TOPSCCC DEMO

Requirements:

RS-485 connection (find out what port number, also!)
Python - tested with 2.7
Python is available from https://www.python.org/downloads/

Usage

To use, open a terminal window and run

```
topsccc_demo.py PORT
```

where **PORT** is the name of the RS485 port you're using.

For Windows, this is usually COMx (i.e. COM0, COM4, etc). This loosely correlates with the number of the port in your motherboard documentation.
For Linux, this is usually /dev/ttySx (i.e. /dev/ttyS0, /dev/ttyS4, etc). Same loose correlation.

The demo will search for a module and select the first one it finds to run the demo on. The commands it is sending will be displayed in the terminal window.
An explanation of its behavior can be found in the code comments. Documentation for the example commands used here, as well as the full list of commands available, can be found in the module-specific manual for each product.

Other available demo options:

```
topsccc_demo.py PORT --term-normal
```

"Normal" mode terminal.
Enter and run normal mode commands as found and described in module-specific documentation.
Ex. To find the name of the module, type: **$AAM** where AA is the hex address of the module ($CDM will ask the module at hex address CD for its name) and press Enter.
The reply is displayed, and then you may enter another command.

```
topsccc_demo.py PORT --term-modbus
```

Modbus mode terminal .
Enter and run modbus commands as described in module-specific documentation.
Input a space-separated list of hex bytes and press Enter to send.
Ex. To read the status of the digital inputs of the module at hex address 0B, type: `0x0B 0x02 0x00 0x00 0x00 0x08` and press Enter.
The response is displayed in the same format, and then you may enter another command.
NOTE: *The modbus checksum (CRC) calculation is done for you. The response packet's checksum is not verified, and is displayed with the rest of the response (the final two bytes).*

`topsccc_demo.py PORT --modbus`

Set the future operating mode of a module in INIT mode to modbus.
Simply a shortcut for the command $00P1.
This only works on a module that has its INIT pin shorted to ground, or INIT switch ON. See module-specific documentation for details.

`topsccc_demo.py PORT --normal`

Set the future operating mode of a module in INIT mode to modbus.
Simply a shortcut for the command $00P0.
This only works on a module that has its INIT pin shorted to ground, or INIT switch ON. See module-specific documentation for details.

TOPSCCC DEMO – MTCP SERIES

Requirements:

Ethernet connection
Python - tested with 2.7
Python is available from https://www.python.org/downloads/

Usage

To use, open a terminal window and run

```
topsccc_demo_mtcp.py
```

The demo will search for a module at the default address of 10.0.0.1 and run the demo if it finds one. The commands it is sending will be displayed in the terminal window.

An explanation of its behavior can be found in the code comments. Documentation for the example commands used here, as well as the full list of commands available, can be found in the module-specific manual for each product.

Other available demo options:

```
topsccc_demo.py --term-normal
```

"Normal" mode terminal.

Enter and run normal mode commands as found and described in module-specific documentation. For MTCP series modules, the address/ID is always 01.

Ex. To find the name of the module, type: **`$01M`** where press Enter.

The reply is displayed, and then you may enter another command.

```
topsccc_demo.py --term-modbus
```

Modbus mode terminal .

Enter and run modbus commands as described in module-specific documentation.

Input a space-separated list of hex bytes and press Enter to send. For MTCP series modules, the address/ID is always 01. Each Modbus packet has the same header (`0x00 0x00 0x00 0x00 0x00 0xLL`) where `0xLL` is the number of bytes in the packet “body”. Additionally, each packet begins with the same ID, `0x01`, so both are prepended automatically to each packet entered in the terminal.

Thus, to read the status of the digital inputs, type: **`0x02 0x00 0x00 0x00 0x0C`** and press Enter.

The terminal program automatically adds the remainder of the packet, so that what is sent is:

```
0x00 0x00 0x00 0x00 0x00 0x06 0x01 0x02 0x00 0x00 0x00 0x0C [CRC]
```

The response is displayed in the same format, and then you may enter another command.

NOTE: *The modbus checksum (CRC) calculation is done for you. The response packet's checksum is not verified, and is displayed with the rest of the response (the final two bytes).*