Python Code

This a hub page for Python code linked to other pages using the `section` tag.

Example of how to embed a code section in another page in DokuWiki:
`{ {section>:python_code#user_string&noheader&nofooter} }`
Remove the spaces between the curly brackets.

Any script with dependencies should use inline script metadata.

Create Button Flag

Python Create Button Flag

asi_create_button_flag.py

```
def create_button_code(at: int = 0, home: int = 0, joystick: int =
0, zero_halt: int = 0) -> int:
    assert at in range(4), "Must be in the range 0-3."
    assert home in range(4), "Must be in the range 0-3."
    assert joystick in range(4), "Must be in the range 0-3."
    assert zero_halt == 0 or zero_halt == 1, "Must be 0 or 1."

    button_code = 0

    # bit masks
    BITMASK_AT   = 0x03 # 00000011
    BITMASK_HOME = 0x0C # 00001100
    BITMASK_JS   = 0x30 # 00110000
    BITMASK_ZERO = 0xC0 # 11000000

    # set bits
    button_code &= ~BITMASK_AT
    button_code |= at & BITMASK_AT

    button_code &= ~BITMASK_HOME
    button_code |= (home << 2) & BITMASK_HOME

    button_code &= ~BITMASK_JS
    button_code |= (joystick << 4) & BITMASK_JS

    button_code &= ~BITMASK_ZERO
    button_code |= (zero_halt << 6) & BITMASK_ZERO

    return button_code

def main():
```

```
    button_code = create_button_code(at=1, home=1)
    print(button_code)
    # prints 5

if __name__ == "__main__":
    main()
```

Parse Button Flag

Python Parse Button Flag

asi_parse_button_flag.py

```
# the value returned from EXTRA M?
button_flag_byte = 127

# bit masks
mask_at   = 0x03 # 00000011
mask_home = 0x0C # 00001100
mask_js   = 0x30 # 00110000
mask_zero = 0xC0 # 11000000

# get the button states from button_flag_byte
btn_at   = button_flag_byte & mask_at
btn_home = (button_flag_byte & mask_home) >> 2
btn_js   = (button_flag_byte & mask_js) >> 4
btn_zero = (button_flag_byte & mask_zero) >> 6

# show the results in decimal and binary
print(f"{button_flag_byte = } (binary {button_flag_byte :08b})")
print(f"{btn_at = } (binary {btn_at:02b})")
print(f"{btn_home = } (binary {btn_home:02b})")
print(f"{btn_js = } (binary {btn_js:02b})")
print(f"{btn_zero = } (binary {btn_zero:02b})")

# console output:
# button_flag_byte = 127 (binary 01111111)
# btn_at = 3 (binary 11)
# btn_home = 3 (binary 11)
# btn_js = 3 (binary 11)
# btn_zero = 1 (binary 01)
```

Status Byte

Python Status Byte

asi_status_byte.py

```python
# /// script
# dependencies = ["pyserial>=3.5"]
# ///
import serial
from enum import Flag, auto

class StatusByte(Flag):
    """
    The Status Byte returned by the RDSBYTE (RB) command.
    The value parameter should be an 8-bit int, 0-255.
    00000001 <- Bit 0 is 1, commanded move in progress.
    """
    COMMANDED_MOVE = auto()
    AXIS_ENABLED = auto()
    MOTOR_ACTIVE = auto()
    JS_KNOB_ENABLED = auto()
    MOTOR_RAMPING = auto()
    MOTOR_RAMPING_UP = auto()
    AT_UPPER_LIMIT = auto()
    AT_LOWER_LIMIT = auto()

def main() -> None:
    # use an empty string for MS2000 (card_address = "")
    card_address = "1"

    # which axes to query (for a single axis use: axes = ["X"])
    # axes_byte_len is the number of bytes that need to be read
for RDSBYTE
    axes = ["X", "Y"]
    axes_str = " ".join(axes)
    axes_byte_len = len(axes) + 3  # 3 bytes for ':', '\r', and
'\n'

    with serial.Serial("COM5", 115200, timeout=1) as serial_port:
        # query the controller for the status byte of each axis
        command = f"{card_address}RB {axes_str}\r"
        serial_port.write(bytes(command, encoding="ascii"))
        response = serial_port.read(axes_byte_len)

        # report and check for errors
        print(f"Send: \"{command[:-1]}\"")
        print(f"Recv: \"{response}\" (interpreted as ASCII)")
        print(f"Number of bytes to read: {axes_byte_len}\n")
```

```
        if b"N" in response:
            print("Error in response...")
            return

        # view as bytes
        response_bytes = [byte for byte in response]
        print(f"{response_bytes = } (raw bytes)\n")

        # get the status byte for each axis and skip the first
byte (':')
        status_bytes = [response[i] for i in range(1, len(axes) +
1)]
        print(f"{status_bytes = } (decimal)")

        # create a dictionary that maps uppercase axis names to
status bytes
        status_bytes_dict = {axis.upper(): StatusByte(status_byte)
for (axis, status_byte) in zip(axes, status_bytes, strict=True)}
        print(f"{status_bytes_dict = }\n")

        # check the status of each axis
        for axis in axes:
            status_byte = status_bytes_dict.get(axis.upper())
            # check a single flag
            is_at_upper_limit = StatusByte.AT_UPPER_LIMIT in
status_byte
            # check multiple flags
            is_js_and_axis_enabled = StatusByte.JS_KNOB_ENABLED |
StatusByte.AXIS_ENABLED in status_byte
            # check if one flag is True and the other is False
            is_motor_on_and_not_cmd_move = StatusByte.MOTOR_ACTIVE
in status_byte and StatusByte.COMMANDED_MOVE not in status_byte
            print(f"{axis} Axis Status: {status_byte.value:08b}")
# print the status_byte in binary
            print(f"{status_byte = }")
            print(f"{is_at_upper_limit = }")
            print(f"{is_js_and_axis_enabled = }")
            print(f"{is_motor_on_and_not_cmd_move = }\n")

if __name__ == "__main__":
    main()
```

User String

Python User String

asi_user_string.py

```
# /// script
# dependencies = ["pyserial>=3.5"]
# ///
import serial

def main() -> None:
    # the input string to send to the controller
    user_string = "abcdefghij1234567890"
    save_settings = True

    # use an empty string for MS2000 (card_address = "")
    card_address = "2"

    # error checking
    if len(user_string) > 20:
        raise Exception("Max stored string length is 20.")

    # open the serial port and send characters to the controller
    with serial.Serial("COM4", 115200, timeout=1) as serial_port:
        # clear the current stored string
        serial_port.write(bytes(f"{card_address}BU Y-\r",
encoding="ascii"))
        serial_port.readline()

        # send the input string to the controller
        for character in user_string:
            serial_port.write(bytes(f"{card_address}BU
Y={ord(character)}\r", encoding="ascii"))
            serial_port.readline()

        # print the stored string and check to see if it's the
same as the input string
        serial_port.write(bytes(f"{card_address}BU Y?\r",
encoding="ascii"))
        response = serial_port.readline().decode().rstrip("\r\n")
        if response == user_string:
            print(f"Successfully stored the input string =>
{response}")
            if save_settings:
                serial_port.write(bytes(f"{card_address}SAVESET
Z\r", encoding="ascii"))
                serial_port.readline()
                print("Settings saved to the controller using
SAVESET Z.")
        else:
            print(f"Error: expected {user_string} but got
{response} instead!")

if __name__ == "__main__":
```

```
    main()
```