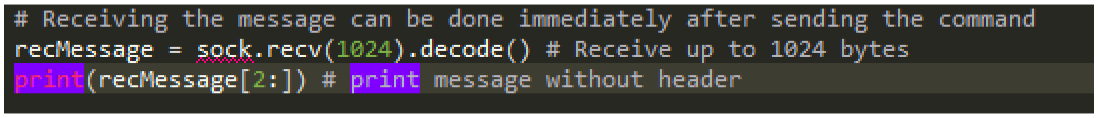
2. Send a command to the drive:



```
#eSCL required header
header = bytes([0x00, 0x07])
end = bytes([0xD])

Message = 'EG20000'
encodeMessage = Message.encode()
Send = header + encodeMessage + end #Full message to send
sock.sendto(Send,(val,UDP_PORT))
```

3. Wait for and handle the response from the drive:



```
# Receiving the message can be done immediately after sending the command
recMessage = sock.recv(1024).decode() # Receive up to 1024 bytes
print(recMessage[2:]) # print message without header
```

Sample Scripts

The following sample scripts have been included in the same zip file as this PDF:

- **RS232Demonstration**: This script shows the basics of initializing a serial port for communication and sending commands. The motor should travel CW then CCW and read the immediate position twice during the movement.
- **eSCLDemonstration**: Similar to the RS232Demonstration script, this script demonstrates the ability to open an Ethernet socket and send commands to a drive. The commands included in the script will rotate the motor CW and then CCW.
- **MultiDriveTerminal**: This script allows a user to communicate with drives at multiple IP addresses simultaneously. Commands can be addressed to individual drives, or if no address is given the commands will be sent to all drives. The commands are the same that you would send in our SCL terminal (e.g. in *SCL Utility* or *Q Programmer*) but with the ability to address multiple drives on the Ethernet network at once, allowing for greater flexibility.

Conclusion

Python provides flexibility to both dynamically stream commands and run scripts for both single drives and for multi-axis control. The minimum requirements for establishing communication and control are quite simple, supporting a variety of possible projects.

For Python application support or general technical questions contact Applied Motion Products at support@applied-motion.com or 1-800-525-1609.