NetLinx Module Interface Specification

for a

Pioneer
Plasma Display

TABLE OF CONTENTS

Revision History

Date	Initials	Comments
1-27-07	DC	Initial release
1-4-12	DC	Added vol control, updated for newer units

Introduction

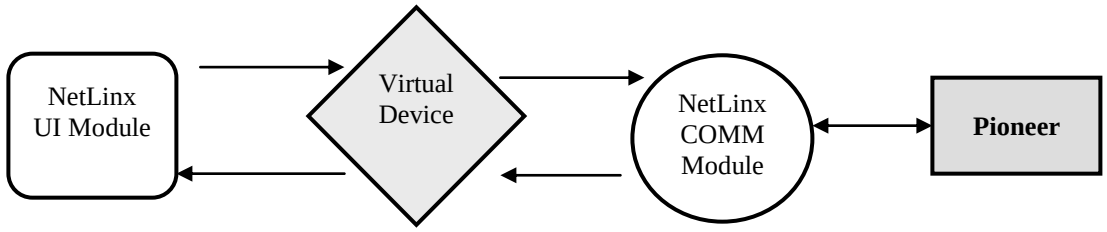
This is a reference manual to describe the interface provided between an AMX NetLinx system and a Pioneer Plasma Display. This module works with the PDP-502MX. The required communication settings are a baud rate of 4800, 8 data bits, 1 stop bit, no parity, and handshaking on.

Overview

The COMM module translates between the standard interface described below and the **Pioneer** serial protocol. It parses the buffer for responses from the PLASMA, sends strings to control the PLASMA, and receives commands from the UI module or telnet sessions.

A User Interface (UI) module is also provided. This module uses the standard interface described below and parses the string responses for feedback.

The following diagram gives a graphical view of the interface between the interface code and the NetLinx module.



Implementation

To interface to the Pioneer module, the programmer must perform the following steps:

1. Define the device ID for the Pioneer that will be controlled.
2. Define the virtual device ID that the Pioneer COMM module will use to communicate with the main program and User Interface. NetLinx virtual devices start with device number 33001.
3. If a touch panel interface is desired, a touch panel file Pioneer,REV0.tpd and module Pioneer_Plasma_UI.axs) have been created for testing.
4. The NetLinx Pioneer module must be included in the program with a DEFINE_MODULE command. This command starts execution of the module and passes in the following key information: the device ID of the PLASMA to be controlled, and the virtual device ID for communicating to the main program.

An example of how to do this is shown below.

```
DEFINE_DEVICE
   dvPioneer          = 5001:1:0     // The Pioneer connected to the NetLinx on 1st RS-232 port
   dvPioneer TPl      = 10001:1:0    // The touch panel used for output

   vdvPioneer         = 33001:1:0    // The virtual device use for communication between the
                                     // Comm module interface and User_Interface (UI) module interface

DEFINE_VARIABLE
//Define arrays of button channels used on your own touch panel
integer nTP_BUTTONS[]={1,2,3,4,5,6,7}

DEFINE_START      // Place define_module calls to the very end of the define_start section.
// Comm module
DEFINE_MODULE 'Pioneer_Comm' mdlPioneer_APP(dvPioneer, vdvPioneer)

// Touch panel module
DEFINE_MODULE 'Pioneer_UI'' mdlPioneer_APP(vdvPioneer, dvTPArray, nTPButtons,
nTuneButtons)
```

Upon initialization the AMX Comm module will communicate with the Pioneer and information will be exchanged.

Channels

The channels supported by the COMM module are listed below. These channels are associated with the virtual device(s) and are independent of the channels associated with the touch panel device.

Note: An '*' indicates an extension to the standard API.

Channel	Description
251	This channel is used for feedback only. ON: Device is online. OFF: Device is offline.
255	This channel is used for feedback only. ON: POWER is ON. OFF: POWER is OFF

Levels

The levels supported by the COMM module are listed below. These levels are associated with the virtual device(s) and are independent of the channels associated with the touch panel device.

Level	Description
1	Level for volume. Range is 0-60.

Command Interface

The UI module controls the PLASMA via command events (NetLinx command *send_command*) sent to the COMM module. The commands supported by the COMM module are listed below.

NOTE: AMX has not defined a standard API for PLASMA devices.

Command	Description
DEBUG=<state>	Set the debug state <state> : 0 OFF 1 ON DEBUG=1
DEBUG?	Request the debug state. DEBUG?
INPUT=<value>	Selects the input source. <value> : 1-8 (8 is tv tuner A) INPUT=3
INPUT?	Request the current input setting. INPUT?
MUTE=<value>	Mutes audio <value> : 0 – Unmuted 1 – Muted MUTE=1
MUTE?	Request the current mute state MUTE?
PASSTHRU=<string>	Allows user the capability of sending commands directly to the Pioneer without processing by the NetLinx module. User must be aware of the protocol implemented by the unit to use this command. This gives the user access to features which may not be directly supported by the module. See "Adding Functions to Modules" section at the end of this document for more information. The COMM module **automatically adds the required checksum to the command string.** <string> : string to send to unit PASSTHRU=RVER
POWER=<state>	Sets the power state. <state> : 0 = off 1 = on POWER=0

POWER?	Request the current power setting. POWER?
VERSION?	Request the current version number of the NetLinx module. VERSION?
XCH=<value>	Set the current tuner station. Feedback for this command is faked. The feedback will return the last stored station number. The valid range depends on the currently selected tuner band. <value> : 1..135 (ANALOG CABLE) XCH=030 (ANALOG_CABLE)
XCH?	Request the current channel XCH?

Table 1 – Send Command Definitions

String Feedback

The NetLinx COMM module provides feedback to the User Interface module for **Pioneer** changes via string events. The strings supported are listed below.

String	Description
INPUT=<value>	Reports the input source. <value> : 1-9 (9 is tv tuner) INPUT=3
POWER=<value>	Reports the power status. <value> : 0 power is off 1 power is on
VERSION=<version>	Reports the current version number of the NetLinx module. <value> : current version number in xx.yy format VERSION=1.06
XCH=<value>	Set the current tuner station. Feedback for this command is faked. The feedback will return the last stored station number. The valid range depends on the currently selected tuner band. <value> : 1..135 (ANALOG CABLE) XCH=030 (ANALOG_CABLE)

Table 2 - String Feedback Definitions

Device Notes

Several of the Pioneer commands have been omitted and may be executed using the PASSTHRU command.

There does not appear to be any response to normal commands back from the Plasma. (No OK or ERR return). Therefore, the program assumes that a valid command has been executed.

It is assumed that the DEVICE-ID of the Plasma is set to 01. This version of the module only provides support for one Plasma connected to the specific serial port. (No chaining).

When using the PASSTHRU command, the 3-character command is the only thing that the main program needs to send. The STX, ID=01, and ETX fields are supplied by the COMM module.

Programming Notes

The COMM module imposes synchronous operation i.e. the COMM will not send a command to the Pioneer until it has gotten a response or a timeout from the previous command.

A queue is implemented so that the commands from can be stacked. Commands are pulled off the queue waiting 3 seconds between commands.

The Request commands return the state that is currently stored in the COMM module. They do NOT send a new Request to the device.

Adding Functions to Modules

Commands to the device

This module supplies a mechanism to allow additional device features to be added to software using the module. This is the PASSTHRU command, which allows protocol strings to be passed through the module. The device-specific protocol must be known in order to use this feature.

As an example, suppose that a module for a projector has not implemented the 'white balance adjustment' feature. The command that the projector protocol requires is 03H, 10H, 05H, 14H, followed by a checksum. The documentation for the PASSTHRU command specifies that the module will automatically generate the checksum. In this case, the following string should be sent from the UI code to implement 'white balance adjustment'.

```
send_command vdvDevice,'"PASSTHRU=',$03,$10,$05,$14"
```

The reason to use PASSTHRU instead of sending a protocol string directly to the device port is that the device may require command queuing, calculation of checksums, or other internal processing, which would not be done if the string was sent directly. Because of this, it is best to filter all communication TO the device through the module. (The module documentation will indicate any processing that will be automatically done to the PASSTHRU string like checksum calculation.)

Additional Feedback from the device

The module documentation indicates what feedback is provided. If additional feedback is required, a CREATE_BUFFER for the device must be implemented in the user code to process the strings from the device manually. Note that the module will still be processing the response strings independently and sending the interpreted feedback up to the user code.