621-Netlinx Client IP Communications

Symptoms

An example is needed of how to handle client IP connections within NetLinx code.

Resolution

See the commented example below for a good example of how to handle client IP communications:

```
(***********************************************************)
(* DEVICE NUMBER DEFINITIONS GO BELOW *)
(***********************************************************)
DEFINE_DEVICE dvIP = 0:2:0; // 0:2:0 is the first available IP port, 0:3:0 is the next, etc.
dvTP = 128:1:0; // The touch panel
(***********************************************************)
(* CONSTANT DEFINITIONS GO BELOW *)
(***********************************************************)
DEFINE_CONSTANT
// Booleans
TRUE = 1;
FALSE = 0;
// TCP/IP constants
TCP = 1;
UDP = 2;
// Time for wait before reopening connection.
RETRY_TIME = 300; // 30 seconds
(***********************************************************)
(* VARIABLE DEFINITIONS GO BELOW *)
(***********************************************************)
DEFINE_VARIABLE
// IP address of server = 192.168.1.100 in this example
```

```
CONSTANT char cServerAddress[13] = '192.168.1.100';
// Using server port 23 (Telnet) in this example.
CONSTANT LONG lServerPort = 23;
INTEGER bClientOnline; // Flag: TRUE when client is connected
INTEGER bClientKeepOpen; // Flag: keep the client open at all times
(***********************************************************)
(* STARTUP CODE GOES BELOW *)
(***********************************************************)
DEFINE_START
// Attempt to open a TCP connection to the server.
IP_CLIENT_OPEN (dvIP.port,cServerAddress,lServerPort,TCP);
(***********************************************************)
(* THE EVENTS GOES BELOW *)
(***********************************************************)
DEFINE_EVENT DATA_EVENT [dvIP]
 {
// Online handler runs when a successful connection is made.
ONLINE:
{
// We have communication. Can send strings to IP device now.
bClientOnline = TRUE;
}
// Offline handler runs when connection is dropped/closed.
OFFLINE:
{
// NOTE: Certain protocols (such as HTTP) drop the connection
// after sending a response to a request. For those protocols,
// this is a better place to parse the buffer than in the STRING
// handler. There will be a complete reply in the buffer.
bClientOnline = FALSE;
```

```
// Attempt to reestablish communications, if desired.
IF (bClientKeepOpen)
WAIT RETRY_TIME
IP_CLIENT_OPEN (dvIP.port,cServerAddress,lServerPort,TCP);
}
// Data event runs when we get messages from the server.
STRING:
{
// Buffer parsing goes here.
// May need to ACK initial message from server to establish connection.
// NOTE: Even if parsing responses in OFFLINE, you should still have
// an empty STRING handler to prevent mainline from running needlessly
// every time a string comes in.
}
// Onerror runs when attempt to connect fails.
ONERROR:
{
SWITCH (DATA.NUMBER)
{
// No need to reopen socket in response to following two errors.
CASE 9: // Socket closed in response to IP_CLIENT_CLOSE.
CASE 17: // String was sent to a closed socket.
{
}
DEFAULT: // All other errors. May want to retry connection.
{
IF (bClientKeepOpen)
WAIT RETRY_TIME
IP_CLIENT_OPEN (dvIP.port,cServerAddress,lServerPort,TCP);
}
```

```
}
}
}
BUTTON_EVENT [dvTP, 1] // Open client.
{
PUSH:
{
IF (!bClientOnline)
IP_CLIENT_OPEN (dvIP.port,cServerAddress,lServerPort,TCP);
bClientKeepOpen = TRUE;
}
}
BUTTON_EVENT [dvTP, 2] // Close client (and keep it closed).
{
PUSH:
{
IF (bClientOnline)
IP_CLIENT_CLOSE (dvIP.port);
bClientKeepOpen=FALSE;
 }
}
```