NetLinx Module Interface Specification

for a

Data Video
MP-6000 DVR

Professional Control Solutions LLC • 3804 Parchment Circle • Henrico, VA • 23233-7034
804-677-6794 • info@professionalcontrolsolutions.com
www.professionalcontrolsolutions.com

TABLE OF CONTENTS

LIST OF TABLES

Revision History

Date	Initials	Comments
07/19/2010	DC	Initial Release v1.0

Overview

The AMX module communicates to the Data Video MP-6000 at 19200, N, 8, 1; without hardware handshaking. The communication cable is a db-9 female with the following connections:

Denon	NetLinx
2 Rx	2 Rx
3 Tx	3 Tx
5 Gnd	5 Gnd (or 1 on NXI Phoenix)

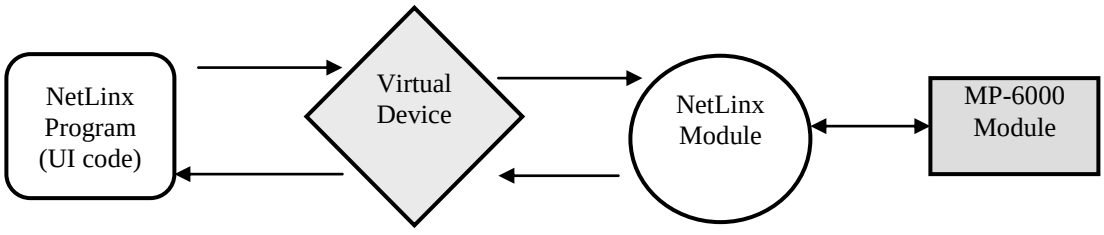
The communication module is instantiated/called by adding the following line of code:

```
DEFINE_MODULE 'MP-6000_Comm' comm._code(virtual_name, real_name)
```

This document will define the common NetLinx module interface for a DATA VIDEO MP-6000 player. Obviously there will always be features one system supports that another does not (or cannot). The model is not designed to be static. It is designed to be ever-growing while always supporting backwards compatibility. It is up to the programmer of each module to adhere to the model and to find the best way to fit the protocol of a piece of equipment to the model.

For features that are not part of the model the programmer may support additional commands that extend beyond the model to support those features. This is desirable because manufacturers want to expose the features of their system that make them unique and differentiate them from their competitors. Exposing control for those features should be done even if they are not part of the model. In this module, the 'PASSTHRU=' command provides this functionality.

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



Channels

The UI module controls the disc device via channel events (NetLinx commands *pulse, on, and off*) sent to the COMM module. 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.

Channel	Description
1	PULSE: Play
2	PULSE: Stop
3	PULSE: Pause
4	PULSE: Next
5	PULSE: Previous
6	PULSE: Fast Forward
7	PULSE: Fast Reverse
8	PULSE: Record
9	PULSE: Cycle Power
10	PULSE: 0 Digit Button
11	PULSE: 1 Digit Button
12	PULSE: 2 Digit Button
13	PULSE: 3 Digit Button
14	PULSE: 4 Digit Button
15	PULSE: 5 Digit Button
16	PULSE: 6 Digit Button
17	PULSE: 7 Digit Button
18	PULSE: 8 Digit Button
19	PULSE: 9 Digit Button
20	PULSE: +10 Button
21	PULSE: Enter Button
44	PULSE: Menu Button
45	PULSE: Cursor Up
46	PULSE: Cursor Down
47	PULSE: Cursor Left
48	PULSE: Cursor Right
49	PULSE: Select
66	PULSE: Setup Button
80	PULSE: Clear Button
99	PULSE: Display Button
100	PULSE: Subtitle Button
104	PULSE: Return Button
112	PULSE: AB Repeat Button
115	PULSE: Top Menu Button
116	PULSE: Zoom Button
125	PULSE: Cycle the repeat state
241	ON: Play is active – provides feedback only
242	ON: Stop is active – provides feedback only
243	ON: Pause/Record Pause is active – provides feedback only
244	ON: Record is active – provides feedback only
251	ON: Device is Online – used for feedback only OFF: Device is not Online
252	ON: Data is Initialized – use for feedback only

	OFF: Data is not Initialized
300	PULSE: Finalize
301	PULSE: Erase Disc (RW)
302	PULSE: Make Compatible (RW)

Command Interface

The interface code will control the DATA VIDEO MP-6000 player via command events (NetLinx command *send_command).* These commands will be sent to the module to affect control. Below are the commands supported.

Command	Description
DEBUG=<state>	Set the state of the debugging flag. <state>: 0 = off 1 = on 'DEBUG=1'
PASSTHRU=<string>	Allows user the capability of sending commands directly to whatever unit is attached 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. The communication does not add automatically any characters to the passthru string. For more detail please read the Adding Functions to Modules section at the end of this document. <string> : string to send to unit PASSTHRU=THIS IS A COMMAND PASSTHRU=RESET
VERSION?	Query for the current version number of the NetLinx module. VERSION?

Table 1 – Send Command Definitions

String Feedback

The NetLinx module will provide feedback to the interface code for DATA VIDEO MP-6000 player changes via string events. Below are the strings supported.

String	Description
DEBUG=<state>	State of the debug flag. Non-solicited feedback. <state>: 0 = off 1 = on 'DEBUG=1'
ERROR=<cmd>:<arg>	Error occurred. <cmd>: command that caused error <arg>: associated faulty argument to command (@#$jvdf8*) AUDIO: <arg> ANGLE: <arg> CURSOR: <arg> NUMPAD: <arg> SCAN: <arg> STEP: <arg> TRANSPORT: <arg> TRAY: <arg> ERROR=TRAY:BOLOGNA
VERSION=<value>	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

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_string 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.