



1. Python

The Digi device server provides a simple scripting environment using Python.

For detailed information on the Python functionality supported on the Digi device server refer to the "Digi International Python Programming Guide".

http://ftp1.digi.com/support/documentation/90000833_b.pdf

2. Python Handling

Python File Management Administration

Python program files and modules can be uploaded to your Digi device server.

To manually execute an uploaded program, access the command line of your Digi device server.

Then type the command: `python filename [program args...]`

Upload Files

Click Browse to select a file to upload to your Digi device server and then click Upload.

Manage Files

Select any files you would like to remove from your Digi device server and click Delete.

Python Configuration

▼ Python Files

Upload Files

Upload Python programs

Upload File:

Manage Files

Action	File Name	Size
No files currently uploaded.		

▶ Auto-start Settings



Digi Connect ES

How to use Python and an application example

Python Auto-start Settings

This allows you to configure Python programs to execute when the Digi device server boots. Up to four entries can be configured.

Enable

When checked, the program specified in the Auto-start command line field will be run when the device boots.

Auto-start command line

Specify the Python program filename to be executed and any arguments to pass to tprogram.

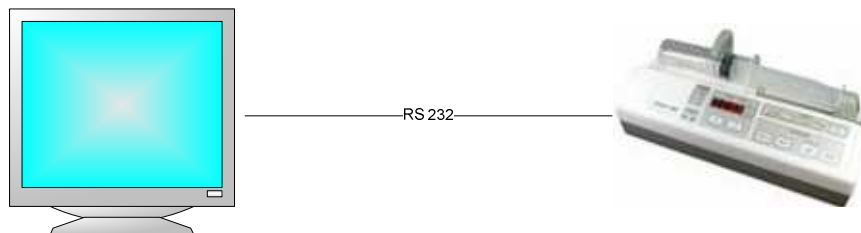
The syntax is: filename [arg1 arg2...]

The screenshot shows a window titled "Python Configuration". Under the "Auto-start Settings" section, there is a heading "Specify python programs to be run when the device boots." Below this, there is a table with two columns: "Enable" and "Auto-start command line (specify program filename to execute and any arguments)". There are four rows, each with a checkbox in the "Enable" column and an empty text input field in the "Auto-start command line" column. At the bottom of the dialog, there are "Apply" and "Cancel" buttons.

3. Python Y-Cable application

Situation and Problem

The data between an infusion pump and a touch screen is transferred via a serial crossover cable. There is no way to store the data to an independent server to record all medical customer details.





Digi Connect ES

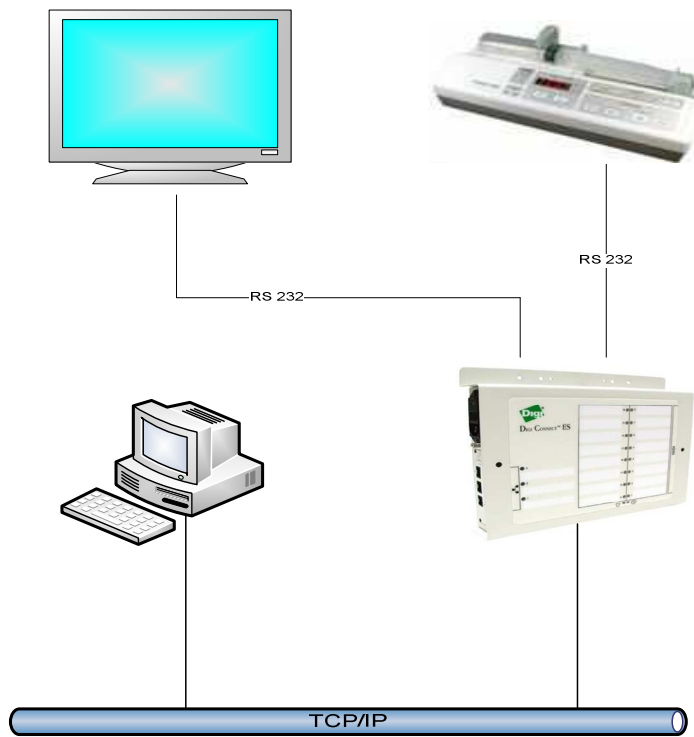
How to use Python and an application example

Solution

Connect the Monitor and pump to the Connect ES and use our python "Y-Cable" python script.

The script does following:

- Transmitting all serial data between the pump and monitor.
- In addition all data will be send out on TCP socket which can be stored on a server
- In the case you experience for example data fragmentation on the pump or monitor you have the option to packet the serial protocol and all messages will be sent in single packets.



4. Data Fragnetation of serial protocol

Here you see the serial data send between the pump and monitor:

```

{bh}{06/11/2008,12:27:52.145}~o~I~d~ij~
E~x~m~2~
0b~I~9b~I~ç~I~ -~I~ -~I~
/~~{be}{bh}{06/11/2008,12:27:52.145}~o~I~d~ij~
E~x~m~2~
~u~p~ÛH~0b~I~ç~I~ -~I~ -~I~
/~~{be}~{bh}{06

```



Digi Connect ES

How to use Python and an application example

To get the data transferred unfragmented you need to understand the protocol. In this example the block starts with “{bh}” and ends with “{be}”. In the script below you find the blocking for that.

To prove the blocking of the protocol you can use the “set trace python” output and / or Wireshark.

```

No.      Time           Source           Destination      Protocol  Info
  1002  9.184750      10.49.2.114     10.49.2.125     TCP       gbjd816
> odn-castraq [PSH, ACK] Seq=1 Ack=1 Win=8739 Len=407

```

```

Frame 1002 (461 bytes on wire (461 bytes captured)
Ethernet II, Src: DigiBoar_29:98:4d (00:40:9d:29:98:4d), Dst: WwPcbaTe_e3:4c:b6 (00:0f:1f:e3:4c:b6)
Internet Protocol, Src: 10.49.2.114 (10.49.2.114), Dst: 10.49.2.125 (10.49.2.125)
Transmission Control Protocol, Src Port: gbjd816 (2626), Dst Port: odn-castraq (2498), Seq: 1, Ack: 1, Len: 407
Data (407 bytes)

```

```

0000 7b 62 68 7d 7b 30 36 2f 31 31 2f 32 30 30 38 2c {bh}{06/11/2008,
0010 31 32 3a 32 37 3a 35 32 2e 31 34 35 7d 0d 0a 7e 12:27:52.145}..~
0020 6f 01 00 03 00 00 a8 e3 12 49 00 00 00 00 64 00 o.....I....d.
0030 00 00 01 00 00 ff 00 00 00 00 00 00 00 00 00 00 .....
0040 00 00 00 00 00 00 00 00 04 02 f4 01 0a 00 c9 00 .....
0050 00 00 19 00 90 01 78 00 f4 01 00 00 32 00 0a 00 .....x...2...
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 01 80 01 80 .....
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00b0 00 00 01 80 00 00 00 00 00 00 00 05 00 00 00 00 .....
00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00d0 00 00 00 00 00 00 00 00 00 f9 b3 fc 48 00 00 00 .....H...
00e0 00 d4 de 0a 49 00 00 00 39 df 0a 49 00 00 00 00 .....I...9..I...
00f0 00 c7 b6 0a 49 00 00 03 b7 0a 49 00 00 82 b7 0a .....I...I...
0100 49 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 I.....
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2f ...../
0190 7e 0d 0a 7b 62 65 7d ~...{be}

```

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=1 Ack=1 Win=8739 Len=407
2	0.109501	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=408 Win=64105 Len=0
3	1.052012	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=408 Ack=1 Win=8739 Len=538
4	1.211113	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=946 Win=63567 Len=0
5	1.796304	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=946 Ack=1 Win=8739 Len=411
6	1.912095	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=1357 Win=63156 Len=0
7	2.571908	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=1357 Ack=1 Win=8739 Len=411
8	2.713233	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=1768 Win=64532 Len=0
9	3.287999	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=1768 Ack=1 Win=8739 Len=411
10	3.414283	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=2179 Win=64101 Len=0
11	4.084219	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=2179 Ack=1 Win=8739 Len=411
12	4.215442	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=2590 Win=63690 Len=0
13	4.788214	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=2590 Ack=1 Win=8739 Len=411
14	4.916410	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=3001 Win=63279 Len=0
15	5.620711	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=3001 Ack=1 Win=8739 Len=411
16	5.817734	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=3412 Win=64532 Len=0
17	6.332140	10.49.2.114	10.49.2.125	TCP	gbjd816 > odn-castraq [PSH, ACK] Seq=3412 Ack=1 Win=8739 Len=411
18	6.518716	10.49.2.125	10.49.2.114	TCP	odn-castraq > gbjd816 [ACK] Seq=1 Ack=3823 Win=64101 Len=0

```

> Frame 1 (461 bytes on wire, 461 bytes captured)
> Ethernet II, Src: DigiBoar_29:98:4d (00:40:9d:29:98:4d), Dst: WwPcbaTe_e3:4c:b6 (00:0f:1f:e3:4c:b6)
> Internet Protocol, Src: 10.49.2.114 (10.49.2.114), Dst: 10.49.2.125 (10.49.2.125)
> Transmission Control Protocol, Src Port: gbjd816 (2626), Dst Port: odn-castraq (2498), Seq: 1, Ack: 1, Len: 407
> data (407 bytes)

```

```

3000 00 0f 1f e3 4c b6 00 40 9d 29 98 4d 08 00 45 00 .....6...M..E.
3010 01 6f 00 02 00 00 40 06 5f e7 0a 31 02 72 0a 31 .....6...1..r..I
3020 02 7d 01 42 09 c2 9b 61 3e 77 40 57 6f 05 10 15 19 ..B...66..p..p
3030 22 23 53 f9 00 00 7b 62 68 7d 7b 30 36 2f 31 31 #S...{bh}{06/11
3040 2f 32 30 30 38 2c 31 32 3a 32 37 3a 35 32 2e 31 /2008,12:27:52.1
3050 54 33 7d 08 0a 7a 6f 01 00 00 00 a8 e3 12 49 45 }...o.....I
3060 00 00 00 00 64 00 00 00 01 00 00 ff 00 00 00 00 .....d.....
3070 00 00 00 00 00 00 00 00 00 00 00 00 00 04 02 .....
3080 04 01 00 00 e9 00 00 19 00 00 01 78 00 f4 01 .....
3090 00 00 33 00 0a 00 00 00 00 00 00 00 00 00 00 .....
30a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
30b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
30c0 00 00 01 80 01 80 00 00 00 00 00 00 00 00 00 .....
30d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
30e0 00 00 00 00 00 00 00 01 80 00 00 00 00 00 00 .....
30f0 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3100 00 00 00 00 00 00 00 00 00 00 00 00 00 f9 .....
3110 03 fc 48 00 00 00 04 0e 0a 49 00 00 00 00 39 .....H...I...9
3120 0f 03 49 00 00 00 c7 89 0a 49 00 00 03 b7 0a I...I...
3130 49 00 00 32 b7 0a 49 00 00 00 00 00 00 00 00 .....
3140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
3190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
31a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
31b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```



Digi Connect ES

How to use Python and an application example

5. Python Y-Cable Code

```
#!/usr/bin/python
"""
Constructs a data "tee" by passing data between two serial ports and presenting a copy of
this data to TCP clients connected to a defined TCP port on the system.
"""

import sys, os
from socket import *
from select import *

PORT_7 = "/com/6"          # Port 7 of the Connect ES
PORT_8 = "/com/7"          # Port 8 of the Connect ES

TCP_PORT = 2626            # The TCP server port to copy data to when a client is present.

MAX_BUFFER_LEN = 8192     # Maximum amount of data to buffer

class TCPClientContext:
    def __init__(self, client_sd, client_addr):
        self.sd = client_sd          # Client socket handle
        self.buffer = ""             # Data buffer for the client.
        self.address = client_addr   # The client's TCP address

def main():
    byteSumIn = 0
    byteSumOut232 = 0
    byteSumOutTCP = 0
    packetStart = False            # packet header received?

    port_a_dev = PORT_7           #this port is only able to send
    port_b_dev = PORT_8           #this port is only able to receive
    server_port = TCP_PORT

    if sys.argv and len(sys.argv) > 1:
        if len(sys.argv) > 1:
            port_a_dev = sys.argv[1]

        if len(sys.argv) > 2:
            port_b_dev = sys.argv[2]

        if len(sys.argv) > 3:
            server_port = int(sys.argv[3])

    port_a_fd = os.open(port_a_dev, os.O_NONBLOCK | os.O_RDWR)
    port_b_fd = os.open(port_b_dev, os.O_NONBLOCK | os.O_RDWR)
    buf_a_to_b = ""
    buf = ""
    dummybuf = ""

    client_dict = { }

    server_sd = socket(AF_INET, SOCK_STREAM)
    server_sd.bind(('', server_port))
    server_sd.listen(4)           # Maximum number of pending clients

    while 1:

        client_disc_q = [ ]      # Reset client disconnection processing queue:

        r1, w1, x1 = ([], [], [])# Initialize lists of descriptors to monitor w/select():

        if len(buf_a_to_b) < MAX_BUFFER_LEN:
            r1.append(port_a_fd)
        if len(dummybuf) < MAX_BUFFER_LEN:
            r1.append(port_b_fd)

        if packetStart:
            if len(buf_a_to_b):
                w1.append(port_b_fd)
            for client_sd in client_dict:
                if len(client_dict[client_sd].buffer):
                    w1.append(client_sd)
            packetStart = False

        r1.append(server_sd)     # Add server socket to read list to accept new connections:

        r1, w1, x1 = select(r1, w1, x1) # Block appl. execution until action is required:
```



Digi Connect ES

How to use Python and an application example

```
if server_sd in r1:                # Process new TCP connections:
    client_sd, client_addr = server_sd.accept()
    client_sd.setblocking(0)
    client_dict[client_sd] = TCPCClientContext(client_sd, client_addr)

if port_b_fd in w1:
    n = os.write(port_b_fd, buf_a_to_b)
    byteSumOut232 += len(buf_a_to_b)
    print "out(232): %i (sum: %i)" % \
          (len(buf_a_to_b), byteSumOut232)
    buf_a_to_b = buf_a_to_b[n:]

for client_sd in client_dict:
    if client_sd in w1:
        try:
            tmp = client_dict[client_sd].buffer
            n = client_sd.send(tmp)
            byteSumOutTCP += len(tmp)
            print "out(TCP): %i (sum: %i)" % \
                  (len(tmp), byteSumOutTCP)
            client_dict[client_sd].buffer = tmp[n:]
        except:
            client_disc_q.append(client_sd)

for client_sd in client_disc_q:    # Process any TCP client disconnections:
    del(client_dict[client_sd])

if port_a_fd in r1:                # Process reads to preform:
    old_len = len(buf)
    read_len = MAX_BUFFER_LEN - len(buf_a_to_b)
    buf += os.read(port_a_fd, read_len)
    print "in      : %s" % (len(buf)-old_len)
    byteSumIn += (len(buf)-old_len)
    index = buf.find("{be}")        # check if packet header received
    if index > -1:
        packetStart = True
        buf_write = buf[:index+4]  # buf till {be} belongs to packet
        buf_a_to_b += buf_write
        for client_sd in client_dict:
            copy_len = MAX_BUFFER_LEN - len(client_dict[client_sd].buffer)
            client_dict[client_sd].buffer += buf_write[:copy_len]
        buf = buf[index+4:] # buf from {be} belongs to next packet

    # please use "index+8" if 0A0D0A0D should be discarded

if port_b_fd in r1:
    read_len = MAX_BUFFER_LEN
    dummybuf = os.read(port_b_fd, read_len)
    print "disregarding input from port b: ", dummybuf

if __name__ == '__main__':
    import sys
    status = main()
    sys.exit(status)
```