Using Tornado’s ioloop for SNAPconnects scheduler

Why would you want to do use Tornado’s ioloop for the scheduler? Tornado has many built in advantages for addressing the use of sockets and connections to external devices. An instance of using Tornado with SNAPconnect would be to have a local web server running that showcases data that is being polled from the SNAP network. Another might be to actually post data to a cloud server and have an open socket for connections to be received from the cloud server while still processing efficiently all the SNAP network details. Tornado uses filedescriptors in general for addressing socket communications and on a Unix clone (Linux, BSD, etc.) it will use them for processing serial data as well. This use of the ioloop for serial data allowing SNAPconnect to perform a standard select on serial data allows for more efficient handling of incoming packets from the connected serial port.

With this version of SNAPconect we now use the tornado ioloop for serial connections when configured to do so. If on windows systems SNAPconnect will poll the serial port as it has always done , even if you are using the tornado ioloop.

To install tornado on a pip enabled machine, just use ‘sudo pip install tornado’.

This example assumes that the user is already working on a system with SNAPconnect installed. While written to work with the E20 gateway from Synapse only the connection parameters for the SNAP bridge should be required to be modified to work on other platforms.

So what needs to be done to use tornado’s ioloop for your snapconnect application?

from apy import ioloop\_scheduler

import tornado.ioloop

# Local system settings for E20 environment

serial\_conn = snap.SERIAL\_TYPE\_RS232

serial\_port = '/dev/snap1'

snap\_addr = None # Intrinsic address on E20

SNAPCONNECT\_POLL\_INTERVAL = 5 # ms

# Create the SNAP rpc function list for your inbound rpcs

snapRpcFuncs = {'updateLEDs' : self.updateLEDs}

cur\_dir = os.path.dirname(\_\_file\_\_)

# Create SNAP Connect instance. Note: we are using TornadoWeb's scheduler.

self.snapconnect = snap.Snap(

license\_file = os.path.join(cur\_dir, 'license.dat'),

addr = snap\_addr,

scheduler=ioloop\_scheduler.IOLoopScheduler(),

funcs = self.snapRpcFuncs

)

# Connect to local SNAP wireless network

self.snapconnect.open\_serial(serial\_conn, serial\_port)

# Tell the Tornado scheduler to call SNAP Connect's internal poll function. Tornado already polls asyncore.

tornado.ioloop.PeriodicCallback(self.snapconnect.poll\_internals, self.SNAPCONNECT\_POLL\_INTERVAL).start()

From this point the snap connect features you would normally use operate the same way as you would if you were using the default asyncore scheduler.