Sending Binary Data

Sockets transmit streams of bytes. Those bytes can contain text messages encoded to bytes, as in the previous examples, or they can be made up of binary data that has been packed into a buffer with `struct` to prepare it for transmission.

This client program encodes an integer, a string of two characters, and a floating point value into a sequence of bytes that can be passed to the socket for transmission.

```
# socket_binary_client.py

import binascii
import socket
import struct
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 10000)
sock.connect(server_address)

values = (1, b'ab', 2.7)
packer = struct.Struct('I 2s f')
packed_data = packer.pack(*values)

print('values =', values)

try:
    # Send data
    print('sending {!r}'.format(binascii.hexlify(packed_data)))
    sock.sendall(packed_data)
finally:
    print('closing socket')
    sock.close()
```

When sending multi-byte binary data between two systems, it is important to ensure that both sides of the connection know what order the bytes are in and how to assemble them back into the correct order for the local architecture. The server program uses the same `Struct` specifier to unpack the bytes it receives so they are interpreted in the correct order.

```
# socket_binary_server.py

import binascii
import socket
import struct
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 10000)
sock.bind(server_address)
sock.listen(1)

unpacker = struct.Struct('I 2s f')

while True:
    print('\nwaiting for a connection')
    connection, client_address = sock.accept()
    try:
        data = connection.recv(unpacker.size)
        print('received {!r}'.format(binascii.hexlify(data)))

        unpacked_data = unpacker.unpack(data)
        print('unpacked:', unpacked_data)
```

```
    finally:
        connection.close()
```

Running the client produces:

```
$ python3 source/socket/socket_binary_client.py
values = (1, b'ab', 2.7)
sending b'0100000061620000cdcc2c40'
closing socket
```

And the server shows the values it receives:

```
$ python3 socket_binary_server.py

waiting for a connection
received b'0100000061620000cdcc2c40'
unpacked: (1, b'ab', 2.700000047683716)

waiting for a connection
```

The floating point value loses some precision as it is packed and unpacked, but otherwise the data is transmitted as expected. One thing to keep in mind is that depending on the value of the integer, it may be more efficient to convert it to text and then transmit, instead of using `struct`. The integer 1 uses one byte when represented as a string, but four when packed into the structure.

See also

- `struct` – Converting between strings and other data types.

Multicast

Non-blocking Communication and Timeouts

This page was last updated 2016-12-17.

Navigation

Multicast
Non-blocking Communication and Timeouts

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

- Blog
- The Python Standard Library By Example