PROFESSIONAL CONTROL SOLUTIONS, LLC

NetLinx Module Interface Specification

for a

Infocus MONDOPAD INF7021 (should work with INF5520)

TABLE OF CONTENTS

Revision History

Date	Initials	Comments
3/19/2014	DC	Initial release

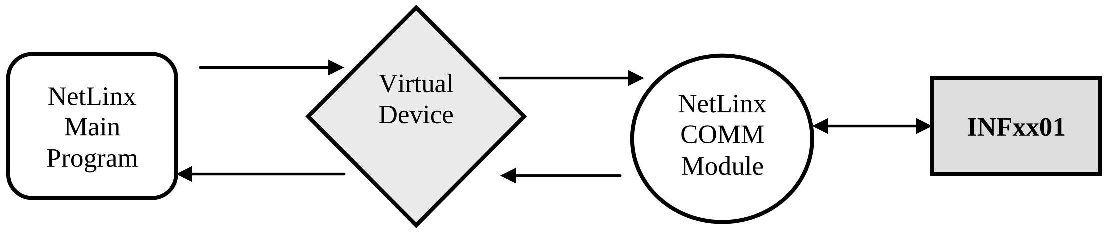
Introduction

This is a reference manual to describe the interface provided between an AMX NetLinx system and a Infocus MONDOPAD display. The required communication settings are a baud rate of 9600, 8 data bits, 1 stop bit, no parity, and handshaking off.   Crossover cable with 2, 3, and 5.

Overview

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

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



Implementation

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

1. Define the device ID for the Infocus MONDOPAD that will be controlled.
2. Define the virtual device ID that the Infocus MONDOPAD COMM module will use to communicate with the main program and User Interface. NetLinx virtual devices start with device number 33001.
3. The NetLinx Infocus MONDOPAD 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 MONDOPAD 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

dvInfocus_MONDOPAD	= 5001:1:0 // The **Infocus MONDOPAD** connected to the NetLinx on 1st RS-232 port
vdvInfocus_MONDOPAD	= 33001:1:0 // The virtual device use for communication between the Comm module interface and User Interface program.

DEFINE_START // Place define_module calls to the very end of the define_start section.
// Comm module
DEFINE_MODULE 'Infocus_MONDOPAD_Comm'
mdlInfocu_MONDOPAD_APP(vdvInfocus_MONDOPAD, dvInfocus_MONDOPAD)

Upon initialization the AMX Comm module will communicate with the Infocus MONDOPAD 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 0-100.

Command Interface

The UI module controls the MONDOPAD 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 MONDOPAD devices.

Command	Description

`AUDIO=<value>`	`Sets the current audio input.`
	
	`<value>: 1 = AUDIO 1`
	`         2 = AUDIO 2`
	`         3 = HDMI 1`
	`         4 = HDMI 2`
	`         5 = PC`
	`         6 = PC SPEAKER`
	
	`AUDIO=2`

`AUDIO?`	`Request for current audio input.`
	
	`AUDIO?`

`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 = HDMI-1`
	`          2 = HDMI-2`
	`          3 = VGA`
	`          4 = Component`
	`          5 = AV`
	`          6 = DP1`
	
	`INPUT=3`

`INPUT?`	`Request the current input setting.`
	
	`INPUT?`

`MUTE=<value>`	`Selects the mute state.`
	
	`<value> : 0 = UN-MUTE`
	`          1 = MUTE`

MUTE?	Request the current mute setting. MUTE?
PASSTHRU=<string>	Allows user the capability of sending commands directly to the Infocus MONDOPAD 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?

Table 1 – Send Command Definitions

String Feedback

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

String	Description

`AUDIO=<value>`	`Reports the current audio input.`
	
	`<value>: 1 = AUDIO 1`
	`2 = AUDIO 2`
	`3 = HDMI 1`
	`4 = HDMI 2`
	`5 = PC`
	`6 = PC SPEAKER`
	
	`AUDIO=2`

`INPUT=<value>`	`Reports the input source.`
	
	`<value> : 1 = HDMI-1`
	`2 = HDMI-2`
	`3 = VGA`
	`4 = Component`
	`5 = AV`
	`6 = DP1`
	
	`INPUT=3`

`MUTE=<value>`	`Reports the mute state.`
	
	`<value> : 0 = UN-MUTE`
	`1 = MUTE`

`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`

Table 2 - String Feedback Definitions

Device Notes

Commands implemented by this interface include those that are commonly used and shared between various MONDOPAD devices. Several of the Infocus MONDOPAD commands have been omitted and may be executed using the PASSTHRU command.

A periodic poll will return the power state and the total number of hours used by the device. Should the MONDOPAD shift into sleep mode, it will be automatically shifted back into power-on mode.

Programming Notes

The COMM module imposes synchronous operation i.e. the COMM will not send a command to the Infocus MONDOPAD 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 (1) when a reply is received or, (2) after the reply timer times out (200 milliseconds). The device is polled every 10 seconds to return the power state and the input selected.

The Request commands return the state that is currently stored in the COMM module. They do NOT send a new Request to the device. The COMM module states are updated either by the periodic poll or by the enforced feedback. When a state changes, the UI is automatically informed.

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.