NetLinx Module Interface Specification

for a

DirecTV Satellite Receiver

TABLE OF CONTENTS

Revision History

Date	Initials	Comments
12-17-07	DC	Initial release
12-20-08	DC	Added info on USB adapter

Introduction

This is a reference manual to describe the interface provided between an AMX NetLinx system and a DirecTV dss receiver. The required communication settings are a baud rate of 9600, 8 data bits, 1 stop bit, no parity, and handshaking off.

The new DSS receivers require the use of a USB to Serial adapter. The recommended adapter is the IOGEAR GUC232A adapter. Older DirecTV units used a baud rate of 115200, but all newer units use 9600. Also, the receiver should have the latest firmware loaded. DirecTV publishes a document that describes how to force a firmware update. Finally, the receiver should be on and through the boot-up process prior to installing the USB adapter.

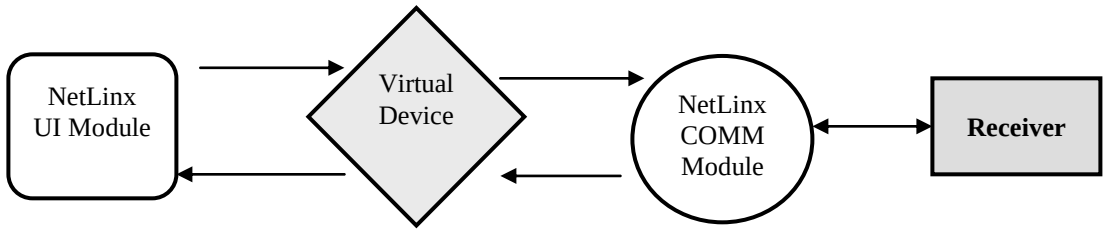
Also included in the distribution file is a program from DirecTV that may be used to test the communications from a PC.

Overview

The COMM module translates between the standard interface described below and the **DirecTV** serial protocol. It parses the buffer for responses from the dss, sends strings to control the dss, 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. This module is NOT intended for production use. The MAIN.AXS and user interface modules are provided to illustrate how to set up the communications and process the commands. The primary method used for control is through channel events.

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



Implementation

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

1. Define the device ID for the DIRECTV that will be controlled.
2. Define the virtual device ID that the DIRECTV 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 DirecTV.tpd and module (DirecTV_UI.axs) have been created for testing.
4. The NetLinx DIRECTV 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 dss 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
   dvSAT             = 5001:1:0     // The DIRECTV connected to the NetLinx on 1st RS-232 port
   dvDIRECTV TPl     = 10001:1:0    // The touch panel used for output

   vdvSAT            = 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 ‘DirecTV_Comm' mdlsat_APP(dvSAT, vdvSAT)

// Touch panel module
DEFINE_MODULE ‘DirecTV_UI' mdlsat_APP(vdvSAT, dvTPArray)
```

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

Channels

The UI module controls the DirecTV receiver 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
1	ON: Play OFF: Release button
2	ON: Stop OFF: Release button
3	ON: Pause OFF: Release button
4	ON: Skip forward OFF: Release button
5	ON: Replay (Jumps back 6 seconds) OFF: Release button
6	ON: Fast Forward OFF: Release button
7	ON: Rewind OFF: Release button
8	ON: Record OFF: Release button
10	ON: 0 digit button (simulates remote control "0") OFF: Release button
11	ON: 1 digit button (simulates remote control "1") OFF: Release button
12	ON: 2 digit button (simulates remote control "2") OFF: Release button
13	ON: 3 digit button (simulates remote control "3") OFF: Release button
14	ON: 4 digit button (simulates remote control "4") OFF: Release button
15	ON: 5 digit button (simulates remote control "5") OFF: Release button
16	ON: 6 digit button (simulates remote control "6") OFF: Release button
17	ON: 7 digit button (simulates remote control "7") OFF: Release button
18	ON: 8 digit button (simulates remote control "8") OFF: Release button
19	ON: 9 digit button (simulates remote control "9") OFF: Release button
20	ON: - button (simulates remote control "-") OFF: Release button
27	PULSE: Set system standby On
28	PULSE: Set system standby Off

45	ON: Move up (simulates remote control "arrow up") OFF: Release button
46	ON: Move down (simulates remote control "arrow down") OFF: Release button
47	ON: Move left (simulates remote control "arrow left") OFF: Release button
48	ON: Move right (simulates remote control "arrow right") OFF: Release button
49	ON: Select Button (simulates remote control "SELECT") OFF: Release button
50	ON: Enter button (simulates remote control "Enter") OFF: Release button
80	ON: Guide button (simulates remote control "Guide") OFF: Release button
81	ON: Active (simulates remote control "Active") OFF: Release button
82	ON: List button (simulates remote control "List") OFF: Release button
83	ON: Exit (simulates remote control "Exit") OFF: Release button
84	ON: Back (simulates remote control "Back") OFF: Release button
85	ON: Menu (simulates remote control "Menu") OFF: Release button
86	ON: Info button (simulates remote control "Info") OFF: Release button
87	ON: Red (simulates remote control "Red") OFF: Release button
88	ON: Green (simulates remote control "Green") OFF: Release button
89	ON: Yellow button (simulates remote control "Yellow") OFF: Release button
90	ON: Blue (simulates remote control "Blue") OFF: Release button
91	ON: CH UP (simulates remote control "Channel ^") OFF: Release button
92	ON: CH Down (simulates remote control "Channel V") OFF: Release button
93	ON: Prev (simulates remote control "Prev") OFF: Release button

Command Interface

The UI module controls the dss 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 DSS devices.

Command	Description
CHANNEL=<value>	Selects a channel. <value> : n go to channel number n CHANNEL=12
CHANNEL?	Request the current channel setting. See device notes relating to tuner queries. CHANNEL?
DEBUG=<state>	Set the debug state <state> : 0 OFF 1 ON DEBUG=1
DEBUG?	Request the debug state. DEBUG?
PASSTHRU=<string>	Allows user the capability of sending commands directly to the DIRECTV 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
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 **DIRECTV** changes via string events. The strings supported are listed below.

String	Description
CHANNEL=<value>	Reports the tuner channel. <value> : CHANNEL=4
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

Programming Notes

Because the DIRECTV responds very slowly to the serial commands, the COMM module imposes synchronous operation i.e. the COMM will not send a command to the DIRECTV 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 (such as channel changing commands that require one command for each digit). 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 query commands return the state that is currently stored in the COMM module. They do NOT send a new query 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 following the existing AMX standard. 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.)

The source code for the COMM module is also supplied, so that commands can be added directly to the module, and eliminate the need for using the PASSTHRU command.

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.

Discalimer and request for feedback & updates

If you do make changes to the package, please send an updated version back to dannycampbell@comcast.net, so I can incorporate your changes. I do not charge for the module, and at the same time do not provide any warranty as to the functionality of this software. It is distributed in a "use at your own risk" form.