NetLinx Module Interface Specification

for the

Vaddio ProductionView HD MV

TABLE OF CONTENTS

Revision History

Date	Initials	Comments
04/17/14	DC	Initial release v1.0

Introduction

This is a reference manual for use by anyone wanting to interface between an AMX NetLinx system and an Vaddio ProductionView Video Camera controller. Communication takes place directly over the built in communications port. The required communication settings are a baud rate of 9600, 8 data bits, 1 stop bit, no parity, and no handshaking**.** The cable for this device is FG#10-1784.

Scope

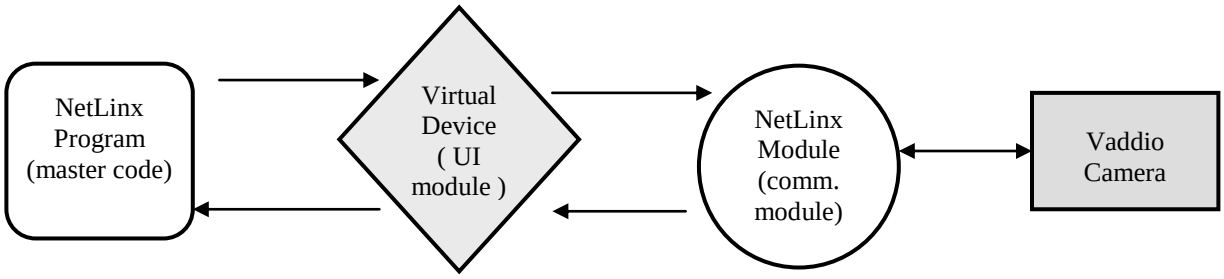
The intent in creating NetLinx modules is to make the interface as generic as possible so all interfaces from NetLinx to cameras will share the same protocol. The Vaddio ProductionViewdoes not support all of the features of this protocol, and in some cases extended commands have been added for unique PRODUCTIONVIEW features.

Overview

The PRODUCTIONVIEW may be controlled using the "Vaddio_ProductionView_Comm.tko" module. . The module implements the actual PRODUCTIONVIEW protocol for communicating to the unit but exposes a more simplified, NetLinx-friendly protocol to the programmer.

The programmer will interface to the NetLinx module through a virtual device. This virtual device is defined by the programmer and is used to control the PRODUCTIONVIEW and receive status updates from the
PRODUCTIONVIEW.

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



Implementation

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

1. Define the device ID for the camera that will be controlled.
2. Define the virtual device ID that the communication module will use to communicate with the main program and User Interface. NetLinx virtual devices start with device number 33001.
3. The NetLinx **VADDIO_PRODUCTIONVIEW_COMM** 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 camera 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
   dvPV_Device        = 5001:1:0    // The ProductionView connected to the NetLinx on 1st RS-232 port
   dvTouchPanel       = 10001:1:0  // The touch panel used for output.

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

DEFINE_START      // Place define_module calls to the very end of the define_start section.
// Comm module
DEFINE_MODULE 'Device_Comm' Comm1(vdvDevice, dvDevice)
```

Upon initialization the Comm module will communicate with the **PRODUCTIONVIEW** and information will be exchanged.

Channels

The main program controls the camera 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.

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

Channel	Description
132	ON: Ramp Tilt Up – provides feedback also OFF: Stop Tilt Ramping
133	ON: Ramp Tilt Down – provides feedback also OFF: Stop Tilt Ramping
134	ON: Ramp Pan Left – provides feedback also OFF: Stop Pan Ramping
135	ON: Ramp Pan Right – provides feedback also OFF: Stop Pan Ramping
158	ON: Ramp Zoom Out – provides feedback also

	OFF: Stop Zoom Ramping
159	ON: Ramp Zoom In – provides feedback also OFF: Stop Zoom Ramping
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
255	ON: Set power on – used for feedback also OFF: Set power off
*301	ON: Set video mute on. (Glass frosted) OFF: Set video mute off. (Glass clear)

Table 1 - Virtual Device Channel Events

Levels

The UI module controls the camera via level events (NetLinx command *send_level*) sent to the COMM module. The levels supported by the COMM module are listed below. These levels are associated with the virtual device(s) and are independent of the levels associated with the touch panel device.

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

Level	Description
*1	Zoom Speed Level (range 0-7)
*2	Pan Speed Level (range 1-24)
*3	Tilt Speed Level (range 1-20)

Table 2 - Virtual Device Level Events

Command Interface

The programmer controls the PRODUCTIONVIEW with the NetLinx command *send_command.* Below are the camera commands showing which are supported, with implementation notes where needed.
* Denotes commands beyond the generic camera API

Command	Description
DEBUG=<val>	Turns on/off the debug feature. <val>: 0 = off 1 = on DEBUG=1
DEBUG?	Query for the debug state. DEBUG?
CAMERAPRESET=<preset>	Recalls a preset. <preset>: 1..6 = preset number CAMERAPRESET=3
CAMERAPRESETSAVE=<preset>	Saves a preset. <preset>: 1..6 = preset number CAMERAPRESETSAVE=3
PASSTHRU=<exact protocol>	Allows user the capability of sending commands directly to whatever unit is attached without processing by the NetLinx modules. User must be aware of the exact protocol implemented by the unit to use this command. This gives the user access to features that may not be directly supported by the modules. The communication module does not add any characters to the passthru string. For more detail see "Adding Functions to Modules" section at the end of this document. USAGE: From Telnet: Connect to the master and at the prompt type SEND_C <virtual dev number>,"'PASSTHRU=',<exact protocol>" Example: SEND_C 33001,"'PASSTHRU=',$81,$01,$04,$00,$02,$FF" will turn camera 1 on.
VERSION?	Query for the current version number of the NetLinx communication module.

Table 3 – Send Command Definitions

String Feedback

The virtual device will send string events to the NetLinx user interface module on change of states of the PRODUCTIONVIEW, and in response to queries by the program.

String	Description
DEBUG=<val>	Reports the debug status. <val>: 0 = off 1 = on DEBUG=1
VERSION=<val>	Reports the communication module version. <val>: string VERSION=1.5

Table 4 - String Feedback Definitions

Programming Notes:

Most of the feedback information is obtained from the packet replies.

Some features of the camera are dependent on other settings. For example if auto-focus is on then the focus can not be adjusted. Please read the camera's documentation for further details.

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.