Duet Module Interface Specification

for a

NigelB 45 Watts Stereo/Mono Amplifier (Plus Series)

TABLE OF CONTENTS

Date	Initials	Comments
05-08-2006	YS	Initial release

Introduction

This is a reference manual to describe the interface provided between an AMX NetLinx system and a NigelB 45 Watts Stereo/Mono Amplifier. The NigelB 45 Watts Stereo/Mono Amplifier supports an RS-232 serial protocol. The required communication settings are a baud rate of 9600**,** 8 data bits, 1 stop bit, no parity, and handshaking off**.** The cable for this device is FG#10-752-10.

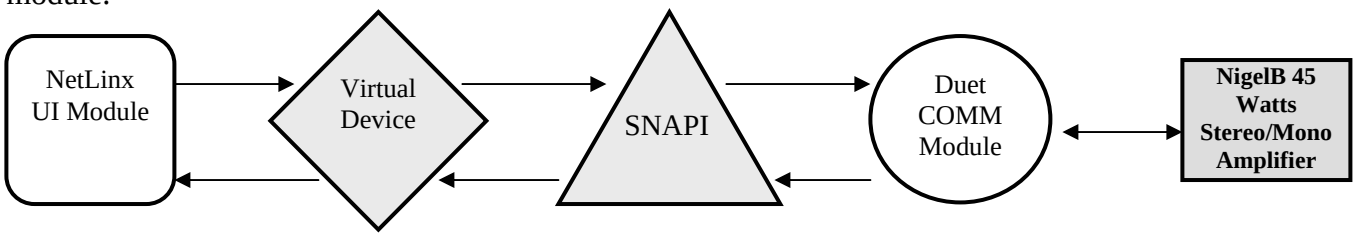
9 Pin Female D-SUB (AMX)		9 Pin Male D-SUB (Nigel B)	
5	GND	5	GND
2	RXD	2	TXD
3	TXD	3	RXD

Overview

The COMM module translates between the standard interface described below and the amplifier serial protocol. It parses the buffer for responses from the amplifier, sends strings to control the amplifier, 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 command responses for feedback.

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



Some functionality in the device interface may not be implemented in the API interface. In cases where device functions are desired but not API-supported, the PASSTHRU command may be used to send any and all device-protocol commands to the device. See the PASSTHRU command and the Adding Functions to Modules section for more information.

A sample UI module and a touch panel file are provided in the module package. These are not intended to cover every possible application, but can be expanded as needed by a dealer to meet the requirements of a particular installation.

Implementation

To interface to the AMX NigelB 45 Watts Stereo/Mono Amplifier module, the programmer must perform the following steps:

1. Define the device ID for the amplifier that will be controlled.
2. Define the virtual device ID that the NigelB 45 Watts Stereo/Mono Amplifier COMM module will use to communicate with the main program and User Interface. Duet virtual devices use device numbers 41000 - 42000.
3. If a touch panel interface is desired, a touch panel file NigelB_PlusAmp45W.TP4 and module (Amplifier Main.axs) have been created for testing.
4. The Duet NigelB 45 Watts Stereo/Mono Amplifier 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 amplifier to be controlled, and the virtual device ID for communicating to the main program.

Please see the Amplifier Main.axs file for an example

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

Port Mapping

This module uses multiple virtual devices in order distinguish events for one zone from another.

Port 1 is the master Volume and displays the average left and right volume.
When a volume change or mute are done on this port it will affect both left and right.

Port 2 is the left volume and only control the left volume.

Port 3 is the right volume and only control the right volume.

Channels

The UI module controls the amplifier 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
24	ON: Ramp Volume Up – used for feedback also OFF: Stop Ramping
25	ON: Ramp Volume Down – use for feedback also OFF: Stop Ramping
26	PULSE: Cycle Volume Mute
199	ON: Set Volume Mute On – used for feedback also OFF: Set Volume Mute Off
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

Table 1 - Virtual Device Channel Events

Levels

The UI module controls the amplifier 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	Volume Level (range 0…255) (control and feedback)

Table 2 - Virtual Device Level Events

Command Control

The UI module controls the amplifier via command events (NetLinx command *send_command*) sent to the COMM module. The commands supported by the COMM module are listed below.

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

Command	Description
?DEBUG	Request the state of the debug feature. ?DEBUG
DEBUG-<value>	Set the state of debugging messages in the UI module and the Comm. module. **Note:** See Programming Notes section. <value> : 1 = set only error messages on 2 = set error and warning messages on 3 = set error, warning & info messages on 4 = set all messages on DEBUG-1
REINIT	Re-initializes the communication link and data. **Note:** This command deletes any messages waiting to go out to the device. REINIT
?VERSION	Query for the current version number of the Duet module. ?VERSION

Table 3 – Send Command Definitions

Command Feedback

The COMM module provides feedback to the User Interface module for amplifier changes via command events. The commands supported are listed below.

PLEASE NOTE: Feedback is only provided when there is a state change. If no state change resulted from the command sent in, then no feedback will be returned.

Command	Description
DEBUG-<value>	Returns the state of debugging messages in the UI module and the Comm. module. <value> : 1 = set only error messages on 2 = set error and warning messages on 3 = set error, warning and info messages on 4 = set all messages on DEBUG-1
VERSION-<version>	Reports the version number of the module. <version> : xx.yy.zz = module version number VERSION-1.0.0

Table 4 - Command Feedback Definitions

Device Notes

- The only functionality the device's protocol supports is set volume for left and right only. Ramp up/down, mute and cycle mute for right, left and master where all faked.
- The device only has 2 channels (left and Right), the master channel was added by the module to make it easer for the user to change both left and right channels at the same time, same for muting.
- If the left and right channels have different values, ramping up/down the master will ramp both left and right equally and once either of the channels reaches it's max/min the other channel will keep increasing/decreasing until it also reaches it's max/min.
- On the other hand when setting the volume on the master, both left and right channels will become the master's volume regardless of their previous values.

Programming Notes

- The module will query for the volume on boot up and every poll.
- Polling is defaulted at every 5 seconds.
- The module uses a multi level queuing and de – queues messages every 200 ms.

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-' command like checksum calculation.)

Responses from the device

The module will automatically interpret replies from the device and pass these on to the application code according to the documented API. Some device replies may not be passed on to the application code. To see all replies from the device unfiltered by the module, enable PASSBACK and use a DATA_EVENT with a string handler in the UI code. Again, the device-specific protocol must be known in order to interpret these responses. Even when PASSBACK is enabled, the module will still interpret device responses according to the standard API.