Command Protocol V03

We use intelligent attachments, with controllers made from Raspberry Picos, ESP32s, Arduinos etc. and communicate from the central Raspberry Pi via a serial link over USB using a command protocol. The version of this used in 2022 was very compact, but lacking in some facilities, so I have added a bit more to it, but as little as possible.

All communication is initiated by the Pi. The controller cannot initiate anything. In normal use clashes are unlikely. However, to prevent clashes when the computers are busy, there is a handshake. For Pi to Pico this is a GPIO to GPIO wire. At the Pico end it is an output GPIO declared as:
`machine.Pin(pin_no, machine.Pin.OUT)`
At the Pi end it is an input GPIO declared as:
`pigpio.set_mode(pin_no, pigpio.INPUT)`
`pigpio.set_pull_up_down(pin_no, pigpio.PUD_UP)`
If the Pico has done nothing, or the wire is disconnected, the Pi value is 1 (not 0) because of the pullup, therefore a value of 1 indicates that it is set off. Because this can be confusing the Pico has functions `set_on()` and `set_off()`, while the Pi has `is_on()` which returns `True` or `False`. The controller sets the handshake to say it is ready to receive one command and unsets it while busy acting on the command. The Pi only sends commands when the Pico is ready.
The Pi has functions:
`send_command()` waits a certain length of time for the handshake, if timed out returns with a status code, otherwise it sends the command and waits for the feedback, which the Pico sends instantly, and then returns while the command is executed.
`send_command_and_wait()` is the same up to the end, where it waits after feedback for the handshake to be set again

All communication is via printable characters. The command format is:
serial_no - a four character string, which gets reflected back to the Pi so the Pi can detect drop-outs if it wants to.
command - a four character command. Command sets vary according to the application, but always include:
`WHOU` - a request for the controller to identify itself, in case there are multiple controllers
`EXIT` - an instruction to close down
data - this is optional and obviously application dependent. If numeric it should consist of 4 digit items to be interpreted as integers.
example: `0001DRIV0050000001000000` is
serial no: 1
command: DRIV
steering: 50
throttle: 0
duration: 100 (milliseconds)
crab: 0

The controller responds immediately to every command by echoing the serial number, followed by OKOK if all is well, or some error message.
example: `0001BADCCommand not understood`
Note that OKOK does not mean that the command can be executed, just that it is received and understood.

There are objects for the Pi (in ColinPiClasses) and the Pico (in ColinPicoClasses). Typical usage on the Pi would be (N.B. code not tested; it's just a hint):
```
import pigpio
gpio = pigpio.pi()
import CommandStream_V05 as CommandStream
handshake = CommandStream.Handshake(4, gpio)
pico_id = 'PICOA'
my_pico = CommandStream.Pico(pico_id, gpio, handshake)
serial_no_sent = 0
serial_no_sent_string = '{:04f}'.format(serial_no_sent)
command = 'WHOU'
serial_no_returned, feedback, data = my_pico.send_command(
    serial_no_sent_string + command)
if serial_no_returned != serial_no_sent_string:
    raise (ColError('sequence error'))
serial_no_sent += 1
```

Typical Pico usage:
```
import CommandStreamPico_V05 as CommandStream
my_handshake = ThisPico.ThisHandshake()
my_stream = CommandStream.CommandStream('From Pi', my_handshake)
serial_no, command, data = my_stream.get()
if command == 'WHOU':
    my_stream.send(serial_no + 'OKOK' + 'PICOA')
```