APPLICATION NOTE

MicroComm DXI

TCP/IP Host Protocol Example Design

Table of Contents

1. **Intent & Scope** 2
2. **Introduction** 2
3. **Special Features** 2
   - 3.1 Non-blocking Connections 2
   - 3.2 Timers 2
     - 3.2.1 RX Timer 3
     - 3.2.2 TX Timer 3
     - 3.2.3 Connect Timer 3
4. **SAC Configuration** 4
5. **Software Design** 6
   - 5.1 main () 6
   - 5.2 initialize() 7
   - 5.3 cleanup() 7
   - 5.4 event_loop() 8
   - 5.5 reconnect_ports() 9
   - 5.6 receive_tcp_message() 10
   - 5.7 process_tcp_packet() 11
   - 5.8 process_tcp_message() 12
   - 5.9 send_message() 14
   - 5.10 open_port() 14
   - 5.11 connect_port() 15
   - 5.12 activate_port() 16
   - 5.13 close_port() 17
   - 5.14 other_port() 19
   - 5.15 synchronize_to_dxi() 19

1. Intent & Scope

The purpose of this procedure is to describe a sample host client that can interface to DXI TCP/IP Host protocol to control the DXI SAC computer. This sample host client is written to make use of redundant SAC computers for fail-safe operation.

2. Introduction

This example shows a simple program using flowcharts that implements the communication to the DXI system using full redundancy. Example "C" source code for this program is available upon request.

The DXI TCP/IP host protocol communicates using simple text strings, with each message being terminated by a <CR> character. The format of these strings and the commands within them are found in the Harding Instruments document "IM-MES-DXI-R14" entitled "MicroComm DXI Host Port Command/Status Messages".

The document "IM-PROT-TCP-R1" entitled "MicroComm DXI TCP/IP Host Communications Protocol" describes the basics necessary to establish a simple connection to the DXI system.

This document focuses mainly on the connection protocol required to communicate with redundant SAC computers, and can be considered a supplement to the above mentioned "MicroComm DXI TCP/IP Host Communications Protocol" document.

This document assumes the reader has some experience with programming TCP sockets.

3. Special Features

Special features this example uses that may require further explanation are described below.

3.1 Non-blocking Connections

This example makes use of non-blocking TCP connections so that a faulted SAC computer or network connection will not cause delays in the system. Most TCP stacks should have some function to allow non-blocking connections.

Non-blocking connections will always return immediately when attempting to connect. A failure to connect will later be reported and handled in the main event_loop.

3.2 Timers

The term "NOOP" is derived from the term **No**-**op**eration. This is a SAC computer command that does nothing – but it can be used to indicate that the system is operational and communicating. The "NOOP" command is used extensively as part of the timer software.

In this example we use three types of timers.

3.2.1 RX Timer

There is a rx timer on the host client for each SAC computer, which gets reset each time a message is received from the SAC computer. If the timer exceeds a given time-out value without receiving a message from the SAC computer, it can be assumed that the SAC computer has failed or that the network connection is down.

Hence, when the rx timer times out, that SAC connection is disconnected and closed.

When configured properly, the DXI system will send a "NOOP" or ping message periodically to allow this detection mechanism to work. To allow for latency in processing and synchronization issues, the actual rx timer time-out on the client side should be double this setting in the SAC software. The time out value is a trade-off dependent on the maximum detection time allowable for dead connections versus available network bandwidth. A value between 5 and 30 seconds for the DXI NOOP timer (i.e. 10 to 60 seconds for rx timer timeout) is generally acceptable.

Rx timers should always be used, as they allow the host client to properly detect when the connection to the DXI system is lost.

3.2.2 TX Timer

There is a tx timer associated with each SAC computer, and it is the inverse of the rx timer. The tx timer gets reset each time a message is sent to the SAC computer. If the timer exceeds a given time-out value without sending a message to the SAC computer, the client host will send a dummy message to the SAC computer to tell it that the client host is still active.

The tx timer can be designed to operate in two different ways. The first method is to design the tx timer to simply send a "NOOP" to the SAC computer and clear the tx timer whenever the tx timer times out. The second method is to send a "NOOP" and clear the tx timer whenever the tx timer times out but also to clear the tx timer every time any other message is sent to the SAC computer. The tradeoff between these two approaches is that the first method, while easier to implement, uses slightly more bandwidth sending more "NOOPs" than the second approach.

With either method, the SAC computer is able to detect a dead host client or connection, display the connection error on a master station, and allow alternate access (such as enabling the keypad/display on masters with Exclusive host enabled) if no host client is available. Much like the rx timer, the time-out value on the SAC software should be twice the tx timer value on the client. Again, a value between 5 to 30 seconds for the tx timer (i.e. 10 to 60 seconds for the DXI inactivity timer timeout) is generally acceptable.

This timer is optional.

3.2.3 Connect Timer

The third timer is a connect timer. This timer will allow the host client to periodically check each SAC computer's connection, and try to re-establish it if it is not currently connected.

When the connect timer times out, check the status of all SAC connections, and try to re-establish any connections that are currently not connected.

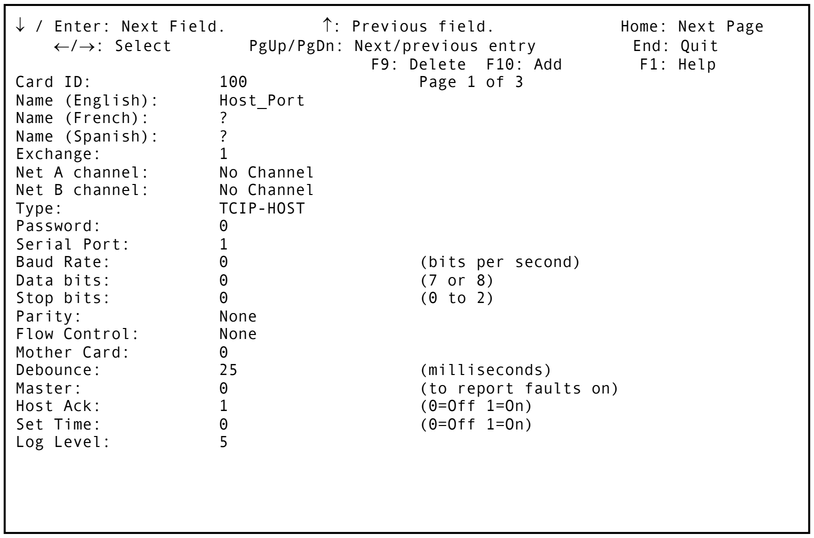
Choosing a value for the connection timer will affect how often you will attempt to re-connect to previously dead connections. For certain applications, particularly if you must use blocking connections, failed connections may result in detrimental performance, so in these cases the connection timer may have to be set to a longer value. This value may typically be between 5 seconds (if using non-blocking connections) to minutes (if using blocking connections)

A connect timer should always be used, as it allows the host client to automatically re-establish connections to the DXI system upon a computer reboot or after network communication problems.

4. SAC Configuration

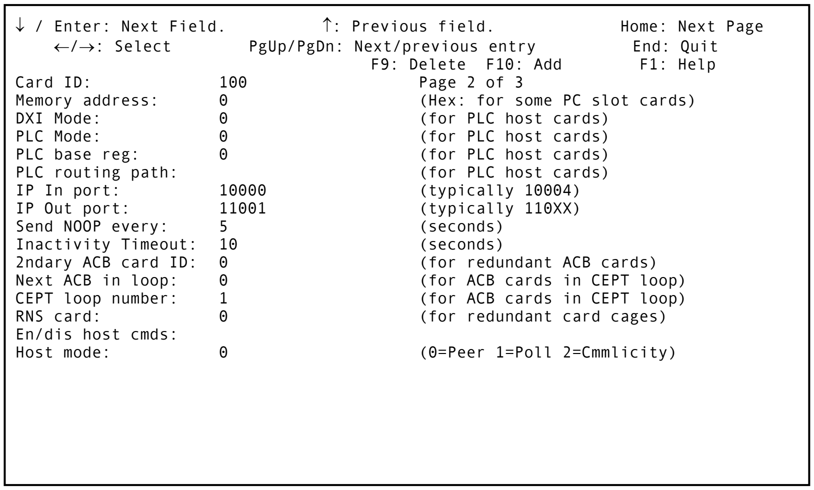
To configure the SAC for the ASCII TCP/IP implementation, the following steps should be followed.

First, a host card needs to be configured. Enter the Mod_HW program by going to the Maintenance/Edit/Modify Hardware menus, then go to the Card database. Create a new card (typically 100). The "Type" of card should be TCPIP_HOST, "Serial Port" should be 1, "Host Ack" should be 1.



```
↓ / Enter: Next Field.          ↑: Previous field.             Home: Next Page
    ←/→: Select         PgUp/PgDn: Next/previous entry          End: Quit
                                     F9: Delete  F10: Add        F1: Help
Card ID:             100                  Page 1 of 3
Name (English):      Host_Port
Name (French):       ?
Name (Spanish):      ?
Exchange:            1
Net A channel:       No Channel
Net B channel:       No Channel
Type:                TCIP-HOST
Password:            0
Serial Port:         1
Baud Rate:           0                    (bits per second)
Data bits:           0                    (7 or 8)
Stop bits:           0                    (0 to 2)
Parity:              None
Flow Control:        None
Mother Card:         0
Debounce:            25                   (milliseconds)
Master:              0                    (to report faults on)
Host Ack:            1                    (0=Off 1=On)
Set Time:            0                    (0=Off 1=On)
Log Level:           5
```

On the second page of the Card database the "IP in port" should be assigned a number – this is the port number that the host client program needs to connect to. Our recommended port number is 10000, but any valid TCP/IP port number is acceptable. Note, that if multiple host cards are assigned on the same SAC computer, each host card must have a unique "IP in port" number. The DXI NOOP timer value (half the rx timer value) should be entered in the "Send NOOP every" field. Finally, if the host will send NOOP commands to the DXI, the DXI inactivity time-out value (double the tx timer value) should be entered in the "Inactivity timeout" field.



```
↓ / Enter: Next Field.          ↑: Previous field.             Home: Next Page
    ←/→: Select         PgUp/PgDn: Next/previous entry          End: Quit
                                     F9: Delete  F10: Add        F1: Help
Card ID:             100                  Page 2 of 3
Memory address:      0                    (Hex: for some PC slot cards)
DXI Mode:            0                    (for PLC host cards)
PLC Mode:            0                    (for PLC host cards)
PLC base reg:        0                    (for PLC host cards)
PLC routing path:                         (for PLC host cards)
IP In port:          10000                (typically 10004)
IP Out port:         11001                (typically 110XX)
Send NOOP every:     5                    (seconds)
Inactivity Timeout:  10                   (seconds)
2ndary ACB card ID:  0                    (for redundant ACB cards)
Next ACB in loop:    0                    (for ACB cards in CEPT loop)
CEPT loop number:    1                    (for ACB cards in CEPT loop)
RNS card:            0                    (for redundant card cages)
En/dis host cmds:
Host mode:           0                    (0=Peer 1=Poll 2=Cmmlicity)
```

Secondly, the SAC has to be told to send a PCsw message when the active PC is switched. To do this, remain in the Mod_HW program, and go to the "Exchange" database. Set the "Side A ID" to the DXI node number of the first computer of this exchange – i.e. the same value that is entered in the "QNX ID (PC A)" field, and the "Side B ID" to the DXI node number of the second computer in this exchange – i.e. the same value that is entered in the "QNX ID (PC B)" field. The "Master" setting should be set to a valid host controlled master number to receive this PCsw message. Exit out to the Maintenance menu screen.

To activate the host card, using the Maintenance/Maintenance/Swap Card menu, swap in the host card.

To configure the masters to send messages to host client, enter the Mod_SW program by going into the Administration/Edit/Modify Software menus, then go to the Master database. For each master to be controlled from the host client, set the "Ext Host Card" to the host card number (100 in this example) and the "External Host" to Parallel (to have it work in conjunction with any IMS keypad/display masters you have).

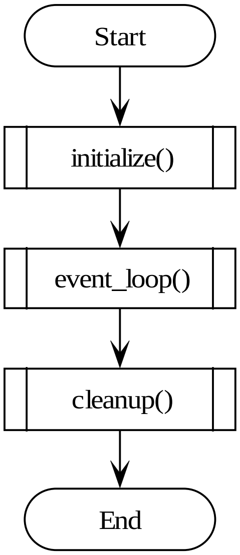
Reboot the SAC for the changes to take effect.

5. Software Design

5.1 main ()

The main program initializes, executes the main event_loop until a quit command is received, then cleans up before quitting.

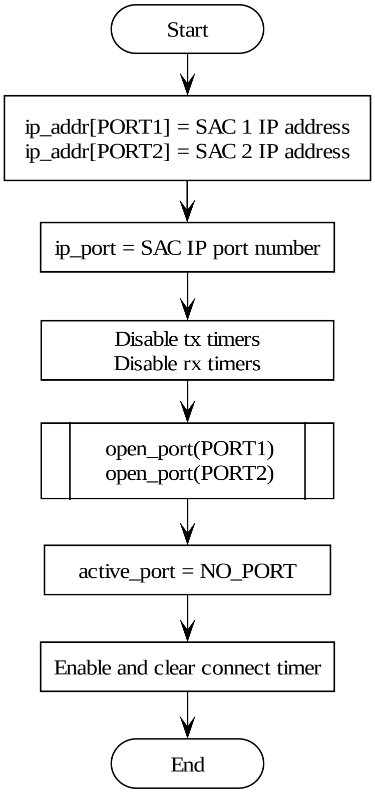
main()



5.2 initialize()

This part of your program should initialize all your variables, then attempt to open the TCP connections to the SAC computers as shown in the initialize flow chart below.

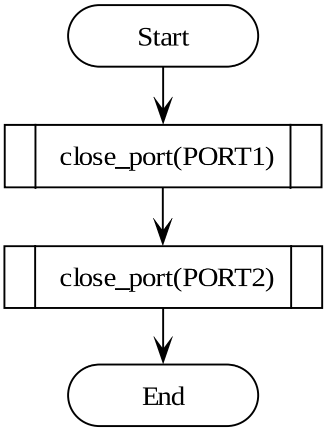
initialize()


Start
ip_addr[PORT1] = SAC 1 IP address
ip_addr[PORT2] = SAC 2 IP address
ip_port = SAC IP port number
Disable tx timers
Disable rx timers
open_port(PORT1)
open_port(PORT2)
active_port = NO_PORT
Enable and clear connect timer
End

5.3 cleanup()

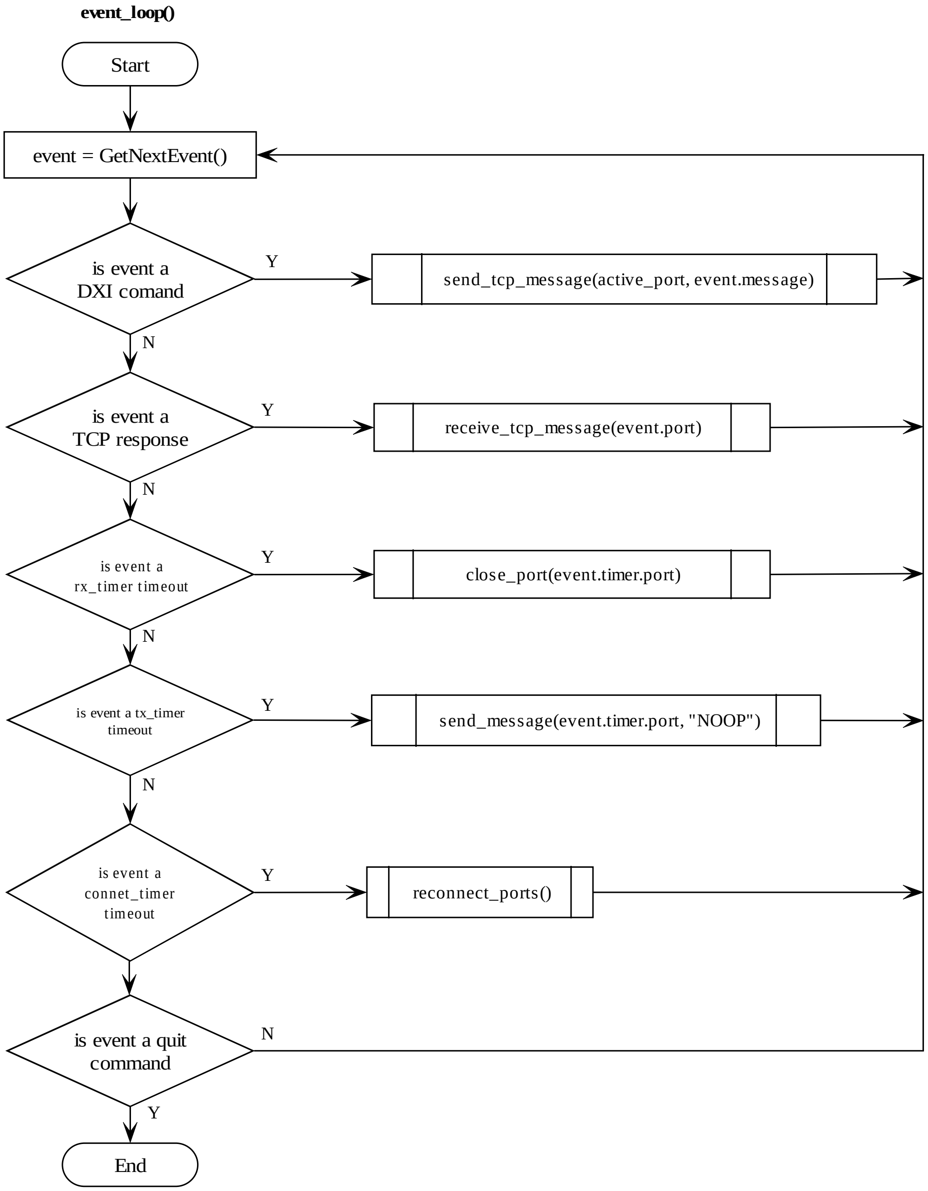
The cleanup simply closes the ports prior to quiting.

cleanup()



5.4 event_loop()

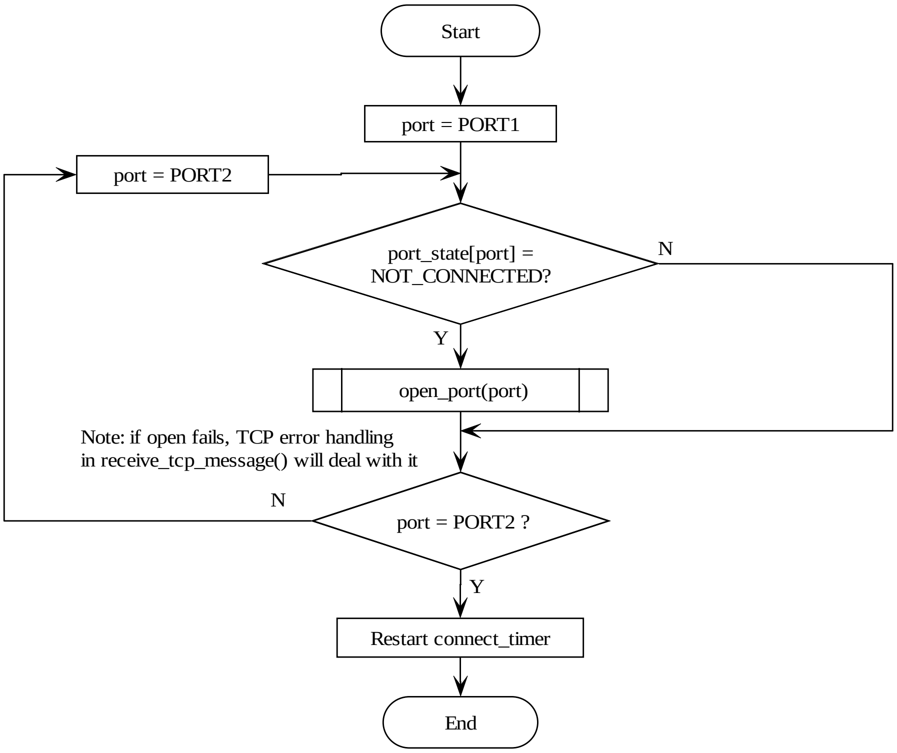
The main loop should perform the functions of the event_loop flow chart – wait for an event, then process that event appropriately. If a command comes from the user, send the command straight through to the DXI system. If a TCP message is received (from the DXI system), process the command as in the receive_tcp_message flowchart. If any rx timer times out, then disconnect that SAC computer. If any tx timer times out, then send a NOOP to the SAC computer and clear the tx timer. Finally, when the connect timer times out, process the commands in reconnect_ports to check the status of all SAC computer connections and try to re-establish any inactive connections.


event_loop()
Start
event = GetNextEvent()
is event a DXI comand
Y
send_tcp_message(active_port, event.message)
N
is event a TCP response
Y
receive_tcp_message(event.port)
N
is event a rx_timer timeout
Y
close_port(event.timer.port)
N
is event a tx_timer timeout
Y
send_message(event.timer.port, "NOOP")
N
is event a connet_timer timeout
Y
reconnect_ports()
is event a quit command
N
Y
End

5.5 reconnect_ports()

The code in reconnect_ports checks the status of all SAC computer connections and tries to re-establish any inactive connections

reconnect_ports()


Start
port = PORT1
port = PORT2
port_state[port] = NOT_CONNECTED?
N
Y
open_port(port)
Note: if open fails, TCP error handling in receive_tcp_message() will deal with it
N
port = PORT2 ?
Y
Restart connect_timer
End

5.6 receive_tcp_message()

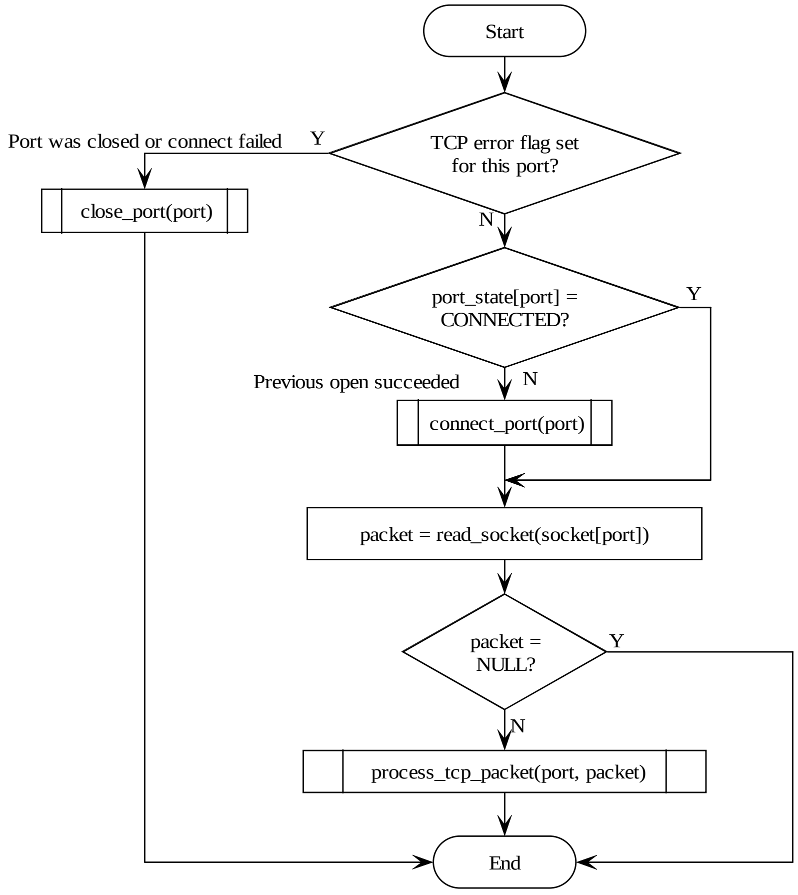
Whenever event_loop indicates that a TCP message is received, receive_tcp_message is executed.

If an error response is received (such as a connection failure, timeout, or disconnect notification from the SAC computer), handle the error appropriately, which likely involves calling the close_port routine. Not only does close_port close the TCP port, but it also handles switchover to a redundant SAC if appropriate.

If the TCP stack instead sends a connect notification (from an earlier non-blocking connection attempt), then set up the state appropriately and start up the rx timer for this port since it should now be receiving messages.

If an actual message packet is received from the SAC computer, then process the message (or messages) in that packet in process_tcp_packet.

receive_tcp_message(port)


Start
TCP error flag set for this port?
Y
Port was closed or connect failed
close_port(port)
N
port_state[port] = CONNECTED?
Y
N
Previous open succeeded
connect_port(port)
packet = read_socket(socket[port])
packet = NULL?
Y
N
process_tcp_packet(port, packet)
End

5.7 process_tcp_packet()

The code in process_tcp_packet takes the TCP packet and splits it into individual messages sent from the SAC computer.

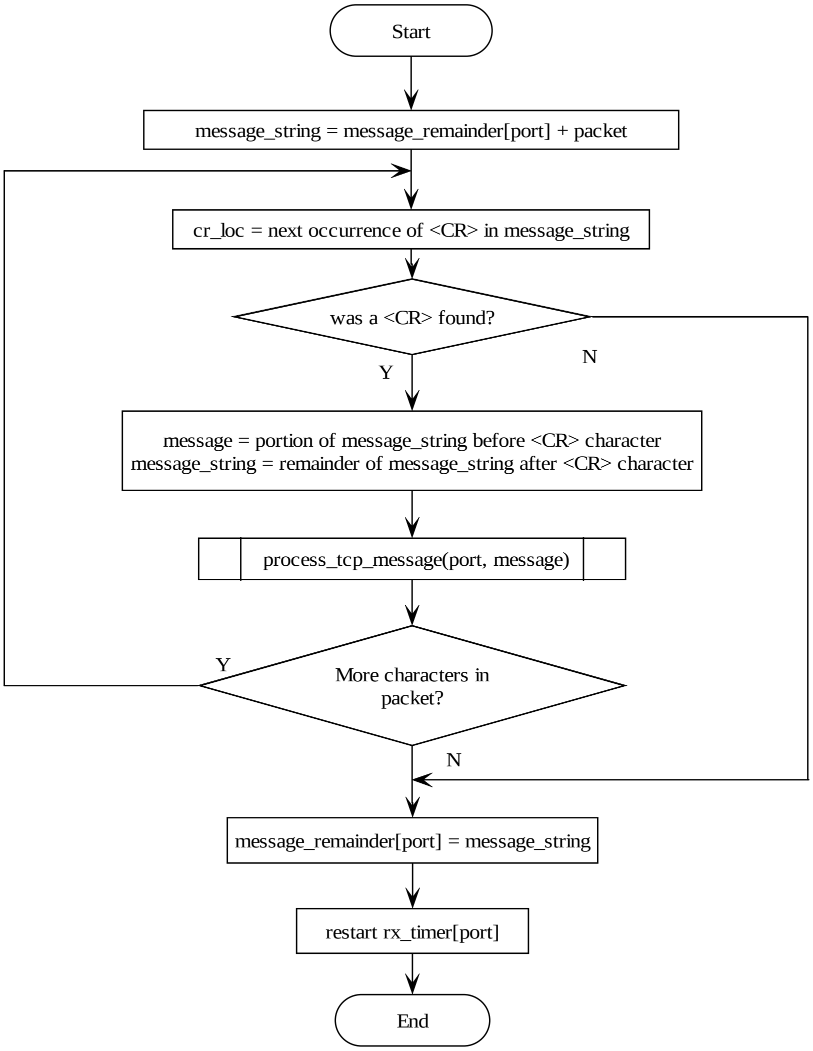
Because TCP/IP is a stream protocol, not a packet protocol, the underlying protocol can (and **will**) concatenate multiple messages into one packet to improve transport efficiency. To prevent message loss, the host client software **must** be able to handle multiple messages in a packet, and even a message split between two packets.

There are many ways to take this stream based input and parse it for <CR> delimited messages; the method in process_tcp_packet is one way, by processing all complete messages within this packet, then storing the remainder of the packet for concatenation with the next received packet. Other ways can involve checking the TCP buffer for a <CR> character and only emptying and processing the TCP buffer up to the <CR> character, or even emptying the TCP buffer one character at a time into a temporary buffer until a complete message is received.

In any case, once a message is received, process the message with process_tcp_message. This will determine if the active SAC port has changed, then process the message.

Repeat the above process until there are no more messages in the TCP packet. Clear the rx timer to indicate that there was activity from the SAC computer, then go back into the main event_loop loop waiting for another event.

process_tcp_packet(port, packet)


Start
message_string = message_remainder[port] + packet
cr_loc = next occurrence of <CR> in message_string
was a <CR> found?
N
Y
message = portion of message_string before <CR> character
message_string = remainder of message_string after <CR> character
process_tcp_message(port, message)
Y
More characters in packet?
N
message_remainder[port] = message_string
restart rx_timer[port]
End

5.8 process_tcp_message()

Once receive_tcp_packet has received a message, it is passed to process_tcp_message.

If the message is a NOOP (activity indicator), then clear the rx timer and exit – this message is only to indicate that the SAC is still operational and communicating.

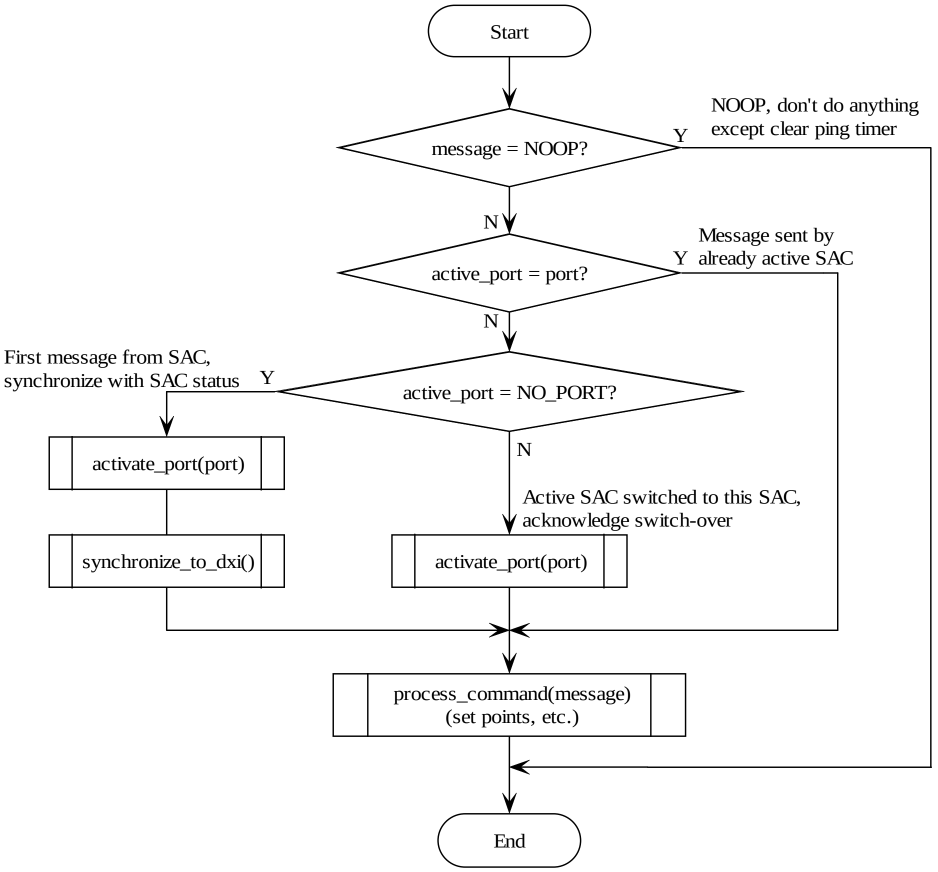
If the message is not a NOOP, then this message is coming from the active SAC computer (since only the active computer sends messages – the inactive SAC computer only sends NOOP messages to indicate it is “alive” but sends no other messages).

If the client determines that it doesn’t have a currently “active” SAC computer, then this is the first connection to the DXI exchange (I.E. a start-up condition, or communications are restored after both exchanges were disconnected). Normally, the first message you would see in this case is ACTV, but this technique shows a robust method that will detect this for any message. In this case, set the active port number to this port. To determine any call requests that were initiated or calls that were established with the stand-alone IMS masters while the communications were down, synchronize the connection and call request status between the host and SAC computer by clearing out all call request and connection bits, then issuing a synchronization command to the SAC computer. To do this, send a STAT <master> command for each master that this client host controls. This is done in synchronize-to-dxi. The SAC computer will then send a series of call connection and call request responses (exactly as if those connections or requests were initiated at the time the STAT message was sent). All types of master messages are sent at this time – active connections, call requests, disabled stations, hardware faults, etc.

However, if there was an active port, but the current port was not the last active port that received a message, this means that the SAC computers have performed a backup switch. Normally the first message you will see in this case is a PCSW message (if the SAC is configured to send this message), but this technique shows a robust method that will detect a switchover on any message. Update the host client's active port number so that any messages from the host client to the DXI system will be sent to the newly-activated primary SAC computer rather than the inactive one, then acknowledge the SAC change by sending an ACTV to the SAC. This is done in activate-port.

After any redundant switchover conditions handled, the message can then be processed. This will usually entail parsing the message and determining the end point devices (master number and station number) and setting your internal point list appropriately.

process_tcp_message(port, message)

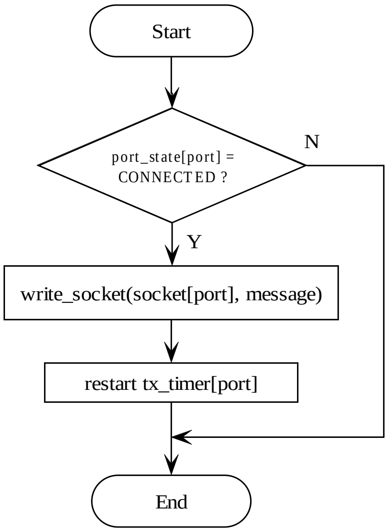


5.9 send_message()

Whenever a user action requires sending a command to the DXI system (such as connecting a master to a station), the operations in send_message are performed.

Simply check if there is an active SAC, then send the command to the SAC and (optionally) clear your tx timer.

send_message(port, message)


Start
port_state[port] = CONNECTED ?
N
Y
write_socket(socket[port], message)
restart tx_timer[port]
End

5.10 open_port()

The operations in open_port are performed whenever the host client attempts to connect to a SAC computer.

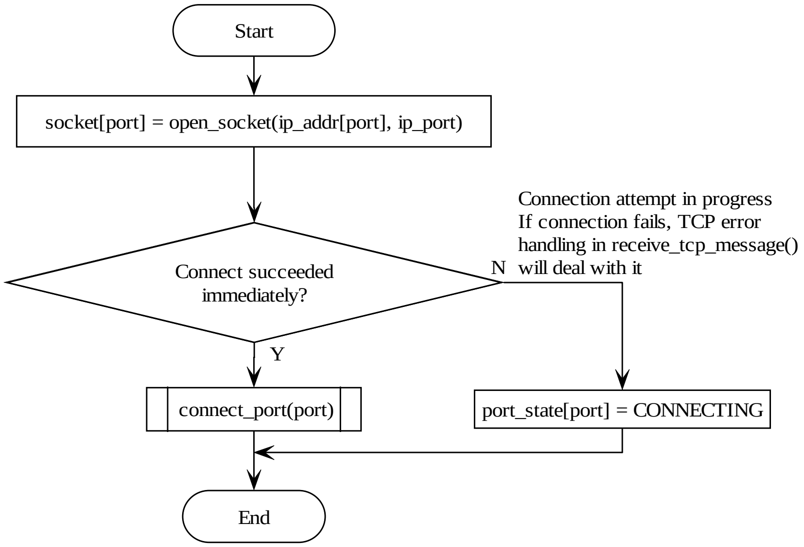
Assign the parameters for the SAC IP address and port number (from the SAC configuration), and make the connection non-blocking.

Then attempt a connection to the SAC.

Because the connection is non-blocking, the connection will likely not be completed immediately, so the connect will return a response that the connection is in progress. If this is the case, then set the port state to connecting.

However, if for some reason the connection had blocked, or if the connect succeeded immediately, then call connect_port to determine whether this SAC is active or not, and to initialize the necessary timers and other connection information.

open_port(port)


Start
socket[port] = open_socket(ip_addr[port], ip_port)
Connect succeeded immediately?
N
Connection attempt in progress
If connection fails, TCP error
handling in receive_tcp_message()
will deal with it
Y
connect_port(port)
port_state[port] = CONNECTING
End

5.11 connect_port()

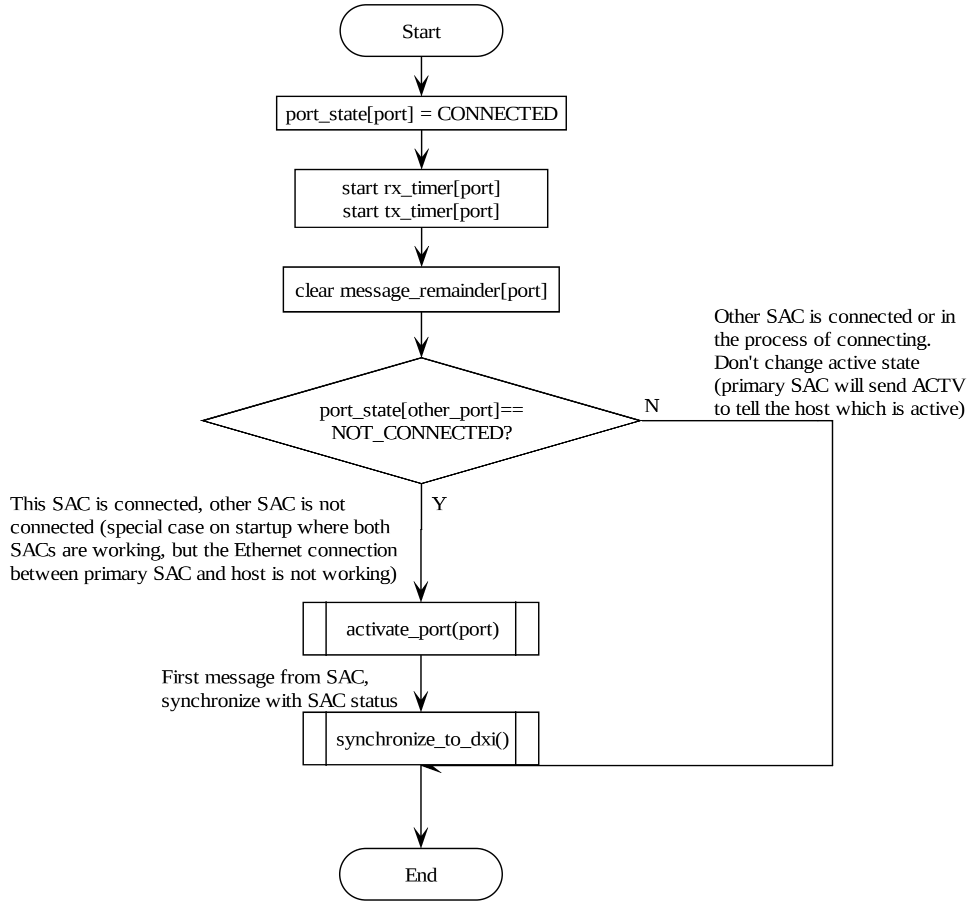
The operations in connect_port are performed whenever the host client has determined that a port in CONNECTING state has received a tcp response.

Set the state of this port to CONNECTED, and enable and clear the rx and tx timers.

If another SAC is already connected (or is in the process of connecting), then simply exit after setting the above state and clearing the rx and tx timers. In this case, the first SAC computer to send the ACTV command will be the active computer.

If the other SAC is not connected, then make this port active, by setting the active_port variable and sending the ACTV command. To synchronize the state of the DXI with the state of the host client, send a STAT <master> command for each master that this client host controls. The SAC computer will then send a series of call connection and call request responses (exactly as if those connections or requests were initiated at the time the STAT message was sent). This is described in more detail in the process_tcp_message description above.

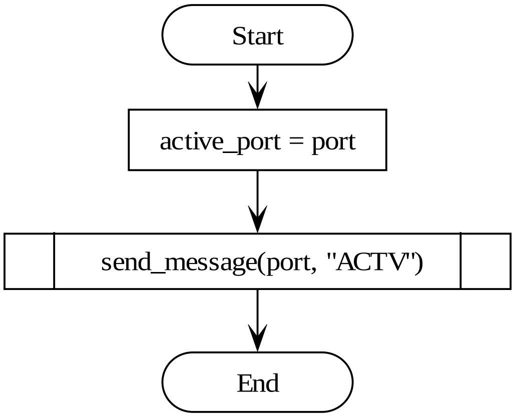
connect_port(port)


Start
port_state[port] = CONNECTED
start rx_timer[port]
start tx_timer[port]
clear message_remainder[port]
port_state[other_port]== NOT_CONNECTED?
N
Other SAC is connected or in the process of connecting. Don't change active state (primary SAC will send ACTV to tell the host which is active)
Y
This SAC is connected, other SAC is not connected (special case on startup where both SACs are working, but the Ethernet connection between primary SAC and host is not working)
activate_port(port)
First message from SAC, synchronize with SAC status
synchronize_to_dxi()
End

5.12 activate_port()

When a connection is made to the active SAC (usually when the SAC sends “ACTV” as the first message sent to the host client), activate_port is called. This sets the active port, and responds to the SAC with an “ACTV” acknowledgement.

activate_port(port)



5.13 close_port()

The functions in close_port() are called whenever a connection is closed.

This closes the port, and switches to the backup SAC if applicable.

The port state is set to not connected, and the rx and tx timers are disabled.

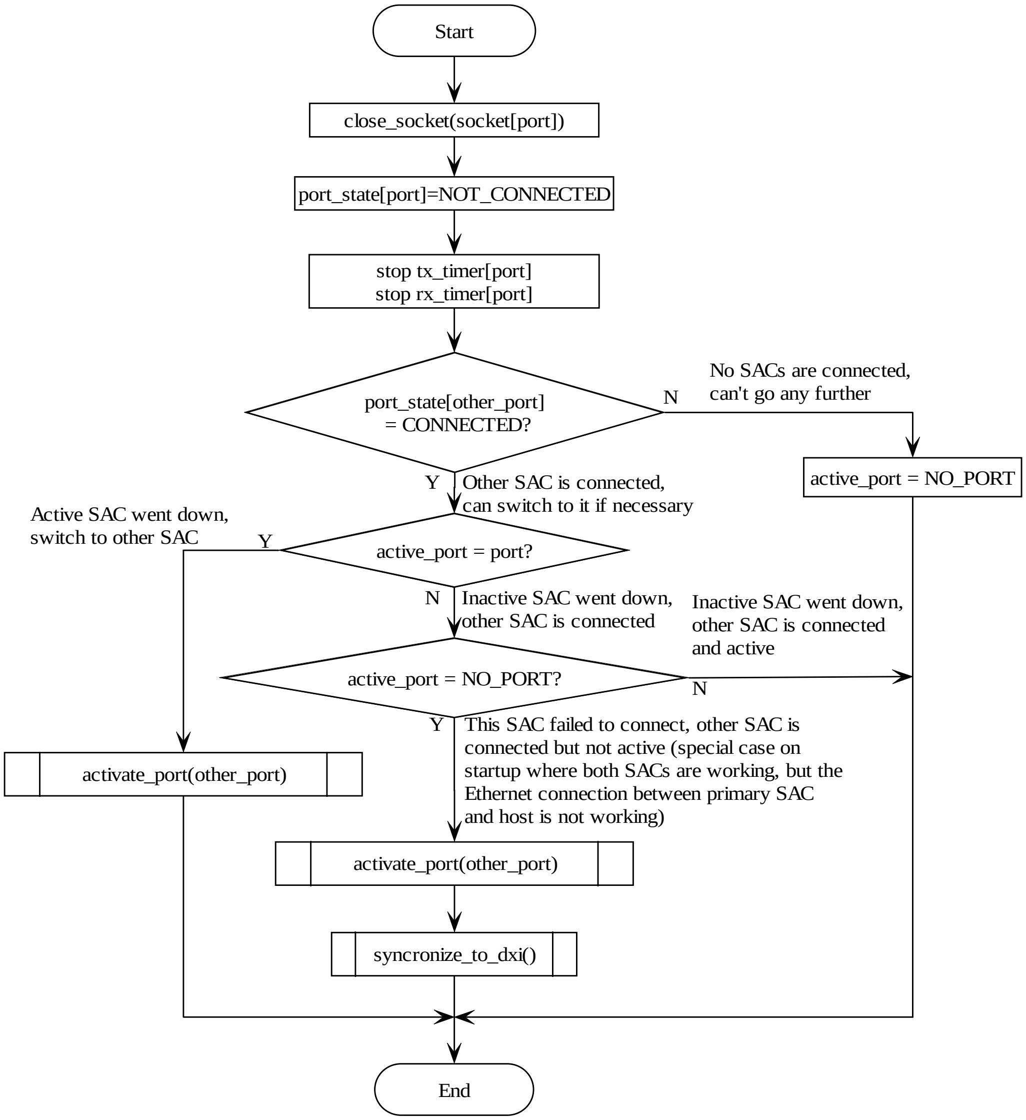
If the other SAC is not connected, then there are no SACs left to make active, so set the active port to none and exit.

If the other SAC is connected, and this SAC was active (before close_port was called), then perform a simple redundant SAC switch to the other SAC by sending ACTV to the other SAC and optionally clearing the tx timer.

If the other SAC is connected, but not active, and this SAC was not active either, then no SACs were currently active (this is usually a start-up condition where the host is waiting for a SAC computer to indicate that it is active). In this case, since this SAC is closing down, and the other SAC hasn't initialized yet, make the other SAC the active one (by sending ACTV) then synchronize its status by sending STAT <master> to all the masters on the other SAC, optionally clearing the tx timer.

Finally, if the other SAC is connected and active, do nothing. This case is where the inactive SAC is rebooted or shut down.

close_port(port)



No SACs are connected, can't go any further

Other SAC is connected, can switch to it if necessary

Active SAC went down, switch to other SAC

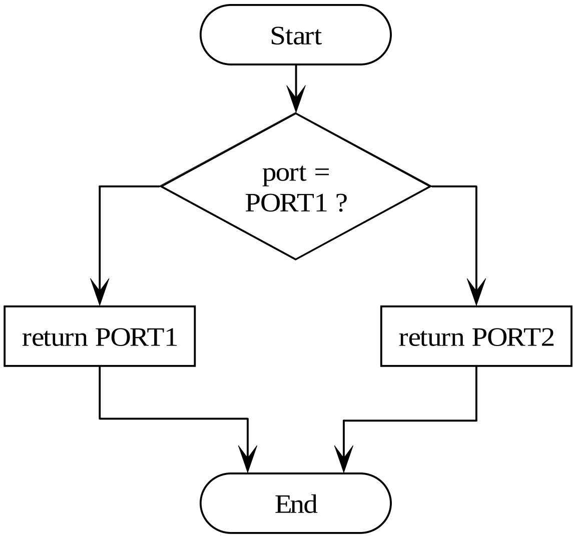
Inactive SAC went down, other SAC is connected

Inactive SAC went down, other SAC is connected and active

This SAC failed to connect, other SAC is connected but not active (special case on startup where both SACs are working, but the Ethernet connection between primary SAC and host is not working)

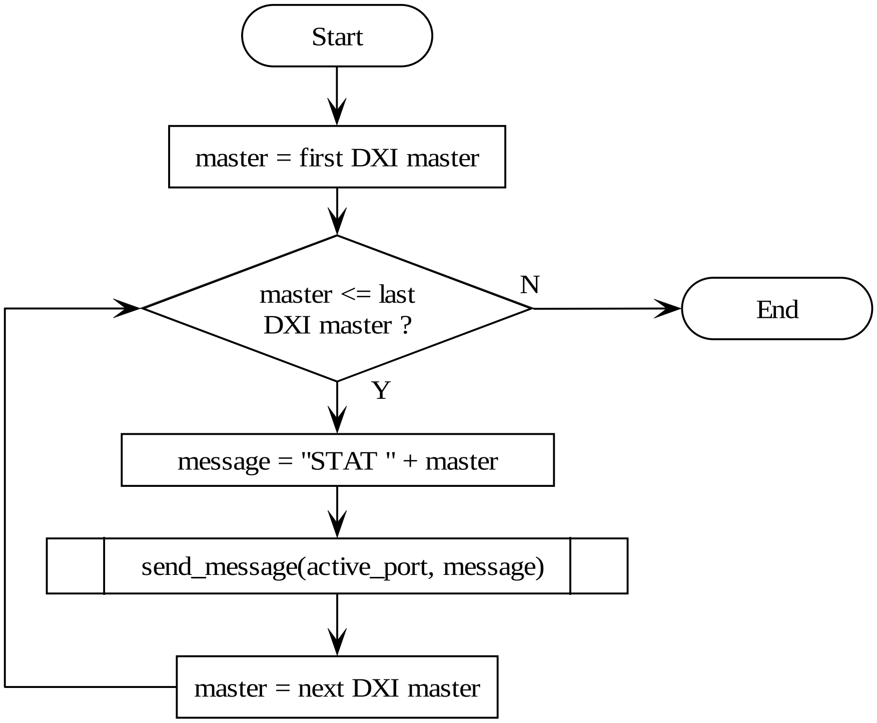
5.14 other_port()

other_port(port)



5.15 synchronize_to_dxi()

synchronize_to_dxi()


Start
master = first DXI master
master <= last DXI master ?
N
End
Y
message = "STAT " + master
send_message(active_port, message)
master = next DXI master