NetLinx Module Interface Specification

for a

Digital Projection Titan and Lightning ‘i’ Series Projector

TABLE OF CONTENTS

Revision History

Date	Initials	Comments
10-22-10	DC	Initial release

Introduction

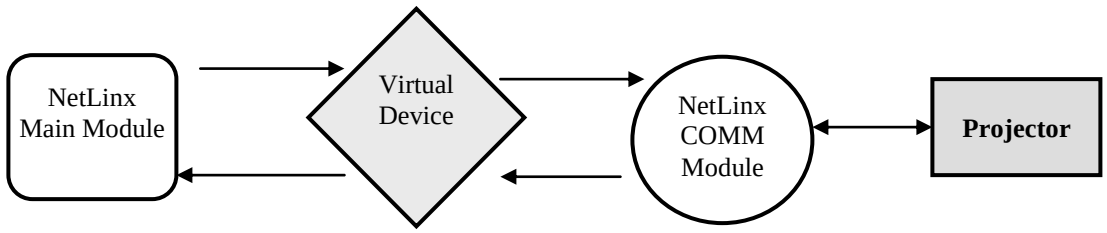
This is a reference manual to describe the interface provided between an AMX NetLinx system and a Digital Projection TITAN projector. The required communication settings are a baud rate of 19200, 8 data bits, 1 stop bit, no parity, and handshaking off. This module should work with most Digital Projection and Christie-Digital projectors with only minor revisions required for specific input and aspect commands. This module supports the two-lamp systems and HDMI input.

Overview

The COMM module translates between the standard interface described below and the **TITAN** serial protocol. It parses the buffer for responses from the PROJECTOR, sends strings to control the PROJECTOR, 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 TITAN module, the programmer must perform the following steps:

1. Define the device ID for the TITAN that will be controlled.
2. Define the virtual device ID that the TITAN comm module will use to communicate with the main program and User Interface. NetLinx virtual devices start with device number 33001.
3. The NetLinx TITAN 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 PROJECTOR 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
   dvTITAN          = 5001:1:0     // The TITAN connected to the NetLinx on 1st RS-232 port
   dvTITAN TPl      = 10001:1:0    // The touch panel used for output

   vdvTITAN         = 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 'DigitalProjection_Titan_Comm' mdlTITAN_APP(vdvTITAN, dvTITAN)
```

Upon initialization the AMX Comm module will communicate with the Digital Projection TITAN 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 UI module controls the video projector 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.

Level	Description
1	Lamp-Hours used lamp 1
2	Lamp-Hours used lamp 2

Command Interface

The UI module controls the PROJECTOR 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 PROJECTOR devices.

Command	Description
ASPECT=<value>	Sets the current aspect ratio. <value>: 1 = Native 2 = Fill 3 = User 4 = 1.33:1 5 = 1.78:1 6 = 2.35:1 7 = 1.66:1 8 = 1.85:1 ASPECT=2
ASPECT?	Request for current aspect ratio. ASPECT?
AUTO=<value>	Auto-adjust PC image <value>: 1 = Execute Auto-Adjust
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 = RGB1 2 = RGB2 3 = DVI 4 = SDI 5 = Composite 6 = S-Video 7 = Component INPUT=1
INPUT?	Request the current input setting. INPUT?

MUTE=<value>	Set the video mute state. <value> : 0 = Mute Off 1 = Mute On MUTE=1
PASSTHRU=<string>	Allows user the capability of sending commands directly to the TITAN 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 **TITAN** changes via string events. The strings supported are listed below.

String	Description
ASPECT=<value>	Reports the current aspect ratio. <value>: 1 = Native 2 = Fill 3 = User 4 = 1.33:1 5 = 1.78:1 6 = 2.35:1 7 = 1.66:1 8 = 1.85:1 ASPECT=2
INPUT=<value>	Reports the input source. <value> : 1 = RGB1 2 = RGB2 3 = DVI 4 = SDI 5 = Composite 6 = S-Video 7 = Component INPUT=3
LAMPTIME=<value>, <value>	Reports lamp usage in hours. (Updated every 5 minutes). Also available as levels 1 and 2. LAMPTIME=30, 31
MUTE=<value>	Reports the video mute state. <value> : 0 = Mute Off 1 = Mute On MUTE=1
POWER=<value>	Reports the power status. <value> : 0 power is off 1 power is on 2 cooldown 3 warmup 4 error condition POWER=2

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 PROJECTOR devices. Several of the TITAN 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.

Programming Notes

The COMM module imposes synchronous operation i.e. the COMM will not send a command to the TITAN 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 (1/2 second). The device is polled every 5 seconds to return the power state and every 10 seconds (when power is on) for 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.