3-Space Sensor Python API Quick-Start Guide

Purpose
The purpose of this document is to introduce users to the 3-Space Sensor Python Application Programming Interface (API). After reading this document, users will be capable of instantiating the API's various classes, interacting with the 3-Space family of sensors, and interpreting any read data in a meaningful way.

Overview
The Python 3-Space Python API is a collection of convenience functions and classes that wrap all normal functionality of the 3-Space wired sensors, wireless sensors, and wireless dongles for use in a Python interpreter. Additional features, such as broadcast commands, have also been added to further convenience of use. This particular module is meant to be used with a system running Python 3 and newer.

Software Requirements
Operating System – Windows XP – Windows 10

Python Interpreter – 32/64-Bit Python 3 or newer

Additional Required Python Modules –

- PySerial 2.6 or newer (may be found here: http://pypi.python.org/pypi/pyserial)
- threading (included with most interpreters)
- struct (included with most interpreters)
- time (included with most interpreters)
- sys (included with most interpreters)
- os (included with most interpreters)
- math (included with most interpreters)
- subprocess (included with most interpreters)
- multiprocessing (included with most interpreters)

Data Structure Descriptions
Before delving into how the API works, lets first describe the various classes and structures a programmer will be interacting with when using the 3-Space Sensor Python API.

threespace_api – This structure is a python module that contains all functions and classes used for interacting with the 3-Space family of attitude and heading reference devices.

TSSensor – This structure is a python class that acts as the interface between the programmer and a USB 3-Space sensor.

TSWLSensor – This structure is a python class that acts as the interface between the programmer and a 3-Space Wireless or Mini Wireless sensor.

TSDongle – This structure is a python class that acts as the interface between the programmer and a 3-Space dongle.

TSBootloader - This structure is a python class that acts as the interface between the programmer and a 3-Space sensor either without firmware or one in its “Bootloader” mode.

TSDLSensor - This structure is a python class that acts as the interface between the programmer and a Data Logging 3-Space sensor.

TSBTSensor - This structure is a python class that acts as the interface between the programmer and a Bluetooth or Mini Bluetooth 3-Space sensor.

TSEMSensor - This structure is a python class that acts as the interface between the programmer and an Embedded 3-Space sensor.

TSLXSensor - This structure is a python class that acts as the interface between the programmer and an LX 3-Space sensor

TSNANOSensor - This structure is a python class that acts as the interface between the programmer and Nano 3-Space sensor

global_async_broadcaster – This structure is a global python object that acts as a convenience tool for starting/stopping asynchronous communication sessions with a collection of 3-Space hardware as well as reading asynchronous data from all requested hardware at once.

Getting Started With The API

When connecting to and communicating with a piece of 3-Space hardware via the Python API, there are a few common steps that must be taken before full communication may take place. These steps are as follows:

1. Import API Module
2. Scan for Available 3-Space Hardware
3. Construct Interface Class Instances

Import API Module

Experienced Python programmers will be familiar with this step in the API communication process. This step imports all the functions and classes provided by the API into the programmer's application. This is done by typing the following line at the top of the application source file:

```
import threespace_api
```

Scan for Available 3-Space Hardware

With the module imported, the programmer now has access to all the functions and classes provided by the 3-Space Python API. Before one can retrieve data from a 3-Space sensor however, they must first detect what devices are available on the host computer and connect to them. This step of the setup process identifies all connected 3-Space devices on the system and details their method of communication. Two provided functions are used to retrieve this information:

getAvailableComPorts – This function scans the host computer for any available serial ports. It returns all discovered ports as a pair of lists. The first list contains all ports that the system has detected as corresponding to 3-Space sensors (using their driver information) and fall under an inputted filter of 3-Space device types. The second list contains all serial ports whose connected device could not be determined without polling the port for information. Each entry in these lists includes the port name, its "friendly name", and the type of 3-Space device connected (if known without polling the device) . Any serial ports not created by the 3-Space drivers will have an empty string for the 3-Space device type.

getComPortInfo – This function interrogates an inputted serial port for information on what type of hardware is connected to it. If a parameter in the function call is set, attempts will be made to write commands to the serial port and identify:

1. Whether the connected device (if any) is a 3-Space device.
2. What type of 3-Space device is connected.
3. The serial number of the connected device.
4. What version of software/hardware is the device (if 3-space).

This function is primarily useful for checking serial ports whose connected device is unknown to the system. It is preferred that users invoke this command from the utilizing application, as automatically polling unknown ports could cause unforeseen errors with non-3-Space devices connected via serial port.

Using the two above functions, a programmer should be able to identify all 3-Space sensors on the system, sort out which ports are desired to connect to, and protect arbitrary com ports from having data written to them and risking undesired behavior from non-3-Space hardware. The couple examples below will show some common applications of the above functions:

Example 1: Find the first available com port

```
tspace_port_lists, unknown_ports = threespace_api.getAvailableComPorts()
#Prints a tuple (PORT_NAME, FRIENDLY_NAME, DEV_TYPE)
print(tspace_port_lists[0])

>>> ('COM7', '3 Space Sensor (COM7)', 'USB')
```

Example 2: Get a list of all available USB 3-Space sensors

```
avail_usb_sens, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["USB"])
for com_port in avail_usb_sens:
#Prints a tuple (PORT_NAME, FRIENDLY_NAME, DEV_TYPE)
      print(com_port)
```

```
>>>('COM7', '3 Space Sensor (COM7)', 'USB')
('COM8', '3 Space Sensor (COM8)', 'USB')
('COM9', '3 Space Sensor (COM9)', 'USB')
```

Example 3: Get information on the com port "COM1"

```
com_info = threespace_api.getComPortInfo("COM1")
#Prints a tuple (FRIENDLY_NAME, DEV_TYPE, SERIAL, FIRMWARE_VER, HARDWARE_VER)
print(com_info)

>>>['3 Space Sensor (COM1)', 'USB', 540, '21DEC2011   ', 'TSS-USB                  v1.0.
0 ']
```

Construct Interface Class Instances

Once the desired serial ports have been identified, the next step is to instantiate the appropriate class for each said ports. As discussed above, there are seven possible device classes to connect. Each class makes available the features unique to that type of sensor. Below are some examples of instantiating a 3-Space device class:

Example 1: Connecting a USB sensor

```
avail_usb_sens, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["USB"])
new_sensor = TSSensor(com_port=avail_usb_sens[0])
```

Example 2: Connecting a Data Logging sensor

```
avail_dl_sens, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["DL"])
new_sensor = TSDLSensor(com_port=avail_dl_sens[0])
```

Example 3: Connecting a Bluetooth sensor

```
avail_bt_sens, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["BT"])
new_sensor = TSBTSensor(com_port=avail_bt_sens[0])
```

Example 4: Connecting an Embedded sensor

```
avail_em_sens, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["EM"])
new_sensor = TSEMSensor(com_port=avail_em_sens[0])
```

Example 5: Connecting a 3-Space Dongle

```
avail_dngl, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["DNG"])
new_dngl = TSDongle(com_port=avail_dngl[0])
```

Example 6: Connecting a Wireless sensor via USB

```
avail_wl_sens, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["WL"])
new_sensor = TSWLSensor(com_port=avail_wl_sens[0])
```

Example 7: Connecting a Wireless sensor via wireless

NOTE: It is assumed that the sensor/dongle are already configured for the same PanID and wireless channel. Furthermore, it is assumed that the sensor being used is already listed in the dongle's wireless table.

```
#First, connect a dongle(assuming it is configured to our sensor)
avail_dngl, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["DNG"])
new_dngl = TSDongle(com_port=avail_dngl[0])
#Next, we get a TSWLSensor representing the sensor listed as the first entry
#in the dongle's wireless table
new_sensor = new_dngl[0]
```

Example 8: Connecting a 3-Space device in Bootloader mode

```
avail_boots, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["BTL"])
new_sensor = TSBootloader(com_port=avail_boots[0])
```

3-Space devices connected either through USB, Serial Port, or Bluetooth may also be instantiated by passing in a string expressing the name of the serial port the device is connected through. The following example shows how to do this for a USB sensor, but the same process applies for all applicable devices.

```
new_sensor = TSSensor(com_port="COM1") #Assuming its connected to COM1
```

Now that the appropriate sensor class has been instantiated, all the functions and features of the connected hardware is available to the programmer. The programmer may now access data from the sensor such as its current orientation via methods of the instantiated class:

```
orientation = new_sensor.getFiltTaredOrientQuat()
```

Bootloader and Dongle devices cannot read orientations, thus the above function is not a method of their classes. The same calling process applies for any commands sent to these devices though.

To tie all of these concepts together a simple example script is listed below. This script creates a new USB 3-Space Sensor object, reads the orientation of the sensor and prints the orientation to the screen 100 times, then closes the sensor object (allowing other programs to access it).

Script Example:

```
#import the 3-Space API Module
import threespace_api
#Get the list of serial ports and instantiate our 3-Space device object
avail_usb_sens, unknown_ports = threespace_api.getAvailableComPorts(filter_list=["USB"])
new_sensor = TSSensor(com_port=avail_usb_sens[0])
#Get the current orientation of the connected sensor (as a quaternion) and print it
for i in range(100):
    orientation = new_sensor.getFiltTaredOrientQuat()
    print(orientation)
#Close the sensor's serial port so other applications may have access to it
new_sensor.close()
```

Details about more advanced features of the discussed functions as well as the full feature set of the 3-Space API is included in the API documentation. Please reference these documents for mastering the 3-Space Python API.

Yost Labs
630 Second Street
Portsmouth, Ohio 45662

Phone: 740-876-4936

www.yostlabs.com