README

Perry Kundert

November 29, 2014

Contents

1 Actuator Position Control via EtherNet/IP

The `cpppo_positioner` module allows control of the position of a set of actuators, by initiating a communication channel, and issuing new position directives via the actuator state machine. The current state is polled as necessary via EtherNet/IP CIP read commands, and data updates and state changes are performed via EtherNet/IP CIP writes.

1.1 Installing

Clone the repository, and run the setup.py installer:

```
$ git clone git@github.com:pjkundert/cpppo_positioner.git
$ cd cpppo_positioner
$ python setup.py install
$ python
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import cpppo_positioner
```

```
>>>
```

1.2 Positioning

A Python API is provided to implement positioning control.

```
from import cpppo_positioner import smc_lec_gen1
gateway            = smc_lec_gen1()
gatway.position( actuator=0, timeout=10.0, position=12345, speed=100, ... )
```

1.2.1 Command- or Pipe-line usage

An executable module entry point (`python -m cpppo_positioner`), and a convenience executable script (`cpppo_positioner`) are supplied.

If your application generates a stream of actuator position data, or if you have some manual positions you wish to move to, you can use the command-line interface. You may supply one or more actuator positions in blobs of JSON data (an actual position would have more entries, such as `acceleration`, `deceleration`, `timeout`, ...):

```
$ position='{ "actuator": 0, "position": 12345, "speed": 100 }'
```

These positions may be supplied either as single parameters on the command line, or as separate lines of input (if standard input is selected, by supplying a '-' option):

```
$ python -m cpppo_positioner --address gateway -v "$position"
$ echo "$position" | cpppo_positioner --address gateway -v -
```

- Quoting double-quotes on Windows Powershell
  Note that on Windows Cmd or Powershell, it is very difficult to quote double-quote characters in strings. In Powershell, you need to use the bash-slash + back-tick before each double-quote. Unexpectedly, using a single-quoted string does **not** allow you to contain double-quotes.

  You can get double quotes into a string:

  ```
  PS > $position = '{ "actuator": 0, "position": 12345, "speed": 100 }'
  PS > $position
  '{ "actuator": 0, "position": 12345, "speed": 100 }'
  PS >
  ```

  However, when you try to use them, they are re-interpreted on inclusion in a command:

```
PS > python -m cpppo_positioner --address gateway -v "$position"
PS > python -m cpppo_positioner -v "$position"
... Invalid position data: { actuator: 0, position: 12345, speed: 100 };
    Expecting property name: line 1 column 3 (char 2)
```

So, the only way to do this is to use the strange back-slash + back-tick double-escape, directly as a command-line argument:

```
PS > python -m cpppo_positioner --address gateway -v '{ \`"actuator\`": 0, .
```

Recommendation: use Linux or Mac, or install Cygwin and use bash on Windows. Trust me; this is just the tip of the iceberg...

1.3 SMC Gateway Simulator

A basic simulator of some of the I/O behaviour of the SMC Gateway is implemented for testing purposes.

Ensure that either you have installed the $\text{cpppo}_{\text{positioner}}$, **or** are in the directory containing the cloned $\text{cpppo}_{\text{positioner}}$ repository): To simulate a number of SMC positioning actuators behind an SMC LEC-GEN1 gateway, run:

```
$ python -m cpppo_positioner.simulator -v \
    --actuators 3 IN=SINT[256]@0x1FF/1/1 OUT=SINT[256]@0x1FF/1/2
...
... NORMAL   main       Creating tag: IN=SINT [256]
... WARNING  __init__   Simulating 3 SMC Positioning Actuators
... NORMAL   __init__   SMC Gateway Tag IN : Class   511/0x01ff, Instance   1, Attr
... NORMAL   main       Creating tag: OUT=SINT [256]
... NORMAL   __init__   SMC Gateway Tag OUT: Class   511/0x01ff, Instance   1, Attr
... NORMAL   main       EtherNet/IP Simulator: ('', 44818)
...
```

Note: the second command-line continuation line above starting `--actuators` contains all the defaults, and can be skipped unless you wish to change some of them. These optional arguments are:

```
--actuators # -- how many actuators you wish to simulate; default: 3
IN=...@C/I/A  -- IDs for Class C (eg. 0x1FF, 511), Instance I and Attr A
```

This starts up the simulator, specifying that the simulated Gateway is at CIP Class ID 0x1FF (511), and the IN and OUT Attributes are at Instance ID 1, Attribute IDs 1 and 2, respectively. It will bind to port 44818 on all interfaces (including 'localhost').

Once the simulator is running, test that the communication works, by issuing a basic positioning command:

```
$ python -m cpppo_positioner -va localhost \
    --config '{ "in": "@0x1FF/1/1", "out": "@0x1FF/1/2" }' \
    '{ "actuator": 0, "position": 12345 }'
... NORMAL   main       Position: {u'actuator': 0, u'position': 12345}
... NORMAL   main       Gateway: localhost:44818 connected
... NORMAL   main       Completed 1/1 actuator positions in   0.021s
$
```

Note: The second command-line continuation line above starting `--config` shows how to pass configurations to the actuator class. It contains defaults, and can be skipped unless you want to change some of them. Presently, the $\text{smc.smc}_{\text{lecgen}}$ allows the following configurations:

```
in       -- The @C/I/D Class, Instance and Attribute of the IN array
out      -- The @C/I/D Class, Instance and Attribute of the OUT array
depth    -- Pipelining depth to use for EtherNet/IP CIP I/O (default: 1)
multiple -- Use CIP Multiple Packet Service for requests (default: False)
timeout  -- Timeout in seconds for all EtherNet/IP CIP I/O (default: 1.0)
```